Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Bachelor Thesis

Design and Development of a mobile application for attendance checking with QR code

Pech Menghour

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Menghour Pech

Informatics

Thesis title

Design and Development of a mobile application for attendance checking with QR code

---

Objectives of thesis

This bachelor thesis deals with the design and development of a mobile application that allows students to check attendance by scanning QR codes. In this application, students must enable location to allow verification. The main objective is to check if they are in the right place during attendance. Then only the students who are in the correct location will be successfully marked as present in the system.

Methodology

The work is divided into a theoretical and a practical part. In the theoretical part, technologies, features of IDE and principles used in the practical part are presented. Android Studio is used as IDE, which is implemented in Java. Selected components of the Android architecture and the Android framework are presented.

In the practical part, the requirements for the application are first defined and then a data model is created. Then the implementation of selected application functions is described and examples of the corresponding source codes are given. We also implement how the mobile application and the website communicate via the database system. Finally, the developed solution is tested and suggestions for further improvements are made.

The proposed extent of the thesis

35-40 pages

Keywords

Java, XML, Mobile Application, Android Studio, HTML, CSS, JavaScript, server side, client side

---

Recommended information sources

Hedgpeth, Robert. R2DBC Revealed : Reactive Relational Database Connectivity for Java and JVM Programmers, Apress L. P., 2021. ProQuest Ebook Central, https://ebookcentral-proquest-com.infozdroje.czu.cz/lib/czup/detail.action?docID=6533343.

Saxena, Swati. Java, BPB Publications, 2019. ProQuest Ebook Central, https://ebookcentral-proquest-com.infozdroje.czu.cz/lib/czup/detail.action?docID=6891836.

S, Dani Akash. JavaScript by Example. Birmingham: Packt Publishing, Limited, 2017. Accessed April 30, 2022. ProQuest Ebook Central.

Späth, Peter, and Friesen, Jeff. Learn Java for Android Development : Migrating Java SE Programming Skills to Mobile Development. Berkeley, CA: Apress L. P., 2020. Accessed April 30, 2022. ProQuest Ebook Central.

Expected date of thesis defence

2022/23 SS – FEM

The Bachelor Thesis Supervisor

Ing. Jiří Brožek, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 31. 10. 2022

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 30. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 13. 03. 2023

Declaration

I have worked on my bachelor thesis titled "Design and Development of a mobile application for attendance checking with QR code" by myself, and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the idea does not break any copyrights.

In Prague on 15th March 2023 _____ Pech Menghour

Acknowledgement

I express my sincere gratitude to my supervisor Ing. Jiří Brožek, Ph. D., who gave me the golden opportunity to do this work under his supervision and provided invaluable guidance. It would not have been possible for me to complete this work without his advice and encouragement.

Design and Development of a mobile application for attendance checking with a QR Code

Abstract

The thesis project aims to develop an android mobile application that allows students to scan a provided QR Code for attendance checking by verifying the student's current location with the QR Code provider's location, which is stored in a database. If the location matches or is in the acceptance range provided, students can submit their attendance.

The main component of this thesis project is an android mobile application, but a complete system must be constructed for it to function correctly. This system must include a web application developed using EJS, CSS, and JavaScript, a server developed using the Express.js framework connected to the NoSQL database MongoDB, and an android mobile application implemented using Java in Android Studio IDE. The implementation of the Android mobile application includes core importing libraries for Android, such as the Code scanner library and Google Maps API.

The theoretical section in the first part defines various technologies, frameworks, and libraries for Android, including the NoSQL database MongoDB and the RESTful API technology used as the server on our system. The system development and mobile application development phases are covered in the practical part, along with the development structure and examples of pertinent source codes to justify the implementation phases. The final portion of this paper includes testing, improvements, and advantages of utilising the system.

Keywords: QR Code, Android Mobile Application, Java, XML, Client-side, JavaScript, HTML, EJS, CSS, Server-side, Express.js, MongoDB, NoSQL.

Návrh a vývoj mobilní aplikace pro kontrolu docházky pomocí QR kódu

Abstrakt

Cílem diplomové práce je vyvinout mobilní aplikaci pro Android, která studentům umožní naskenovat poskytnutý QR kód pro kontrolu docházky ověřením aktuální polohy studenta s polohou poskytovatele QR kódu, která je uložena v databázi. Pokud se umístění shoduje nebo je v uvedeném rozsahu přijatelnosti, studenti mohou odeslat svou účast.

Hlavní součástí tohoto projektu je mobilní aplikace pro Android, ale pro její správné fungování je nutné zkonstruovat kompletní systém. Tento systém musí obsahovat webovou aplikaci vyvinutou pomocí EJS, CSS a JavaScript, server vyvinutý pomocí frameworku Express.js propojený s NoSQL databází MongoDB a mobilní aplikaci pro Android implementovanou pomocí Javy v Android Studio IDE. Implementace mobilní aplikace pro Android zahrnuje základní importní knihovny pro Android, jako je knihovna Code scanner a Google Maps API.

Teoretická část v první části definuje různé technologie, frameworky a knihovny pro Android, včetně NoSQL databáze MongoDB a technologie RESTful API používané jako server v našem systému. V praktické části jsou popsány fáze vývoje systému a mobilní aplikace, spolu s vývojovou strukturou a příklady příslušných zdrojových kódů pro odůvodnění fází implementace. Poslední část tohoto dokumentu obsahuje testování, vylepšení a výhody používání systému.

Klíčová slova: QR kód, mobilní aplikace pro Android, Java, XML, na straně klienta, JavaScript, HTML, EJS, CSS, na straně serveru, Express.js, MongoDB, NoSQL.

Table of content

1 Introduction

Checking attendance is one of the options that teachers can use to track if theirs students attend sessions regularly. The record of attendance might be beneficial for teachers to set a maximum number of absences students can miss, upgrade final exam grades, and give a bonus point based on their attendance, etc.

As one of the students, I used to see a couple of traditional ways of attendance tracking, such as passing an attendance paper from one student to another for signing, calling a student's name one by one, etc. However, this is not an efficient way to do so since we are now living in a modern technology generation. It does not provide accuracy in cheating prevention and takes time. That is why the idea of developing attendance checking on mobile with a QR code will guarantee that students must be within the permissible range of the QR code provider's location, and students can simultaneously submit their attendance.

The present smartphone includes a camera, GPS, and internet connectivity as its minimum three basic features; this creates the opportunity to construct an android mobile application that might address the problem at hand.

The implementation of this mobile application and the entire system will be covered in this project, along with the usage of Java, new Android XML capabilities, and other frameworks and technologies.

2 Objective And Methodology

2.1 Objective

The main goal of this work is to implement an Android mobile application for attendance checking by scanning provided QR Code during the session, which could ensure that a student who can submit attendance must be within the permissible range of the QR code provider's location. This project will consist of an android mobile application as a core of this work, a web application and a NoSQL database deployed on a web server. This thesis will also discuss new features and APIs of android for developers, including additional frameworks and technologies that were utilised to develop our whole system, including QR Code, Express.js, EJS, and MongoDB, a NoSQL database.

2.2 Methodology

The thesis methodology is based on researching relevant documentation and credible scientific sources of information about the Android platform. The Developers Android Documentation website is the leading and most trustworthy source of information and guidelines; additional sources include books about the technologies and ideas used.

This project is divided into theoretical and practical parts. The three components of the system are examined separately in each piece. The theoretical part covers the technologies and frameworks based on the composite of learned documentation and book references. The mobile application and the system architecture are explained on practical in detail, along with the methods used and source code for each component, the development phases, and a solution for each problem.

The final portion of this work includes testing, improvements, and advantages of utilising the system.

3 Literature Review

3.1 Client-side

Prior to progressing, it is essential to comprehend the definition of the client-side. This term pertains to the software programs that operate on the device of the client or user, particularly within their web browser, enabling interactive engagement by the user (Team, Indeed Editorial, 2021).

When building a site, HTML (Hypertext Markup Language) is one of the core technologies for defining a Web page structure, along with CSS (Cascading Style Sheets) for various devices (W3C Corporate Team, n.d.). Considering a programming language is also essential for making a web page responsive. When discussing web development, JavaScript is the first programming language that comes to mind. Angular, React, and Vue.js are the JavaScript libraries and frameworks developed over the years. These frameworks can also use a higher level at build time compiled into imperative JavaScript code, which is then used to execute the actual code on the browser. According to the latest RADEX reports, among the 1.8 billion websites in the world by 2022, JavaScript, including its framework, will be used by 98% of them (Raval, 2022).

3.1.1 HTML & EJS & CSS

- **HTML**

HTML is a standard markup language for creating web pages. It enables the developing and structuring of sections, paragraphs, and links through HTML elements such as tags and attributes (S., 2023). Many versions of HTML have been released since the first version (HTML 1.0) was first created by “Tim Berners-Lee” in 1991. The current version of HTML is HTML5.

HTML gives authors the means to:

- Publish online documents with headings, text, tables, lists, photos, etc.
- Retrieve online information via hypertext links at the click of a button.
- Design forms for conducting transactions with remote services to search for information, make reservations, order products, etc.
- Include spreadsheets, video clips, sound clips, and other applications directly in their documents. (W3C Corporate Team, n.d.)

- **EJS**

Ejs.co, the official site, defines *"EJS is a simple templating language that lets you generate HTML markup with plain JavaScript. No religiousness about how to organise things. No reinvention of iteration and control flow. It is just plain JavaScript."* (EJS official site, n.d.)

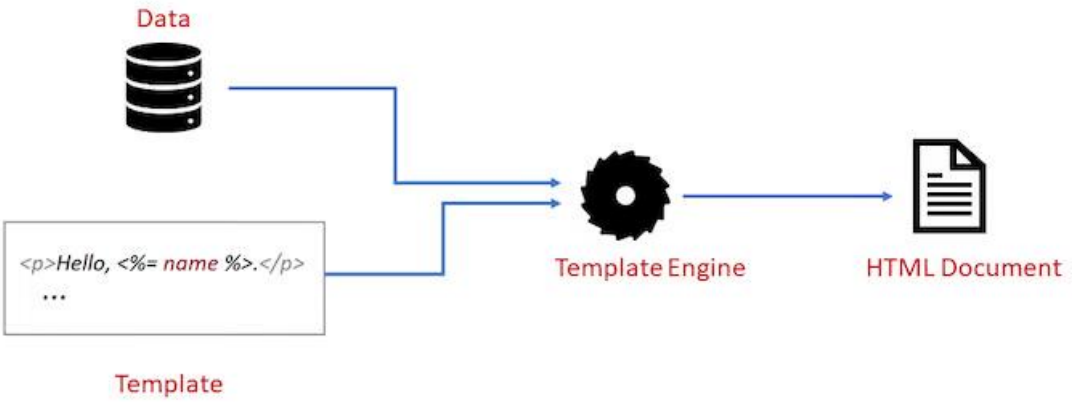

Figure 1 Template Engine EJS into HTML (Olusola, 2021)

To simplify, the stated template engine facilitates the embedding of JavaScript code within a template language, which generates HTML. Additionally, it features its syntax and tags. Utilizing EJS instead of HTML is advantageous when designing a dynamic page that includes data from multiple sources. The template can be bound with the dynamic data, and the browser will be provided with the final output in HTML (Braktim99, 2021). The process of the template engine EJS transforming into HTML is illustrated in *Figure 1*.

EJS could be used on both the client and server sides of rendering. However, in most cases, EJS is used as a template engine by Node.js. Regarding the server side, this topic will go into detail.

- **CSS**

According to w3.org, the official site defines *"CSS is the language for describing the presentation of Web pages, including colours, layout, and fonts. It allows one to adapt the presentation to different devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language."* (W3C Corporate Team, n.d.)

To clarify, CSS does not involve the web page structure. However, it works with HTML to provide the best UI for a web page, such as various colours, responsiveness to multiple screen sizes and more. The main objective was to distinguish between the content of documents and their presentation, which includes design elements like colour, layout,

and fonts. CSS handles an internet page's design and aesthetics (Pulkitagarwal03pulkit, 2020).

3.1.2 JavaScript

In brief, JavaScript is an interpreted programming language that does not require prior compilation of the source code before it is transmitted to the browser. The responsibility of executing the JavaScript source code lies with the browser engine (Hiwarale, 2018). As mentioned above, JavaScript is a popular scripting language for the web. However, how does it become so?

A brief history, they invented JavaScript because they first wanted to create richer user interactions rather than just a hyperlink that connected documents, mainly driven by the desire to save roundtrips for simple tasks such as form validation. In 1995, Brendan Eich introduced LiveScript, which became JavaScript at Netscape. Eventually, there was an effort to standardise the various implementations of the language, which is how ECMAScript was born. ECMA (European Computer Manufacturers Association) created the standard ECMA-262, which describes the core part of the JavaScript programming language without browser and web pager-specific features.

JavaScript encompasses three pieces:

- ECMAScript: the core language, including variables, functions, loops, etc. ECMAScript is independent of the browser and can be used in many other environments.
- Document Object Model (DOM): this provides ways to work with HTML and XML documents.
- Browser Object Model (BOM): it provides access to items that expose the characteristics of the web browser. (Stoyan Stefanov, Kumar Chetan Sharma, 2013)

Prior to the release of Node.js, JavaScript was limited to running solely on a browser engine. The following chapter will delve into Node.js in greater detail. Our current focus is gaining a deeper comprehension of how the JavaScript source code operates atop a browser. It is important to note that various browsers utilize different JavaScript engines, such as Chrome V8, a well-known JavaScript engine used by Google Chrome, SpiderMonkey engine utilized by Firefox, Chakra engine used by Internet Explorer, and WebKit engine utilized by Safari.

The subsequent area of focus pertains to an analysis of the functionality of JavaScript source code within the context of a browser. Historically, JavaScript was solely classified as an interpretation-based programming language. Nevertheless, in present times, many JavaScript engines adopt a hybrid approach, which combines interpretation with compilation through the utilization of JIT (Just in Time) compilation.

First, JavaScript source code is parsed into Abstract Syntax Tree (AST), which means splitting up each line of code into language-relevant components, such as constant and function keywords, and then saving all these pieces into a form of a node in a properly structured manner. At this step, it does also check syntax errors. If nothing occurs, the final tree result is utilised to create machine code (Siddiqui, 2022).

The created AST from the previous step is then compiled into machine code. Then, this machine language gets executed. However, the JavaScript Engine's call stack is where machine code is executed. During the execution, other un-optimized machine code must be recompiled and optimised in the background. This is because the JavaScript engine, beginning after JavaScript source code turned into AST, creates an up-optimized code that allows a script to be executed as fast as possible. This is a general process of the JIT JavaScript engine. However, Different JavaScript engines have other internal optimising processes (Siddiqui, 2022). The entire process of turning from readable JavaScript source code into Bytecode is shown in *Figure 2.*

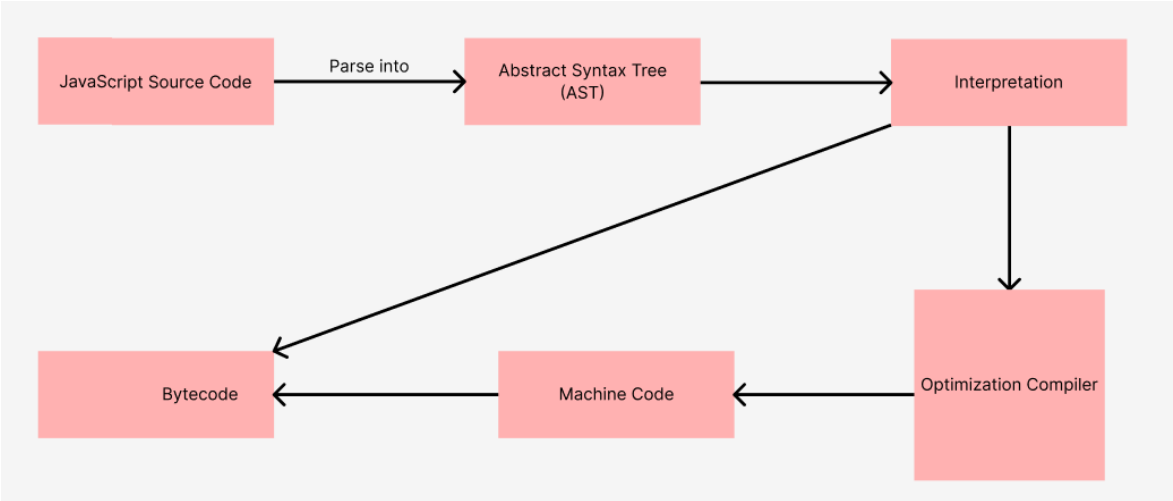

Figure 2 Just-In-Time compilation JavaScript (by author)

3.1.2.1 jQuery

The official site of jQuery (jQuery.com) defines *"jQuery is a fast, small, and feature-rich JavaScript library. It simplifies things like HTML document traversal and manipulation, event handling, animation, and Ajax with an easy-to-use API that works across many browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript."* (jQuery Corporate team, n.d.)

In a simple explanation, jQuery is a JavaScript library that provides developers not only the shortest lines of code to traverse a Document Object Model (DOM) but also much more than that. This article aims to explore the capabilities of jQuery.

- Access elements in a document in shortage syntax with leverage knowledge of CSS selectors. Generally, the element selection syntax is `$('').`

  For example: `$('.container').find('p');`
- Modify the appearance of a web page. jQuery fills in the gaps between CSS and the same unsupported standards because jQuery relies on the same standard across browsers in short consistency. jQuery can also alter a class or isolated style to an element of a document even after pages have been rendered.

  For example: `$('ul > li:first').addClass('active');`
- Alter the content of a document. jQuery not only can modify the content of a document, such as texts, images, and list order, and rewrite and extend the whole HTML structure with its API.

  For example: `$('.container').append('<p>Hello World<p/>');`
- Respond to a user's interaction. The jQuery library offers ingenious ways to head off various events by providing a list of reusable event functions and API.

  For example: `$('div.element').show();`
- Animate changes being made to a document. jQuery provides a list of effects and transitions, such as fades and wipes.
- One of the cool features is that it allows information retrieved from the server with a page refreshing. This pattern is known as Ajax, which stands for Asynchronous JavaScript and XML.

  For example: `$('div.element).load('more.html #content');`

- In some cases, jQuery could also simplify basic JavaScript tasks. For example, a basic JavaScript construct for iteration and array manipulation. (Jonathan Chaffer, Karl Swedberg, Karl Swedberg, 2013)

3.1.2.2 Axios

"Axios is a promise-based HTTP Client for node.js and the browser. It is isomorphic (= it can run in the browser and node.js with the same codebase). On the server side, it uses the native node.js http module, while on the client (browser), it uses XMLHttpRequests." (Axios Official Corporate team, n.d.)

Axios is also a JavaScript library built on top of Ajax, relies on XMLHttpRequests wrapped in a friendly way and can make an HTTP request from both the browser client side and the node.js server side. Several options allow making an HTTP request, such as Fetch API, GraphQL, and SuperAgent. However, a couple of features make Axios unique from others.

Core features of Axios:

- Make XMLHttpRequests from the browser. To use Axios on the browser, including this library by Using jsDelivr CDN or unpkg CDN like the other JavaScript library.
- Make http requests from node.js.
- Support the Promise API. Therefore, code can be written better and cleanly through async/await.
- Request and response interception enables access to request headers, bodies, and response headers.
- Request and response transforming uses data property.
- Timeout for response and cancel requests
- Data transforming automatically in JSON form without adding .json() conversion.
- Providing support on the client side against Cross-site request forgery. (Axios Official Corporate team, n.d.)

3.2 Server-side

The server side in web development refers to the web server. Generally, a web browser is a place where a file or data is displayed to a user. The question is, “How does a web browser get this needed file or data?”.

The short answer to this question is “Web Server”. Web browser and Web server communicate via HTTP (Hypertext Transfer Protocol). When a file or data is needed on the browser, the browser sends an HTTP request to the server. After receiving a request, the server proceeds to find the requested documents. Then send a response to the browser by HTTP Response (MDN contributors, 2023). The essential communication between a Web server and a Web browser is illustrated in *Figure 3*.

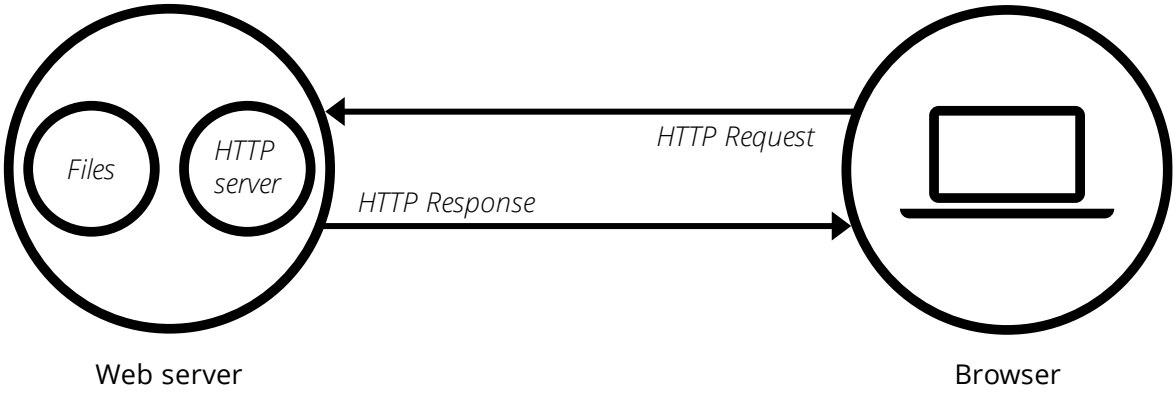

Figure 3 Communication of Web Server and Web Browser (MDN contributors, 2023)

There are two types of web servers: static and dynamic. To publish a website, either one of them is needed. However, the selection is based on the requirement of a website. Static web servers comprise a computer (hardware) and an HTTP server (software). It is called "static" because your browser receives the hosting files as they are.

On the other hand, the dynamic web server consists primarily of a static web server and extra software, namely an application server and a database. Before sending a response file or content back to the browser, an application server first pull-out data from a database and then dynamically update content on the hosting file with getting data. Finally, a response sends back to the client (MDN contributors, 2023).

3.2.1 Node.js

“Node.js is an existing new platform for developing web applications, application servers, any network server or client, and general-purpose programming. It is designed for

extreme scalability in networked applications through an ingenious combination of server-side JavaScript, asynchronous I/O, and asynchronous programming. (Herron, 2020)*"*

A brief history, Ryan Dahl first wrote Node.js in 2009. It combines a V8 JavaScript engine, a low-level I/O API, and an event loop in separate modules, which implement in C/C++ (Magaji, 2020).

Before Node.js, JavaScript was a popular programming language for front web development. However, it was only possible to use JavaScript in the browser, not anywhere else, especially for a web server, to make a complete website. Therefore, Developer had to go for a traditional approach architecture using other programming languages such as Java, PHP, Python or Ruby on Rails (Herron, 2020).

The arrival of Node.js has brought a massive change in web development, primarily JavaScript. JavaScript has become a full-stack web development programming language. Furthermore, Node.js architecture deviates from a typical decision from application platforms where threads are frequently utilised to grow an application to occupy the CPU; Node.js forgoes threads due to their inherent complexity. According to these, single-thread event-driven systems provide a small memory footprint, fast performance, a better latency profile under load, and a more basic programming style than others (Herron, 2020).

It is possible to develop JavaScript applications outside of a web browser using the Node.js framework. As a result, certain browser-specific functionalities are unavailable in Node.js. For instance, Node.js does not offer an HTML DOM. To accomplish tasks, Node.js provides numerous built-in and installed modules, in addition to its inherent capability to execute JavaScript code. The next chapter will cover Node.js modules.

3.2.1.1 Node Modules

"Modules are the basic building blocks for constructing Node.js applications. A Node.js module encapsulates functions, hiding details inside a well-protected container and exposing an explicitly declared API." (Herron, 2020)

As mentioned, the Node.js module can be a built-in or an installed module. A developer could load a built-in core module without installation into a project. On the other hand, an installed module needs to be downloaded or installed via npm (Node Package Manager) into a project before it is available for loading.

For example, to install Express, use the command: `npm install express`

The Node.js module can be regarded as either a JavaScript library or a class in another object-oriented programming paradigm. Consequently, it is possible to categorize associated functions or details within a separate module, resulting in a reduction of the global object and enhancing its reusability.

There are two types of Node.js module systems, CommonJS modules and ECMAScript modules, which are used nowadays. The first method of bundling JavaScript code for Node.js was through CommonJS modules. The reserved keyword to pull in the node.js module is *"require"*. Given a module identification, the *require* function looks for the module named by that identifier. If it does, it locates the module definitions and loads them into the Node.js runtime, making their functions accessible. For a best practice, a declaration of a loaded module should be *const* to avoid mistakenly reassigning value to this variable (Official Node.js Corporate team, n.d.).

For example: `const http = require('http');`

The official standard format for storing JavaScript code for reuse is ECMAScript modules. There are numerous import and export lines used to define modules. However, the ECMAScript modules approach is widely used in browsers because browsers do not support CommonJS (Official Node.js Corporate team, n.d.).

For example: `import { addTwo } from './addTwo.mjs';`

3.2.1.2 Express.js

"Express is a minimal and flexible Node.js web application framework that provides a robust set of web and mobile applications features." (Expressjs Official Corporate team, n.d.)

There are several common tasks in web development that Node.js does not support by nature. For instance, if a developer wants to construct unique route handling for various HTTP, process requests independently for different URL paths, serve static files, or utilise templates to create the response dynamically; Node.js will not be very useful. The first option is to write our code from scratch, or the second option, which is best to go, is to choose a web framework to avoid reinventing the wheel (MDN contributors, n.d.).

Express.js is one of the most famous Node.js web application frameworks, and it is a Node.js module available through the *npm* registry. Therefore Express.js needs to be installed via the *npm* command on the terminal, like other modules.

Here are the core features that Express provides mechanisms to:

- By implementing handlers for requests with different HTTP verbs at distinct URL paths (routes), it becomes possible to segregate request routes into a separate module, leading to the creation of a more organized main app.js file.
- Integrate with the view template engine that allows data to be inserted dynamically into a template. Several template engines are available for Express, such as EJS, mentioned previously, Vash, Haml and Handlebars.
- Identify the port to connect and the location of templates used to render responses.
- Any point in the request handling pipeline can be extended with additional "middleware". (MDN contributors, n.d.)

3.2.1.3 Middleware Function

"Middleware functions are functions that have access to the request object (req), the response object (res), and the next function in the application's request-response cycle. The next function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware." (Expressjs Official Corporate team, n.d.)

A developer can build a pipeline or series of processes for a single request route using middleware functions. If the current middleware function succeeds, the *next* function in this middleware function will be invoked. As a cycle, the next middleware function in the pipeline will follow the same tasks until the final function is reached. For instance, in a use case, that web application required authentication before users could see their home page. Validation user's token is required as a middleware function. It proceeds to verify the user's token. If the token is valid, the next function in this middleware invokes and passes control to the following middleware until reaching the final function to send a response back to the client.

Middleware functions could be used to perform the following tasks:

- To execute any code in the sequence of pipeline
- To modify the request and response objects
- Finish the request-response cycle if there is no *next* function added
- Call the next middleware in the stack (Expressjs Official Corporate team, n.d.)

3.2.2 RESTful APIs

Generally, an application programming interface (API) specifies the rules for interacting with other software systems. By exposing or creating APIs, developers enable other applications to communicate with theirs.

Representational State Transfer, or REST, is a software architecture that specifies how APIs must work. Initially, REST was designed to manage communication over a complex network like the internet. The REST-based architecture supports high-performance and reliable communication at scale (AWS Official Corporate team, n.d.).

Express.js is a Node.js backend framework for building RESTful APIs. So, what is RESTful APIs?

A RESTful APIs allows two computer systems to exchange information securely over the Internet. It is common for business applications to communicate with other internal and third-party applications to perform various tasks. It is called REST APIs when APIs follow a REST architectural style. A RESTful web service is a web service that implements the REST architecture. A RESTful API is generally a RESTful web API. Although often used interchangeably, REST API can also be called a RESTful API (AWS Official Corporate team, n.d.).

The following illustrates the core features of the REST architecture:

1. Uniform interface
2. Stateless
3. Layered system
4. Cacheable
5. Client-server architecture
6. Code on demand (Optional) (Lahoti, 2019)

Below are the general steps on how RESTful APIs work:

1. The client requests the server if a resource is needed. The request format must follow API documentation so that the server can understand.
2. The server verifies the client's identity and validates that the client is authorised to make the request.
3. The request is received by the server and processed inside.

4. The server responds to the client. The answer comprises information that tells the client whether the request was successful. The response also provides any information that the client requested. (AWS Official Corporate team, n.d.)

A valid request format from a client in REST APIs contains:

1. Unique resource identifier: The server commonly conducts resource identification for REST services by utilising a Uniform Resource Locator (URL), which can be called an endpoint.
2. Method: One from CRUD Operation (GET, PUT, POST, DELETE), including PATCH.
3. HTTP headers: a metadata exchange between client and server.
4. Data: Data is primarily included in the POST and PUT.
5. Parameters: Give the server more information about what needs to be done; this information is typically included in the URL. The parameters could be cookie parameters for quick client authentication, query parameters requesting more information about the resource, or path parameters that provide URL specifics. (AWS Official Corporate team, n.d.)

3.2.3 Mongoose

"Mongoose is a Node.js-based Object Data Modelling (ODM) library for MongoDB. It is akin to an Object Relational Mapper (ORM) such as SQL Alchemy for traditional SQL databases." (Ado Kukic, Stanimira Vlaeva, 2021)

Using Mongoose has the advantage of providing a schema to work within an application code and a clear connection between MongoDB documents and the Mongoose models in the application. It enables quick and easy creation and management of data relationships. Although using Timestamps is relatively simple, it is nevertheless essential to highlight. Change "timestamps" to "true" when specifying a schema to turn on the feature. Timestamps will show when the model was created and last updated. Furthermore, Model definition and schema validation are handled by Mongoose in one step. Based on the type defined in mongoose schema, it does validate before saving to MongoDB.

3.2.4 MongoDB

"MongoDB is a document database with the scalability and flexibility you want with the querying and indexing you need." (MongoDB Official Corporate team, n.d.)

The term document database means storing data in tables and rows to JSON objects instead of storing data in tables and rows. MongoDB is a so-called nonrelational database management system or NoSQL database. As one of the most established NoSQL databases, MongoDB offers advanced features such as data aggregation, ACID transactions (Atomicity, Consistency, Isolation, Durability), horizontal scaling, and charts. Furthermore, data is essential to making data-driven business decisions, explicitly storing, analysing, and visualising the data (Amit Phaltankar, Juned Ahsan, Michael Harrison, and Liviu Nedov, 2020).

Due to these reasons, MongoDB has been used by more than 35,000 customers, such as Toyota Material Handling Europe Creates a Smart Factory, Vodafone, Expedia, KPMG, Royal Bank of Scotland, and Barclays (MongoDB Official Corporate team, n.d.).

Feature	Relational Database	Nonrelational Database
Schema	Relational databases have a strict schema that requires tables to define all desired columns and their types. Any attempt to manipulate data that does not adhere to the schema results in an error.	Non-relational databases store unstructured and dynamic data without a rigid schema.
Data model/ Storage Structure	Data in a relational database is organized in tables with each record represented as a row containing information about all the columns. Modifying a table can have repercussions on other tables and applications.	Data is stored in various formats depending on the provider, such as documents, graphs, key values, and wide columns. The database is designed to adapt to dynamic data and seamlessly integrate with the application without requiring any modifications.
Normalisation	Normalization removes duplicate data and prevents data anomalies. A relational database utilizes normalization to avoid data redundancy, requiring the storage of data in separate tables with established relationships between them.	NoSQL databases focus more on fast data retrieval, and the data can be denormalisation.
Scaling	Scaling is the database's ability to grow or shrink as needed. Relational databases are challenging to scale and are usually scaled vertically, which increases the computing and storage of the machine.	NoSQL databases provide both vertical and horizontal scaling. The data can be distributed in horizontal scaling across different machines/ clusters.

Table 1 Differences between relational and nonrelational databases (Amit Phaltankar, Juned Ahsan, Michael Harrison, and Liviu Nedov, 2020)

3.3 Android mobile application

Mobile phones and tablets use Android OS, an operating system built on a Linux foundation. The Android platform includes an operating system built on the Linux kernel, a GUI, a web browser, and end-user applications that can be downloaded. Although early Android demonstrations used a basic QWERTY smartphone with a vast VGA screen, the operating system was created to run on reasonably priced handsets with conventional numeric keypads. Android has been used and developed over two decades after a project started in 2003 by the American technology company Android Inc (Mixon, 2022).

The building blocks of an Android app are called app components. An app's components serve as entry points for the user or system. Components are interdependent. There are four different main types of app components:

- Activities
- Services
- Broadcast receivers
- Content providers (Android Official Corporate team, 2023)

3.3.1 Android App components

3.3.1.1 Activities

Interacting with the user begins with an activity. In this case, it represents a single screen with an interface. For example, a YouTube app might have one activity showing a playing video and a list of related videos, a search option. Each activity in the YouTube app is independent of the others, even though they all work together to create a seamless user experience. Hence, another app may begin any of these tasks if this app permits (Android Official Corporate team, 2023).

An activity is a crucial interaction between a system and an app. The following are the function of activity in android:

- To ensure that the system remains running the process hosting the activity, keep track of what the user is currently interested in (what is displayed on the screen).
- The user tends to prioritise retaining processes that contain activities they may need to return to, and these processes may have been used previously.
- Assisting the application in managing the situation where its process is terminated so that the user can resume their activities with the previous state being restored.

- Enabling applications to incorporate user flows between one another and facilitating the system in managing and organising these flows. (Android Official Corporate team, 2023)

An application comprises one or multiple activities, and each activity class has its lifecycle with six fundamental callbacks: *onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy()*. The system triggers the corresponding callback when an activity transitions into a new state.

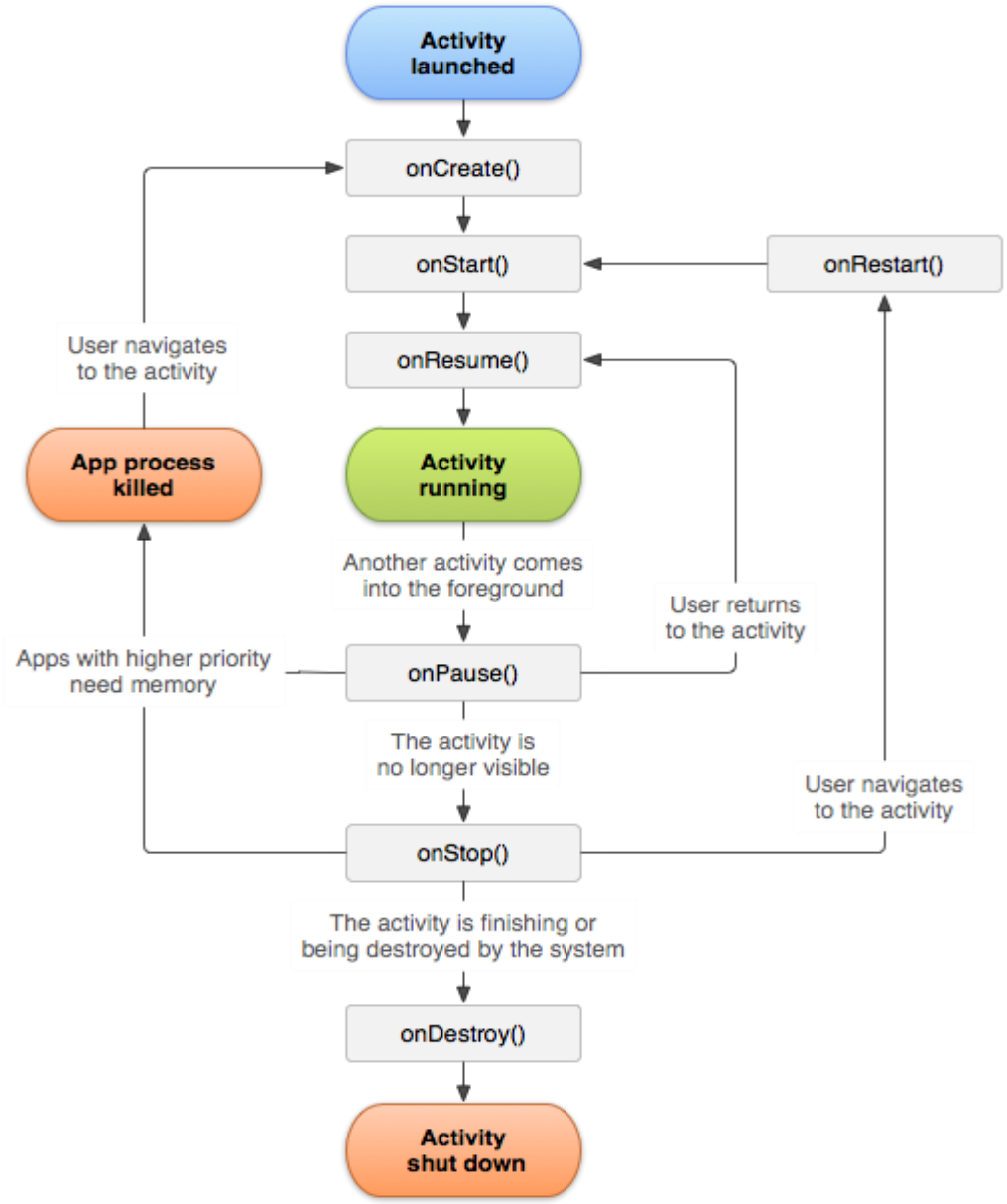

Figure 4 Activity Lifecycle model (Android Official Corporate team, 2022)

3.3.1.2 Services

A service is a versatile means to keep an application active in the background for various purposes. It functions as a background component to execute lengthy operations or to assist with tasks for external processes. Notably, a service does not need a user interface (Android Official Corporate team, 2023).

Even if an application is closed or the user switches to another application, a service can operate continuously in the background. Additionally, application components can link to a service to facilitate inter-process communication (IPC). There are three types of Services in android: Foreground Services, Background Services and Bound Services. However, among these three types, each can follow one of two paths to complete its life cycle: Started or Bounded.

- Started Service: When an application component calls the *startService()* method, it triggers the initiation of the service. Once the service has been initiated, it can continue to run in the background even if the component responsible for starting the service is destroyed.
- Bounded Service: The Started path of service can be likened to a server in a client-server interface. Android application components can send requests to the service using this path and receive results. On the other hand, a service is considered Bounded when an application component binds to it using the *bindService()* method. To halt the execution of a Bounded service, all components must unbind from the service using the *unbindService()* method. (RISHU_MISHRA, 2020)

3.3.1.3 Broadcast receivers

An Android BroadcastReceiver is a passive component of the Android system programmed to monitor and respond to system-wide broadcast events or intents. When a broadcast event or intent is triggered, the BroadcastReceiver can activate an application by performing a specific task, such as creating a status bar notification. Unlike activities, a BroadcastReceiver does not have a user interface. Typically, a BroadcastReceiver is used to delegate tasks to services based on the type of intent data received (Chugh, 2022).

There are two types of Broadcast Receivers:

- Static Broadcast Receivers: receivers that are declared in the app's manifest file and can receive broadcast events even if the app is not running in the foreground or has been

closed. These receivers are always available to the system and help receive events important to the app, even if the app is not running.

- Dynamic Broadcast Receivers are receivers registered at runtime by an app component, such as an activity or service. These receivers are active only when the app component that registered them is running, such as active or minimised. (Lavishgarg26, 2022)

To set up a Broadcast Receiver in an Android application, the following steps are typically necessary:

- Creating a BroadcastReceiver: This involves defining a class that extends the BroadcastReceiver class and overrides the *onReceive()* method to specify the actions to be taken when the BroadcastReceiver receives a broadcast event.
- Registering a BroadcastReceiver: This involves registering the BroadcastReceiver with the Android system by either statically declaring it in the AndroidManifest.xml file or dynamically registering it using the *registerReceiver()* method. By enlisting the BroadcastReceiver, the application specifies the events or intents it wants to listen to and handle. (Chugh, 2022)

3.3.1.4 Content providers

A Content Provider is an Android component that manages access to a shared data set. It acts as a central repository for data that other applications can access. While an app with a content provider often includes its user interface for working with the data, content providers are primarily designed to be used by other applications (Android Official Corporate team, 2023).

In an Android application, UI components like Activities and Fragments use a CursorLoader object to send queries to the ContentResolver. The ContentResolver then acts as a client, sending create, read, update, and delete requests to the ContentProvider. Once the ContentProvider receives a request, it processes it. It returns the appropriate result to the ContentResolver, which returns the result to the UI component that made the initial query. This allows efficient and secure access to data stored in a shared repository, enabling data sharing and inter-app communication (RISHU_MISHRA, 2020).

3.3.2 XML in Android

XML is an acronym for Extensible Markup Language, a type of markup language used to represent data, like HTML. In the context of Android development, XML is

utilised to implement UI-related data, and it serves as a lightweight markup language that does not weigh down the layout. XML only consists of tags, which are invoked when implementing them (Adityamshidlyali, 2022).

In an app, a layout specifies the arrangement of the user interface, for example, within an activity. The layout consists of a hierarchical structure of View, and ViewGroup objects, with all elements in the layout, created using these. A View is responsible for rendering something visible to the user and allowing interaction. In contrast, a ViewGroup is an invisible container that establishes the layout structure for View and other ViewGroup objects. The layout is stored on an android application as XML syntax. The following figure illustrates the layout structure for View and other ViewGroup objects (Android Official Corporate team, 2022).

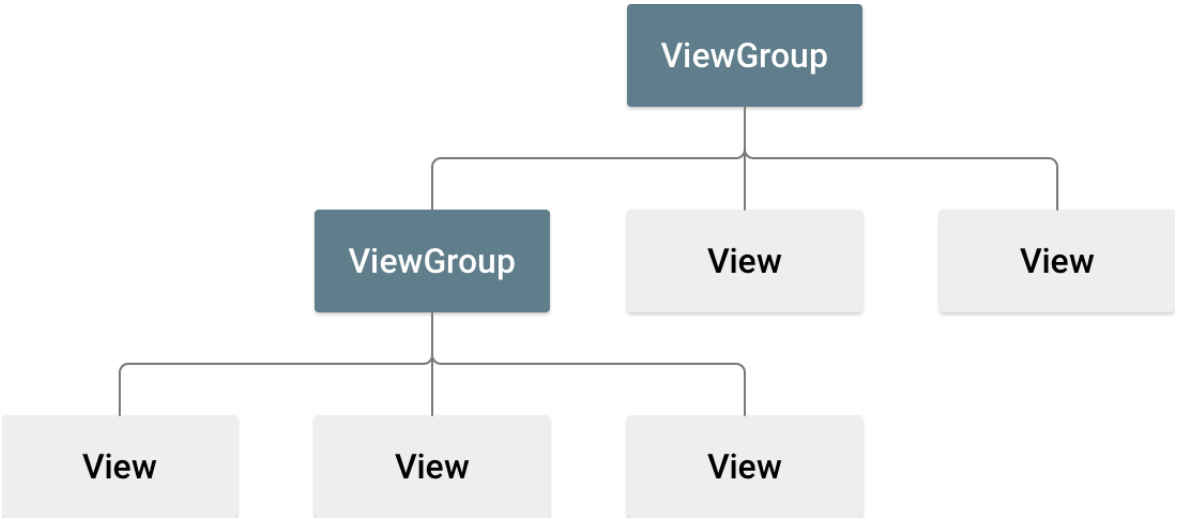

Figure 5 An illustration of a view hierarchy defines a UI layout (Android Official Corporate team, 2022)

Apart from the layout, Android Studio uses various XML files for various purposes, such as:

- AndroidManifest.xml file: contains crucial information about the application, such as its package name, which should match the code's namespaces and the different components that make up the application, such as activities, services, broadcast receivers, and content providers. Permissions are also included in this file.
- strings.xml file: contains text values for various components, such as TextView widgets. Keeping text values in a separate file allows for code reusability and easier application localisation into different languages.
- themes.xml file: defines the application's base theme and customised themes.

- drawable XML files: provide graphics for elements such as custom backgrounds for buttons, ripple effects, and other graphics, including vector graphics like icons.
- colors.xml file: holds all the colours used in the application.
- dimens.xml file: is used to hold dimensions for views in the application. (Adityamshidlyali, 2022)

3.3.3 Java

"Java is a programming language and computing platform first released by Sun Microsystems in 1995. It has evolved from humble beginnings to power a large share of today's digital world by providing a reliable platform upon which many services and applications are built. New, innovative products and digital services designed for the future also rely on Java." (Java Official Corporate team, n.d.)

For simplicity, the Android platform is a Linux-based operating system and open-source software platform designed for mobile devices. It provides a platform for developers to create and manage Android devices using managed code written in Java. The Android SDK and Java programming languages are used to develop Android applications, making it essential for programmers to understand Java (Kanjilal, 2016).

Native apps are built using tools and libraries a specific platform provides, such as Android. While it is possible to develop apps using other programming languages, a framework is often required to convert the code into a native app for the platform's API. Android does not use the JAVA Virtual Machine (JVM) to execute programming files, instead relying on the Dalvik Virtual Machine (DVM) or Android Run Time (ART) starting with Android 5.0, which are not actual JVM. To run the files, they must first be converted into the DEX format and then bundled into an Android Package (APK) before being installed and run on an Android device (Nordeen, 2018).

Until the launch of Android 4.4 KitKat, the DVM was designed for mobile devices as a virtual machine to execute Android applications. Subsequently, with the introduction of ART (Android Run Time) as a runtime environment, it replaced Dalvik entirely in Android 5.0 (Lollipop). The most noticeable distinction between ART and DVM is that ART employs AOT (Ahead-Of-Time) compilation, whereas DVM uses JIT compilation. However, ART has recently adopted a hybrid approach of AOT and JIT (Dora & Arthur, 2020).

In simple terms, a combination of two different approaches is being utilised, which yields its benefits:

- More efficient compilation: When an app is launched, the compiler can better understand its operation than with static analysis. As a result, more appropriate optimisation techniques are applied for each situation.
- Conserving RAM and storage: Bytecode is more concise than machine code. By solely performing an AOT compilation of specific parts of an application and not compiling unused applications, NAND-memory space can be significantly preserved.
- A significant increase in installation speed and first boot after a system update: There is no AOT compilation and, thus, no delay. (Dora & Arthur, 2020)

The following figure illustrates a schema of the hybrid approach of AOT and JIT for ART.

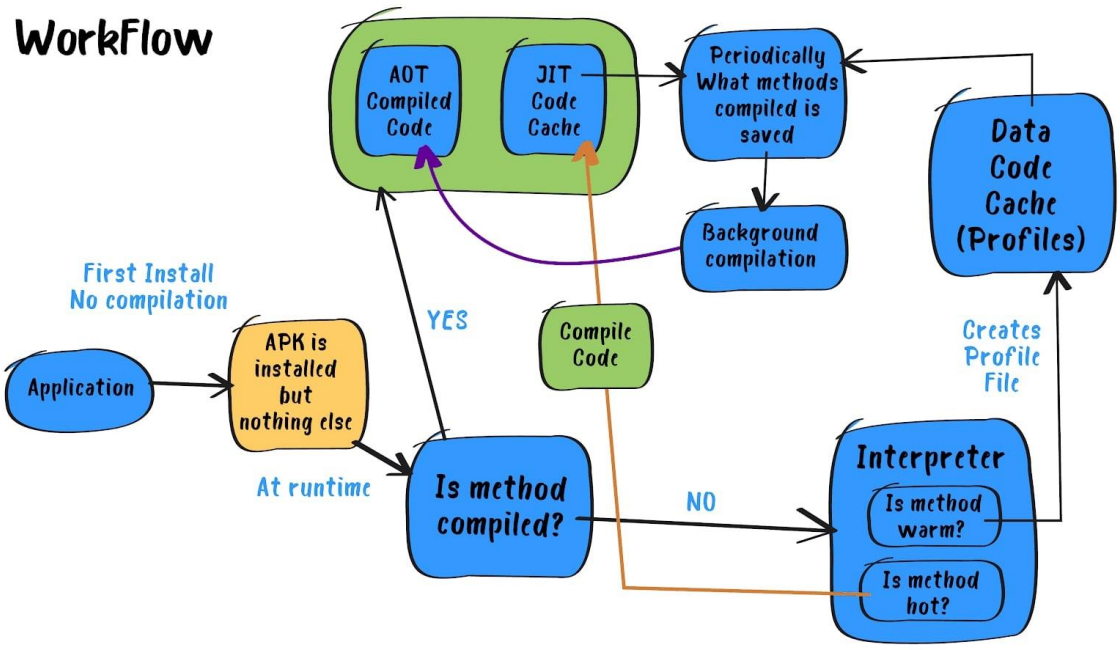

Figure 6 Workflow of ART compilation using AOT & JIT (Dora & Arthur, 2020)

3.3.4 Android Java Libraries

3.3.4.1 Google maps

The Maps SDK for Android allows a developer to easily integrate maps into an activity using a basic XML snippet to add a map as a fragment. This new map functionality provides a range of exciting features, including 3D maps, indoor maps, satellite maps, terrain maps, hybrid maps, vector-based tiles for efficient caching and drawing, animated

transitions, and numerous other capabilities. To utilize Google Maps in your Android app development project, it is necessary to configure the Google Play services SDK since the Maps SDK for Android is part of the Google Play services platform (Android Official Corporate team, 2021).

In an Android app, there are two methods to acquire location data. The first method involves utilising the LocationManager class, which allows apps to obtain the device's geographic location or receive regular updates. Also, it provides notifications when the device is close to a specific location. This LocationManage class is an Android Location API built-in since API level 1 (Android Official Corporate team, 2023).

The second method is a novel feature that utilises Google's Location API. This approach involves using the fused location provider to obtain the device's last known location. One of the location APIs Google Play services provides is the fused location provider. It handles the underlying location technology and presents a simple API for defining requirements at a high level, such as high accuracy or low power consumption. It optimises the device's battery power usage. Accessing the fused location provider necessitates including Google Play services in an app's development project. The second method is preferable since it offers numerous benefits, including a more straightforward implementation, greater accuracy, and reduced battery usage in normal mode (Android Official Corporate team, 2023).

3.3.4.2 Retrofit

Retrofit is a REST client for Android and Java that is type-safe and designed to simplify the process of consuming RESTful web services. Retrofit 2, the most recent version of Retrofit, has an updated internal API and several new features. It is built on top of OkHttp, its default networking layer. When handling the JSON response, *Retrofit 2* automatically converts it to a POJO (Plain Old Java Object) using a pre-defined JSON structure (Chugh, 2022).

To define an HTTP request in Retrofit 2, every method must include an HTTP annotation that specifies the request method and relative URL. Eight built-in annotations are available in Retrofit 2, such as HTTP, GET, POST, PUT, PATCH, DELETE, OPTIONS, and HEAD. The annotation also includes the relative URL for the requested resource (Retrofit Official Corporate author, n.d.).

In addition to serialising JSON responses into POJOs, Retrofit 2 provides several other converters that allow parsing JSON into different necessary types, such as Jackson, Moshi, Protobuf, Wire and GSON as a first conversion (Chugh, 2022).

3.3.4.3 QR Code Scanner

Implemented using the ZXing platform, the code scanner library is a lightweight Android library that prioritises fast and efficient QR code scanning with minimal dependencies. In addition to scanning multiple QR codes simultaneously, it also allows customisation of the scanning user interface (Budiyev, 2018).

This library offers several features, including autofocus and flash control, support for portrait and landscape screen orientations, front and back-facing camera options, a customisable viewfinder, compatibility with Kotlin, and touch focus capability (Budiyev, 2018).

4 Practical Part

The upcoming chapter will provide an overview of the various stages of developing a comprehensive system comprising a mobile application for attendance verification using QR codes, a web application, and a server. The steps include planning, analysing, designing, implementing, and testing, all underpinned by system development life cycle (SDLC) management principles (Admin BWC, 2019).

4.1 Motivation and Planning

This project aims to enhance the accuracy and efficiency of attendance tracking in the university while minimising the time required. The traditional method of calling out students' names or passing around a paper to record attendance is prone to errors and can be time-consuming, potentially reducing teaching time. Therefore, the proposed solution addresses these issues and provides a more reliable way of tracking attendance.

I have harboured this idea since my high school days in 2017, and I am grateful for the opportunity provided by the Czech University of Life Sciences Prague (ČZU), especially Mr Ing. Jiří Brožek, Ph. D., who has given me a chance to develop this concept into an actual project for my thesis.

As a result, I began contemplating and conducting research to identify a superior approach to implementing this idea. Since nearly all students possess a smartphone, it presents an opportunity to leverage the device's camera to work with QR Codes and access their GPS location, thereby enabling an improved attendance tracking system. Developing native apps is crucial for achieving optimal performance. I have pursued Android development for this project because Android has a significant market share.

Therefore, it is feasible to implement this system using the following components:

- Android for the mobile application: This app will enable students to scan a QR code provided by the web application provider.
- Markup languages such as HTML, CSS, and JavaScript for the web application client-side: This will allow professors to request and display QR codes for students.
- Node.js with the Express.js framework for the server-side implementation: This will be used for data and access management for mobile and web applications.

4.2 Requirement Analysis

The analysis phase, which is the most significant stage in the software development life cycle, will be covered in this section. Functional and Non-functional Requirements are the two primary areas through which the analysis will be provided. Functional requirements are the essential features that the end user expects the system to provide, and they must be included in the system as part of the contract. These requirements play a crucial role in defining the structure of the software or system, which can then be segmented into web applications, servers, and mobile applications. In contrast, non-functional requirements are quality constraints that the system must adhere to according to project contract terms. The degree to which these factors are implemented can vary depending on the project (Chitrasingla2001, 2022).

4.2.1 Functional Requirements

- Authentication and Authorization

  Authentication is a crucial use case for accessing the system as it ensures that each user is assigned a unique profile, access privileges, and restrictions within the system. In this project, the system provides three access levels, student, teacher, and admin, with the latter being the highest.

  Upon successfully logging in to the web application or mobile application, the server will authenticate the login credentials and issue a token that will be stored on the client side. Each subsequent request made to the server will include the authentication key for authorisation verification purposes.

  Only users with admin privilege access on the web application can perform account registration; this feature will not be available on the mobile application.

- Data Manipulation

  Upon logging into the web application, a user with teacher privileges can:

  - Create a new course and append it to any of the root categories.
  - Enrol students in the course.
  - Create attendance records for courses created by that user.

  Users with admin privileges, on the other hand, have the same tasks and rights as a teacher, but they also can:

- Create new user accounts for students, teachers, or admins themselves.
- Add new course categories to the system.

- Data Retrieval

  The mobile application requires students to log in initially. Once the student has been authenticated, the application will redirect them to a home page where a list of courses they have enrolled in will be displayed. The student can click on each course to view its details. Upon clicking on a specific course, the application will navigate the student to a course detail page with four primary tabs: course, participants, attendance, and grades.

  Upon accessing the web application, teachers are required to provide their login credentials. Upon successful authentication, they will be directed to a homepage displaying three tabs: Home, Dashboard, and My Courses. By selecting the My Courses tab, the system will retrieve and display all the courses the teacher has created. The teacher can then click on any course to navigate to a new page containing detailed information about the course. The course detail page comprises several tabs, including course, participants, attendance, grades, and more. Clicking on these tabs will retrieve and display the relevant data correspondingly.

- Data Accessible

  Data access in the system will be encrypted using a digital key. When a request is made from a web application to create an Attendance QR Session on the server, the server will verify the request and compute all necessary information, such as course details, attendance checking duration, maximum permissible range, and the current geolocation of the requester. The newly generated GUID and this information are then saved into a database. Once saved, the GUID is encrypted and returned to the client as a digital key represented by a string. The client then translates this key into a QR code and displays it.

- QR Code Reading

  The project development necessitates the implementation of QR code scanning as a core functional requirement. As previously stated, the web application will display a QR code generated by the teacher, who can be referred to as the QR provider. The mobile application has a QR scanning button that the student can click to access a new activity, which involves scanning the QR code. Upon inspecting the QR code, the

system decodes the key and converts it into a string value. Then, a callback function will send an HTTP request to the server, which includes the decrypted key. Once the server receives the request, it will search for the key in the database and send a response back. The mobile application will initiate the verification process upon receiving the necessary information. If the verification process is successful, the system will permit the student to submit their attendance.

4.2.2 Non-Functional Requirements

- Database

  In the context of this project, the database of choice shall be MongoDB Atlas. The rationale behind this selection is that MongoDB Atlas is a cloud-based, fully managed database that effectively handles the intricacies associated with deploying, managing, and recovering database deployments on a preferred cloud service provider, including but not limited to AWS, Azure, and GCP. Deploying, operating, and scaling MongoDB in the cloud is best accomplished using MongoDB Atlas (MongoDB Official Corporate team, n.d.).

- Geolocation access

  Geolocation plays a crucial role in the attendance-checking process. The mobile and web applications must utilise geolocation services to obtain the current location. The QR provider will be prompted to enable location services to capture their position before being permitted to create a QR session. Similarly, in the case of the mobile application, students must grant permission for location access. The location data obtained from both parties is necessary to calculate the distance between them and determine whether the student is within an acceptable range to mark attendance.

- Publishing

  The source code for the server and client components of the system will be uploaded to a GitHub repository. As the server component serves as an intermediary between the mobile application, client-side web application, and database, a cloud platform capable of hosting this server is required. Heroku is a cloud platform that facilitates the development, delivery, monitoring, and scaling of applications for organisations (Heroku Official Website, n.d.). As such, the source code for both the client and server will be stored in GitHub and connected to Heroku for hosting purposes.

4.3 Design

This chapter focuses on establishing a system architecture that aligns with the system requirements discussed in the previous chapter. This plan encompasses various elements such as designing the Name and Logo, determining the Communication and Data models, and creating the Wireframes for both the mobile and web applications.

4.3.1 Name and Logo

The system has been named *"E-System"* by the author, inspired by the term *"Electronic Education System"*. Furthermore, two logo variations have been created - one for the mobile application (as depicted in *Figure 7*) and another for the web application (illustrated in *Figure 8****Error! Reference source not found.***).

Figure 7 Mobile Application Logo (By author)

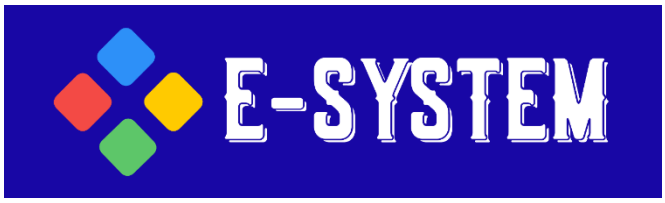

Figure 8 Web Application Logo (By author)

4.3.2 Communication model

The attendance session can be accessed through a QR code displayed on the web application, with each QR code being used for a single attendance session. The QR code is requested and then displayed on the client side of the web application. The entire process is done through the server and stored in a database. The communication model, which explains each step, is illustrated in the accompanying *Figure 9*. This communication diagram focuses specifically on the Attendance QR Session process, which is the central process of this system development, for the sake of simplicity.

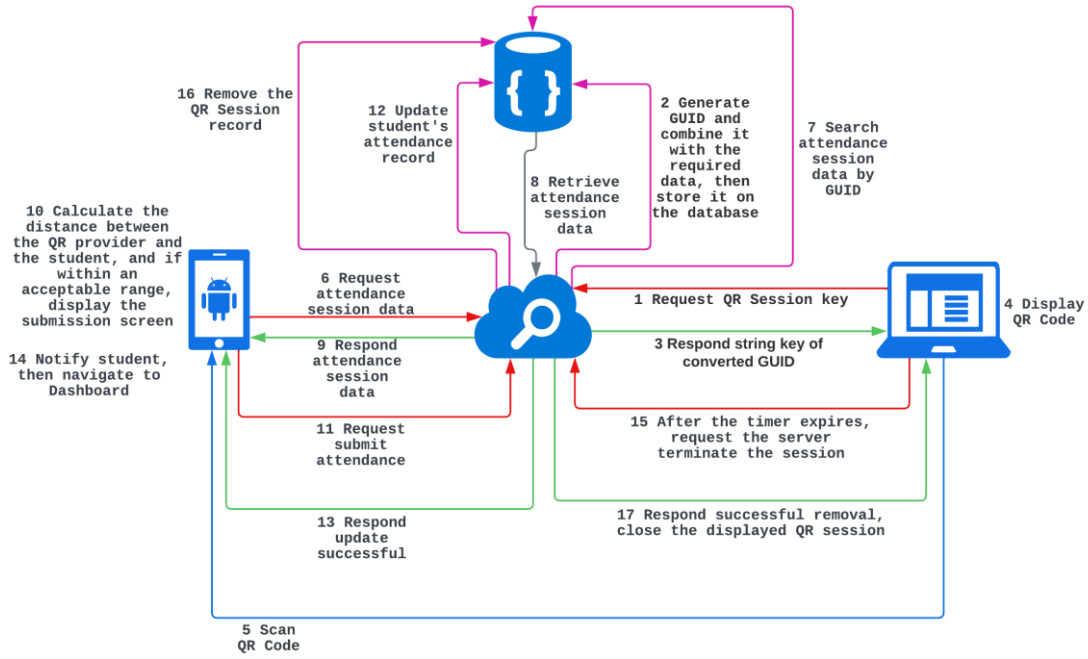

Figure 9 Communication model for Submitting attendance from QR session (By author)

1. A teacher clicks on the Generate QR Code to request for QR session.
2. Upon receiving a request, the server generates a GUID and creates a new JSON object that includes the GUID and other necessary data for future verification. This object is then stored in the database.
3. After saving, the server converts GUID to a string key and responds to the client.
4. After receiving the string key, the client displays it as a QR Code on Canvas.
5. After scanning the QR Code, the mobile application stores the resulting session GUID in string format.
6. The mobile application requests the server for session data filtered by GUID.
7. The server receives the request and searches the database for session data using the provided GUID.
8. If a matching GUID is found, retrieve the session data.
9. Send the session data back to the mobile application in response.
10. Once the mobile application receives the session data from the server, it obtains the current device location and calculates the distance between the QR provider's and student's locations. It then compares the calculated distance with the maximum allowed distance specified in the session data. The application navigates the student to the submission screen if the distance is within the allowed range.

11. Clicking the 'Present' button sends a request to the server to update the attendance record.
12. After receiving the request, the server connects to the database and updates the record.
13. The server sends a response with a success message indicating that the record has been updated to the mobile application.
14. The mobile application notifies the student about the update and navigates to the home page.
15. When a QR Code is displayed, the timer starts and requests the server to terminate the session when it expires.
16. The server receives the request and connects to the database to remove the session data.
17. The server successfully removed the record, responded to a client, and closed the QR session.

4.3.3 Data Schema

Complex entity relationships may be necessary for a real-world university system to fulfil various system requirements. However, for the sake of simplicity, the author has chosen to focus on a specific entity crucial to the system's functioning.

As a NoSQL database will be utilized in this defined system, the conceptual schema depicts the entities and their relationships in the data model. The data model is illustrated in the following *Figure 10*.

The following note and its explanation should be emphasized in this context:

- The model presented here only encompasses a small portion of the overall university system in the real world, with numerous entities and attributes, such as course content and teacher and student details, not included. Nevertheless, these few entities are sufficient for implementing the system following the requirements.
- The *Figure 10* depicts each entity with an automatically generated ID by Mongoose, as NoSQL databases do not require explicitly defined primary and foreign keys. An entity in NoSQL databases refers to a particular data type in the system. It can be represented as an object or a document representing a single instance of that data type. As a result, the relationships displayed in the figure provide a general indication of how each entity relates to others.

- While it is possible to use ERDs to model the relationships between entities in NoSQL databases, it is only sometimes necessary or practical. Instead, the data model in NoSQL databases is often designed to suit the application's specific requirements, and the relationships between entities are often modelled using techniques such as embedding, linking, or denormalization.
- This model diagram does not split many-to-many relationships into two one-to-many relationships, which is a common practice. While many-to-many relationships are not usually used in SQL databases, they are present in NoSQL databases where entities or objects are related through embedding. For instance, a course can have multiple students enrolled, and a student can be enrolled in several courses. In this case, the array of student IDs is embedded into each course object. The implementation chapter will illustrate this.
- Although not visible in the *Figure 10*, some entities may be necessary to meet the query document requirements in NoSQL databases. For instance, to query information related to Student, Teacher, or Admin accounts, a connection entity or schema must allow navigation to the correct user role. This connection entity or schema is essential to retrieve the appropriate data when querying the database.

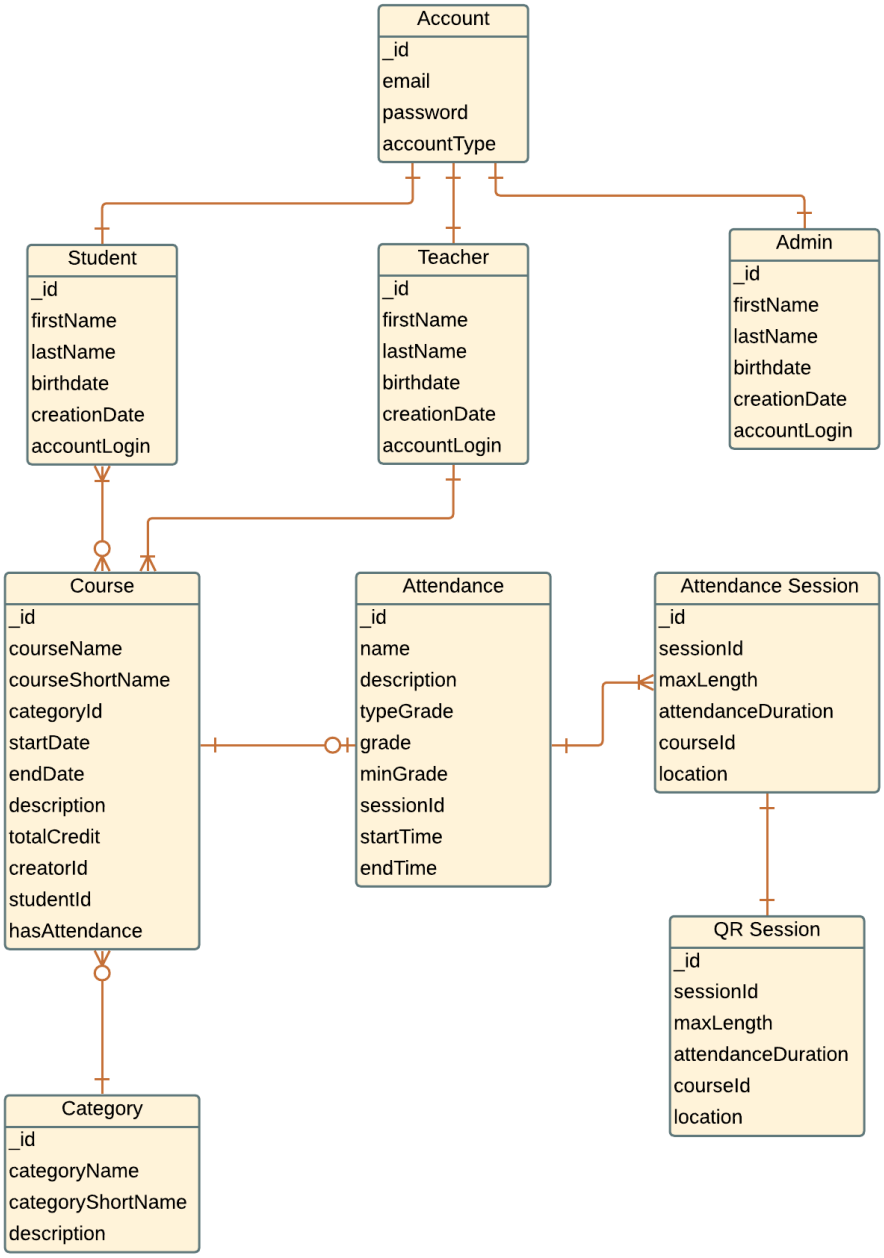

Figure 10 Conceptual Data Model (By author)

4.3.4 Wireframe

Wireframes are an essential aspect of design as they provide a comprehensive outline of the page structure, layout, information architecture, user flow, functionality, and intended behaviours. UX designers commonly use them to ensure that all stakeholders agree on the placement of information before developers begin building the interface with code (HANNAH, 2022).

This project will include several wireframe pages, but for simplicity, this chapter will only showcase Request and Display QR Code on the web application and two pages from the Mobile application: scanning and displaying a submission screen.

The header section of the web application, which typically includes menu navigation, a logo, account options, and a heading title, may appear on almost every page. The attendance session page consists of two primary components: a "Generate QR Code" button and a table displaying information about students' names and attendance records. When the user clicks on the "Generate QR Code" button, a QR session popup window appears, containing three pieces of information: a title for the QR session, the QR code itself, and a timer that starts counting down the duration. These Wireframes illustrate in *Figure 11* and *Figure 12*.

Same to the mobile app, the scanning page of the application only displays a camera frame that is prepared for scanning. Once a QR code is successfully scanned, the app guides the student to a submission screen. These Wireframes illustrate in *Figure 13* and *Figure 14*. This screen contains three main pieces of information:

- A map displays the device's current location.
- The course title.
- A "Present" button that the student can click to submit an attendance.

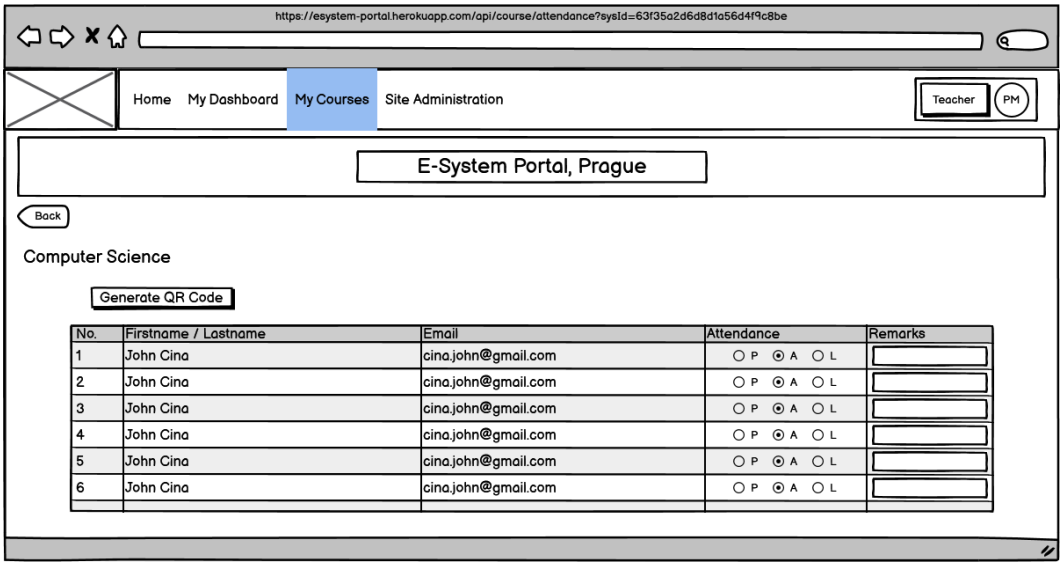

Figure 11 Attendance Session on Web Application (By author)

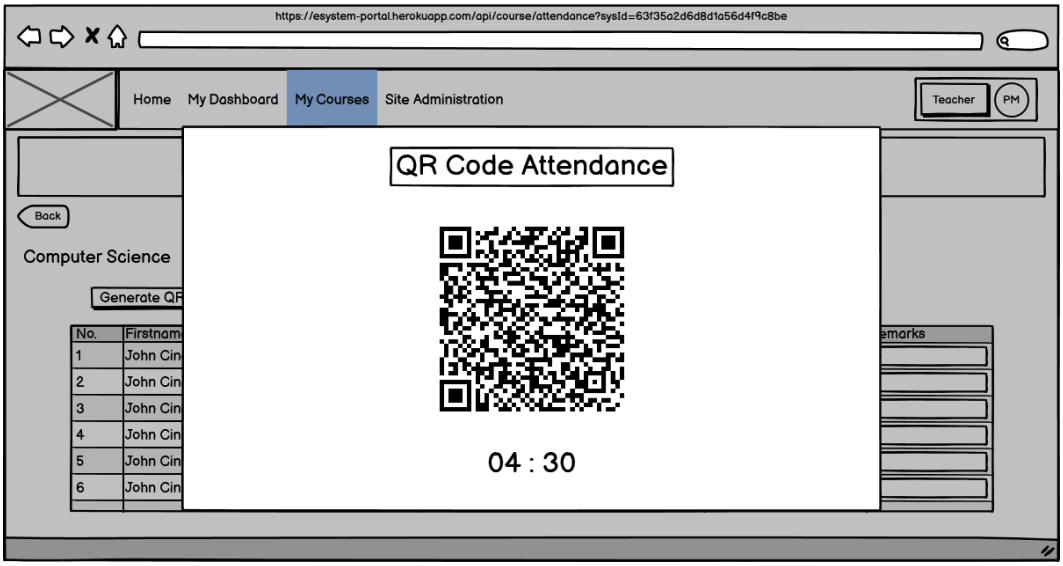

Figure 12 QR Code popup with timer on Web Application (By author)

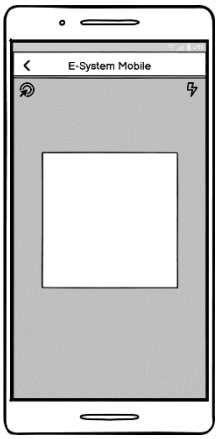

Figure 13 Scanning Fragment (By author)

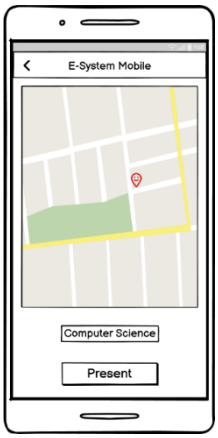

Figure 14 Submission Fragment (By author)

4.4 Implementation

The implementation of the E-System prototype will be divided into three main parts. The first part will involve implementing the server, which will handle the web API route and establish a connection to the database. The second part will focus on implementing the web application (client), while the last position will entail implementing the mobile application.

This project utilizes two different Integrated Development Environments (IDEs): Visual Studio Code is used for server and web development, while Android Studio is used for developing native Android mobile applications.

Due to the extensive volume of files and lines of code, not all code will be explained in this implementation part. Only the code essential for achieving this thesis's objective will be covered. Nonetheless, the complete source code is available in the GitHub repository.

The project folder in Visual Studio Code contains four main folders and one primary file, as displayed in the *Figure 15*. These include assets, middleware, server, views, and index.js.

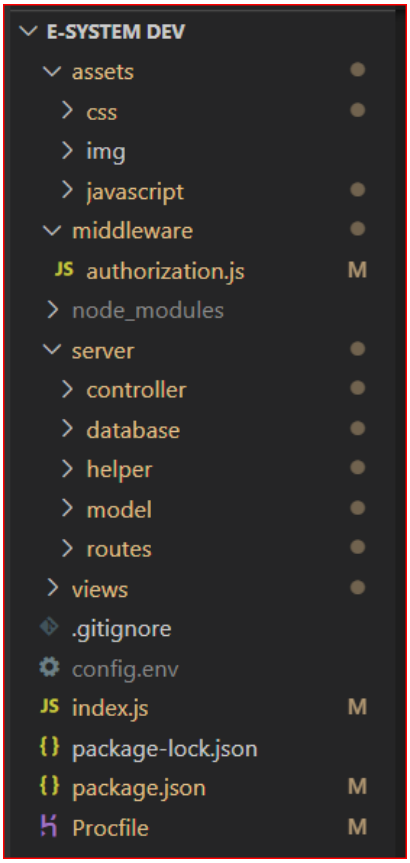

Figure 15 Project structure in Visual Studio Code (By author)

4.4.1 Server-Side

As mentioned in the theoretical section, Express.js is a web framework for Node.js that facilitates server-side development of websites and web applications. It generates APIs to simplify communication by handling HTTP requests and responses. The upcoming chapter will provide a detailed explanation of each implementation step, starting with the setup of the Express.js project.

4.4.1.1 Express.js development environment setup

As indicated earlier, NodeJS is a JavaScript run-time environment that is compatible with multiple platforms and enables the execution of JavaScript code outside of the browser environment via command-line tools. To initiate a new node project using express.js, the following steps should be adhered to:

First, create a project folder and subsequently open it in Visual Studio Code. Following this, open a terminal or command line in VS Code and input the command *"npm init"*. This command will create a new node folder within the project folder.

Next, to install express, enter the command *"npm i express"* into the terminal. This will install the express module into the current project. Finally, to install additional modules, repeat the command and input the name of the module as follows: *"npm i (name of the module)"*.

Here is a list of the node modules that have been utilized in this project:

```
{
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.0",
    "cookie-parser": "^1.4.6",
    "core-js": "^3.26.0",
    "datejs": "^1.0.0-rc3",
    "dotenv": "^16.0.2",
    "ejs": "^3.1.8",
    "express": "^4.18.1",
    "joi": "^17.6.2",
    "jsonwebtoken": "^8.5.1",
    "lodash": "^4.17.21",
    "mongoose": "^6.7.2",
    "qrcode": "^1.5.1",
    "uuid": "^9.0.0",
    "nodemon": "^2.0.20"
  }
```

Source Code 1 Install Modules (By author)

4.4.1.2 Index.js

In Node.js projects, *"index.js"* serves as the primary entry point that concatenates all an application's files into one. This file contains the application logic, including the setup of the web server, definition of routes, database connections, middleware functions, and handling of requests and responses.

```
const cookieParser = require("cookie-parser");
const express = require("express");
const morgan = require("morgan");
const dotenv = require("dotenv");
const bodyparser = require("body-parser");
const path = require("path");
const connectDB = require("./server/database/connection");
const app = express();

app.connectDB;
app.use(cookieParser());

app.set("views", path.resolve(__dirname, "views"));
app.set("view engine", "ejs");

app.use(express.static("assets"));
app.use("/css", express.static(path.resolve(__dirname, "assets/css")));
app.use("/javascript",
  express.static(path.resolve(__dirname, "assets/javascript"))
);
app.use("/img", express.static(path.resolve(__dirname, "assets/img")));

app.use(bodyparser.urlencoded({extended: true}));
app.use(bodyparser.json());
app.use(express.json());
app.use("/", require("./server/routes/router"));

dotenv.config({path: "config.env"});
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server listen on port http://localhost:${PORT}`);
```

Source Code 2 Index.js (By author)

Source Code 2 refers to an exemplary version of the *index.js* code that has been implemented in the project. In the initial section of this source code, the *'require'* function is utilized to import all the required modules, including the *'express'* module. The 'express' function is then invoked and stored in a constant variable named *'app'*, which is utilized as an application.

After initializing the *'app'* variable as an application, the *'app'* invokes a sequence of functions. Firstly, the *'connectDB'* function is called to establish a connection with the database, which is loaded from the root folder path named *'server/database'*.

Next, the *'set'* function is employed to set the template engine for rendering. As per the previously mentioned theory, Ejs is utilized for web page rendering. The following function is *'use'*, which is employed to enable the *'app'* to use all the required middleware functions.

Furthermore, the *'use'* function is also utilized by the *'app'* to serve a static file for client-side use, such as images, CSS, and JavaScript. As per the best practice guidelines, all the request routes are segregated into a separate file named *'router'* and need to be loaded and utilized in the *'app'*.

Finally, the *'listen'* function is invoked by the *'app'* to listen on the provided port. For security purposes, it is recommended to store the port in an environment variable.

4.4.1.3 Model Schema and Database Connection

Due to the nature of NoSQL databases like MongoDB being document-oriented and lacking in relational structures, it is essential to establish a proper schema to define the structure of documents in a collection. Mongoose is utilized to specify properties such as data types, default values, and validation rules to ensure consistency and integrity in the data stored in the database. By implementing the schema, the database only accepts valid data. Below is the source code for the schemas utilized in this project to achieve the desired outcome.

Source Code 3 defines a user *"Account"* schema that comprises three properties: *"email"*, *"password"*, and *"accountType"*. It also includes a function named *"generateAuthJWT"* that generates a JWT token. This function is utilized in the case of a user logging in with the correct credentials.

Source Code 4 showcases a student schema that contains five properties: *"firstName"*, *"lastName"*, *"birthdate"*, *"creationDate"*, and *"accountLogin"*.

'*accountLogin*' is of *ObjectId* data type and references an *Account* schema described in *Source Code 3*. As there are two types of schema relationships in Mongoose, namely *embedded sub-document*, and *ObjectId* as a reference to another schema, this project will utilize both approaches based on *use cases* and *performance efficiency*. By

storing the *ObjectId* as a reference in *Source Code 4*, the account schema can be retrieved from the *"Student"* schema using the *"populate"* method.

```
const getRoleUser = require("../controller/getRoleUser");

const accountSchema = new mongoose.Schema({
  email: { type: String, required: true, trim: true, min: 1, max: 255 },
  password: { type: String, required: true, min: 1, max: 255 },
  accountType: { type: String, required: true },
});

dotenv.config({ path: "config.env" });
accountSchema.methods.generateAuthJWT = async function () {
  const user = await getRoleUser(this.accountType, this._id);
  return jwt.sign({
      id: user._id, email: this.email, firstname: user.firstName,
      lastname: user.lastName, role: this.accountType,
    }, process.env.TOKEN_SECRET_KEY );
};

module.exports = mongoose.model("Account", accountSchema);
```

Source Code 3 Account Schema Model (By author)

```
const mongoose = require("mongoose");
const ObjectId = mongoose.Schema.Types.ObjectId;

const studentSchema = new mongoose.Schema({
  firstName: { type: String, required: true, min: 1, max: 255, trim: true },
  lastName: { type: String, required: true, min: 1, max: 255, trim: true },
  birthdate: { type: Date, required: true },
  creationDate: { type: Date, default: Date.now() },
  accountLogin: { type: ObjectId, ref: "Account" },
});

module.exports = mongoose.model("Student", studentSchema);
```

Source Code 4 Student Schema Model (By author)

```
const mongoose = require("mongoose");

const dynamicProfileSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, refPath: "accountType" },
  accountType: { type: String, required: true,
    enum: ["Student", "Teacher", "Admin"]},
});

module.exports = mongoose.model("DynamicProfile", dynamicProfileSchema);
```

Source Code 5 Dynamic Account Reference on Role (By author)

The conceptual Data Model in the design phase illustrates a one-to-one relationship between the *account* and the *student*, *teacher*, or *admin*. The *Dynamic Reference* schema in

Source Code 5 serves as a connection bridge to differentiate between user roles after successful authentication, as the *accountLogin* type ObjectId is embedded in the *student*, *teacher*, and *admin* schemas. This *Dynamic Reference* schema aids Mongoose in navigating to the appropriate schema based on the user's role.

```
const mongoose = require("mongoose");

const courseSchema = new mongoose.Schema(
  {
    creator: { type: mongoose.Schema.Types.ObjectId, ref: "DynamicProfile",
      required: true, trim: true },
    courseName: { type: String, required: true, trim: true,
       min: 1, max: 255 },
    courseShortName: { type: String, required: true, trim: true,
      min: 1, max: 255 },
    category: { type: mongoose.Schema.Types.ObjectId,
      ref: "Category", required: true, trim: true },
    startDate: { type: Date, required: true },
    endDate: { type: Date, required: true },
    description: { type: String, trim: true, max: 2505 },
    totalCreadit: { type: Number, min: 1, max: 1000 },
    student: [{ type: mongoose.Schema.Types.ObjectId, ref: "Student" }],
    hasAttendance: { type: mongoose.Schema.Types.ObjectId,
              ref: "Attendance" },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Course", courseSchema);
```

Source Code 6 Course Schema Model (By author)

The Course schema in *Source Code 6* comprises ten properties, namely *"creator"*, *"courseName"*, *"courseShortName"*, *"category"*, *"startDate"*, *"endDate"*, *"description"*, *"totalCredit"*, *"student"*, and *"hasAttendance"*. The *"creator"* property denotes the current user account, which could be a teacher or an admin and is embedded with the data type ObjectId reference to *"DynamicProfile"*. The other properties are obtained from the user input during course creation and stored in the database except for *"student"* and *"hasAttendance"*. As each course may have multiple students, the *"student"* property is an array of ObjectId reference Student schema that can be enrolled or removed via the enrollment use case, which will be covered on the client side. Similarly, *"hasAttendance"* data type ObjectId reference to Attendance schema does not exist yet, and its creation will be addressed in another use case for attendance creation on the client side.

Source Code 7 illustrates the Attendance schema, as mentioned in *Source Code 6*. Each course can only have one attendance, which may consist of one or more attendance sessions without overlapping start and end dates. The Attendance schema has six properties, including *"name," "description," "typeGrade," "grade," "minGrade,"* and *"sessions"*. The *"sessions"* property contains an array of embedded objects that include *"attendanceType," "startTime," "endTime"*, and *"sessionRefs"*. The *"sessionRefs"* property is an ObjectId type reference to the *"Session"* schema defined in *Source Code 8*. The *"Session"* schema contains only one array of ObjectId references to the "SingleSession" schema. For example, a course can have multiple attendance sessions, and each session can have multiple single sessions for each repeating date, storing information about enrolled students on that specific single session date. *"SingleSession,"* defined in *Source Code 9*, is derived from the Course schema, and contains only the exact date of attendance taking and an array of student information about their attendance on that date.

```
const mongoose = require("mongoose");

const attendanceSchema = new mongoose.Schema({
  name: { type: String, required: true, trim: true, min: 1, max: 255 },
  description: { type: String, trim: true, max: 2505 },
  typeGrade: { type: String, trim: true, required: true },
  grade: { type: Number, trim: true },
  minGrade: { type: Number, trim: true },
  sessions: [
    {
      sessionRefs: { type: mongoose.Schema.Types.ObjectId, ref: "Session" },
      attendanceType: { type: String, trim: true, min: 1, max: 255,
        required: true, default: "All students" },
      startTime: { type: String, trim: true, required: true },
      endTime: { type: String, trim: true, required: true },
    }],
});

module.exports = mongoose.model("Attendance", attendanceSchema);
```

Source Code 7 Attendance Schema Model (By author)

```
const mongoose = require("mongoose");

const sessionSchema = new mongoose.Schema({
  sessions: [{type: mongoose.Schema.Types.ObjectId, ref: "SingleSession"}],
});

module.exports = mongoose.model("Session", sessionSchema);
```

Source Code 8 Attendance Sessions Schema Model (By author)

```
const mongoose = require("mongoose");
const student = require("./student");

const singleSchema = new mongoose.Schema({
  date: {type: Date, required: true},
  students: [{
      student: {type: mongoose.Schema.Types.ObjectId, ref: "Student"},
      isPresent: { type: String, min: 1, max: 250, required: true,
        default: "Absent" },
      remark: {type: String, max: 2550},
       point: {type: Number, min: 0, max: 100, required: true, default: 0},
    }],
});

module.exports = mongoose.model("SingleSession", singleSchema);
```

Source Code 9 Single Session Schema Model (By author)

```
const mongoose = require("mongoose");

const qrsessionSchema = new mongoose.Schema({
  sessionId: {type: String, required: true, max: 40},
  objectLocation: {
    latitude: {type: Number, required: true, min: -90, max: 90},
    longitude: {type: Number, required: true, min: -180, max: 180},
  },
  maxLength: {type: Number, required: true, min: 10, max: 255},
  attendanceDuration: {type: Number, required: true, min: 5, max: 255},
  singleSessionId: { type: mongoose.Schema.Types.ObjectId,
    ref: "SingleSession" },
  courseId: { type: mongoose.Schema.Types.ObjectId, ref: "Course" },
  createdAt: { type: Date, default: Date.now, expires: 2100 },
});

module.exports = mongoose.model("QRSession", qrsessionSchema);
```

Source Code 10 QR Code Session Schema Model (By author)

This *Source Code 10* defines the object structure for a QR session that includes attributes such as *"sessionId," "objectLocation," "maxLength," "attendanceDuration," "singleSessionId," "courseId,"* and excludes *"createdAt"* during automation. For this schema, *"createdAt"* begins counting from the time data is stored in the database for a maximum of 35 minutes and will be removed upon client deletion request failure. The difference between *"sessionId"* and "singleSessionId" is that *"sessionId"* is a key generated on the server for data transmission security purposes, while *"singleSessionId"* is an ObjectId reference to the SingleSession schema used for student verification.

The *"connection.js"* file, which can be found in the root directory *"server/database"* of the *Source Code 11*, describes how to set up a Mongoose connection to MongoDB. The

connection string URL is saved as an environment variable for security reasons. The code specifies two possible outcomes: *"error"* for connections to MongoDB that fail, and *"open"* for connections that succeed.

```
const mongoose = require("mongoose");
const dotenv = require("dotenv");
dotenv.config({path: "config.env"});

mongoose.connect(process.env.MONGODB_URL);
const conn = mongoose.connection;
conn.on("error", () => console.error.bind(console, "connection error:"));
conn.once("open", () => console.info(`Connected to Database:${conn.host}`));

module.exports = conn;
```

Source Code 11 MongoDB Connection (By author)

4.4.1.4 Router

This chapter explains the defined route handlers for request and response in the *index.js* file, as previously mentioned. The server in this project facilitates communication between the mobile and web applications, as depicted in the communication schema during the design phase. To maintain consistency, separate routes are created for mobile and web applications. However, as the server will be uploaded to *GitHub* and hosted by *Heroku* for student subscription, it only provides one free domain without subdomains. To address this limitation, the author decided to create a rule for the URL path. The URL path starting with *"/mobile/"* is used for mobile communication, while *"/api/"* is used for web application communication.

```
const controller = require("../controller/controller");
const express = require("express");
const router = express.Router();
const authorization = require("../../middleware/authorization");
const globalObjects = require("../helper/globalObjects");

// Android Java RESTful web services
router.get( "/mobile", authorization.tokenValidation,
  authorization.studentAccess, controller.mobileUserLogin
);
router.post("/mobile/auth", controller.authMobileUser);
router.get(
  "/mobile/course/attendances/qrsession", authorization.tokenValidation,
  authorization.studentAccess, controller.getQRSessionOneStudent
);
router.patch(
  "/mobile/course/attendances/qrsession", authorization.tokenValidation,
  authorization.studentAccess, controller.submitAttendance
);
```

```
// Web Application RESTful API server route handler
router.get("/auth", authorization.publicAccess, (req, res) => {
  res.render("login", {userInfos: req.user});
});
router.post("/auth", controller.authUser);
router.get(
  "/api/courses/my_courses", authorization.tokenValidation,
  authorization.teacherAccess, controller.getMyCourses
);
router.get(
  "/api/course", authorization.tokenValidation,
  authorization.teacherAccess, controller.displayCourse
);
router.get(
  "/api/course/participants", authorization.tokenValidation,
  authorization.teacherAccess, controller.enrolledParticipants
);
router.get(
  "/api/course/participants/enrolls", authorization.tokenValidation,
  authorization.teacherAccess, controller.enrollsParticipantsOption
);
router.put(
  "/api/course/participants/enrolls", authorization.tokenValidation,
  authorization.teacherAccess, controller.updateEnrolledStudents
);
router.get(
  "/api/course/attendance", authorization.tokenValidation,
  authorization.teacherAccess, controller.getAttendance
);
router.post(
  "/api/course/attendance", authorization.tokenValidation,
  authorization.teacherAccess, controller.addAttendance
);
router.get(
  "/api/course/session_check", authorization.tokenValidation,
  authorization.teacherAccess, controller.getSessionSingleCheck
);
router.post(
  "/api/course/session_check/qrCode", authorization.tokenValidation,
  authorization.teacherAccess, controller.createQRSession
);
router.delete(
  "/api/course/session_check/qrCode", authorization.tokenValidation,
  authorization.teacherAccess, controller.deleteQRCodeSession
);
router.get("/*", (req, res) => { res.render("404"); });
module.exports = router;
```

Source Code 12 Route Handlers (By author)

In this *Source Code 12*, both routes are included in one file. The first part of the routes is for mobile RESTful web services, which defines four routes:

1. The first route handles a request for courses to display on the home activity in case the user is already logged in.
2. The second route handles an authentication request.

3. The third route handles a request for a QR Session. This request is made after the QR Code has been scanned.
4. Finally, this route handles a request to submit an attendance. This is the final process after the student is validated, and an attendance can be submitted.

The second part of the routes pertains to Client-side RESTful web services, which consist of twelve routes.

1. The first route, *"/auth"*, is a GET route that directs the user to a login page.
2. The second route, *"/auth"*, is a POST route that handles authentication requests from the login page.
3. The third route, *"/api/courses/my_courses"*, is a GET route that redirects the user to a webpage displaying all courses created by that user.
4. The fourth route, *"/api/course"*, is a GET route that directs the user to a single course detail page.
5. The fifth route, *"/api/course/participants"*, is a GET route that requests participants for the course when the user clicks on the *"Participants"* tab on the course detail page.
6. The sixth route, *"/api/course/participants/enrolls"*, is a GET route that displays a list of students who have and have not enrolled in the course.
7. The seventh route, *"/api/course/participants/enrolls"*, is a PUT route that updates the enrollment status of the students in the database.
8. The eighth route, *"/api/course/attendance"*, is a GET route that displays the list of attendance for each course.
9. The ninth route, *"/api/course/attendance"*, is a POST route that creates a new attendance if there is no existing attendance for the course.
10. The tenth route, *"/api/course/session_check"*, is a GET route that displays attendance details for each date, with a button to create a QR Code.
11. The eleventh route, *"/api/course/session_check/qrCode"*, is a POST route that creates and stores a QR Session in the database upon clicking the "Create QR Code" button on the client side.
12. The twelfth route, *"/api/course/session_check/qrCode"*, is a DELETE route that deletes the QR Session from the database.
13. Finally, the *"/*"* route handles a case where there is no existing route and directs the user to a 404 page.

4.4.1.5 Controller

This section will focus on the implementation of a controller for each route. As evidenced in *Source Code 12*, each route request is ultimately passed to the final function, which serves as a controller that sends a response back to the user. In simpler terms, a controller in Express.js can be defined as a JavaScript function that connects the user's request to the server's response by processing incoming requests and generating appropriate responses. It typically includes the necessary business logic and database access logic to handle the request and produce the response. Due to space limitations, not all controllers of this project will be covered in this explanation. However, the complete source code for the controllers can be found on the disc at the back side of this thesis or on GitHub. All the source code described below is from the *"controller.js"* file located in the root path "server/controller".

```
module.exports.authUser = async (req, res, next) => {
  const {error} = validation.authValidation(req.body);
  const alert = UIMessage.alert_invalid_login;
  if (error) return res.status(400).render("login", {alert});

  const user = await Account.findOne({email: req.body.email});
  if (!user) return res.status(400).render("login", {alert});

  const validPassword = await bcrypt.compare(req.body.password,
      user.password);
  if (!validPassword) return res.status(400).render("login", {alert});

  const token = await user.generateAuthJWT();
  res.cookie("x-auth-token", token, { httpOnly: true, sameSite: "lax" });
  return res.status(200).redirect("/");
};
```

Source Code 13 Authentication User Controller (By author)

In *Source Code 13*, a controller for user authentication is implemented. Firstly, the *"authValidation()"* function validates the user input and prompts the user to re-enter their input if any fields are empty. The user's account is then searched for in the *"Account"* collection based on their email. If the account is not found, authentication fails, and the user is notified accordingly. *"compare()"* method from *"bcrypt"* is used to compare the input password with the password found in the database. If the passwords match, a token is generated and stored in a cookie on the client side.

The *Source Code 14* contains two controllers: the Display Course Controller and the Show List of Enrollment Students Controller. The Display Course Controller retrieves

the courseId from the URL's query string (sysId), searches for a matching course in the Course collection by ID and sends the course object as a response to the Client-side.

In contrast, the Show List of Enrollment Students Controller retrieves all students from the *"Student"* collection. It then uses the courseId from the URL's query string (sysId) to locate a course in the *"Course"* collection and extract a list of enrolled students for that course. Finally, it applies the *"filter()"* method of the JavaScript array to obtain a list of students who have not enrolled in the course by comparing the array of all students with the array of enrolled students. The response will send both the enrolled and unenrolled student lists to the client-side.

```
module.exports.displayCourse = async (req, res, next) => {
  if (ObjectId.isValid(req.query.sysId)) {
    const course = await Course.findById(req.query.sysId);
    if (course) {
      return res.render("courseDisplay", {
        course: course, userInfos: req.user,
        navMenu: globalObjects.indexNavigation,
        tabId: globalObjects.indexNavigation.myCourses.id,
      });
    }
  } else return res.redirect("/api/courses/my_courses");
};

module.exports.enrollsParticipantsOption = async (req, res, next) => {
  if (ObjectId.isValid(req.query.sysId)) {
    const allStudents = await Student.find().select("_id firstName
      lastName");

    const enrolledStudents = await Course.findById(req.query.sysId)
      .populate({ path: "student", populate: {path: "accountLogin",
      select: "email"}}).select("student");

    const unenrolledStudents = allStudents.filter(
      (element) => !enrolledStudents.student.find(({_id}) =>
       element._id.toString() === _id.toString()));
    return res.status(200).send({enrolledStudents, unenrolledStudents});
  } else next();
};
```

Source Code 14 Display Course and Show Enrolled Students Controller (By author)

```
module.exports.updateEnrolledStudents = async (req, res, next) => {
  const courseId = req.query.sysId;
  const update = {student: req.body.student};
  const courseSingleSessionsId = [];
  if (!ObjectId.isValid(courseId)) return res.redirect("/api/course");

  const course = await Course.findById(courseId).populate({
    path: "hasAttendance", populate: { path: "sessions.sessionRefs",
      populate: { path: "sessions" } } });

  if (course.hasAttendance == undefined) {
    await Course.findOneAndUpdate({_id: courseId}, update, {
      returnOriginal: true });
    return res.status(200).send(UIMessage.successUpdated);
  }

  const studentsToRemove = course.student.filter(
    (elem) => !req.body.student.includes(elem.toString()));

  const studentsToEnroll = req.body.student.filter(
    (elem) => !course.student.includes(elem));

  const moveStudentsSession = await connection.startSession();
  try {
    moveStudentsSession.startTransaction();
    course.hasAttendance.sessions.forEach((element) => {
      element.sessionRefs.sessions.forEach((session) => {
        courseSingleSessionsId.push(session._id);
      });
    });

    await SingleSession.updateMany({_id: {$in: courseSingleSessionsId}},
    { $push: { students: {
      $each: studentsToEnroll.map((studentId) => ({student:studentId})) }}},
    {moveStudentsSession});

    await SingleSession.updateMany({_id: {$in: courseSingleSessionsId}},
    { $pull: { students: { student: { $in: studentsToRemove }}}},
    {moveStudentsSession}
    );

    await Course.findOneAndUpdate({_id: courseId}, update, {
      returnOriginal: true, moveStudentsSession,
    });

    await moveStudentsSession.commitTransaction();
    res.status(200).send(UIMessage.successUpdated);
  } catch (error) {
    await moveStudentsSession.abortTransaction();
    res.status(500).send("Couldn't update the enrollment students!");
    next();
  } finally {
    await moveStudentsSession.endSession();
  }
};
```

Source Code 15 Update Enrollment Student Controller (By author)

Source Code 15 provides an Update Enrollment Student Controller. It begins by finding a course by its ID using the courseId from the URL's query string (sysId) and populating an attendance. It then checks whether the attendance has already been created using an if condition. If not, it simply calls the *"findOneAndUpdate"* function to update an array of enrolled students listed in the *"Course"* collection. If the attendance has already been created, two collections need to be updated: *"Course"* and *"SingleSession"*. Since a transaction is used to prevent failures from one update, it will be rolled back. Before starting the transaction, two lists of students need to be obtained, one for students that will unroll and another for students who are going to enrol in the course by using *"filter()"*. The transaction is wrapped in a *try-catch block* and a *finally block* to end the transaction. As mentioned in *Source Code 9*, the *"SingleSession"* collection is a small unit of attendance in this project that stores information on attendance on each date. Therefore, in the *try block*, all single sessions belonging to this course are first retrieved. Then, the *"SingleSession"* collection is updated twice: first, to remove the students to be unenrolled, and then to add the students to be enrolled. Finally, the array list of enrolled students is updated in the course. After the transaction is committed, a response message is sent to the client side indicating the successful update.

Source Code 16 demonstrates the addition of an Attendance controller. Initially, the code retrieves a course by its ID using the courseId from the URL's query string (sysId). If a course is found, the *"attendanceValidation()"* function is called to validate the user's input. If no errors are detected, *"createArrayDates()"* is invoked to create an array of dates by passing three arguments: *"req.body.repeatDay"*, *"data.startDate"*, and *"data.endDate"*. Since this process involves working with multiple collections in sequence, a transaction is required to roll back in case of failure, as explained in the previous *Source Code 15*. First, *"insertMany()"* is called to insert an array of dates corresponding to an array of enrolled students into the *"SingleSession"* collection. Then, *"createArrayIds()"* is invoked to create an array of IDs for the single sessions that were just created. This array of IDs is stored in a *"Session"* collection. Once the session record is saved, the ID of this record is then saved in the *"Attendance"* collection, along with additional data elements such as *"name"*, *"description"*, *"typeGrade"*, *"grade"*, and *"minGrade"*. Finally, the transaction is committed, and the code retrieves the course data corresponding to an attendance and sends a response back to the client side.

```
module.exports.addAttendance = async (req, res, next) => {
  const data = req.body;
  const courseId = req.query.sysId;
  let arrayDates;
  if (ObjectId.isValid(courseId)) {
    const course = await Course.findById(courseId);
    if (!course) return res.status(400). redirect("/api/course");

    const {error} = validation.attendanceValidation(
      req.body, course.startDate, course.endDate);
    if (error) return res.status(400).render("attendance",
       {alert: error.details[0].message, course});

    arrayDates = extFuntion.createArrayDates(req.body.repeatDay,
    data.startDate, data.endDate);

    const addSessionsAttendance = await connection.startSession();
    try {
      addSessionsAttendance.startTransaction();

      const students = course.student;
      const singleSessionsSaved = await SingleSession.insertMany(
        extFuntion.createSessions(arrayDates, students),
        {session: addSessionsAttendance}
      );
       const arraySingleSessionIds =
        extFuntion.createArrayIds(singleSessionsSaved);

      const sessionsSaved = await Session({sessions: arraySingleSessionIds})
             .save(addSessionsAttendance);

       const attendance = new Attendance(_.pick(data, ["name", "description",
       "typeGrade", "grade", "minGrade"]));

       attendance.sessions.push({ sessionRefs: sessionsSaved._id,
         attendanceType: data.attendanceType, startTime: data.startTime,
        endTime: data.endTime });

       const attendanceSaved = await attendance.save({
         session: addSessionsAttendance });
      course.hasAttendance = attendanceSaved._id;
      await course.save({session: addSessionsAttendance});

      await addSessionsAttendance.commitTransaction();

      await Course.findById(courseId).populate({ path: "hasAttendance",
        select: "sessions", populate: {
        path: "sessions.sessionRefs", populate: { path: "sessions" }}})
      return res.status(200)
             .redirect(`/api/course/attendance?sysId=${courseId}`);
    } catch (error) { await addSessionsAttendance.abortTransaction();
      res.status(500).send("Couldn't create attendance sessions!");
      next();
    } finally { await addSessionsAttendance.endSession(); }
  } else return res. status(400).redirect("/api/course");
};
```

Source Code 16 Add Attendance Controller (By author)

```
module.exports.createQRSession = async (req, res, next) => {
  const courseId = req.query.sysId;
  const sessionId = req.query.sessionId;

  if (ObjectId.isValid(sessionId)) {
    await QRSession.findOneAndDelete({ singleSessionId: sessionId, });
  } else return res.status(400).redirect("/api/course");

  const qrSession = req.body;
  qrSession.courseId = courseId;
  qrSession.sessionId = extFuntion.createUUID();
  qrSession.singleSessionId = sessionId;

  if (ObjectId.isValid(courseId) && ObjectId.isValid(sessionId)) {
    const course = await Course.findById(courseId)
      .populate({ path: "hasAttendance", select: "sessions",
        populate: { path: "sessions.sessionRefs",
          populate: { path: "sessions.students.student" } } })
      .select("courseName");

    const session = extFuntion.getSingleSession(
      course.hasAttendance.sessions, sessionId );

    if (session) {
      const qrCodeSession = await QRSession(qrSession).save();
      return res.status(200).send(qrCodeSession);
    } else return res.status(400).redirect("/api/course");
  } else return res.status(400).redirect("/api/course");
};
```

Source Code 17 Create QR Session Controller (By author)

```
module.exports.deleteQRCodeSession = async (req, res, next) => {
  const courseId = req.query.sysId;
  const sessionId = req.query.sessionId;
  const qrSession = req.body;

  if (ObjectId.isValid(courseId) && ObjectId.isValid(sessionId)) {
    const result = await QRSession.findOneAndDelete({
      singleSessionId: sessionId,
    });
    res.status(200).send(result);
  } else next();
};
```

Source Code 18 Delete QR Session Controller (By author)

The controller in *Source Code 17* is designed to create a temporary QR session for scanning. The process starts by calling a *"findOneAndDelete()"* function to search for an existing session with the "sessionId" from the query URL in the *"QRSession"* collection. If a record is found, it is removed. The controller then assigns an object variable to *"req.body"* and adds additional elements such as *"courseId"*, *"singleSessionId"*, and *"sessionId"* generated by the *"createUUID()"* function. The *"getSingleSession"* function is

used to confirm that the *"sessionId"* from the query URL exists in the current course before saving the new record to the *"QRSession"* collection. Finally, the controller sends a response of the newly saved record to the client side.

Source Code 18 demonstrates a QR Session Delete Controller that contains a single main function, *"findOneAndDelete(),"* which searches for a record with the *"sessionId"* from the query URL in the *"QRSession"* collection and deletes it if found. The controller then sends a response indicating that the removal was successful.

```
module.exports.authMobileUser = async (req, res, next) => {

  const errorMessage = UIMessage.alert_invalid_login;

  const {error} = validation.authValidation(req.body);
  if (error) return res.status(400).send({errorMessage});

  const user = await Account.findOne({email: req.body.email});
  if (!user) return res.status(400).send({errorMessage});

  const validPassword = await bcrypt.compare(req.body.password,
      user.password);
  if (!validPassword) return res.status(400).send({errorMessage});

  const token = await user.generateAuthJWT();
  const student = jwt.verify(token, process.env.TOKEN_SECRET_KEY);

  const courses = await Course.find({student: student.id})
    .populate("category", "categoryName -_id")
    .select("courseName courseShortName categoryName");

  if (!courses) return res.status(200).send([]);
  return res.status(200).send({token, courses});
};
```

Source Code 19 Mobile User Authentication Controller (By author)

Source Code 19 defines a controller that provides authentication for a mobile user. The controller is like the one implemented in *Source Code 13*, but it includes a few differences to handle requests from a mobile application. In case of an error, the controller sends a response in JSON format. Since the home activity of the mobile application displays a list of enrolled courses, the controller invokes the *"find()"* method to retrieve all courses enrolled by the student. Instead of storing the *JWT token* on a cookie, the controller sends an array of courses and the *JWT token* in JSON format to the mobile application.

```
module.exports.getQRSessionOneStudent = async (req, res, next) => {
  let sessionStored = {};
  const userId = req.user.id;
  const sessionId = req.query.sessionId;

  const result = await QRSession.findOne({sessionId: sessionId})
    .populate("singleSessionId").populate("courseId");

  if (result) {
    const students = result.singleSessionId.students;
    for (let student of students) {
      if (student.student == userId) {
        sessionStored.maxLength = result.maxLength;
        sessionStored.objectLocation = result.objectLocation;
        sessionStored.unitSessionId = result.singleSessionId._id;
        sessionStored.courseName = result.courseId.courseName;
      }
    }

    if (Object.keys(sessionStored).length != 0)
      return res.status(200).send(sessionStored);

    return res.status(404).send(UIMessage.invalidRequested);
  }
  return res.status(400).send(UIMessage.invalidRequested);
};
```

Source Code 20 Mobile Retrieve Attendance Session for one Student Controller (By author)

```
module.exports.submitAttendance = async (req, res) => {
  const userId = req.user.id;
  const unitSessionId = req.query.unitSessionId;

  if (ObjectId.isValid(unitSessionId) && ObjectId.isValid(userId)) {
       const updatedSingleSession = await
       SingleSession.findByIdAndUpdate(unitSessionId,
      {$set: {"students.$[elem].isPresent": "present"}},
      {arrayFilters: [{"elem.student": userId}], new: true}
    );

    return res.status(200).send(UIMessage.successUpdated);
  }
  return res.status(400).send(UIMessage.invalidRequested);
};
```

Source Code 21 Mobile Attendance Submission Controller (By author)

Source Code 20 provides a controller that retrieves attendance session data for a specific student during a QR scanning process in a mobile application. The controller first searches the *"QRSession"* collection for a matching record using the *"sessionId"* from the query string URL and then populates the corresponding *"SingleSession"* and *"Course"* collections. As implemented in *Source Code 16*, each single session record contains an

array of enrolled students and their attendance data on that date. Therefore, the controller uses a for loop to compare each *student's enrolled ID* with the current *user's logged-in ID*. If a match is found, the controller appends the element to an empty element called *"sessionStored"*. Finally, the controller checks if the *"sessionStored"* object is empty. If it is not empty, the controller sends this *"sessionStored"* object as a response to the mobile application.

Source Code 21 provides a controller for submitting attendance, specifically updating an attendance record after a student clicks the "Present" button on a mobile application. The controller uses only one method, *"findByIdAndUpdate()"* to find a record by *"unitSessionId"* from the query string URL and update the "SingleSession" collection by setting the value of the *"isPresent"* object key from the default *"absent"* (as shown in *Source Code 9*) to "present" for the matching *userId*.

4.4.1.6 Middleware function

Source Code 22 showcases a middleware authentication function which can access request and response objects to perform pipeline processes and subsequently pass on the command to another function using the *"next()"* method. This middleware function is designed to authenticate users before they can access a controller function, which consists of five methods.

The first method, *"tokenValidation()"*, is responsible for decoding a token that is stored on the client, such as in a cookie or query string, by utilizing the *"verify()"* method and passing in the token and a *"TOKEN_SECRET_KEY"* as arguments. If the token is successfully decoded, the "next()" method is invoked to pass on the command. The remaining four methods, *"publicAccess()"*, *"studentAccess()"*, *"teacherAccess()"*, and *"adminAccess()"*, are used to determine the level of authorization of access. The access levels are assigned a numerical value, starting from 0 for *"publicAccess()"* and increasing in value for each subsequent level, with *"adminAccess()"* being the highest. Higher level access automatically grants access to lower levels. For example, a teacher role could access any route that requires public or student access, in addition to the routes that require teacher access. Each function contains an if-else condition, and if the condition is met, the *"next()"* method is invoked to pass on the command to the next function in line. However, if the condition is not met, a response indicating access has been denied will be sent to the client.

```
const jwt = require("jsonwebtoken");
const dotenv = require("dotenv");
const globalObjects = require("../server/helper/globalObjects");

dotenv.config({path: "config.env"});

function tokenValidation(req, res, next) {
  const token = req.query.token || req.cookies["x-auth-token"];
  if (!token) return res.status(401).send("access denied!");
  try { const decoded = jwt.verify(token, process.env.TOKEN_SECRET_KEY);
    req.user = decoded;
    next();
  } catch (ex) { res.status(400).send("Invalid token!"); }
}
function publicAccess(req, res, next) {
  const token = req.cookies["x-auth-token"];
  if (token) {
    try { const decoded = jwt.verify(token, process.env.TOKEN_SECRET_KEY);
      req.user = decoded;
      next();
    } catch (ex) { res.status(400).send("Invalid token!"); }
  } else next();
}
function adminAccess(req, res, next) {
  if (req.user.role === globalObjects.accountType.admin) { next(); }
  else return res.status(401).send("access denied!");
}
function teacherAccess(req, res, next) {
  if ( req.user.role === globalObjects.accountType.admin ||
    req.user.role === globalObjects.accountType.teacher
  ) { next(); }
  else return res.status(401).send("access denied!");
}
function studentAccess(req, res, next) {
  if ( req.user.role === globalObjects.accountType.student ||
    req.user.role === globalObjects.accountType.teacher ||
    req.user.role === globalObjects.accountType.admin
  ) {
    next();
  } else return res.status(401).send("access denied!");
}

module.exports = {
  tokenValidation, publicAccess, adminAccess, teacherAccess, studentAccess
};
```

Source Code 22 Middleware Authentication Function (By author)

4.4.1.7 Views

In the previous chapter, it was mentioned that the project utilizes a hybrid rendering approach. In this approach, the views folder contains all *.ejs* files, which are used for server

rendering. Due to the large number of *.ejs* files used in the project, this chapter will cover only three particularly important source codes.

Source Code 23 displays a table record of *"My Courses"*. The table consists of five columns, and for maintainability reasons, the body of the table has been separated into a separate file called *"_courseRecords.ejs"*.

```
<table class="w3-table-all records_tables" id="myCoursesTable">
  <thead>
    <tr class="w3-light-grey">
      <th></th>
      <th>Courses</th>
      <th>Categories</th>
      <th>Course Duration</th>
      <th>Creator</th>
      <th class="w3-right-align"></th>
      <th class="w3-right-align">Students</th>
    </tr>
  </thead>
  <tbody><%- include('include/_courseRecords') %></tbody>
</table>
```

Source Code 23 EJS My Course (By author)

```
<div>
  <div class="w3-margin-bottom">
     <button class="w3-button w3-blue w3-hover-light-blue w3-round-large"
             id="enrollStudents">Enrol users</button>
  </div>
  <div>
   <table class="w3-table-all records_tables" id="participantsTable">
     <thead>
       <tr class="w3-light-grey">
         <th></th>
         <th>First name/Surname</th>
         <th>Email address</th>
         <th>Last Login</th>
         <th>Actions</th>
       </tr>
     </thead>
     <tbody><%- include('include/_participantRecords') %></tbody>
   </table>
  </div>
 </div>
```

Source Code 24 EJS Participants (By author)

Source Code 24 illustrates a table record of *"Participants"*. The table comprises five columns, and for better maintainability, the body of the table has been separated into a separate file named *"_participantRecords.ejs"*.

```
<div class="fullWidth">
  <% if(typeof course.hasAttendance !== "undefined"){%>
       <%-include('include/_attendanceRecords') %>
  <%}else{%>
       <%-include('include/_newAttendance') %>
  <%}%>
</div>
```

Source Code 25 EJS Attendance (By author)

Source Code 25 displays the implementation of an attendance feature for a course. This code block includes an if-else statement. The if-block checks whether the attendance has already been created, and if so, it loads a record from another included file named *"_attendanceRecords.ejs"*. If the attendance has not yet been created, the else block will be executed, and a form for creating new attendance will be shown from an included file named *"_newAttendance.ejs"*. This form allows the user to create a new attendance record.

4.4.2 Client-Side

In this chapter, the focus will be on user interaction, as well as retrieving or storing data from a database, specifically using JavaScript code on the client side. As a result, this chapter will cover two main use cases that have an impact on the project objective.

4.4.2.1 Use Case for Enrollment Students to Course

Source Code 26 demonstrates the functionality of the *"Enroll Student"* button, which is identified by the button ID *"enrollStudents"*. Upon clicking the button, jQuery is utilized to locate the button by its ID and initiate a click event. The event triggers an inner function called *"showParticipants()"*. This function begins by constructing a URL with *"/enrolls?"* inserted in the middle and proceeds to utilize Axios to retrieve data regarding both *"enrolledStudents"* and *"unenrolledStudents"* for the course. The resulting list of students is displayed in a panel using the *"enrollment()"* function found in *"assets\javascript\actionHelper.js"*. After selecting the desired students for enrollment and clicking the save button, the *"updateEnrollment()"* function is invoked with a constructed URL from the *"showParticipants()"* function and an array list of students on the enrollment side as arguments. This data is sent to the server as a request body object via a put request. Upon successful saving, the browser is reloaded, and the updated list of enrolled students is displayed to the user.

```
$(function () { $("#enrollStudents").click(function () {
       showParticipants() })});

function showParticipants() {
  const urlSplit = window.location.href.split("?");
  const pathUrl = window.location.pathname;
  const url = `${BASE_URL}${pathUrl}/enrolls?${urlSplit[1]}`;
  $("html").css("overflow-y", "hidden");

  axios.get(url)
    .then((response) => {
     const enrolledStudents = response.data.enrolledStudents.student;
     const unenrolledStudents = response.data.unenrolledStudents;
     enrollment(unenrolledStudents, enrolledStudents, "Cancel",  "Save",
       url);
    }).catch((error) => { console.log(error); });
}
async function updateEnrollment(url, students) {
  try {
    const result = await axios({ method: "put", url: url,
      data: {student: students}});
    if (result.status == 200) location.reload();
    else location.reload();
  } catch (error) { console.log(error); }
}
```

Source Code 26 Enrollment Students button clicked (By author)

4.4.2.2 Use Case for QR Code Attendance Displaying

Source Code 27 demonstrates the implementation of a click event for a button labeled "Generate QR Code" with an Id of *"btnGenerateQR"*. The button is obtained using its Id through jQuery and a click event listener is set. Upon clicking the button, the browser navigator is checked to verify if it is enabled. If enabled, the *"getCurrentPosition()"* function is invoked from the *"geolocation"* property of the navigator to obtain the current location of the browser in terms of *longitude* and *latitude*, which is then stored in an object named *"location."* This *"location"* object is passed as an argument to the callback function of *"qrCodeInputBox()"*, which can be found in *"assets\javascript\actionHelper.js"*. The purpose of *"qrCodeInputBox()"* is to display a panel to the user that allows for the selection of a *maximum range distance* and *attendance session duration* through two dropdown buttons. Once the user has made their selections, they click the "*Generate"* button to invoke the callback function *"createQRCode()"*, which is illustrated in *Source Code 28*.

The function *"createQRCode()"* begins by creating an object variable called *"data"* to represent a *request body*. This object is then updated with additional properties such as *"objectLocation"* from an argument, *"attendanceDuration"*, and *"maxLength"* from a

dropdown value that was selected. Next, a URL path is constructed by adding *"/qrCode?"* to the *current URL*. A fetch request using Axios is then sent to the server via the *constructed URL* with the *constructed data* sent in a *Post request*. After receiving a response from the server, the function *"displayQRCode()"* is invoked, which is displayed to the user as a QR Code on *Source Code 29*.

```
$(function () { $("#btnGenerateQR").on("click", function () {
    if (navigator.geolocation) {
      $("html").css("overflow-y", "hidden");
      navigator.geolocation.getCurrentPosition(
        (position) => {
          const location = {
            latitude: position.coords.latitude,
            longitude: position.coords.longitude,
          };
          qrCodeInputBox( "Generate QR Code", "Generate", "Cancel",
             qrCodeBody, function () { createQRCode(location) });
        },
        (error) => { warningBox("Warning", warningContent, "Close"); }
      );
    } else {
      warningBox("Warning", "Geolocation is not supported on your browser!",
             "Close");
      }
  });
});
```

Source Code 27 Generate QR Code button clicked (By author)

```
const createQRCode = async (objectLocation) => {
  const data = {};

  data.objectLocation = objectLocation;
  $(".qrcode-form select").each(function (index) {
    let value = $(this).val();
    let key = $(this).attr("name");
    if (key === "maxLength") { data.maxLength = value;}
    else if (key === "attendanceDuration") {
      data.attendanceDuration = value;
    }
  });
  const url = `${window.location.pathname}/qrCode?${
    window.location.href.split("?")[1]}`;

  try {
    const response = await axios({ url: url, method: "post", data: data });
    displayQRCode(response.data.sessionId,
              response.data.attendanceDuration, url);
  } catch (error) { alert(new Error(error)); }
};
```

Source Code 28 QR Code Creation function (By author)

```
function displayQRCode(sessionId, duration, url) {
  const $content = `<div class='dialog-ovelay'>
   <div class="qrBackground w3-card-4">
    <h3>QR Code Attendance</h3>
    <div id="qrcodeCanvas"></div>
    <p class="timeDisplay"></p>
   </div></div>`;
  $("body").prepend($content);
  $("html").css("overflow-y", "hidden");

  let intervalId;
  let timer = duration * 60;
  intervalId = setInterval(() => {
    let minutes = Math.floor((timer % (60 * 60)) / 60)
      .toString().padStart(2, 0);
    let seconds = Math.floor(timer % 60).toString().padStart(2, 0);
    $(".timeDisplay").html(minutes+" min" + " : "+seconds+" s");
    timer--;

    if (timer < 0) { clearInterval(intervalId);
      $(".timeDisplay").html("Time up!");
      removeQRCodeSession(sessionId, url);
      $("html").css("overflow-y", "visible");
      $(".dialog-ovelay").fadeOut(200, function () { $(this).remove(); });
      window.reload();
    }
  }, 1000);
  new QRCode(document.getElementById("qrcodeCanvas"), {
    text: sessionId, width: 250, height: 250, colorDark: "#000000",
    colorLight: "#ffffff", correctLevel: QRCode.CorrectLevel.M });
}
```

Source Code 29 QR Code Display function (By author)

```
async function removeQRCodeSession(sessionId, url) {
  await axios({ method: "delete", url: url, data: {sessionId: sessionId} });
}
```

Source Code 30 QR Session Removal (By author)

Source Code 29 exemplifies the use of the *"displayQRCode()"* function to append a QR Code block template in *HTML* syntax, stored as the string variable *"$content"*, to the *"body"* element using jQuery. The *"setInterval()"* function is employed to create a countdown timer that executes a code snippet or calls a function every second. In parallel, the *"QRCode()"* function from the JavaScript library is utilized to transform the string of *"sessionId"* into a QR Code. Once the timer concludes, it halts and triggers the *"removeQRCodeSession()"* function from *Source Code 30* to remove the QR session from the database. The function *"removeQRCodeSession()"* uses *Axios* to send a *Delete* request to the server to delete a QR Session using a *constructed URL* argument and the *"sessionId"* as the request body.

4.4.3 Android Mobile Application

As the primary aim of this project is to design a native Android mobile application, utilizing the Android Studio IDE provides a significant advantage. The project structure is established within the Android Studio IDE and is illustrated in the *Figure 16* below.

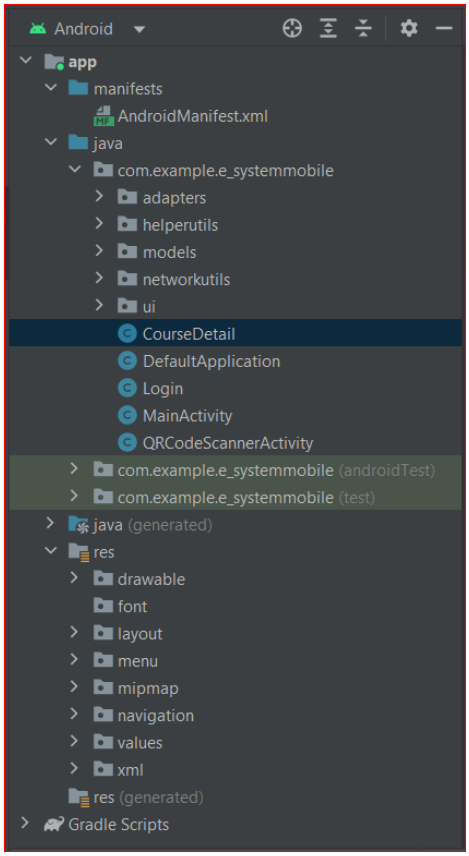

Figure 16 Android Studio Project Structure (By author)

The *"app"* folder contains three primary sub-folders: *"manifests,"* which houses the "AndroidManifest.xml" file that contains package information, including application components; *"java,"* which contains additional *Java code* sub-folders, such as *"adapters," "helperutils," "models," "networkutils," and "ui,"* which also include other *Java activity* files. Lastly, the *"res"* folder encompasses sub-folders such as *"layout," "drawable," "navigation," "menu," "value," and "font."* This folder predominantly contains *.xml* files.

4.4.3.1 Model

As Java is a statically typed language, it is crucial to define data types accurately to handle request and response objects from the server, particularly in JSON format. In this regard, Google's GSON Java library will be utilized to convert JSON into Java objects and vice versa.

Source Code 31 depicts a class defined for the *QR Provider's location*, which includes two variables: *"longitude"* and *"latitude."*

Source Code 32 defines a class for managing the QR session response object, which comprises four properties: *"maxLength," "unitSessionId," "courseName,"* and *"providerLocation"* from the *"ProviderLocation"* class.

```
public class ProviderLocation {
    private Double latitude;
    private Double longitude;
    public ProviderLocation(Double latitude, Double longitude) {
        this.latitude = latitude;
        this.longitude = longitude;
    }
    public Double getLatitude() { return latitude; }
    public Double getLongitude() { return longitude; }
}
```

Source Code 31 QR Provider Location Model (By author)

```
public class QRSession {
    private String maxLength;
    @SerializedName("objectLocation")
    ProviderLocation providerLocation;
    private String unitSessionId;
    private String courseName;
    public QRSession(String maxLength, ProviderLocation providerLocation,
        String unitSessionId, String courseName) {
        this.maxLength = maxLength;
        this.providerLocation = providerLocation;
        this.unitSessionId = unitSessionId;
        this.courseName = courseName;
    }
    public String getCourseName() { return courseName; }
    public String getMaxLength() { return maxLength; }
    public ProviderLocation getLocation() { return providerLocation; }
    public String getUnitSessionId() { return unitSessionId; }
}
```

Source Code 32 QR Session Model (By author)

4.4.3.2 HTTPs Request with Retrofit

This chapter will discuss the implementation of Retrofit to send an *HTTP request*, as previously mentioned in the theoretical part. *Source Code 33* presents an interface for defining a method to send an *HTTP request*, where each request is defined using the *"Call<T>()"* function, where *T* refers to the type of response object.

Source Code 34 presents a class that encapsulates a *Retrofit object*. The *"getClient()"* function checks if the *Retrofit object* is null, and if so, it constructs the *URL* to create the *Retrofit object* and returns it as a return type.

In summary, the class *"ApiUtils"* in *Source Code 35* returns a *Retrofit object* by passing the *"BASE_URL"* argument and creating an instance of the *RESTService interface*. The *RESTService interface* class enables the application to send *HTTP requests* to the *REST API* defined in the interface by calling the *"create()"* method of the *Retrofit client object*.

```
public interface RESTService {
    @GET("mobile/course/attendances/qrsession")
    Call<QRSession> getQRSession(@Query("token") String token,
@Query("sessionId") String sessionId);

    @PATCH("mobile/course/attendances/qrsession")
    Call<SubmittedResponse> getResponseUpdatedAttendance(
@Query("token") String token, @Query("unitSessionId") String unitSessionId);
}
```

Source Code 33 RESTService Retrofit interface (By author)

```
public class RetrofitClient {
    private static Retrofit retrofit = null;
    public static Retrofit getClient(String url) {
        if (retrofit == null) {
            retrofit = new Retrofit.Builder()
                    .baseUrl(url)
                    .addConverterFactory(GsonConverterFactory.create())
                    .build();
        }
        return retrofit;
    }
}
```

Source Code 34 RetrofitClient class base URL and GSON converter (By author)

```
public class ApiUtils {
    public final static String BASE_URL =
        "https://esystem-portal.herokuapp.com";

    public static RESTService getRESTService(){
        return RetrofitClient.getClient(BASE_URL).create(RESTService.class);
    }
}
```

Source Code 35 ApiUtils instantiates RetrofitClient (By author)

4.4.3.3 Android Permission

From Android SDK 23 onwards, users have greater control over granting permissions to apps to access specific functionalities such as camera and location. (Android Official Corporate team, 2019) This chapter demonstrates the permissions used in this project through the following code snippet.

```
binding.buttonQrcodeScan.setOnClickListener(v -> {
    intent = new Intent(getApplicationContext(),
        QRCodeScannerActivity.class);
    askPermission();
});

private void askPermission() {
    mPermissions = new String[]{Manifest.permission.CAMERA,
        Manifest.permission.ACCESS_FINE_LOCATION};
    if (!isPermissionsGranted(this, mPermissions)) {
        ActivityCompat.requestPermissions(this, mPermissions,
        PERMISSION_CODE_CAMERA_LOCATION);
    } else { startActivity(intent); }
}
```

Source Code 36 QR Code scanning button click's event (By author)

The aim of this mobile application development is to utilize the *camera* for QR Code scanning and *location* to access the user's current location. As a result, when the button is clicked, *"askPermission()"* function shown in *Source Code 36* is called to verify whether the user has granted these two permissions. This function also invokes another function called *"isPermissionsGranted()"* shown in Source Code 37, which returns true if all permissions are granted. If consent is not given, use the *"requestPermissions()"* function of the *"ActivityCompat"* class to request permissions from the user. The *"onRequestPermissionsResult()"* override function is called to determine the user's selection. If the user grants all permissions, start a new activity that involves QR codes. If not, display a dialogue box explaining why the permissions are necessary.

```
private boolean isPermissionsGranted(Context context, String...mPermissions)
{   for (String permission : mPermissions) {
        if (ActivityCompat.checkSelfPermission(context, permission)
            != PackageManager.PERMISSION_GRANTED) {
            return false; }
    }
    return true;
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
  super.onRequestPermissionsResult(requestCode, permissions, grantResults);
  switch (requestCode) {
      case PERMISSION_CODE_CAMERA_LOCATION:
          if (grantResults.length > 0 && grantResults[0] ==
        PackageManager.PERMISSION_GRANTED && grantResults[1] ==
        PackageManager.PERMISSION_GRANTED) {
              startActivity(intent);
          } else {
              AlertDialog.Builder builder = new AlertDialog.Builder(this);
              builder.setTitle(   ConstantVariable.Permission_Access_Denied)
                 .setMessage(ConstantVariable.Denied_Approved_By_User);
                 builder.setPositiveButton("OK", (dialog, which) ->
                 dialog.cancel());
                 builder.show();
          }
        return;
  }
}
```

Source Code 37 Function for checking Permissions (By author)

4.4.3.4 Layout

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".ui.qrscan.QRScanFragment">

    <com.budiyev.android.codescanner.CodeScannerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/qrcode_scanner_view"
        app:frameColor="@color/red"/>
</FrameLayout>
```

Source Code 38 Budiyev QR Scan Frame layout (By author)

This chapter will discuss two layouts, namely the *QR Scan fragment* and the *Attendance submission fragment*. The *QR Scan fragment* is demonstrated in *Source Code 38* and comprises a single view, namely *"com.budiyev.android.codescanner.CodeScannerView"*. Conversely, the *Attendance submission fragment* is presented in *Source Code 39* and

consists of three views, namely a fragment with Id *"@+id/fragment_map"* to display Google maps, a text view with *"@+id/textView_course_name"* to show a course name, and a button with Id *"@+id/button_present"* that allows attendance submission upon clicking.

```
<fragment
    android:id="@+id/fragment_map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_above="@id/textView_course_name" />
<TextView
    android:id="@+id/textView_course_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginVertical="20dp"
    android:textSize="@dimen/textView_size2"
    android:textColor="@color/white"
    android:layout_above="@+id/textView_date"/>
<Button
    android:id="@+id/button_present"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/present"
    android:layout_marginHorizontal="20dp"
    android:textColor="@color/white"
    android:backgroundTint="@color/dark_red"
    android:layout_alignParentBottom="true"
    android:layout_marginVertical="5dp"/>
```

Source Code 39 Attendance Submission Layout (By author)

4.4.3.5 Java Class

This session will cover the creation of a Java class for each fragment involved in the QR Code scanning process, as well as several additional auxiliary functions.

Source Code 40 demonstrates the implementation of code that decodes a QR Code into a text string. To begin, an object of the *"CodeScanner"* class is created and instantiated as *"mCodeScanner"*. A method called *"setDecodeCallback()"* is then defined for this object to manage the scanning process. A callback function is returned with the *"result"* argument, and *"runOnUiThread()"* is invoked in this function to permit code execution on the *UI thread*. An if-else condition is used to check if the result from decoding is not empty. If it is not empty, the string of the result is passed as an *argument bundle* and the app navigates to an *attendance fragment*. Lastly, a *"startPreview()"* method of *"mCodeScanner"* is defined to initiate the camera.

```
public class QRScanFragment extends Fragment {
    private FragmentQRScanBinding binding;
    private CodeScanner mCodeScanner;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup    container,
Bundle savedInstanceState) {
        binding = FragmentQRScanBinding.inflate(inflater, container, false);
        View root = binding.getRoot();
        return root;
    }
    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
         savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        mCodeScanner = new CodeScanner(getContext(),
         binding.qrcodeScannerView);
        mCodeScanner.setDecodeCallback(result ->
         getActivity().runOnUiThread(() -> {
             if (!result.getText().isEmpty()) {
                 Bundle args = new Bundle();
                args.putString(ConstantVariable.LOCATION_COURSE,
         result.getText().trim());
                 Navigation.findNavController(getActivity(),
         R.id.fragment_QRCode_ContainerView)
.navigate(R.id.action_QRScanFragment_to_attendanceFragment, args);
             }
         }));
        mCodeScanner.startPreview();
    }
}
```

Source Code 40 QR Scan Fragment Java Class (By author)

After navigating from the *"QR Scan Fragment"* to the *"Attendance Fragment,"* the latter initializes several private variables required for use within the class (shown in *Source Code 41*). Within the *"onCreateView()"* function, the *"mRestService"* variable of the *"RESTService"* interface is initialized using the "ApiUtils" class that implements the interface as per the Retrofit implementation. Additionally, the *"mPreferences"* object of the *"SharedPreferences"* class is initialized. In the *"onViewCreated()"* function, the *"fusedLocationProviderClient"* variable of the Interface is first initialized, followed by the initialization of the *"locationCallback"* object of the *"LocationCallback"* class and *"locationRequest"* object of *"LocationRequest"* class. The *"locationCallback"* object retrieves the *user's latest location*, if available, while the *"locationRequest*" object updates the *user's location* if the latest location is unavailable. Furthermore, an object *"mapFragment"* is instantiated from the *"SupportMapFragment"* class, and the *"getMapAsync()"* function is invoked to synchronize a map into a fragment. Finally, the

"verifyUserAndQRSession()" function (shown in *Source Code 42*) is called to verify whether the user meets the necessary conditions.

```
public class AttendanceFragment extends Fragment implements
OnMapReadyCallback {
    private QRSession mQRSession = null;
    private FragmentAttendanceBinding binding;
    private Location mLastLocation;
    private Marker mMarkerLastLocation;
    private GoogleMap mMaps;
    private final Float DEFAULT_ZOOM = 19f;
    private FusedLocationProviderClient fusedLocationProviderClient;
    private RESTService mRestService;
    private SharedPreferences mPreferences;
    private LocationCallback locationCallback;
    private LocationRequest locationRequest;
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    binding = FragmentAttendanceBinding.inflate(inflater, container, false);
    View view = binding.getRoot();
    mRestService = ApiUtils.getRESTService();
    mPreferences = getActivity().getSharedPreferences
        (ConstantVariable.SHARED_PREFERENCES, Context.MODE_PRIVATE);
    return view; }
@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    this.fusedLocationProviderClient =
        LocationServices.getFusedLocationProviderClient(getContext());
    locationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(@NonNull LocationResult locationResult)
          {
            if (locationResult != null &&
                  locationResult.getLastLocation() != null) {
                mLastLocation = locationResult.getLastLocation();
               fusedLocationProviderClient
                  .removeLocationUpdates(locationCallback)}
        }
    };
    locationRequest = new
        LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCURACY)
            .setIntervalMillis(5000).setMinUpdateIntervalMillis(2000)
            .setMinUpdateIntervalMillis(4).build();
    SupportMapFragment mapFragment =
        (SupportMapFragment)getChildFragmentManager()
         .findFragmentById(R.id.fragment_map);
    mapFragment.getMapAsync(this);
    verifyUserAndQRSession(mPreferences.getString("Token", ""),
        getArguments().getString(ConstantVariable.LOCATION_COURSE));
    binding.buttonPresent.setOnClickListener(v -> {
        submitAttendance(mPreferences.getString("Token", ""),
        mQRSession.getUnitSessionId()) });
}}
```

Source Code 41 Attendance Fragment Java Class (By author)

```
private void verifyUserAndQRSession(String token, String sessionId) {
    Call<QRSession> call = mRestService.getQRSession(token, sessionId);
    call.enqueue(new Callback<QRSession>() {
    @Override
    public void onResponse(Call<QRSession> call, Response<QRSession> response) {
    if (response.isSuccessful()) {
    mQRSession = response.body();
    if (mQRSession != null) {
    if ((ContextCompat.checkSelfPermission(getContext(),
      Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED)
      && isLocationEnabled()) {
    Task<Location> locationTask = fusedLocationProviderClient.getLastLocation();
    locationTask.addOnSuccessListener(location -> {
    if (location != null) {
        mLastLocation = location;
        float[] result = new float[3];
        Location.distanceBetween(mQRSession.getLocation().getLatitude(),
        mQRSession.getLocation().getLongitude(), mLastLocation.getLatitude(),
        mLastLocation.getLongitude(), result);

        if (result[0] <= Float.parseFloat(mQRSession.getMaxLength()) +
        ConstantVariable.STANDARD_ERROR_DISTANCE) {
        binding.textViewCourseName.setText(mQRSession.getCourseName().toUpperCase());
        mMaps.moveCamera(CameraUpdateFactory.newLatLngZoom(new
        LatLng(mLastLocation.getLatitude(),
               mLastLocation.getLongitude()),DEFAULT_ZOOM));
        finishLoading();
    } else {
        showAlertDialogBadRequested(getContext(), R.string.invalid_request,
        R.string.outside_permissible_range, binding.pbLoadingSubmitAttendance);
        }
    } else {
        fusedLocationProviderClient.requestLocationUpdates(locationRequest,
        locationCallback, Looper.myLooper());
        requestedStatusAlertDialog(getContext(), R.string.invalid_request,
        ConstantVariable.UNABLE_LOCATION, binding.pbLoadingSubmitAttendance);
    }
    }).addOnFailureListener(e -> {
        requestedStatusAlertDialog(getContext(), R.string.invalid_request,
        e.getMessage(), binding.pbLoadingSubmitAttendance);
    });
    } else {
        showAlertDialogBadRequested(getContext(), R.string.invalid_request,
        R.string.missing_gps_permission_internet, binding.pbLoadingSubmitAttendance);
}
    } else {
        showAlertDialogBadRequested(getContext(), R.string.invalid_request,
        R.string.not_enrolled_student, binding.pbLoadingSubmitAttendance); }
    } else {
        showAlertDialogBadRequested(getContext(), R.string.invalid_request,
        R.string.not_enrolled_student, binding.pbLoadingSubmitAttendance); }
    }

    @Override
    public void onFailure(Call<QRSession> call, Throwable t) {
        Toast.makeText(getContext(), t.getMessage(), Toast.LENGTH_SHORT).show();
        getActivity().finish(); }
    });
```

Source Code 42 Verify User on QR Session function (By author)

The *Source Code 42* contains a function called *"verifyUserAndQRSession()"*, which requires two arguments: *"sessionId"* and *"token"*. This function begins by calling the

"getQRSession()" method of an object instantiated as *"mRestService"*. This method sends a request to the server to retrieve a *QR session*. The function then uses *"enqueue()"* to handle the response from the server. If there is no response, the *"onFailure()"* method is invoked to handle the error. If there is a response, the *"onResponse()"* method is called with the response as an argument. The function checks if the response was successful and assigns the response body to *"mQRSession"*. The function then checks if *"mQRSession"* is not empty and confirms that the user has granted all necessary permissions. If so, the function calls *"getLastLocation()"* to obtain the user's latitude and assigns it to an object called *"locationTask"* of type *"Task<Location>"*. The function invokes the *"addOnSuccessListener()"* method of *"locationTask"* if the operation is successful, and the *"addOnFailureListener()"* method if there is an error. The *"addOnSuccessListener()"* method returns a callback function with the *user's location* as an argument. The function checks if the location is not null and calculates the distance between the *provider's location* and the *current device location* using the *"distanceBetween()"* method. Finally, the function checks if the calculated distance is less than or equal to the maximum range provided, considering a *standard error distance*. If the distance is within range, the function displays a *"Submission Fragment"* to the user. If there is any failure or unsuccessful scenario, the function displays a *dialog box* to the user explaining the reason for the error. Once the user has acknowledged the message, the function navigates the user to the *home activity*.

The conclusive *Source Code 43* incorporates a function named *"submitAttendance()"*, which necessitates two parameters: *"unitSessionId"* and *"token"*. The aforesaid function is triggered once the user selects the *"Present"* button to submit the attendance. The function initiates a request to the server to update the user's attendance record by calling the *"getResponseUpdatedAttendance()"* method. The response to the request is handled using the *"enqueue()"* method, as discussed in *Source Code 42*. If a response is received, the *"onResponse()"* method is executed with the response as an argument. In both *successful* and *unsuccessful* scenarios, a *dialog box* appears on the user's screen displaying a success message or providing an explanation for the error. Upon acknowledging the message, the function redirects the user to the *home activity*.

```
private void submitAttendance(String token, String unitSessionId){
    startLoading();
    Call<SubmittedResponse> call =
        mRestService.getResponseUpdatedAttendance(token, unitSessionId);
    call.enqueue(new Callback<SubmittedResponse>() {
        @Override
        public void onResponse(Call<SubmittedResponse> call,
        Response<SubmittedResponse> response) {
            if(response.isSuccessful()){
                requestedStatusAlertDialog(getContext(), R.string.success,
                ConstantVariable.SUCCESS_RESPONSE,
                binding.pbLoadingSubmitAttendance);
            }else {
                requestedStatusAlertDialog(getContext(),
                R.string.failure_updated,
                ConstantVariable.UNABLE_UPDATE_ATTENDANCE,
                binding.pbLoadingSubmitAttendance);
            }
        }
        @Override
        public void onFailure(Call<SubmittedResponse> call, Throwable t) {
            Toast.makeText(getContext(), t.getMessage(),
                 Toast.LENGTH_SHORT).show();
            getActivity().finish();
        }
    });
}
```

Source Code 43 Submission Attendance function (By author)

4.5 Testing

The practical component of this project culminates with the testing phase, which is also critical to guarantee that the software is deployable and suitable for use by end-users. Proper testing ensures that the software is relatively defect-free after deployment. This section focuses solely on system testing and usability testing.

4.5.1 System testing

This testing level validates the software application as an integrated whole, ensuring that the system functions meet the specified requirements. The test is divided into two parts: individual testing of each system component (server, web application, and mobile application) during the implementation phase for each requirement case. This testing includes creating new course records, enrolling students in a course, generating attendance records, generating QR codes, scanning QR codes, and authentication, among other scenarios. All requirement and exception scenarios are tested to determine if the system can handle and cover all cases. If individual testing passes, the system is considered

acceptable. Otherwise, the system is debugged to identify and resolve the cause of the error.

After the local implementation and testing for each component are completed, the source code is uploaded to GitHub. The system's server is then hosted on Heroku web hosting to facilitate communication over the internet. The author then performs testing starting from the initial scenario, which involves creating a new user, creating a course step-by-step, and finally submitting attendance from the mobile application using a student account.

4.5.2 Usability testing

The primary objective of usability testing is to determine whether a system or software is user-friendly for end-users. During a usability testing session, a facilitator or moderator asks a participant to complete tasks using one or more specific user interfaces. As the participant performs each task, the researcher observes their behaviour and listens to their feedback. (Moran, 2019)

The system's usability was evaluated was evaluated by five participants through the following tasks:

- Logging into the web application with provided admin credentials.
- Creating a new student account.
- Creating a new course.
- Enrolling students into the created course.
- Creating a new attendance.
- Requesting to display a QR Code for the attendance session.
- Logging into the mobile application with created student credentials who were enrolled to courses.
- Scanning the QR Code displayed by the web application.
- Clicking "present" to submit an attendance.
- Attempting to login with a student who was not enrolled in any course using the same pattern.

After completing the tasks, participants were asked to provide feedback and suggestions for improvement.

5 Results and Discussion

5.1 Test results

The testing results of the system's internet communication were mostly successful, except for one significant issue. During the testing process of scanning a QR code to submit attendance, the author encountered rejections in 9 out of 10 attempts due to the mobile device's location being outside the permissible range.

The author thoroughly debugged the process and found that the problem arose from the step of taking the QR provider's location by accessing a navigator from a computer, as stated in *Source Code 27*. Current technology limits the ability of computers to obtain an accurate geolocation, with the navigation from a computer only providing the geolocation of the nearest network antenna or connectivity. Therefore, the issue occurred because the computer did not provide an accurate location of the current device.

As this defect is a crucial requirement of the project objective, a solution must be found. The author has proposed an idea to address this issue, which will be discussed in the next chapter titled *"System adjustment"*.

Please note that the usability testing conducted involved addressing the issues discovered during the system's testing phase. The testing was carried out by five participants, who completed a total of nine tasks as previously specified.

During the initial six tasks, the web application was utilized, and all 5 participants were able to successfully complete the tasks. However, two out of the five participants reported having trouble in resizing the web application, indicating that it is not responsive to varying screen sizes. Additionally, one participant expressed dislike towards the background colour of the application due to its brightness.

In the subsequent three tasks involving the use of a mobile application, all 5 participants were able to complete the tasks from logging in to submitting attendance. However, one participant encountered an issue with the application's location services during the first attempt, resulting in a dialogue box appearing with a message and directing user to the home activity. This participant was able to complete the task on their second try. One out of the five participants expressed dissatisfaction with the course icon being the same for all courses as it remained static.

Upon being requested to provide feedback, all participants responded positively towards the entire system despite some issues with its responsive layout and colour scheme. Furthermore, three out of the five participants acknowledged the system's quick responsiveness.

Participants provided the following recommendations to enhance the system:

- Introduction of a loading indicator for the login view of the mobile application.
- Introduction of a search and sort functionality for courses, participants, and attendances.
- Introduction of customizable options on the mobile application, enabling the modification of icons and background colours.
- Implementation of a responsive layout for various screen resolutions.

5.2 Discussion

5.2.1 System adjustment

The forthcoming session will encompass the modifications made to the system to address the issues raised in the preceding chapter *"Test results"*.

To resolve the issue, a dropdown selection was implemented to enable the selection of classroom numbers, rather than retrieving the location from the navigator geolocation. The select options retain a value of the geolocation in key-value pairs and are dynamically populated by fetching records from the database. Additionally, a new server route, form input and mongoose schema were implemented to handle requests for creating new key-value pairs of classrooms and their geolocations, accessible solely to admins.

5.2.2 Proposed optimizations

The present endeavour represents a prototype of an education system intended for practical use. It is contended that the said system may be refined and developed for public use. To be precise, the author advocates for enhancements in the following aspects:

- Enhancing security measures for both client-side and mobile applications, which entail storing tokens securely, expiring tokens after logging out, and preventing unauthorized access.

- Introducing additional functionalities for users on both client-side and mobile applications, such as modifying and deleting courses, users, customizing themes, adding bookmarks or favourites, setting grades and points, content downloading, creating calendars, meetings, and receiving notifications, among others.
- Improving the layout and design of both client-side and mobile applications to cater to varying screen sizes.
- Expanding the data structure by extending the database and implementing additional Mongoose schemas for other information, including course content, timestamps for each transaction, among others.

5.2.3 Capturing UI Output in Implemented Applications

This chapter will present several illustrations depicting the User Interface (UI) of implemented applications.

- Web Application

Figures 17-19 depicted deployed web application screenshots. *Figure 17* depicted an attendance session with a default absence selection, showcasing a list of enrolled students. *Figure 18* displayed a popup window prompting supplementary input in a QR Session request, which included classroom, maximum attendance duration, and allowable range limit. Lastly, *Figure 19* depicted the display of a scannable QR Code.

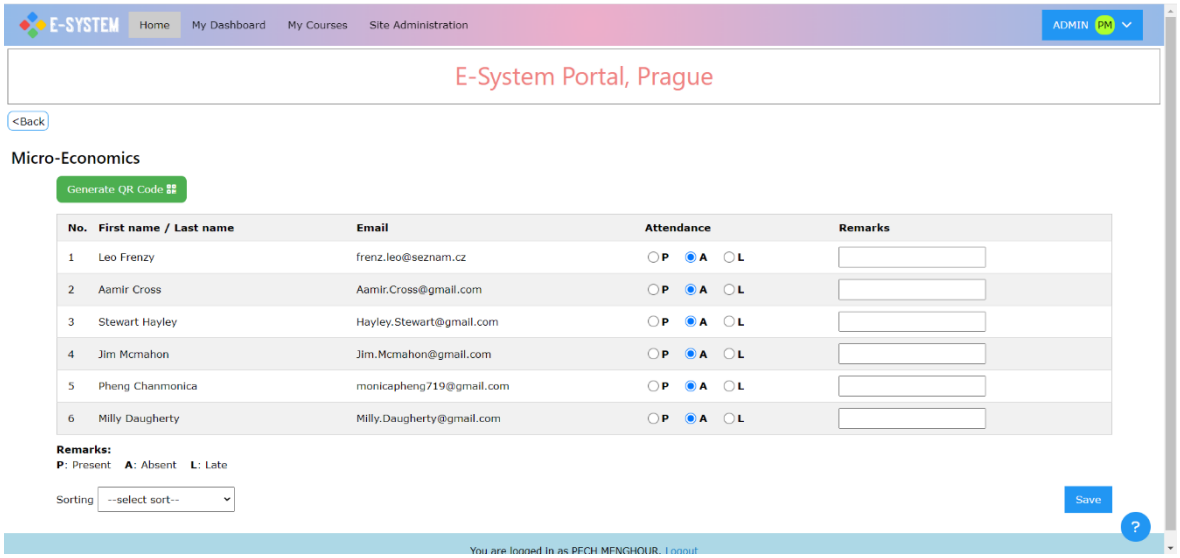

Figure 17 Web Application of Attendance Session (By author)

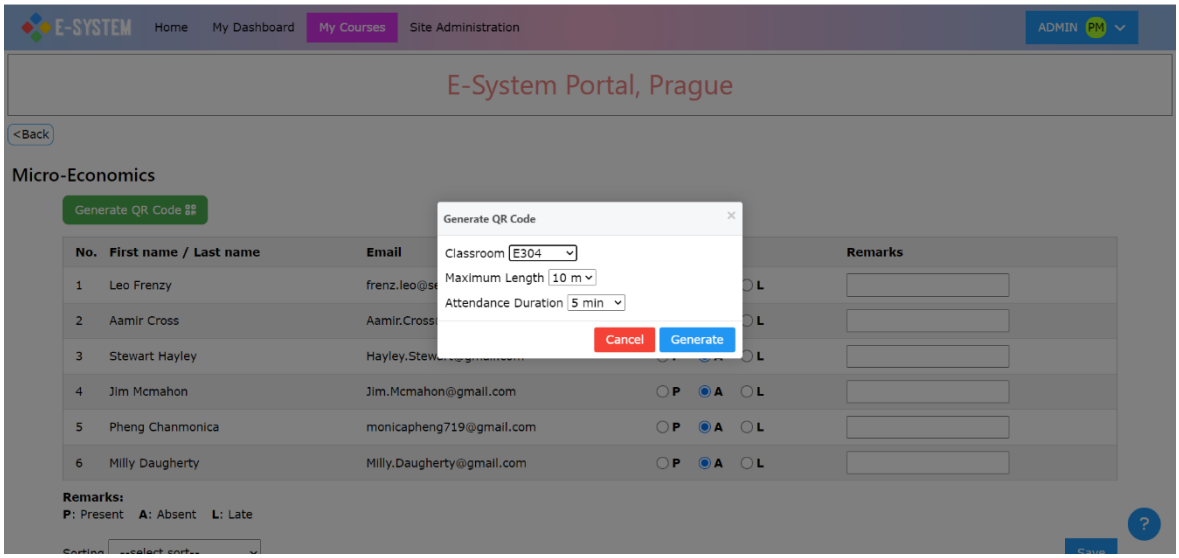

Figure 18 Web Application of QR Session requesting (By author)

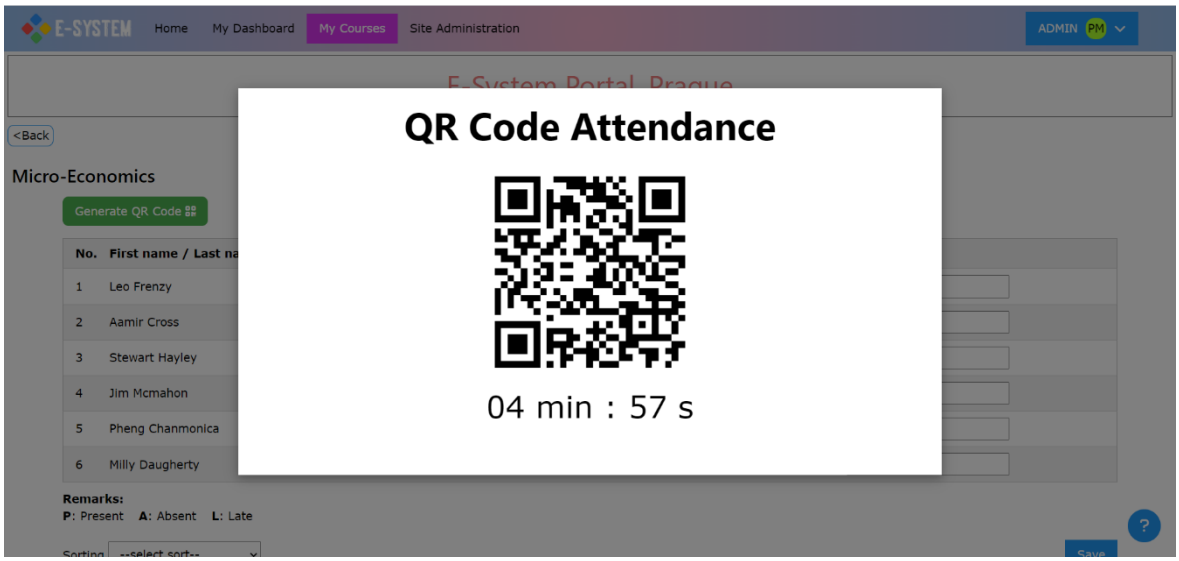

Figure 19 Web Application of QR Code display (By author)

- Mobile Application

Deployed mobile application screenshots were shown in Figures 20-22. Figure 20 depicted a dashboard activity displaying a list of enrolled courses and a QR scan button in the lower right corner. Figure 21 displayed a QR scanning fragment, while Figure 22 depicted an attendance submission fragment where students could submit attendance by clicking on a present button.

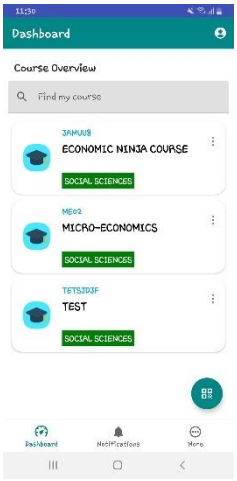

Figure 20 Home Activity (By author)

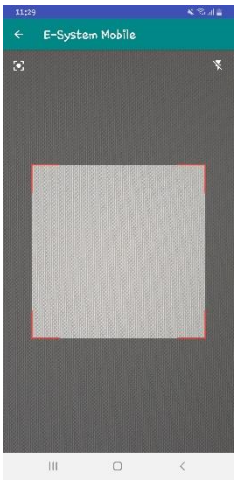

Figure 21 QR Scanning fragment (By author)

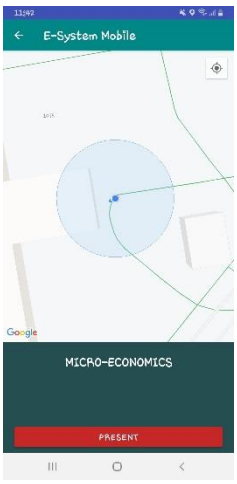

Figure 22 Attendance Submission fragment (By author)

6 Conclusion

The primary objective of that thesis project was to develop an Android mobile application that enabled students to scan a QR code for attendance verification. However, to accomplish that goal, a comprehensive system consisting of a web application and server was necessary. Several technologies, such as JavaScript, Java, Express.js, HTML, EJS, CSS, XML, and MongoDB with Mongoose for defining schemas, were employed in the system implementation. The theoretical section of the thesis outlined the concepts that underpinned the system implementation, while the discussion section examined the prototype's advantages and recommended improvements that could be implemented in the system.

This work proposed three critical propositions. Firstly, it suggested generating a QR Code session that was saved on a database for a pre-determined period and then erased. Secondly, it proposed introducing an education system portal that leveraged QR Code scanning to check attendance. Lastly, the work proposed incorporating a mechanism within the mobile application to verify that students were indeed present in the classroom. This was accomplished by verifying the student's current location against the stored location of the QR code's provider in a database. If the system deemed the location acceptable, the student could confirm their attendance.

The theoretical advantages of each technology and framework were described, and the software development process involved planning as the first phase and testing as the final phase. Each technology was utilized based on its specific role within the system. The system exhibited improved performance in objective processes after undergoing testing. Due to the system's communication through the internet using request and response, MongoDB was selected for its fast data-driven database, Express.js for rapid implementation of asynchronous request and response, Retrofit for its efficient HTTP request capabilities, and the new Play-service-location feature for more precise and expeditious location information in contrast to the previous LocationManager approach.

In my viewpoint, the technologies and frameworks employed in this project have the potential to construct a cohesive system that integrates a web application, mobile application, and server. The system, which adheres to a designated design pattern, is anticipated to yield enhanced performance and efficient data processing, particularly for commercial websites.

7 References

Adityamshidlyali, 2022. *A Complete Guide to Learn XML For Android App Development.* [Online]
Available at: https://www.geeksforgeeks.org/a-complete-guide-to-learn-xml-for-android-app-development

Admin BWC, 2019. *Software Development Life Cycle (SDLC).* [Online] Available at: https://bigwater.consulting/2019/04/08/software-development-life-cycle-sdlc [Accessed 22 02 2023].

Ado Kukic, Stanimira Vlaeva, 2021. *MongoDB & Mongoose: Compatibility and Comparison.* [Online]
Available at: https://www.mongodb.com/developer/languages/javascript/mongoose-versus-nodejs-driver/

Amit Phaltankar, Juned Ahsan, Michael Harrison, and Liviu Nedov, 2020. *MongoDB Fundamentals : A Hands-On Guide to Using MongoDB and Atlas in the Real World.* First ed. s.l.:Packt Publishing, Limited.

Android Official Corporate team, 2019. <*uses-permission-sdk-23*>. [Online] Available at: https://developer.android.com/guide/topics/manifest/uses-permission-sdk-23-element
[Accessed 07 03 2023].

Android Official Corporate team, 2021. *Add maps.* [Online] Available at: https://developer.android.com/training/maps

Android Official Corporate team, 2022. *Layouts.* [Online] Available at: https://developer.android.com/develop/ui/views/layout/declaring-layout

Android Official Corporate team, 2022. *The activity lifecycle.* [Online] Available at: https://developer.android.com/guide/components/activities/activity-lifecycle

Android Official Corporate team, 2023. *Application Fundamentals.* [Online] Available at: https://developer.android.com/guide/components/fundamentals

Android Official Corporate team, 2023. *Content provider basics.* [Online] Available at: https://developer.android.com/guide/topics/providers/content-provider-basics

Android Official Corporate team, 2023. *Get the last known location.* [Online] Available at: https://developer.android.com/training/location/retrieve-current

Android Official Corporate team, 2023. *LocationManager.* [Online] Available at: https://developer.android.com/reference/android/location/LocationManager

AWS Official Corporate team, n.d. *What Is A RESTful API?.* [Online] Available at: https://aws.amazon.com/what-is/restful-api/

Axios Official Corporate team, n.d. *Getting Started Axios.* [Online] Available at: https://axios-http.com/docs/intro

Braktim99, 2021. *Difference between index.ejs and index.html.* [Online] Available at: https://www.geeksforgeeks.org/difference-between-index-ejs-and-index-html/

Budiyev, Y., 2018. *yuriy-budiyev/code-scanner.* [Online] Available at: https://github.com/yuriy-budiyev/code-scanner

Chitrasingla2001, 2022. *Functional vs Non Functional Requirements.* [Online] Available at: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements [Accessed 23 02 2023].

Chugh, A., 2022. *Android BroadcastReceiver Example Tutorial.* [Online] Available at: https://www.digitalocean.com/community/tutorials/android-broadcastreceiver-example-tutorial

Chugh, A., 2022. *Retrofit Android Example Tutorial.* [Online] Available at: https://www.digitalocean.com/community/tutorials/retrofit-android-example-tutorial

Dora & Arthur, 2020. *Android Runtime Environment: DVM vs ART.* [Online] Available at: https://intexsoft.com/blog/android-runtime-environment-dvm-vs-art

EJS official site, n.d. *EJS.* [Online]
Available at: https://ejs.co/

Expressjs Official Corporate team, n.d. *Express : Fast, unopinionated, minimalist web framework for Node.js.* [Online]
Available at: https://expressjs.com/

Expressjs Official Corporate team, n.d. *Writing middleware for use in Express apps.* [Online]
Available at: https://expressjs.com/en/guide/writing-middleware.html

HANNAH, J., 2022. *What Exactly Is Wireframing? A Comprehensive Guide.* [Online]
Available at: https://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide [Accessed 26 02 2023].

Heroku Official Website, n.d. *Heroku is a cloud platform that lets companies build, deliver, monitor and scale apps — we're the fastest way to go from idea to URL, bypassing all those infrastructure headaches..* [Online]
Available at: https://www.heroku.com/what
[Accessed 24 02 2023].

Herron, D., 2020. *Node. js Web Development : Server-Side Web Development Made Easy with Node 14 Using Practical Examples.* Firth ed. s.l.:Packt Publishing, Limited.

Hiwarale, U., 2018. *How does JavaScript and JavaScript engine work in the browser and node?.* [Online]
Available at: https://medium.com/jspoint/how-javascript-works-in-browser-and-node-ab7d0d09ac2f

Java Official Corporate team, n.d. *What is Java technology and why do I need it?.* [Online]
Available at: https://www.java.com/en/download/help/whatis_java.html

Jonathan Chaffer, Karl Swedberg, Karl Swedberg, 2013. *Learning JQuery.* Fourth ed. s.l.:Packt Publishing, Limited.

jQuery Corporate team, n.d. *jQuery.* [Online]
Available at: https://jquery.com

Kanjilal, J., 2016. *How Java is Used in Android App Development.* [Online] Available at: https://www.developer.com/mobile/java-mobile/java-mobile-programming-for-android

Lahoti, S., 2019. *Defining REST and its various architectural styles.* [Online] Available at: https://hub.packtpub.com/defining-rest-and-its-various-architectural-styles/

Lavishgarg26, 2022. *Broadcast Receiver in Android With Example.* [Online] Available at: https://www.geeksforgeeks.org/broadcast-receiver-in-android-with-example

Magaji, J., 2020. *The History of Node.js.* [Online] Available at: https://www.section.io/engineering-education/history-of-nodejs/

MDN contributors, 2023. *What is a web server?.* [Online] Available at: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

MDN contributors, n.d. *Express/Node introduction.* [Online] Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

Mixon, E., 2022. *Android OS.* [Online] Available at: https://www.techtarget.com/searchmobilecomputing/definition/Android-OS

MongoDB Official Corporate team, n.d. *Customer success stories.* [Online] Available at: https://www.mongodb.com/who-uses-mongodb

MongoDB Official Corporate team, n.d. *MongoDB Atlas Tutorial.* [Online] Available at: https://www.mongodb.com/basics/mongodb-atlas-tutorial [Accessed 24 02 2023].

MongoDB Official Corporate team, n.d. *What Is MongoDB?.* [Online] Available at: https://www.mongodb.com/what-is-mongodb

Moran, K., 2019. *Usability Testing 101.* [Online] Available at: https://www.nngroup.com/articles/usability-testing-101 [Accessed 07 03 2023].

Nordeen, A., 2018. *Why Java is the best programming language for mobile applications.* [Online]
Available at: https://irishtechnews.ie/why-java-is-the-best-programming-language-for-mobile-applications

Official Node.js Corporate team, n.d. *Node.js v19.6.0 documentation.* [Online] Available at: https://nodejs.org/api/packages.html

Olusola, S., 2021. *How to use EJS to template your Node.js application.* [Online] Available at: https://blog.logrocket.com/how-to-use-ejs-template-node-js-application

Pulkitagarwal03pulkit, 2020. *Advantages and Disadvantages of CSS.* [Online] Available at: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-css

Raval, N., 2022. *Top 10 JavaScript Usage Statistics to Watch Out for in 2022.* [Online] Available at: https://radixweb.com/blog/top-javascript-usage-statistics

Retrofit Official Corporate author, n.d. *Retrofit.* [Online] Available at: https://square.github.io/retrofit

RISHU_MISHRA, 2020. *Content Providers in Android with Example.* [Online] Available at: https://www.geeksforgeeks.org/content-providers-in-android-with-example

RISHU_MISHRA, 2020. *Services in Android with Example.* [Online] Available at: https://www.geeksforgeeks.org/services-in-android-with-example

S., A., 2023. *What Is HTML? Hypertext Markup Language Basics Explained.* [Online] Available at: https://www.hostinger.com/tutorials/what-is-html#The-History-of-HTML

Siddiqui, T., 2022. *What is the JavaScript Engine and How It Works.* [Online] Available at: https://www.htmlgoodies.com/javascript/what-is-the-javascript-engine-and-how-it-works/

Stoyan Stefanov, Kumar Chetan Sharma, 2013. *Object-oriented JavaScript.* Second Edition ed. s.l.:Packt Publishing, Limited.

Team, Indeed Editorial, 2021. *Client-Side vs. Server-Side: What's the Difference?*. [Online]
Available at: https://www.indeed.com/career-advice/career-development/client-side-vs-server-side

W3C Corporate Team, n.d. *HTML & CSS.* [Online]
Available at: https://www.w3.org/standards/webdesign/htmlcss

WHATWG official site, 2023. *HTML.* [Online]
Available at: https://html.spec.whatwg.org/multipage/introduction.html

8 List of pictures, tables, graphs, and abbreviations

8.1 List of figures

8.2 List of tables

8.3 List of abbreviations

AOT – Ahead-of-Time

APK – Android Package Kit

API – Application Program Interface

ART – Android Runtime

CSS – Cascading Style Sheets

DOM – Document Object Model

DVM – Delvik Virtual Machine

EJS – Embedded JavaScript templates

HTML – HyperText Markup Language

JSON – JavaScript Object Notation

JVM – Java virtual machine

NoSQL – not only SQL

QR – Quick Response

REST – Representational State Transfer

SDK – Software Development Kit

SDLC - Software Development Lifecycle

SQL – Structured Query Language

UI – User Interface

URL – Uniform Resource Locator

W3C – World Wide Web Consortium

XML – Extensible Markup Language