Department of Computer Science
Faculty of Science
Palacký University Olomouc

BACHELOR THESIS

Mobile and Web Application International Student Guide

2020

Dominika Gajdová

Supervisor: Mgr. Jiří Zacpal, Ph.D.

Study field: Applied Computer Science, full-time form

Bibliografické údaje

Autor:	Dominika Gajdová
Název práce:	Mobilní a webová aplikace International Student Guide
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Jiří Zacpal, Ph.D.
Počet stran:	42
Přílohy:	1 CD/DVD
Jazyk práce:	anglický

Bibliographic info

Author:	Dominika Gajdová
Title:	Mobile and Web Application International Student Guide
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Jiří Zacpal, Ph.D.
Page count:	42
Supplements:	1 CD/DVD
Thesis language:	English

Anotace

Tato práce se zabývá tvorbou mobilní a webové aplikace Student Guide. Jejím obsahem je především technická specifikace a uživatelská dokumentace. Cílem práce bylo zmodernizovat přístup k informacím zahraničním studentům univerzity a zároveň tak poskytnout efektivnější způsob pro aktualizaci. Mezi hlavní funkce mobilní aplikace patří prohlížení článků, kontaktů, map a dalších důležitých informací. Webová aplikace je určena pro tvorbu a aktualizaci samotných dat.

Synopsis

This thesis concerns with the creation of a Student Guide mobile and web application. It mainly consists of a technical specification and user documentation. The aim of this thesis was to modernize access to information for the foreign university students and also provide a more effective way of keeping information up to date. The main features of the mobile application include reading articles, contacts, map and other important information. The web application serves the purpose of creating and updating the data itself.

Klíčová slova: Android; mobilní aplikace; ASP.NET Core; webová aplikace; REST API; SQL; průvodce pro zahraniční studenty, webový editor dat, Student Guide

Keywords: Android: Mobile Application; ASP.NET Core; Web Application; REST API; SQL; Guide for International Students, Web Data Editor, Student Guide

I would like to thank Mgr. Jiří Zacpal Ph.D for making this project possible, my family for continual support and my beloved one for being by my side and helping me see things from a different perspective.

I hereby declare that I have completed this thesis including its appendices on my own and used solely the sources cited in the text and included in the bibliography list.

date of thesis submission	author's signature

Contents

List of Figures

List of Tables

1 Introduction

The Student Guide project started from the initiative of Dana Gronychová who works at the student department of the Faculty of Science where she mainly works with foreign students. She addressed Mgr. Jiří Zacpal, Ph.D., who is the supervisor of this bachelor thesis, with the need of an application which would give students the ability to get answers to their questions in an easy and modern way.

The goal of the mobile application is mainly to modernize information access for international students. It is going to be used as a replacement for the printed version of the guide which nowadays is not that popular (which is understandable; students prefer to use their phones to access information rather than carry a printed version with them). It also simplifies the way of information sharing — no updated versions need to be printed which is very important nowadays when things change every day.

With the way the mobile application should work, there also grew the need of a data editor for the employees of student departments. And that is what the second half of this thesis focuses on: an online data-editing system available anywhere with an internet connection with access to only authorized employees.

2 Existing Solutions

There are many existing applications of this kind, but they differ greatly among each other. All the apps that I have come across were made by universities abroad. Not many of them were specifically created for foreign students though; they are mostly meant to serve as a student guide for all university students. Some focus on showing off the campus buildings (maps of the buildings as well as directions), other come closer to what this project is meant to be: a university/travel guide full of important information starting from getting a visa to local tips for clubs and places to visit.

2.1 Example of an Existing Application

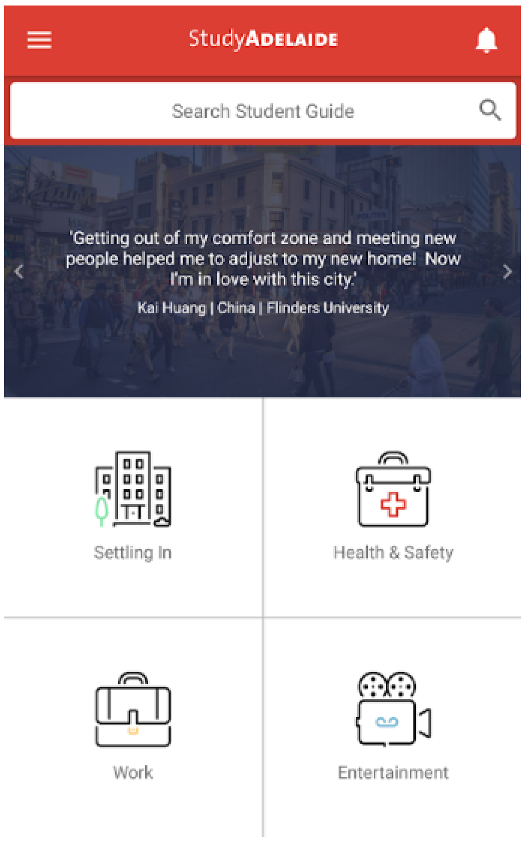

StudyADELAIDE
Search Student Guide
'Getting out of my comfort zone and meeting new people helped me to adjust to my new home! Now I'm in love with this city.'
Kai Huang | China | Flinders University
Settling In
Health & Safety
Work
Entertainment

Figure 1: University of Adelaide student app

The Australian StudyAdelaide app is the closest example of what the International Student guide is supposed to be like. Apart from important study information it also contains features such as a map or news section. Both of these features are implemented in UPlikace — the university's app for Palacky students.

Maps and the possibility of custom news from faculties were also added to the Student Guide Application because UPlikace is only available for students that are already officially a part of the university. Student Guide is also intended to be used as a pre-university source of information for students that are thinking about applying.

The Adelaide guide is divided into well organized categories and then each category contains subcategories and each subcategory has another list of articles. I have decided to take a similar approach to designing Student Guide. The difference is that all the articles for a given subcategory are displayed in one page with links on top for easier navigation so the user has a choice: he can either navigate to the specific information he wants to obtain, or he can scroll through the page and read all the articles one by one.

Another thing featured in the Student Guide application is a search bar which is practically necessary for an application of this kind.

3 Solution Specification

3.1 Mobile Application

3.1.1 Platform

The targeted mobile operating system is Android which is, in spite of Apple's growing popularity, the most commonly used phone operating system in the world.

3.1.2 Java

Java still remains at the top of programming languages list when it comes to programmer's choice. Heavily used in the enterprise area, Java also happens to be the language of choice for Android development."Write once, run anywhere"[1] — a well known slogan nicely describes the purpose of Java. Rather than compiling to machine code directly it compiles into Java Bytecode, a specific set of instructions that is not dependent on computer architecture. Any computer running the Java Virtual Machine can then compile and run the program.

3.1.3 Android SDK and IDE

Android SDK (Android software development kit) is an Android toolbox library provided by Google and created for development. It contains all the components necessary for android development such as libraries or an emulator.

The most compatible Android IDE and the IDE that I used is Android studio created by JetBrains and Google. It uses custom Gradle build tool system that is used to build android packages (apk files) which contain the generated byte code as well as resources (images, UI, xml files etc). It also manages project dependencies and takes care of their newest versions.[1]

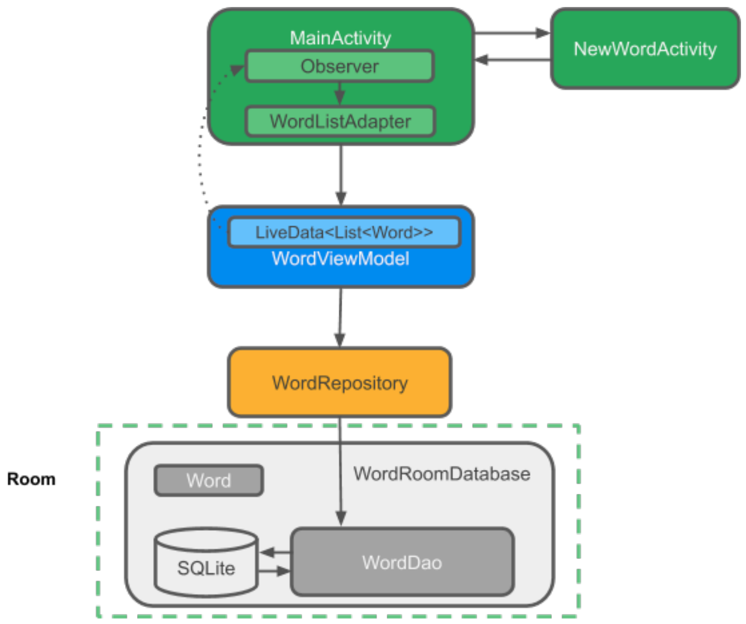
3.1.4 Android Jetpack

Android Jetpack is a collection of Android software components which make creating an Android application easier and more effective. The Student Guide's architecture is based on the Jetpack Architecture components, and the Android recommended MVVM: Model-View-ViewModel architecture was used in its creation.

The base of the architecture is represented by SQLite database abstracted by Room persistence library. Room creates a layer of abstraction from the database by allowing the user to access data without writing boilerplate code. It also serves as a single source of truth, meaning it is the application's main source of data.

Creating a table is as simple as annotating a model class with Room annotations.[2]

[1]A 1995 slogan created by Sun Microsystems



Source: https://codelabs.developers.google.com/codelabs/android-room-with-a-view/index.html?index=..%2F..index#14

Figure 2: MVVM architecture

When it comes to defining relationships between entities, there are two possible ways, each one having a slightly different purpose.

First one is described in the code example above, using the @ForeignKey annotation. When defining a new entity, one can specify a foreign key relationship between two entities. In the example above, each Article belongs to a Subcategory and contains the SubcategoryId which acts as the foreign key to a Subcategory id.[3]

Another way to create a relationship between two entities is using the @Relation annotation. The relationship is defined using a new class that unites the two entities between which you want to define a relationship. The difference from using the @ForeignKey annotation is that an @Embedded (nested) attribute and the @Relation attribute, which is a list of all the tuples containing the embedded's attribute primary key as a foreign key, have to be specified. This provides access to an element and a list of all related elements (subcategory and all the articles which belong to it) which is very useful in some cases.[4]

```
@Entity(tableName = "Article",
      foreignKeys = @ForeignKey(entity = Subcategory.class,
            parentColumns = "Id", childColumns = "SubcategoryId")
public class Article {

   @PrimaryKey
   @ColumnInfo(name = "Id")
   private int id;

   @ColumnInfo(name = "Title")
   private String title;

   @ColumnInfo(name = "Content")
   private String content;

   @ColumnInfo(name = "SubcategoryId")
   private int subcategoryId;

// getters, setters and constructors
}
```

Source code 1: @Entity Model

```
public class SubcategoryAndArticle {
   @Embedded
   private Subcategory subcategory;
   @Relation(
         parentColumn = "Id",
         entityColumn = "SubcategoryId"
   )
   private List<Article> articles;
}
```

Source code 2: @Relation relationship

Next component used in the bottom layer is a DAO (Data Access Object) interface that defines the operations that abstract access to the application database.[5]

Room transforms the custom query into a parametrized SQL query in order to avoid potential SQL injection attack.

Query results are returned as LiveData objects. LiveData is an observable data holder class that is lifecycle aware; it is usually paired with LifecycleOwner (class that holds the application's lifecycle) and its observer will be notified about modifications of the wrapped data only if the paired Lifecycle owner is active. With every database change (when Insert, Update or Delete operations are executed), Room database will notify all active observers and corresponding views will be updated.[6]

```
@Dao
public interface ArticleDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insertArticle(Article... articles);

    @Update
    void updateArticle(Article... articles);

    @Query("DELETE FROM Article WHERE Id=:id")
    void deleteArticle(long id);

    @Query("SELECT * FROM Article")
    LiveData<List<Article>> getAllArticles();

    @Query("SELECT * FROM Article WHERE SubcategoryId=:subId")
    LiveData<List<Article>> getAllArticlesForSubcategory(int subId);
}
```

Source code 3: @Dao

```
 //observes on LiveData
 articleViewModel.getAllArticles().observe(getViewLifecycleOwner(),
 //updates view
  articles -> recyclerAdapter.setArticles(articles));
```

Source code 4: Observing LiveData

Next layer is the Repository class which is just another level of abstraction for mixing different sources of data, such as database or fetching data from an API. It contains methods for accessing data from the DAO, as well as methods for asynchronous execution of insert, update and delete operations because Room does not allow main thread queries and it is understandable; all time-consuming operations should be executed in a background thread without blocking the UI thread. LiveData return values asynchronously by default.

ViewModel is the last part of the architecture and the final layer of abstraction. Its only responsibility is to manage data for the UI. ViewModel objects are scoped to the Lifecycle passed to the ViewModelProvider.[7] ViewModel is also used as a data sharing object where more fragments or activities can communicate. For example, login fragment might share its ViewModel with registration fragment where registration passes new data into ViewModel and login then uses the data to log the user in.

Apart from architecture components, the application also uses the new Navigation UI component, which makes it easier and faster to switch between fragments. The application consists of only one MainActivity that holds the toolbar, the navigation and the hosting fragment. All other views are fragments that live inside of the activity and that replace and get replaced by other fragments. The navigation component abstracts the developer from creating fragment transactions. Instead, a navigation graph can be created — an XML file which consists of fragments and actions defined between them. Using the SaveArgs plugin, corresponding classes are generated to pass and receive arguments through Bundles.

3.1.5 Retrofit

Retrofit is an easy to use type-safe HTTP client for making REST API calls. [8] First, a service interface which defines the REST calls the developer will make to a specific URL must be created. Then, both synchronous and asynchronous calls can be made and a JSON response received through a callback (in case of asynchronous calls) or through directly executing an API call. The JacksonJson converter is used in the application, but there are many other JSON converters supported.

```
public interface SyncDatabaseService {

    @GET("API/Changes/{key}")
    Call<List<Change>> getChanges(@Path("key") String APIKey);

    @DELETE("API/Changes/{key}")
    Call<Void> deleteChanges(@Path("key") String APIKey);

     @GET("API/Subcategories/{key}")
    Call<List<Subcategory>> getSubcategories(@Path("key") String
        APIKey);

}
```

Source code 5: Retrofit service API calls

3.1.6 Proguard

Size of the final application and its APK file is also very important, considering the Student Guide android application targets phones from API level 19 (Android 4.4 KitKat). This choice of API level ensures around 94% phone coverage. Users nowadays have many heavy applications installed on their smartphones (Facebook, Instagram, etc$\cdots$) so the plan was to make the Student Guide as lightweight as possible.

Proguard is a tool that helps programmers minify, obfuscate and optimize their code. There are two settings one can use:

- minifyEnabled = if true will obfuscate and minify all code that is not annotated with @Keep annotation

- shrinkResources = will remove all unused methods, classes, fields and attributes as well as unused resources in app and its dependencies

[9]
The application structure contains ProguardRules.pro file where the developer can set rules which will exclude chosen classes and libraries from being minified and obfuscated.

3.2 Web Application

3.2.1 C# and ASP.NET Core

C# is another popular programming language created by Microsoft in 2000 as a Java competitor. C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. .NET framework mainly consists of four things: the language itself, Visual Studio, Virtual Machine (CLR) and libraries (FCL).[10]

ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-based, Internet-connected applications. ASP.NET Core is a redesign of ASP.NET 4.x, with architectural changes that result in a leaner, more modular framework.[11] The Student Guide Web Editor is based on the 2.2 version.

ASP.NET Core is a rather complex web framework. There are many features that must be learnt before one is able to create anything, which makes it quite challenging for new developers. On the other hand, there are many features that are built in which save developers a lot of time; such as generating CRUD razor pages for a model or the Identity system which takes care of the login functionality.

3.2.2 MVC vs Razor Pages

ASP.NET Core has two architecture options: classic MVC or nowadays popular Razor Pages. MVC stands for Model View Controller and the architecture structures the logic of the application into three independent parts:

- Model — data abstractions (database tables)
- View — presents data to the user (HTML pages)
- Controller — reacts to events from the user and enables communication between Model and View

Razor Pages are very similar to the MVC view component. It has the same functionality and it also uses the Razor Syntax. The main difference is the file organization; each page in the application is self-contained with its own view and code organized together which makes it less complex than the MVC and closer

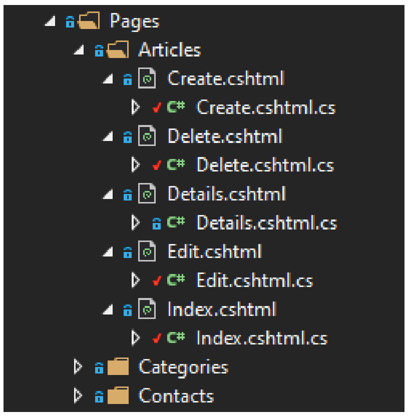


Figure 3: Razor page view and controller

to an MVVM architecture. Each .cshtml file has to be annotated with @Page and has to be located in the Pages folder.

Razor is a syntax developed specifically for C# language. It enables writing C# code along with HTML code in an interactive way. There are many razor pages templates available to be generated which makes it easier and time-saving for the developer. There are other technologies that can be used along with ASP.NET Core development. Because the C# code itself represents the server backend, technologies such as Javascript (vanilla or React, Angular), JQuery[2], Bootstrap[3] can be used for the frontend part of the application.

3.2.3 ASP.NET Core API & REST

API stands for application programming interface which is used for communication and data sharing between applications. In case of the Student Guide, there was a requirement to share data between the server database and the mobile application. ASP.NET core Web API lets you create a REST API (Representational State Transfer).

REST is a design pattern for creating and managing APIs. It takes into account two key parts:

- Client — the person who uses the API, for example a developer who needs to get data access will use the API endpoints to get the data they need
- Resource — any object that the API exposes information about, in case of the Student Guide, it can be subcategories, articles etc.

REST is based on using HTTP protocol and its GET, PUT, POST and DELETE methods. When we need to get certain object data, an HTTP request is made to a specific endpoint[4] and we receive a response in the form of either a JSON or XML.

[2]Javascript library which simplifies events, CSS animations, Ajax call etc.
[3]CSS library that contains predefined styles for the most commonly used web page elements
[4]Endpoint is a location in the server from which APIs can access the resources they need.

3.2.4 Entity Framework

Entity Framework is an open-source framework for mapping objects to a relational database. Its main purpose is to create a level of abstraction for working with the database. Instead of writing plain SQL commands, the developer can work with table data as objects and create queries with LINQ[5]. The DTOs (Data transfer objects) represent each database entity.

In order to use Entity Framework, all that is needed is creating a class that inherits from DbContext (in my case IdentityDbContext) and defines DbSet objects which represent each database table. Then, a new migration via the Nuget console has to be added and the database and all tables have to be updated, so that relationships between them are formed.

```
public class ApplicationDbContext : IdentityDbContext {
      public ApplicationDbContext(DbContextOptions<
         ApplicationDbContext> options) : base(options) { }

      public DbSet<st_guide_2.Models.Article> Article { get; set; }

      public DbSet<st_guide_2.Models.Contact> Contact { get; set; }

      public DbSet<st_guide_2.Models.FacArticle> FacArticle { get;
         set; }
}
```

Source code 6: Entity Framework DbContext class

Because ASP.NET Core uses dependency injection principles[6], the database instance will be saved in a dependency injection container therefore one does not create a new instance of the database each time but request the instance from the dependency injection service, and it will be injected to the desired class through its constructor.

3.2.5 ASP.NET Core Identity

Identity is a framework which provides login functionality to the application out of the box. Users can then create an account and have the information securely stored in Identity. It composes of database tables which handle the data storing and the Identity Razor pages, such as the login or register page.

Another useful feature is the roles system. Each user can have a role assigned to them for a better control of who can have access to what. For example, the

[5] Query language used to conveniently process data from arrays, enumerables, relational databases and more.

[6] Software design principle that states that objects that depend on other objects should not have the responsibility of managing the dependency and instead they should only have the dependency passed through a constructor which ensures the class is independent of its dependencies.

Admin can access everything but other users have a role assigned to them based on which faculty they are from, so they only get to see and edit articles from their faculty. Adding [Authorize] annotation to a class ensures that a person who is not logged in will not have access to the page and instead will be redirected to log in. Adding [Authorize(Roles= 'Admin')] will ensure only Admin will get access to the particular page.

Different login providers can be used with Identity, such as Facebook login or Google login etc.

4 Programmer's Documentation

4.1 Database

4.1.1 Shared Entities

These are the main data entities that are shared between the server and the mobile application. They consist of four synchronizable entities (Subcategory, FacArticle, Article, Contact) and two complementary tables (Faculty, Category).

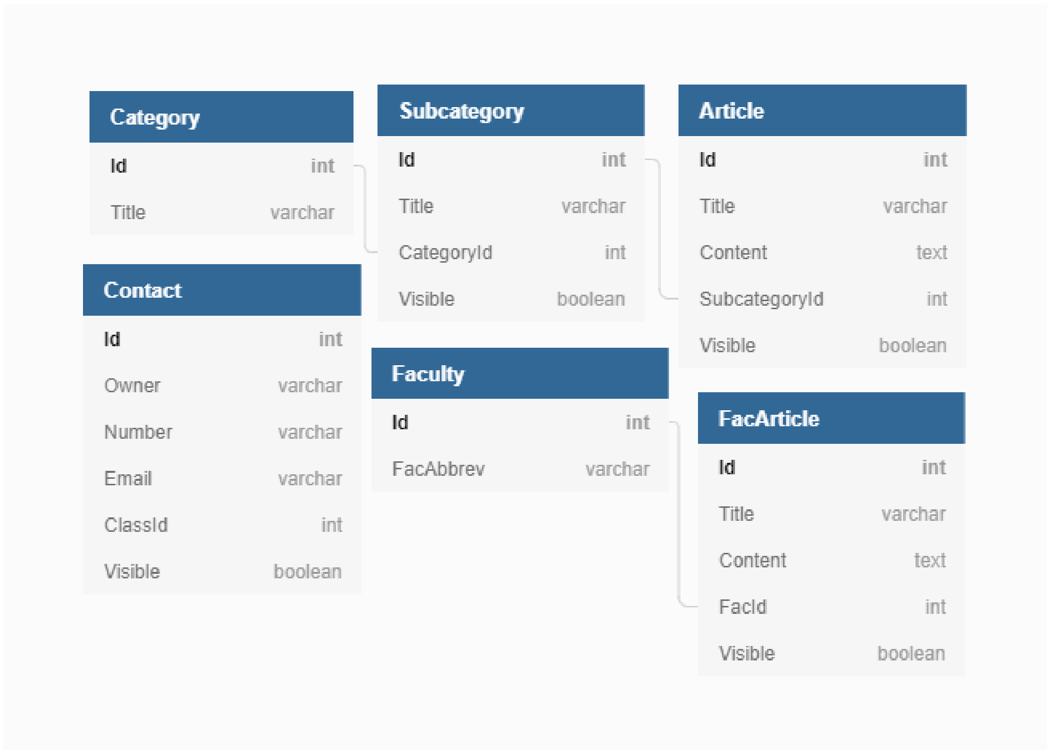


Figure 4: Shared tables - relation model

On the server side, all editable entities have an extra column called RowVersion. It is of a TimeStamp type and serves as a unique row number used during optimistic concurrency — if there are two users editing the same entity and one of them saves the changes, the other one is editing an outdated entity and should be notified.

4.1.2 Web Server

Division of the database into three logically separate parts:

1. Idenity tables — tables generated by the Identity framework. The main tables used in the project are:

    - AspNetUsers — stores secured user login information

- AspNetRoles — stores individual roles, each user can have a role assigned

2. Entity framework tables — shared tables
3. Image — represents an image uploaded to the server by user
4. Synchronization tables — tables related to sync functionality with the mobile application:
   - Change — keeps track of changes for each mobile application user
   - APIUser — keeps track of users using the mobile application
   - Token — keeps track of API users

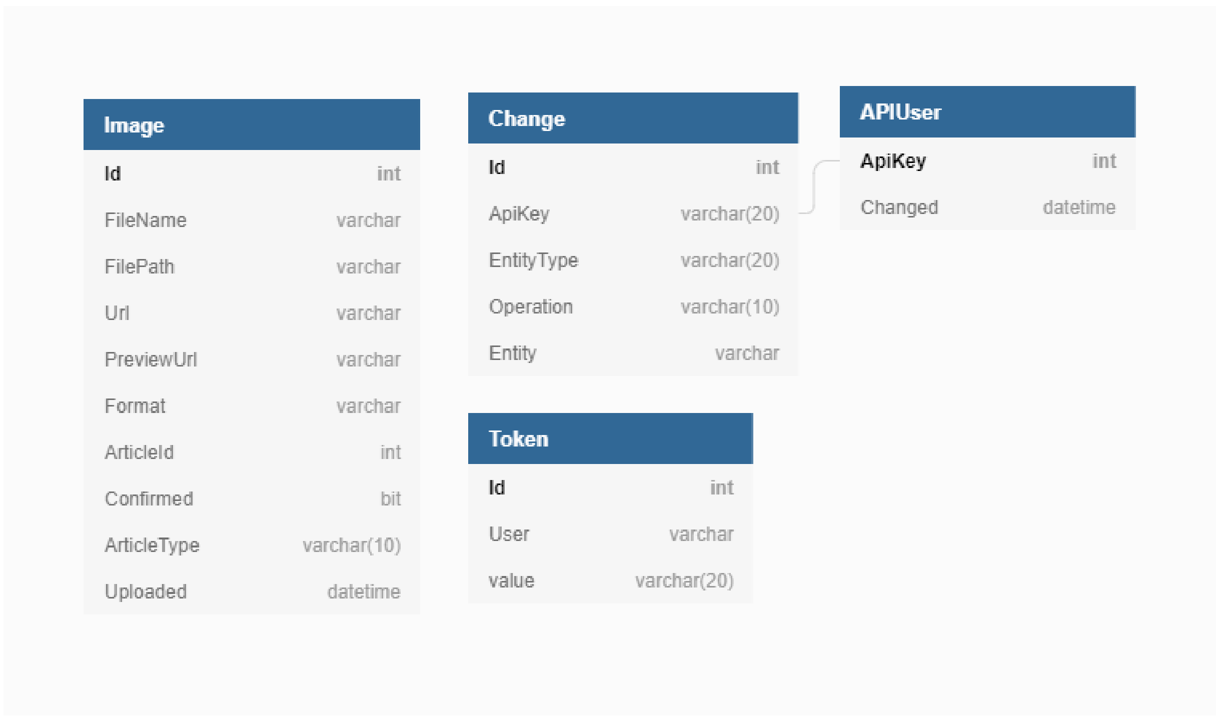


Figure 5: Sync related tables — relation model

4.2 API

The API is logically divided into two parts: Data API and Changes API. Data API provides elementary data such as articles, subcategories etc. Changes API is used to register an individual user and track changes specifically for them (sync functionality).

I differentiate between two key types:

- API Token — ID of an application having access to the API, is assigned to the user after being registered in the Tokens page of the server, can access data and call AddUser.
- API Key — generated when calling AddUser in order to track changes for a specific user, changes will be saved in the server for API User under the API Key, can access data and all other Changes requests.

In order to be able to use the API, one needs to have Admin register their application into the system. Then, an API Token will be generated, which will allow access to Data API as well as Changes API.

4.2.1 Data API

All API calls require an API Key as an argument. It can either be an API Key acquired by sending an AddUser API request from the Changes API or an API Token created in the Tokens section.

Request	GetCategories
URL	API/Categories/{key}
Return value	List<Category>

Request	GetFaculties
URL	API/Faculties/{key}
Return value	List<Faculty>

Request	GetSubcategories
URL	API/Subcategories/{key}
Return value	List<Subcategory>

Request	GetArticles - HTML
URL	API/Articles/{key}
Return value	List<Article>

Articles contain HTML content which preserves the form and style of elements created by user.

Request	GetArticles - plain text
URL	API/Articles/Text/{key}
Return value	List<Article>

Articles contain plain text without any HTML elements. All form, images and style are removed.

Request	GetFacArticles (Faculty Articles) - HTML
URL	API/FacArticles/{key}
Return value	List<FacArticle>

Request	GetFacArticles (Faculty Articles) - plain text
URL	API/FacArticles/Text/{key}
Return value	List<FacArticle>

FacArticles contain HTML content which preserves the form and style of elements created by user.

Articles contain plain text without any HTML elements. All form, images and style are removed.

Request	GetContacts - HTML
URL	API/Contacts/{key}
Return value	List<Contact>

Contacts contain HTML content which preserves the form and style of elements created by user. Email uses mailto attribute and number uses tel attribute to make them clickable and interactive.

Request	GetContacts - plain text
URL	API/Contacts/Text/{key}
Return value	List<Contact>

Contacts contain plain text without any HTML elements. All form and style are removed. Email and number attributes are not interactive nor clickable.

4.2.2 Changes API

The Changes API provides the basic sync functionality to keep the user up to date with the server database. Use AddUser to register a user in the system. Changes will be tracked for every registered user under their user API Key. AddUser and GetChanges are two core request functions, other functions are either optional (they are there if needed it is not necessary to use them) or recommended (good practice).

Request	AddUser
URL	API/Changes/AddUser/{key}
Return value	string (API Key)
Use	required

AddUser request checks whether provided Token is registered. If it is, it returns a new API key which will serve as the user identification — every application user registered in the Tokens section will have a unique API Key assigned and changes will be tracked for the user under this API Key.

Request	GetChanges
URL	API/Changes/{apiKey}
Return value	List<Change>
Use	required

Request	CheckKeyExists
URL	API/Changes/Check/{apiKey}
Return value	boolean
Use	recommended

Checks whether API Key is saved in the database. It is recommended to check if the key still exists in the server before making GetChanges request because the server deletes API Keys after long inactivity (year+).

Request	RemoveUser
URL	API/Changes/RemoveUser/{key}
Return value	void
Use	optional

Will delete API Key user from the server as well as his pending changes.

4.3 Mobile application

4.3.1 Use Case Diagram

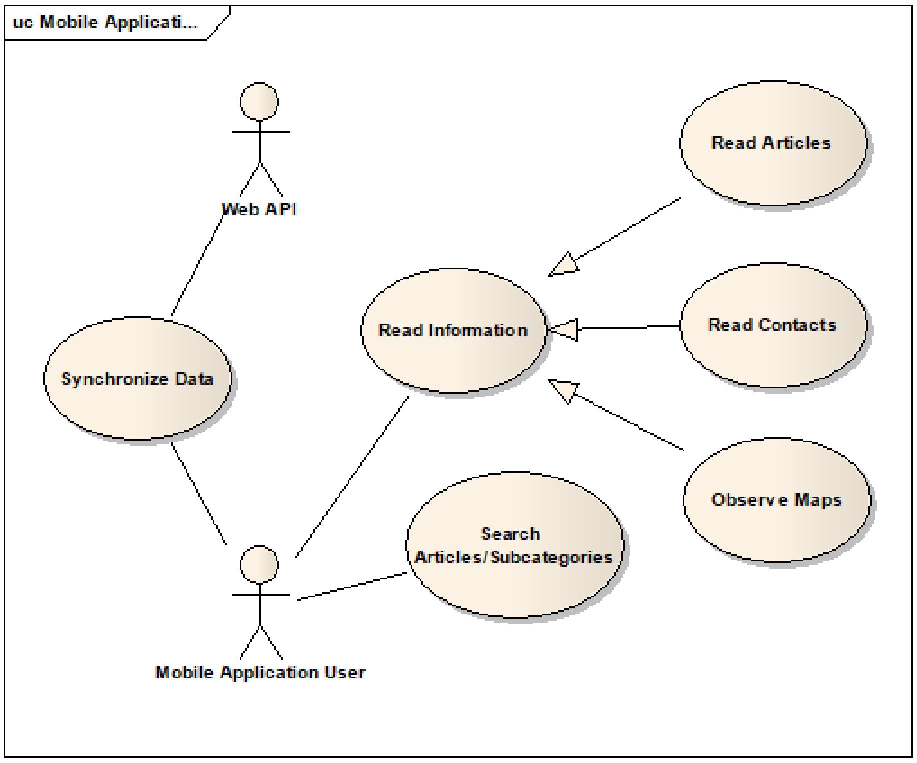


Figure 6: Mobile Application UCD

4.3.2 Activities

In the application there are two Activities:

1. SplashActivity — shows the splash screen with Student Guide logo and redirects to the MainActivity.

2. MainActivity — holds the toolbar (with search), the fragment container and the bottom navigation. The synchronization functionality also happens in the MainActivity because the swipe to refresh feature should work no matter which fragment is currently displayed on screen (except for map fragments as explained: OverViewSwipeLayout.

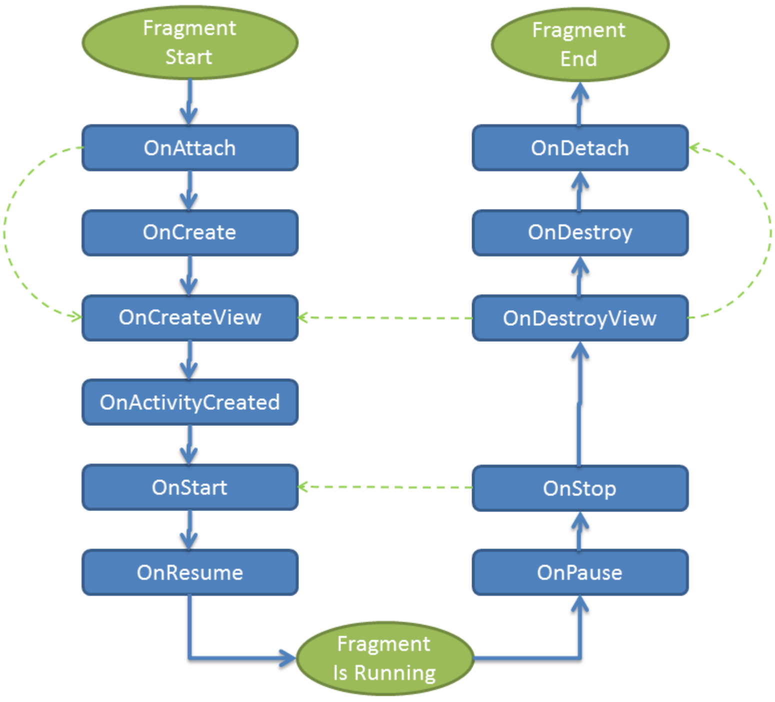
4.3.3 Fragments

The main building stones of the application are fragments. Every visible screen is a fragment living inside the MainActivity.

Fragments have their own lifecycle that is somewhat independent of, but still affected by, the lifecycle of the hosting Activity. For example, when an Activity pauses, all of its associated Fragments are paused. [12]

Navigation UI component takes care of fragment transactions and the actions between fragments are defined in the *mobile_navigation.xml*.

In Source code 7, there is an example of how each fragment is defined.



Source: https://docs.microsoft.com/en-us/xamarin/android/platform/fragments/creating-a-fragment

Figure 7: Fragment lifecycle methods

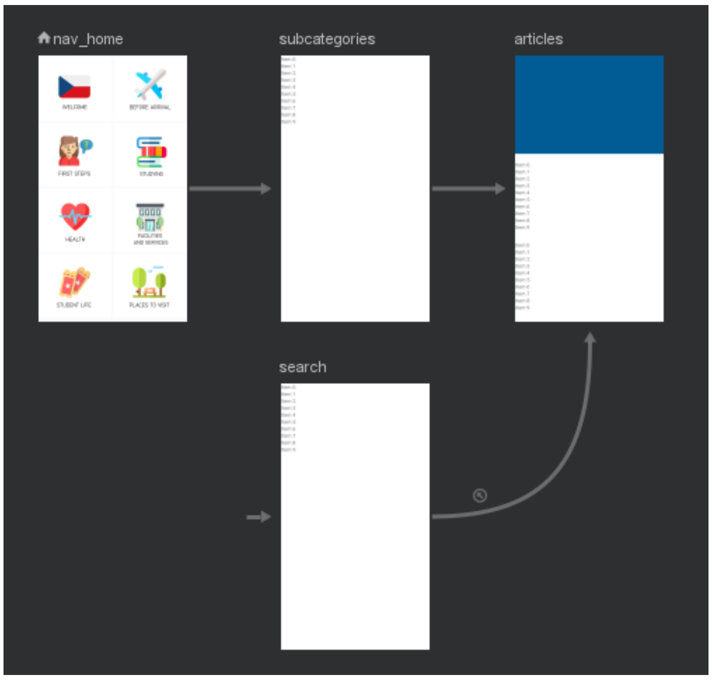


Figure 8: Navigation graph of fragments

```xml
<fragment
    android:id="@+id/subcategories"
    android:label="{title}"
    android:name="ui.subcategories.SubcategoriesFragment"
    tools:layout="@layout/fragment_subcategories">
    <action
       android:id="@+id/navToArticles"
       app:destination="@id/articles" />
    <argument
       android:name="catId"
       app:argType="integer"
       android:defaultValue="0"/>
    <argument
       android:name="title"
       app:argType="string"
       android:defaultValue="default"/>
 </fragment>
```

Source code 7: Navigation Graph XML

In the action attribute, it is specified which fragment should the action navigate to. Arguments can be specified with the argument attribute.

Fragments used in the application are the following:

Home

- HomeFragment — the first fragment loaded and also the fragment that remains as the base fragment when all other fragments are popped from the back stack. It contains tiles with categories that link to corresponding subcategories.

Subcategories

- SubcategoryFragment — displays a list of subcategories with keywords (article titles) in a RecyclerView (using SubcategoriesAdapter) based on chosen category.

Search

- SearchFragment — contains a RecyclerView (using SearchAdapter) and displaying a list of all subcategories with keywords (article titles). It uses SubcategoryViewModel to obtain data for the UI.

Articles

- ArticleFragment — displays articles in RecyclerView (using ArticleAdapter) based on previously chosen subcategory. At the top of the fragment an image is loaded with Glide[7] based on which category the article belongs to.

  The links are a part of another RecyclerView (using ArticleLinksAdapter). In order to create links to the title of each article, the articles have to be already drawn on the screen (so they have coordinates to scroll to). The ArticleFragment implements ArticlesBindingListener which the ArticleAdapter uses to call back when the articles are finished rendering so the ArticleFragment can start binding the article links.

Contacts

- ContactFragment — contains a ViewPager with ContactsPagerAdapter.
- EmergencyTelFragment — contains a RecyclerView (using ContactsAdapter). Contacts with ClassId = 1 (Emergency contacts) are displayed in the list.
- UniTelFragment — contains a RecyclerView (using ContactsAdapter). Contacts with ClassId = 2 (University contacts) are displayed in the list.
- ContactDetailFragment — displays contact details.

Map

Map fragments use the Google Maps API. The Maps API key can be found in the *AndroidManifest.xml*.

- FacultyMapFragment — displays a map with markers specific for the chosen faculty.
- CategoryMapFragment — displays a map with three categories (Menzas 'n dorms, Food 'n drink, Todo tips). Markers change depending on the selected category.

Apart from Google Maps interfaces, both fragments implement PermissionService which ensures correct behavior when using the MapProvider class.

Faculties

- FacultiesFragment — contains tiles with links to each faculty fragment.

Faculty

- FacultyFragment — contains a ViewPager with FacultyPagerAdapter.
- InfoFragment — displays faculty's study department information and articles in a RecyclerView.

[7]Fast and efficient image loading library for loading images, either from the web or from the mobile storage.

Credits

- CreditsFragment — displays icons credits and author.

More

- MoreFragment — displays links to other university applications and CreditsFragment.

4.3.4 ViewModels

ViewModel classes contain methods to query entities as well as insert, update and delete operations. Queries return LiveData objects which are then observed in the corresponding fragments to obtain data for the UI.

- HomeViewModel
- SubcategoryViewModel
- ArticleViewModel
- ContactViewModel
- FacultiesViewModel
- FacArticleViewModel

4.3.5 Adapters

Adapter acts as the mediator between UI components and data sources. Its main function is to fill data in the UI component. For example, the RecyclerView adapter holds the list of items to be displayed in the UI list and takes care of binding each item to its corresponding view (views are held in the ViewHolder class) in the list.

4.3.5.1 RecyclerView Adapters

- ArticleAdapter — binds articles to views in the RecyclerView in ArticleFragment.
- ArticleLinksAdapter — binds keywords with article links to views in the RecyclerView in ArticleFragment.
- ContactsAdapter — binds contacts to views in the RecyclerView in EmergencyTelFragment and UniTelFragment.
- SearchAdapter — binds all subcategories to views in the RecyclerView in SearchFragment.
- SubcategoriesAdapter — binds subcategories to views in the RecyclerView in SubcategoriesFragment.

4.3.5.2 ViewPager Adapters

ViewPager adapters provide pages (fragments) for the ViewPager widget.

- ContactsPagerAdapter — provides EmergencyTelFragment and UniTelFragment for the ViewPager in ContactsFragment.

- FacultyPagerAdapter — provides InfoFragment and FacultyMapFragment for the ViewPager in FacultyFragment.

4.3.6 Database Service

Database service consists of Room database POJOS (models), Dao interfaces defining database operations and repositories for abstraction purposes as explained in Solution Specification.

AppDatabase class builds the Room database and returns its instance. Database migrations can be defined and applied there.

Table 1: Database layers

Model	Dao	Repository
Article	ArticleDao	ArticleRepository
Category	CategoryDao	CategoryRepository
Contact	ContactDao	ContactRepository
FacArticle	FacArticleDao	FacArticleRepository
Faculty	FacultyDao	FacultyRepository
Subcategory	SubcategoryDao	SubcategoryRepisotory

There is also a SubcategoryAndArticle model with one-to-many relationship defined with @Relation — used for fetching subcategories and its articles.

4.3.7 Interfaces

- ArticlesBindingListener — used when an adapter has finished binding data to views.

- MapService — class implementing this interface provides google service, GPS and permission checking functionality to a map.

- PermissionService — fragment implementing this interface ensures location permission can be checked.

- SeedDatabaseListener — used for communication between MainActivity and other classes which may need to invoke database (re)seeding. Contains an onSeedingFinished callback to notify the main thread that seeding has been completed.

4.3.8 Synchronization Classes

- SyncDatabaseService — retrofit API calls interface.
- APIProvider — singleton containing Retrofit instance with the server URL.
- DatabaseSeeder — class used to seed the database asynchronously. It gets a new API key and seeds the database with new data. If the downloading process does not finish properly (user closes the application or internet becomes unavailable), the mobile application deletes its current API key from the server (to prevent data corruption), obtains another one and repeats the seeding process.
- Change — POJO representing an update from the server database to the mobile application.
- ChangeSubmitter — executes changes over database.
- SyncProvider — class used to update mobile application's database with changes from the server. First, it checks if the saved API key is saved on the server (the server deletes API keys which have not been used for year), then it fetches available changes and executes them over the database.

4.3.9 Helpers and Other Classes

Map related:

- MapMarkers — class that holds data for map markers.
- MapProvider — class providing correct map functionality - CheckAll function checks whether GoogleService is available, whether permission to location is granted, whether GPS is turned on (so the user's location can be displayed on the map.) It implements the MapService interface.
- MapMarkersDrawProvider — class containing methods for drawing markers on a map.
- TitledLatLang — marker specification data class.

Global:

- GlideModule — required for Glide usage.
- GlideImageGetter — HTML image getter for getting images from the URL provided. Used when displaying article images.
- ModulePreferences — persistent key-value storage used for storing the API key and other flags.
- HtmlStripper — helper class for removing html tags from a string.

- KeywordsMaker — helper class for creating keywords out of articles.
- OverviewSwipeRefreshLayout — class extending the SwifeRefreshLayout and overriding its onInterceptTouchEvent function. It resolves touch interception problems with maps and other scroll layouts.
- TopScroller — helper class for adding scroll top functionality to a button in a scroll layout.

4.4 Web Application

4.4.1 Use Case Diagram

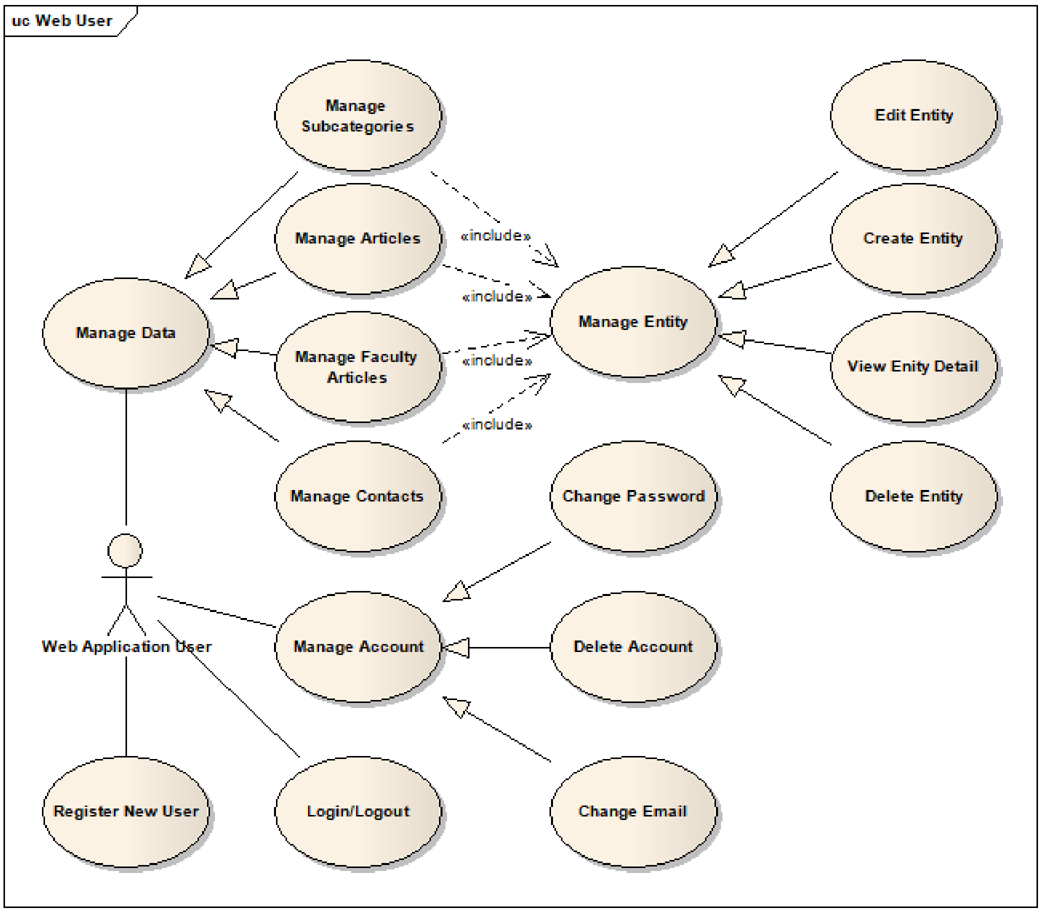


Figure 9: Web application user UCD

4.4.2 Razor Pages and Controllers

The building blocks of the ASP.NET Core framework are Razor Pages and their corresponding controllers. Pages contain the HTML code as well as Javascript and use the Razor Syntax. Controllers deal with the backend and provide correct data for the UI as well as process form data coming from users.

APIDoc

- ChangesAPI — provides documentation about ChangesAPI. Only Admin access.
  URL: /ChangesAPI
- DataAPI — provides documentation about DataAPI. Only Admin access.
  URL: /DataAPI

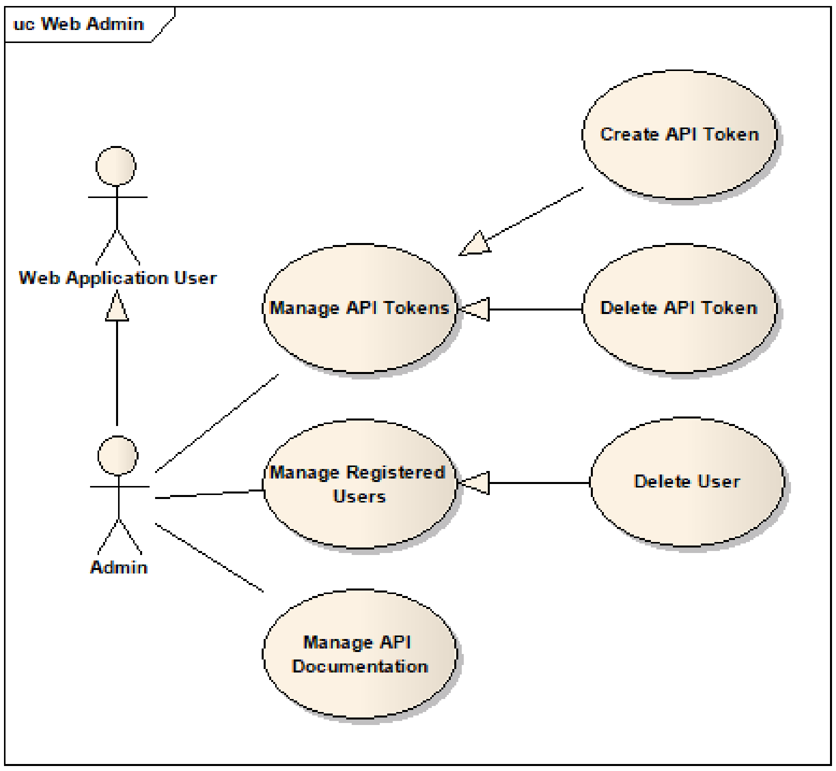


Figure 10: Web application admin UCD

Articles, Subcategories, Faculty Articles, Contacts

Each of these synchronizable entities has an Index page and Create, Edit, Detail, Delete pages. Index page displays a list of all entities which the user has access to. There is also search and paging functionality.

All CEDD pages' URL is in the form of: /entity/operation?id=id_number for example /Articles/Delete?id=10.

Create, Edit and Delete pages all use the ChangesTrackerService to track changes made to the entities so the mobile application users can stay in sync with the changes made on the server. All the Index pages include a checkbox-ajax script — when user changes the visibility of the entity, an Ajax call is made to post back and track the visibility change.

Articles and Faculty Articles both contain _PartialEditor for editing articles.

Faculty Articles

Faculty Articles are user restricted, meaning a user registered under the faculty of science can only read, edit, delete and create faculty articles for the faculty under which they are registered. Admin has access to all faculty articles.

ContactsPath

- Index — contains a link to university contacts and a link to emergency contacts. It is used as a crossroads for the user to pick which category of contacts they want to see.
  URL: /ContactsPath

Users

- Index — displays a list of all registered users. Only available to Admin.
  URL: /Users
- Delete — gives Admin the ability to delete any user from the system.
  URL: /Users/Delete?id=user_id

Tokens

- Index — displays a list of all registered tokens. Only available to Admin.
  URL: /Tokens
- Delete — gives Admin the ability to delete a registered token from the system.
  URL: /Users/Delete?id=token_id
- Create — gives Admin the ability to register a new token.
  URL: /Users/Create

Images

- Index — retrieves an image by its name. Does not contain any view.
- Add — contains image upload/delete image functionality used in editor. Does not contain any view.

Editor

- _PartialEditor — partial view representing the article editor. It consists of the input area and action buttons (bold, italics, list etc.). It depends on the Javascript file editor.js which provides all editor functionality. The editor uses a Trix Editor library which does not use contenteditable and execCommand APIs (deprecated).

4.4.3 API Controllers

ASP.NET Core API controllers provide endpoints to API users' data requests. They all inherit the ControllerBase class and define which data is going to be sent (in case of HTTP GET) and how it is going to be presented. All API endpoints are described in the API documentation section of this document.

- ArticleController — defines endpoints for articles data.

- CategoriesController — defines endpoints for categories data.
- ContactsController — defines endpoints for contacts data.
- FacArticlesController — defines endpoints for faculty articles data.
- FacultiesController — defines endpoints for faculties data.
- SubcategoriesController — defines endpoints for subcategories data.
- ChangesController — defines endpoints for synchronization functionality.

4.4.4 Models

Model classes correspond with database entities described in the web server database chapter of this document.

4.4.5 Services

All services are registered in the Startup.cs and use an interface abstraction. Services can then be injected through a constructor and used accordingly.

- HtmlStripper — removes html tags from text.
- KeyGenerator — generates a random key of chosen length.
- EmailService
  - EmailConfiguration — DTO representing the applications' email configuration.
  - Message — DTO representing the message sent to the use.r
  - EmailSender — class providing email sending functionality using MimeKit and MailKit Nuget packages.
- UnusedImagesRemoverService — background service which deletes unused images from the server once a week. Unused images which are not confirmed for more than a week will be deleted by a background service. This situation can happen when user uploads images through the editor but leaves the page unsaved.
- InactiveUsersCleanerService — background service which deletes inactive APIUser instances as well as their changes from the database after a period of fixed time (356 days). Because there is no way to find out whether a user deleted the mobile application from his mobile phone, the server will check inactive users who have not opened the mobile application for more than a year. Assuming most students come to Olomouc for one or two semesters, after a year of inactivity it is most likely the user has deleted the application and therefore his data can be deleted from the server. If

the user opens the application after more than a year of time and his data are no longer on the server, he will receive a new API key and content will be redownloaded.

- ImageManagerService — manages unused (unconfirmed) articles. Each uploaded image is registered under the article it belongs to. When the user saves changes of the article, the images in the article itself are compared with the images from the database. Differences are resolved and used images confirmed.
- ChangeTrackerService — registers changes for API users.

4.4.6 Entity Framework

Entity Framework uses the *ApplicationDbContext* class which inherits from the IdentityDbContext. All database models are registered here.

4.4.7 Identity Framework

Identity framework generates all account related pages. Individual pages can be scaffolded and changed to the programmer's likings. Identity files can be found in the Areas folder.

4.4.8 Javascript

Javascript files provide interactive frontend functionality.

- editor — provides editor functionality. It uses the Trix Editor library developed by Basecamp.
- unsaved-changes — warns the user before leaving page with editing content.
- checkbox-ajax — contains an Ajax call which is executed when user changes visibility of an entity by checking the checkbox in the Index page.

5 User Documentation

The following two sections serve as a user manual. It is divided into two parts — user documentation of the mobile application and a user documentation of the web application. The web application is described to a bigger extent seeing as the mobile application is fairly simple to use.

5.1 Mobile Application

The Student Guide mobile application can be installed on devices running on Android 4.4 and higher. Google services need to be installed and work correctly in order for the Student Guide application to work properly.

5.1.1 First Run

Users can download the mobile application from the Play Store. When the application is on its first run, the internet is needed in order to download data from the web server. If there is no internet available or the user interrupts the downloading process, it will not let the user proceed further until data is downloaded correctly.

5.1.2 Navigation

The application contains a bottom navigation for better support of gestures. The bottom navigation consists of five options: Contacts, Faculties, Home, Map and More. Each option will take the user to the corresponding screen.

Contacts screen displays emergency and university contacts.

Faculties give the user the option to look at specific information about his faculty. Each faculty shows contact information and faculty news. When the user swipes right, there is a map containing markers of faculty buildings. Markers are clickable and show more information when clicked.

Home screen shows the main categories which lead to articles.

Map screen displays a map with three marker options above: Menzas ’n dorms, Food ’n drink, To do tips. More information can be obtained by clicking on the marker as well.

More screen contains links to other university related applications such as MobilKredit or UPlikace.

5.1.3 Search

Search functionality is located in the toolbar at the top of the screen as shown in Figure 9. It enables the user to search subcategories and articles. It only scans titles, not content. After clicking on the search icon, a list of all subcategories and all article titles will appear. Once the user starts typing, the closest matches will be showed. Once the user submits the word they want to search for by pressing confirm button on the keyboard, only results with full matches will be showed.

5.1.4 Permissions

If the user uses Android 6+, when opening maps for the first time, the user will be prompted with granting location permission. This is used for displaying the user's current location on the map. The choice remains with the user as deciding not to grant the permission will not make the map unusable — it will simply not display the user's location.

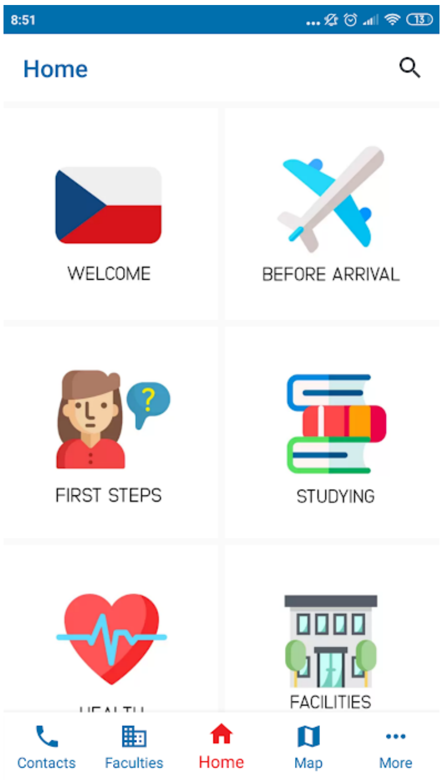
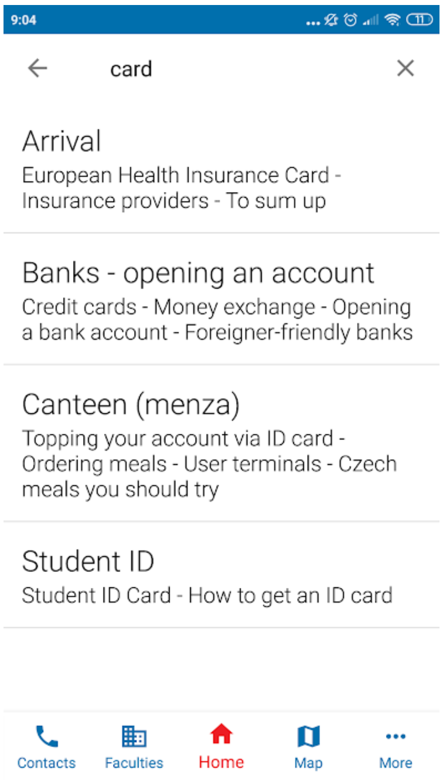


Figure 11: Mobile application home screen



Arrival
European Health Insurance Card - Insurance providers - To sum up

Banks - opening an account
Credit cards - Money exchange - Opening a bank account - Foreigner-friendly banks

Canteen (menza)
Topping your account via ID card - Ordering meals - User terminals - Czech meals you should try

Student ID
Student ID Card - How to get an ID card

Figure 12: Mobile application search

5.1.5 Updating Data

New data updates are downloaded every time user opens the application (assuming the user has internet access). If the user wants to invoke an update, they can do so by swiping down — a refresh icon will be shown indicating that the update has started. There will be a pop up message at the bottom of the screen, informing the user about the success/failure of data update.

5.2 Web Application

The Student Guide web application was mainly developed for and is recommended to be used in Chrome web browser, although other web browsers should work fine as well. Javascript is required for correct application behaviour so it is necessary that it is enabled in the browser.

5.2.1 Login & Registration

The user needs to be registered in the system before access is permitted. The registration can be realized either by Admin or by another user.

Every user is registered under their faculty and therefore has certain restrictions to the data management.

5.2.2 Logout

To log out, the user needs to click the logout icon in the top right corner of the screen. After that, the user will be redirected to the login page.

5.2.3 Forgotten Password

On the login page, there is an option to restore forgotten password. The user will be redirected to a page where they will fill in their email. The email has to be the email under which they are registred. The user will receive an email with a link to a page dedicated to password change.

5.2.4 Managing Account

Once logged in, the user can see their username in the top right corner of the screen. By clicking on the username, the user can access their user profile and proceed to various management options. This can also be accessed directly from the homepage.

The account management options include changing user email, changing the password and deleting the account. The username cannot be changed once registered.

5.2.5 Managing Data

On the homepage, there are four main links for accessing the data — Contacts, Subcategories, Articles, Faculty Articles. After clicking on each of those links, the user will be presented with the available content. As each user is registered only under one faculty, alterations to other faculties' Faculty Articles are forbidden. When navigating to Contacts, the user will be given a choice to manage either Emergency Contacts or University Contacts.

Each content page includes the option of searching through the data based on their title. In Subcategories and Articles, there is the option to filter content by the category (or subcategory) they belong to.

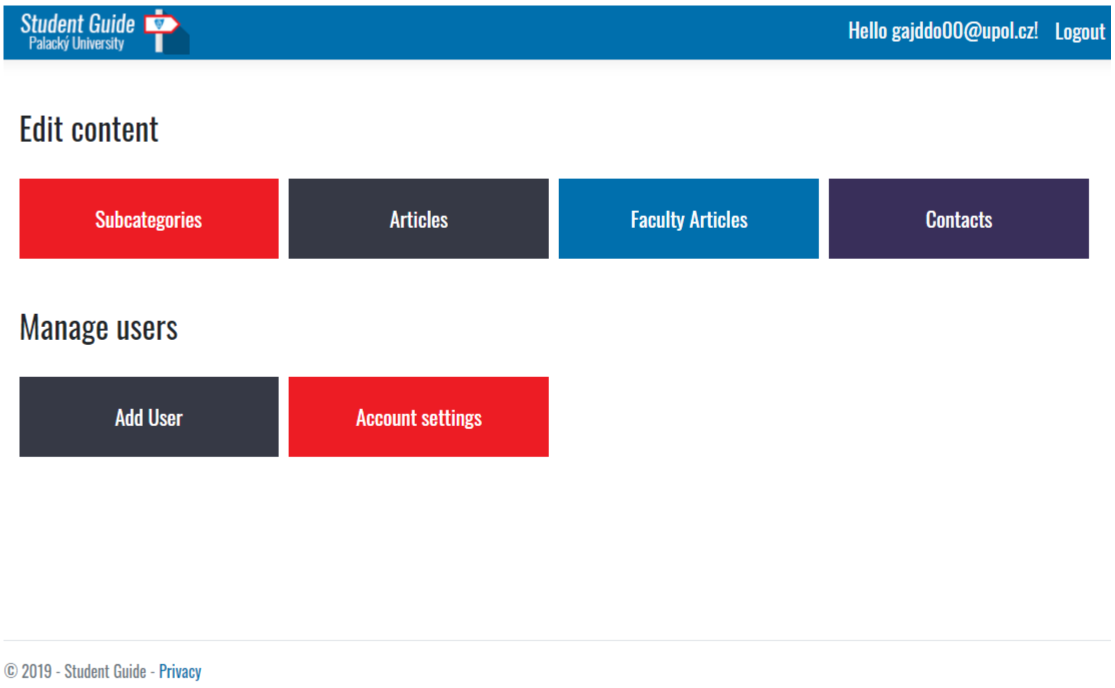
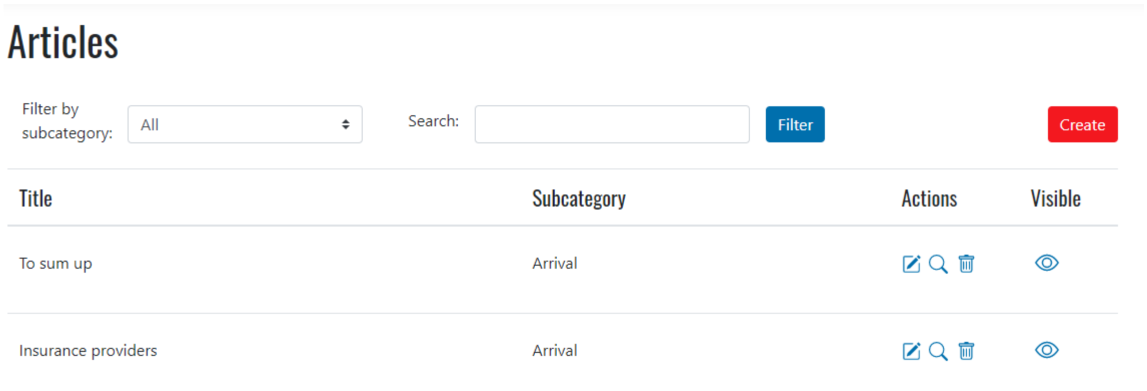


Figure 13: Homepage

Apart from searching, there are four other actions for working with the presented content — Create, Edit, Detail, Delete. Each content has a visibility option which states whether the particular content will be visible in the mobile application.



Title	Subcategory	Actions	Visible
To sum up	Arrival		
Insurance providers	Arrival		

Figure 14: Content page

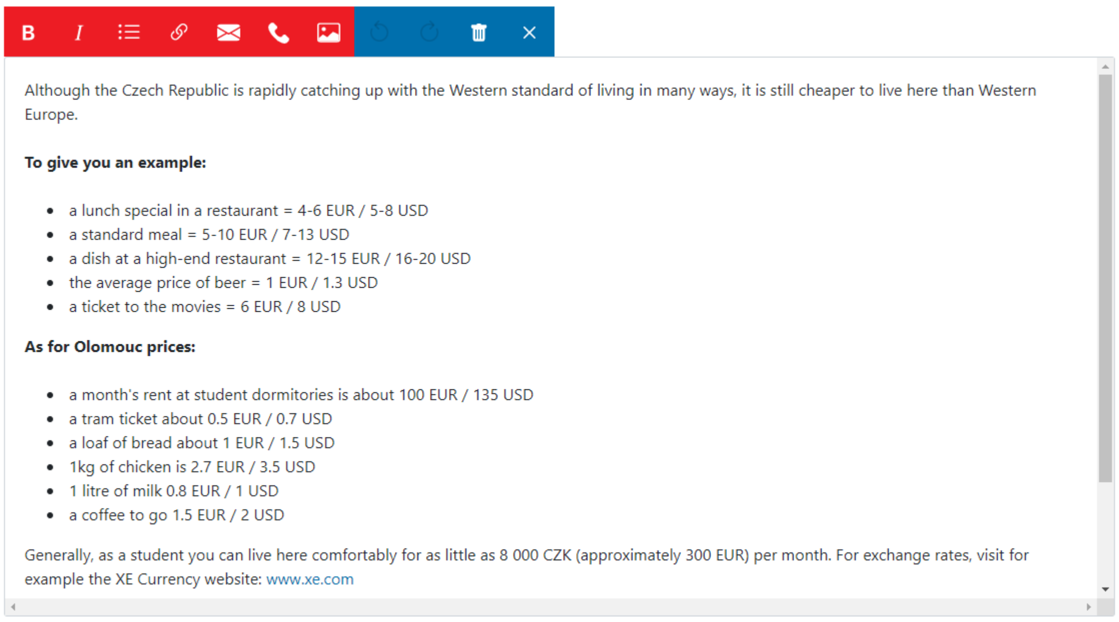
Managing data is very straightforward. Creating and editing is simple — just filling out individual fields. When user enters improper data by passing in values that do not match the required format, they will be informed and asked to fill proper values.

The user will also be notified when they are trying to edit/delete content which has already been edited/deleted, meaning they are not editing the most recent version because someone else had done changes while they were editing.

When leaving page without saving, the user will be prompted to confirm they are sure to leave the page without saving changes.

5.2.6 Article Editor

When working with articles, there is an editor dedicated to creating formatted content.



Although the Czech Republic is rapidly catching up with the Western standard of living in many ways, it is still cheaper to live here than Western Europe.

To give you an example:

- a lunch special in a restaurant = 4-6 EUR / 5-8 USD
- a standard meal = 5-10 EUR / 7-13 USD
- a dish at a high-end restaurant = 12-15 EUR / 16-20 USD
- the average price of beer = 1 EUR / 1.3 USD
- a ticket to the movies = 6 EUR / 8 USD

As for Olomouc prices:

- a month's rent at student dormitories is about 100 EUR / 135 USD
- a tram ticket about 0.5 EUR / 0.7 USD
- a loaf of bread about 1 EUR / 1.5 USD
- 1kg of chicken is 2.7 EUR / 3.5 USD
- 1 litre of milk 0.8 EUR / 1 USD
- a coffee to go 1.5 EUR / 2 USD

Generally, as a student you can live here comfortably for as little as 8 000 CZK (approximately 300 EUR) per month. For exchange rates, visit for example the XE Currency website: www.xe.com

Figure 15: Editor

All actions in the editor require for the editor field to be focused, meaning the cursor should be inside.

Features described from left to right:

- Bold — will make the selection bold or will turn on/off bold mode.
- Italic — will make the selection italic or will turn on/off italic mode.
- List — will turn on/off list mode.
- Link — will create a link out of a selection. The user will be prompted with a field to fill in the website URL.
- Email — will create an email out of a selection.
- Phone Number — will create a phone number out of a selection.
- Image — will insert an image into the editor. Supported image formats are JPG, JPEG, PNG and GIF. Maximum image size is 10 MB. There are three ways of adding an image:

1. Drag&Drop — image will be inserted by dragging the image straight from system folder and dropping it into the editor.
2. URL — image will be inserted after entering a URL and pressing INSERT.
3. Upload — image will be added by uploading the image from a folder and pressing UPLOAD Images cannot be used with undo and redo. Redo operation will not upload the image again.

- Undo — will undo changes.
- Redo — will redo changes.
- Delete All — will delete all editor content.
- Remove Formatting — will remove all formatting from a selection.

5.2.7 Admin

Admin has unlimited access to all content and its modifications. Apart from being able to see and edit all content, there are a few things that only Admin can do:

- Manage Users — Admin can see all registered users and delete them if necessary.
- Manage API Tokens — register new users who will request access to website content, e.g. new application will be created and will require Student Guide data.
- Manage API Documentation — add/edit information about API functionality.

Conclusions

Cílem této práce bylo vytvořit Android aplikaci určenou pro zahraniční studenty Univerzity Palackého a také vyvinout online webový editor, který by umožnil zaměstnancům zahraničních oddělení vytvářet a aktualizovat obsah pro mobilní aplikaci. První verze aplikace byla vytvořena v zimním semestru 2019 a byla dostupná na Google Play do doby (včetně) obhajoby této práce. Jednalo se o statickou verzi, jejíž brzké nahrání bylo vyžadováno paní Gronychovou. Nová verze aplikace bude k dispozici po obhajobě této práce.

Při práci na tomto projektu jsem nasbírala mnoho cenných zkušeností, jelikož jsem neměla předtím žádnou zkušenost s programováním větších projektů, programováním pro Android a ani s programováním webových aplikací. Tento projekt mi rozšířil mou paletu praktických dovedností v oboru a jsem velmi vděčná za to, že jsem si mohla vyzkoušet různé technologie a návrh projektu od píky.

Dana Gronychová byla s vývojem a spoluprací velmi spokojená a sama si mobilní i webovou aplikaci vyzkoušela. K projektu se vyjádřila následovně:

> „O této aplikaci jsem se bavila s kolegy, zmínila ji na ZO RUP a pan proděkan o ní rovněž mluvil na zasedání zahraničních proděkanů. O aplikaci jsme informovali naše budoucí zahraniční studenty a zahraničním studentům, kteří budou přijati ke studiu, bude link ke stažení této aplikace zaslán."

Conclusions

The aim of this work was to create an Android mobile application intended for foreign students of Palacky University as well as an online web editor which would allow the employees of the foreign study departments create and update content for the mobile application. The first version of the mobile application was created in winter semester 2019 and was available on the Google Play Store up until (including) the defence of this thesis. It was a static version of the final mobile application which was required to be published to Google Play by the start of the mentioned semester by Dana Gronychová. The new version will be available after this thesis will have been defended.

This work has given me plenty of experience, which is quite valuable for me seeing as I had no previous programming experience when it comes to bigger projects nor did I have any experience working with Android development or web application development. This project has broadened my practical skills palette, and I am very thankful I got to use various technologies and try to design a project from scratch.

Dana Gronychová was very pleased with our cooperation, and she has tested both the mobile application and the web application herself. Here's what she said about the project:

> "I discussed this application with my colleagues, presented this [mobile] application to the ZO RUP and the faculty's vice dean consulted the application with the rest of the university's vice deans of foreign affairs at a meeting. We have informed our future university foreign students about the existence of this application and the ones who will be accepted to the university will receive the link for the download in an email."

A Contents of attached CD/DVD

`bin/`

- **`web-editor`**
  Complete folder structure of the Student Guide Web Application (in ZIP archive) for copying to a web server. This folder also contains all necessary runtime libraries and other files needed for trouble free operation on web server.

- **`android`**
  Apk file for installation of the Student Guide Mobile Application on a suitable Android device.

`doc/`

Thesis text in PDF format, created with the use of obligatory style KI PřF UP in Olomouc intended for use in the creation of the final thesis, including all its appendices and all necessary files needed for trouble free generation of the PDF document (in ZIP archive), i.e. source text, inserted images etc.

`src/`

- **`web-editor`**
  Complete source codes of the Student Guide Web Application with all necessary (eventually taken over) source codes, libraries and other files needed for trouble free creation of the folder structure intended to be copied on a web server.

- **`android`**
  Complete source codes of the Student Guide Mobile Application with all necessary (eventually borrowed) source codes, libraries and other files needed for trouble free creation of the final Android apk file intended for application installation.

`readme.txt`

Instructions for the web application deployment to a web server, including all requirements for its trouble-free operation and the web address hosting the web application intended for the purpose of testing when making a thesis report and for the purpose of the defence of the thesis.

CD/DVD also contains:

`data/`

Sample and testing data used in the thesis and for the purpose of testing when making a thesis report and for the purpose of the defence of the thesis.

`install/`

Installation files of applications, runtime libraries and other files needed for the Student Guide Web Application to run, which are not a standard part of the operating system devoted to operation of the web application.

All materials attached on CD/DVD that have been borrowed are either not subject to copyright or their further distribution is allowed by the author. All (cited) materials, for which this statement is not true, therefore they are not included on the CD/DVD, have their source attached in the bibliography, the text itself or in the file `readme.txt`

References

[1] Google. *Meeting Android Studio*. 2020. Online; accessed 2020-03-05, last updated 2020-02-14. Available also from WWW: ⟨https://developer.android.com/studio/intro/⟩.

[2] Google. *Defining data using Room entities*. 2019. Online; accessed 2020-03-05, last update 2019-12-27. Available also from WWW: ⟨https://developer.android.com/training/data-storage/room/defining-data⟩.

[3] Google. *Foreign Key*. 2019. Online; accessed 2020-03-05, last update 2019-12-27. Available also from WWW: ⟨https://developer.android.com/reference/android/arch/persistence/room/ForeignKey⟩.

[4] Google. *Define relationships between objects*. 2020. Online; accessed 2020-03-05, last update 2020-01-13. Available also from WWW: ⟨https://developer.android.com/training/data-storage/room/relationships⟩.

[5] Google. *Accessing data using Room DAOs*. 2019. Online; accessed 2020-03-05, last update 2019-12-27. Available also from WWW: ⟨https://developer.android.com/training/data-storage/room/accessing-data⟩.

[6] Google. *LiveData Overview*. 2020. Online; accessed 2020-03-05, last update 2020-01-22. Available also from WWW: ⟨https://developer.android.com/topic/libraries/architecture/livedata⟩.

[7] Google. *ViewModel Overview*. 2020. Online; accessed 2020-03-05, last update 2020-01-22. Available also from WWW: ⟨https://developer.android.com/topic/libraries/architecture/viewmodel⟩.

[8] Inc., Square. *Retrofit: A type-safe HTTP client for Android and Java*. Online; accessed 2020-03-05. Available also from WWW: ⟨https://square.github.io/retrofit/1⟩.

[9] Google. *Shrink, obfuscate, and optimize your app*. 2020. Online; accessed 2020-03-05, last update 2020-02-14. Available also from WWW: ⟨https://developer.android.com/studio/build/shrink-code⟩.

[10] Microsoft. *Introduction to the C# language and the .NET Framework*. 2015. Online; accessed 2020-03-05, last update 2015-07-20. Available also from WWW: ⟨https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework⟩.

[11] Microsoft. *Introduction to ASP.NET Core*. 2019. Online; accessed 2020-03-05, last update 2019-11-12. Available also from WWW: ⟨https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2⟩.

[12] Microsoft. *Creating A Fragment*. 2018. Online; accessed 2020-03-05, last update 2018-02-07. Available also from WWW: ⟨https://docs.microsoft.com/en-us/xamarin/android/platform/fragments/creating-a-fragment⟩.

[13] Allen, Grant. *Android 4: Průvodce programováním mobilních aplikací.* First. Brno: Computer Press, 2013. dclvi, 656 pp. ISBN 978-80-251-3782-6.

[14] Lacko, Ľuboslav. *Vývoj aplikací pro Android.* First. Brno: Computer Press, 2015. cdlxxii, 472 pp. ISBN 9788025143476.