Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

Bachelor Thesis

Designing e-commerce website with Django framework

Milena Krapivenko

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Milena Krapivenko

Informatics

Thesis title

Designing e-commerce website with Django framework

Objectives of thesis

To Develop a functional e-commerce website on the framework Django, for illustrating its functionality, and evaluate the framework by software quality standards.

The partial objectives:
– To make a literature review on the topic of web-development
– To understand the main principles of UML diagrams
– To understand software quality standards

Methodology

The methodology for this thesis involves conducting an experiment to evaluate the effectiveness of specific features and functionalities of the Django framework. The theoretical part of the research will involve studying and analyzing relevant literature. The practical part will consist of a step-by-step procedure for designing an e-commerce website using Django. Different diagrams will be used to aid in various aspects of the project, and both front-end and back-end technologies will be utilized.

The proposed extent of the thesis

40

Keywords

Django framework, E-commerce website, Admin panel, Web development, Software standards, UML diagrams, Software quality, Object-Relational Mapping (ORM),

Recommended information sources

JAMES, McGAW. Beginning Django E-Commerce (Expert's Voice in Web Development) 1st ed. Edition, 2009. ISBN-10 1430225351.

LUTZ, M. *Programming Python.* Boston: O´Reilly Media, 2011. ISBN 978-0-596-15810-1.

MATTHES, E. *Python crash course : a hands-on, project-based introduction to programming.* San Francisco: No Starch Press, 2019. ISBN 978-1-59327-928-8.

Nigel George, Mastering Django, 2020. ISBN – 10 0648884414.

Expected date of thesis defence

2021/22 SS – FEM

The Bachelor Thesis Supervisor

Ing. Jakub Konopásek, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 13. 3. 2023

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 14. 3. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 14. 03. 2023

Declaration

I declare that I have worked on my bachelor thesis titled " Designing e-commerce website with Django framework" 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 thesis does not break any copyrights.

In Prague on 15th of March   _____

Acknowledgement

I would like to thank Jakub Konopásek and Čajka Martin and my mommy for supporting, for their advice and support during my work on this thesis.

Designing e-commerce website with Django framework

Abstract

This thesis examines the benefits and process of website development using Django. The research includes a literature review on website development and software quality standards, as well as an experimental study on the effectiveness of Django's features and functionalities. The thesis details each stage of the development process, including designing the website's architecture and developing the website. Additionally, the research describes the techniques and tools used throughout the project and evaluates the software quality of the resulting website. The end result is a fully functional website that demonstrates the effectiveness of using Django for website development.

Keywords: Django framework, E-commerce website, Admin panel, Web development, Software standards, UML diagrams, Software quality, Object-Relational Mapping (ORM),

Návrh e-commerce webových stránek s použitím frameworku Django.

Abstrakt

Bakalářská práce je věnována procesu tvorby webových stránek pomocí Django a přínosům této metody. Provedený výzkum zahrnoval práci s odbornou literaturou zabývající se problematikou vývoje internetových stránek a také problémy standardů kvality software. Mimo jiné je v práci zohledněn experimentální výzkum efektivity funkcí frameworku Django. V bakalářské práci je detailně rozebrána každá jednotlivá fáze procesu vývoje včetně sestavení architektury webové stránky a její samotná tvorba. Navíc, v práci jsou popsány techniky a nástroje využité pro účely projektu, a výsledná webová stránka je autorem vyhodnocena z hlediska kvality software. Plná funkčnost webové stránky vzniklé v rámci projektu tak jasně demonstruje efektivitu využití Django ve vývoji webových stránek.

Klíčová slova: Django framework, E-commerce webová stránka, administrativní panel, vývoj webových stránek, standardy software, UML diagramy, kvalita software, Mapování objektově relačních dat (ORM)

Table of content

1 Introduction

Nowadays, e-commerce websites play important role in our lives. Online sales have increased, and it is important to have for any size of businesses online website to reach a wider audience. The development of e-commerce websites requires advanced technology and software frameworks to create a user-friendly, responsive, and secure platform. One of the frameworks that reached a good popularity among developers for developing e-commerce websites is Django.

Django is an open-source Python-based web framework that simplifies the development of complex web applications. It is designed to create scalable and maintainable websites and has a wide range of built-in features that make website development faster and more efficient. In this thesis, we will consider the process of designing an e-commerce website step-by-step using Django and discuss the benefits of using this framework.

The purpose of this thesis is to understand the process of website development using Django and evaluate its effectiveness by using software quality standards. The thesis will cover the main stages of website development, including designing the website's architecture, and developing the website.

Overall, this thesis will contribute to the knowledge and understanding of website development using Django and provide insights into the benefits and challenges of using this framework.

2 Objectives and Methodology

2.1 Objectives

To evaluate the effectiveness of the Django framework in developing a functional e-commerce website and assess its compliance with software quality standards. Specific partial objectives include:

- Conducting a comprehensive literature review on web development and relevant framework.
- Utilizing UML diagrams to design the website's architecture and improve its functionality.
- Applying the core principles of website design to ensure a user-friendly and efficient interface.
- Developing dynamic pages that align with industry standards and customer preferences.

2.2 Methodology

The methodology for this thesis involves conducting an experiment to evaluate the effectiveness of specific features and functionalities of the Django framework. The theoretical part of the research involved studying and analysing relevant literature. First, we considered web development and defined the types of websites, then we briefly considered dynamic and static websites and understood their main aspects. As any website follows some principles, from the research and according to the different authors we defined 4 main aspects. The research also defined Front-end, Back-end, and database concepts, and analysed relevant literature on the Django framework, examining its rationale, architecture, built-in functions, and differences between MVC and MTV architectures. In addition, the research explored additional website design tools such as UML diagrams. The practical part consists of a step-by-step procedure for designing an e-commerce website using Django. At the beginning, we designed 3 main diagrams that helped us to understand better our classes and interaction of interested parties. For the front end, we used bootstrap for getting the visible part of the website and defined its hierarchy. According to the UML diagram that has been designed, we created the models for our apps, and then for each app we created the business logic. As according to theoretical part, for avoiding repeating the code, we created some additional HTML files and then included them to other HTML files. After filling the database, we simulated the behaviour of the customer to understand if the functions work

properly. After, all mentioned procedures, we evaluated framework by software quality standard.

3 Literature Review

3.1 Web development

3.1.1 Web site types

A website is a collection of web pages or web content and multimedia content which is typically identified by common Domain Name. A website is usually dedicated to a single topic and all of its WebPages will convey various information related to the dedicated topic. A computer called web server (including web server And Database server software) is dedicated to host the website. A website can be accessed 24/7 through internet by referencing the respective URL which identifies the website. A website Is more or like a book, as by the content of the book we can navigate to the respective pages in the book. Likewise, each website definitely has Home Page which is also a webpage and it contains various hyperlinks. Through these hyperlinks, we can navigate to the respective webpage. All publically accessible websites make up the World wide Web and private websites are accessible only via Intranet. The website come in many ranges and developed for various functionalities. They are:

- Personal Website

- Government website

- E-commerce Website

- Social Networking Website

- News and E-learning Website

- E-governance Website

On the other hand, web pages are the building blocks of a website and it is written by the dedicated markup language HTML. A protocol named "http" is entirely dedicated to interpret and transport the webpage from web server to the web browser. (Karthik, 2018)

3.1.2 Static and Dynamic

Nowadays, any websites can be static or dynamic and it is depending on the requirements from the users. That's why it is important to understand the difference between mentioned types to avoid unnecessary functionalities that will not be used.

Static

A static website consists of a fixed number of pre-built files stored on a web server. These files are written in HTML, CSS, and JavaScript, which are called "client-side" languages because they run in the user's web browser. When a user requests a page from a server using a URL, the server returns the HTML file specified in the URL and any accompanying CSS and/or JavaScript files.
During this exchange, the web server does not modify the files before they are sent to the user, so the web page will look the same to everyone who requests it. The content is

"static" - the only way to change the look and feel of a website is to manually change the content of the files

Most of the benefits of static websites come from their simplicity. Static sites are the simplest kind of site that you can create and maintain from scratch. If you want to launch a basic website quickly and cheaply, static sites are a good option. With knowledge of HTML and CSS, you can write decent code without much effort or expense.

Static websites also tend to be faster than dynamic websites on the user side. This is because the pages on static websites are already built and require minimal internal processing. The server only needs to receive the requested files and deliver them to the client. Static websites are also easier to cache due to the lack of content variety. Site speed, also referred to as site performance, is critical to a positive user experience and also affects search engine ranks. (Jamie Juvile, 2022)

Dynamic

Unlike a static website that displays the same content to all visitors in the same format, a dynamic website presents different information to different visitors. The content that a visitor sees may be determined by several factors such as their location, local time, settings and preferences and/or actions they take on the website (such as shopping habits), making it more personalized and interactive.

Dynamic websites make it much easier to update the entire site. Administrators can quickly and easily make sweeping changes to their site without having to update the source code for every HTML file. On websites that frequently update content and appearance to keep up with their industries, this is a must.

Finally, dynamic websites are more scalable than static websites because the server does not store a fixed number of pages. Instead, the server creates the page when needed. (Jamie Juvile, 2022)

3.1.3   Core website design principles

It is important to understand main principles during the designing any website, otherwise, the website will be not "User-friendly", or even not visible in search results what can be a huge problem for business. Although there are many principles for designing the right website, there are basically four main ones:

a.   Usability

User interface design involves the examination of who will be using the product and how it will be used to support the performance of tasks (Hix & Hartson 1993). Activities, such as profiling the user classes that are performed to design user interfaces can also be performed during web page design. While web pages may seemsimple to develop, a stronger examination of their audience, category, and content lends itself to understanding their usability goals. So key activities in the design of a web site include: identifying target audiences and classifying them into user classes, identifying what type of web site this will be, and determining the content of the site and any constraints or boundaries to it. Profiling these user classes, selecting a category, and understanding the nature of the content are fundamental activities which every web page designer needs to consider. How to perform these tasks is less obvious. (Calongne, 2004)

b. Accessibility

Each website should be user-friendly, if in Usability the main idea is content, then in Accessibility the main idea is to use content and layout it correctly so that it is easy to use. Highlight the main recommendations:

- Provide appropriate alternative text

- Ensure links make sense out of context

- Caption and/or provide transcripts for media

- Do not rely on color alone to convey meaning

- Make sure content is clearly written and easy to read

c. Semantic web

The Semantic Web is a vision about an extension of the existing World Wide Web, which provides software programs with machine-interpretable metadata of the published information and data. In other words, we add further data descriptors to otherwise existing content and data on the Web. As a result, computers are able to make meaningful interpretations similar to the way humans process information to achieve their goals. (Jeen Broekstra, 2023)

d. SEO

Search engine optimization is a process that make website visible in search results of search engine. A higher ranked site gets more visitors in search and Internet is a biggest marketing channel. More than 70000 million user's searches information on search engine every month. A top rank of site provides a great opportunity to reach customers. Specific keywords and back link building can be used for search engine optimization. There are two type of optimization individual webpages (on-Site) and the whole websites(off-site) optimization. Keyword Analysis: keyword must be analyzed for websites when optimization process will perform. Some important rules for selection of keywords analysis are: - The specific keywords list must be used for analysis of keywords. - Use those keywords which takes webpages into first 5 links of search results. - Keyword in links must be found on target. Meta tag and content of page is very important to search individual webpages. There are many keyword analysis tools and site that can help to examine the density of keywords. Density of keywords should be in 4-7 to word in 50 words and these words used in making individual webpages. Link Forming: After selection of keywords concise and famous links should be used for link building as a URL. (Tariq Mahmood, 2019)

3.1.4 Frontend

The concept of frontend means the development of an interface visible to the user and all the functions with which he can interact. In fact, when you go to any site, you see buttons, text, various animations and other components there - all this is implemented using the

frontend. Three different languages are used to create these elements - HTML, CSS and JavaScript.

- **HTML**

  The main tool in this area is the HTML hypertext markup language. It is needed mainly for marking up a document, that is, a page in a browser. With it, the developer creates a structure, adds headings, lists, and performs other content formatting.

- **CSS**

  CSS (Cascading Style Sheets) language is responsible for the appearance of the page. With it, you work with colors, fonts and the arrangement of various blocks. In simple terms, CSS is used to beautifully design a page and customize its appearance after the main structure has been written using HTML.

- **JavaScript**

  With the help of JavaScript, various actions are performed on the page, that is, animation and response to user requests are added. For example, the page reacts to cursor movement and mouse clicks, changing the behavior of elements in accordance with user actions. Thanks to JS, data is sent and received from the server without the need to reload the page, which means that some tasks are implemented more simply, for example, when it comes to sending and receiving messages.

Frontend is an area in which the developer creates interface elements visible to the user and all the functions for interacting with sites and applications. All this is closely related to the backend, which will be discussed next paragraph. (Morris Fletcher , 2017)

3.1.5 Backend

Backend, also known as server-side, refers to the part of a web application or software system that is responsible for handling tasks that are not directly visible to the user. This includes tasks such as data storage, data processing, server-side logic, and server configuration. In the context of web development, the backend typically consists of a server, an application or web server (such as Apache or Nginx), and a database. Backend developers are responsible for designing and building the server-side of the application, which involves writing code in server-side programming languages such as PHP, Python, Ruby, Java, or Node.js. The backend also includes APIs (Application Programming Interfaces) that allow different parts of the application to communicate with each other, and with third-party services such as payment gateways or social media platforms.

3.1.6 Database

A database is a collection of organized data that can be easily accessed, managed, and updated. It is used to store and organize information in a way that makes it easy to search, retrieve, and update when necessary.

Databases are used in a wide range of applications, from small personal databases to large enterprise systems that store and manage vast amounts of data. They can be used to store a variety of data types, including text, numbers, images, and multimedia files.

There are many types of databases, including relational databases, NoSQL databases, object-oriented databases, and hierarchical databases. Relational databases are the most common type of database and are based on the relational model, which organizes data into tables that are related to each other through common fields.

Database management systems (DBMS) are used to manage and manipulate data in a database. Some popular DBMS include MySQL, Oracle, Microsoft SQL Server, and MongoDB.

The design and implementation of a database is an important part of any software project, and requires careful planning and consideration of the requirements of the application and the data it will be storing.

3.2 Django

3.2.1 Introduction

Django is a high-level Python web framework that allows you to quickly create secure and maintainable websites. Built by experienced developers, Django takes most of the hassle out of web development so you can focus on writing your web application without having to reinvent the wheel. It's free and open source, has a growing and active community, great documentation, and plenty of both free and paid support options. (Big Nige, 2016)

3.2.2 Reason to use Django

Django is one of many web frameworks available; however, over the last decade, Django has distinguished itself as a leading framework for developing scalable, secure and maintainable web applications. Django has its rough edges, but its pragmatic approach to getting stuff done is where it really stands out from the crowd. (Big Nige, 2016)

Python

The framework itself is built on the basis of the Python programming language, which is gaining popularity every day, as it is quite easy to learn, has many useful various additional libraries that help and facilitate the work with the project, and so on. The language also supports asynchrony, which helps to make dynamic sites, which were discussed in the previous sections.

Batteries included

Django is completely free, and it's also open source; This means that if you have a suggestion or change to the framework that you think many people will also find useful, you can submit your request or even make changes yourself and make a pull request! With the philosophy of having everything work together in this framework, everything runs smoothly and follows consistent design principles. This is important because you don't have to deal with any part of the configuration; It's all set up for you. Django includes access to some popular databases and gives you a secure way to manage accounts and passwords. Again, this functionality is out of the box (batteries included!) Of course, this functionality will be fully tested and means you will avoid any common security pitfalls

such as posting session information (cookie data) in a public place or forgetting the hash (scramble and mangle) passwords.

Sites framework
Allows you to host multiple sites in one application and share data between them.

Scalable
Rapid scaling is part of the framework's philosophy, so the project structure is built around applications from the very beginning. They are not attached to anything and can be used in different projects (but in practice such a rare thing happens).

Battle tested
For testing, there is pytest - convenient, fast and flexible

Reports
Django writes very detailed error reports. And although django-debug-toolbar is right next to it, which shows everything for your application in general (including database queries with timings!)

Templates
In Django, a template is a text file that defines the structure and layout of the output that is sent to the client's web browser. Templates are used to separate the presentation logic from the business logic of an application, which makes it easier to maintain and modify the code.

3.2.3 Architecture

The architecture in Django is based on the Model-View-Controlled (MVC) software architectural pattern, but it has its own extensions and changes which is called to Model-Template-View (MTV).

The main differences MVC and MTV :
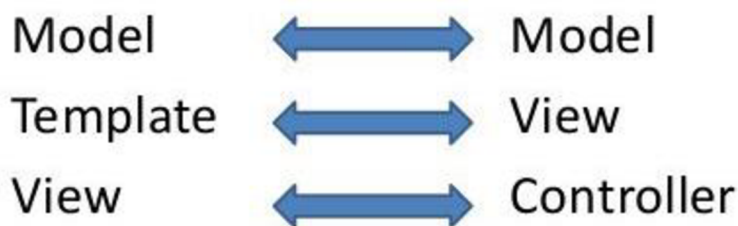
Figure 1 MTV vs MVC

18

Django's Model layer

In Django, a model is a class which is used to contain essential fields and methods. Each model class maps to a single table in the database. Django Model is a subclass of django.db.models.Model and each field of the model class represents a database field (column). Django provides us a database-abstraction API which allows us to create, retrieve, update and delete a record from the mapped table. Model is defined in Models.py file. This file can contain multiple models. (Javapoint, 2021)

Django's View layer

A view is a place where we put our business logic of the application. The view is a python function which is used to perform some business logic and return a response to the user. This response can be the HTML contents of a Web page, or a redirect, or a 404 error. All the view function are created inside the views.py file of the Django app. (Dhulam, 2019)

Django's Template layer

Django provides a convenient way to generate dynamic HTML pages by using its template system. A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

3.2.4   ORM

ORM (Object-Relation Mapping) is a general name for frameworks or libraries that allow you to automatically link a database to code. They try to hide the existence of the database as much as possible. Instead, the programmer is given the opportunity to operate with data in the database through a special interface. Instead of building SQL queries, the programmer calls simple methods, and the ORM takes care of the rest of the work. By using different methods that's are written by more skilled people it increase the quality of the each query. The ORM automatically transfers data from a database, such as PostgreeSQL or MySQL, to objects that are used in the application code. Below the scheme of ORM:
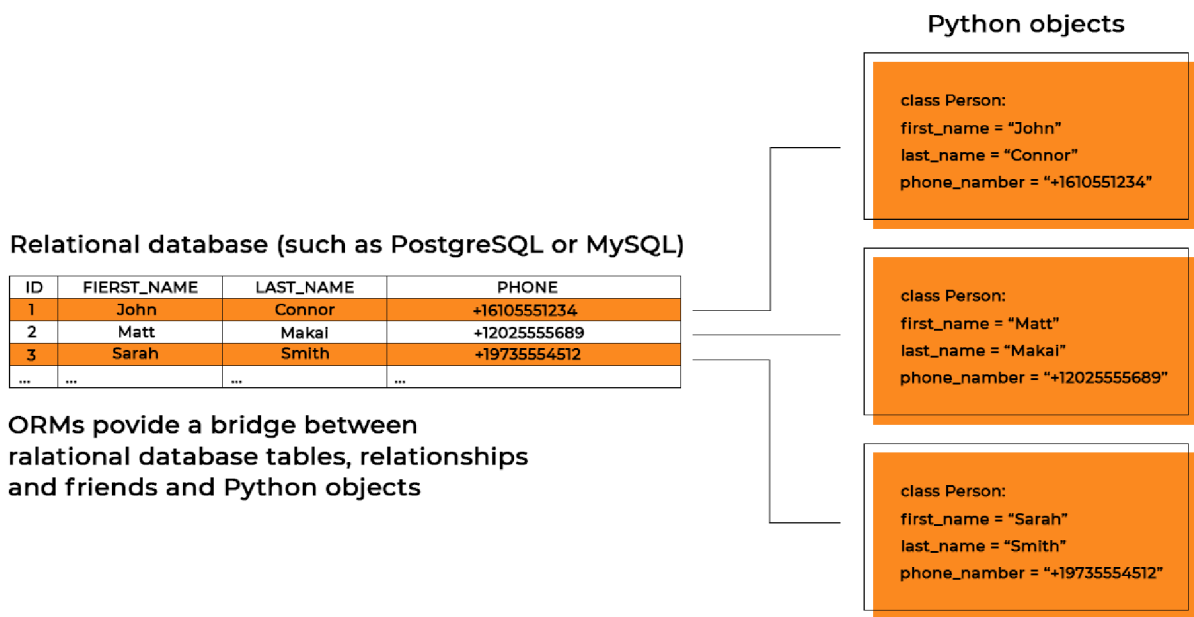
Figure 2 Compare the objects in Python and in Relational databases

Also, ORM allows you to quickly switch between databases with minimal code changes. For example, you can use SQLite on a local server and then switch to MySQL on a production server.

Model

Each object of this class corresponds to one entry in the table. The ORM is responsible for converting data into objects and vice versa. The programmer only needs to use the correct methods to retrieve objects and modify them. Each object of this class corresponds to one entry in the table. The ORM is responsible for converting data into objects and vice versa. The programmer only needs to use the correct methods to retrieve objects and modify them.

Query Builder

(query builder). It's an abstraction over SQL that simplifies query generation. It usually looks like a chain of functions, each of which is responsible for a specific part of the SQL, for example: ORDER, SELECT or WHERE.
Django ORM allows you to describe fairly complex queries, and you can solve most of the tasks using the Query Builder. However, it is also possible to make requests directly:
Another responsibility of the ORM is to change the database schema: adding, deleting, and modifying tables. This is done, as a rule, not in pure SQL, but with the help of a special language. This allows you to work with ORM without being distracted by the specifics of specific databases.
In Django ORM, models are paramount - you change the code and then use tools to bring the database schema up to date with the new state of the code. This approach is called Code First

20

Migrations

Any database changes during the life of an application. New tables are added to it, old ones are deleted, some are changed. This process never ends. Changes to the database are made using the migration mechanism.
In Django ORM, a migration is a Python module that contains a description of the actions that change the database schema. The tutorial project contains an example migration, but if you open this file in an editor, the amount of code might confuse you. Fortunately, this code is not written by hand - Django generates migrations on its own. Modifying migration modules manually is quite rare.

3.2.5 Forms

Django's form uses the same techniques, display receives a request, performs the necessary actions, including reading data from models, generating and returning an HTML page (from a template, which is passed a context containing the data that will be shown). What makes this process more difficult is that the backend needs to further process the data provided by the user and, in case of errors, redraw the page again.
The diagram below shows the process of working with a form in Django, starting with a request for the page containing the form (highlighted in green).
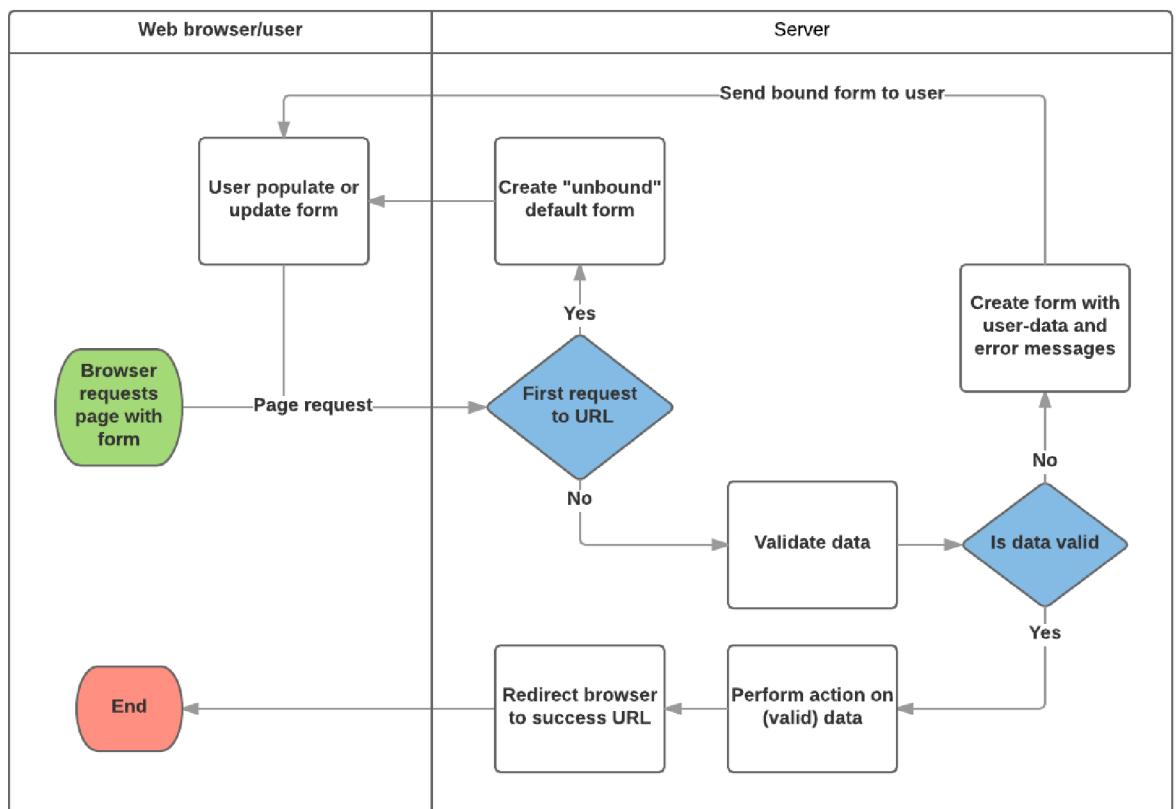

Figure 3 Forms' procedure

According to this diagram, the main points that Django forms take on are:

1. Show the form by default on the first request from the user.

21

The form can contain empty fields (for example, if you create a new record in the database), or they (fields) can have initial values (for example, if you modify a record, or want to fill it with some initial value).
The form is currently unbound because it is not associated with any user input (although it may have initial values).

2. Getting data from a form (from an HTML form) on the client side and binding it to the form (form class) on the server side.

   Associating data with a form means that the data entered by the user, as well as possible errors, when redrawing in the future, will refer specifically to this form, and not to any other.

3. Data cleaning and validation.

   Scouring data is about checking it for possible values, or insertions into input fields (that is, sanitizing is about removing bad characters that could potentially be used to send malicious content to the server), and then converting the scraped data to appropriate Python data types.
   Validation checks whether the field values (for example, the correctness of the entered dates, their range, and so on)

4. If any data is incorrect, then redraw the form, but this time with the data already entered by the user and error messages describing the problems encountered.

5. If all the data is correct, then performing the necessary actions (for example, saving data, sending emails, returning a search result, downloading a file, and so on)

6. When all actions have been successfully completed, then the user is redirected to another page.

3.2.6   User authentication and permissions

Django comes with a user authentication system. It handles user accounts, groups, permissions and user sessions based on cookies
Django's authentication system handles both authentication and authorization. In short, authentication checks if the user is who they say they are, and authorization determines what the authenticated user is allowed to do. Here, the term authentication is used to refer to both tasks. The authorization system consists of:

- Users

- Permissions: Binary (yes/no) flags indicating whether the user can perform a particular task.

- Groups: A general way to apply shortcuts and permissions to multiple users.

- Customizable password hashing system

- Forms and browsing tools for user login or content restrictions

- Pluggable backend system

The authentication system in Django tends to be very general and does not provide some of the features commonly found in web authentication systems. Solutions to some of these common problems have been implemented in third party packages:

- Password strength check

- Throttling login attempts

- Third party authentication (such as OAuth)

- Object level permissions

3.2.7   Caching

A fundamental trade-off in dynamic websites is, well, they're dynamic. Each time a user requests a page, the web server makes all sorts of calculations – from database queries to template rendering to business logic – to create the page that your site's visitor sees. This is a lot more expensive, from a processing-overhead perspective, than your standard read-a-file-off-the-filesystem server arrangement.
For most web applications, this overhead isn't a big deal. Most web applications aren't washingtonpost.com or slashdot.org; they're small- to medium-sized sites with so-so traffic. But for medium- to high-traffic sites, it's essential to cut as much overhead as possible.
That's where caching comes in.
To cache something is to save the result of an expensive calculation so that you don't have to perform the calculation next time. Here's some pseudocode explaining how this would work for a dynamically generated web page:

```
given a URL, try finding that page in the cache

if the page is in the cache:

    return the cached page

else:

    generate the page

    save the generated page in the cache (for next time)

    return the generated page
```

Django comes with a robust cache system that lets you save dynamic pages so they don't have to be calculated for each request. For convenience, Django offers different levels of cache granularity: You can cache the output of specific views, you can cache only the pieces that are difficult to produce, or you can cache your entire site.
Django also works well with "downstream" caches, such as Squid and browser-based caches. These are the types of caches that you don't directly control but to which you can

provide hints (via HTTP headers) about which parts of your site should be cached, and how. (Holovaty & Kaplan-Moss, 2010).

3.2.8   Admin panel

One of the most powerful parts of Django is the automatic admin interface. It reads metadata from your models to provide a quick, model-centric interface where trusted users can manage content on your site. The admin's recommended use is limited to an organization's internal management tool. It's not intended for building your entire front end around.
The admin has many hooks for customization, but beware of trying to use those hooks exclusively. If you need to provide a more process-centric interface that abstracts away the implementation details of database tables and fields, then it's probably time to write your own views.

3.3   Tools for designing web-development

Website design requires auxiliary tools that will facilitate the process in the practical part. That's why, in this paragraph will be described the Bootstrap framework and the Unified Modeling Language their main principles, pros and cons etc.

3.3.1   UML (tools for designing the website)

The Unified Modeling Language (UML) is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system. It captures decisions and understanding about systems that must be constructed. It is used to understand, design, browse, configure, maintain, and control information about such systems. It is intended for use with all development methods, lifecycle stages, application domains, and media. The modeling language is intended to unify past experience about modeling techniques and to incorporate current software best practices into a standard approach. UML includes semantic concepts, notation, and guidelines. It has static, dynamic, environmental, and organizational parts. It is intended to be supported by interactive visual modeling tools that have code generators and report writers. The UML specification does not define a standard process but is intended to be useful with an iterative development process. It is intended to support most existing object oriented development processes. (James Rumbaugh, 1999)

There are many different diagrams in the UML, each diagram has its own specific purpose and the choice depends on what exactly needs to be described, and so on. In this paragraph, only those diagrams that are supposed to be used in practice will be described.
      1. Structural diagram
          a. class diagram
A class is a diagram element denoting a set of objects that have the same internal structure, behavior, and relationships with objects of other classes. The class is shown on the diagram as a rectangle divided into three sections:

1. class name

2. List of class fields

3. List of class methods

Visibility level:
1. "+" - open field. Analogue of public in programming languages. Means that the field can be accessed from any part of the program.
2. "-" - closed field. Analogue of private in programming languages. Means that the field can only be accessed within the class.
3. "#" is a protected field. Analogue of protected in programming languages. Means that the field can be accessed inside the class and inside derived classes

Multiplicity

Multiplicity - an interval that determines the range of the number of elements in the array. If the multiplicity is specified for the field, then it should be considered an array. The number of elements in such an array will be determined by the specified interval.
For multiplicity indicate one or two values:
[m..n] - interval from m to n inclusive (m <= n). Such an entry will mean that the collection can store from m to n values, inclusive.
[n] is an interval that can be considered as an abbreviation for [0..n].

Relationships

1. Association relationship
An association relation is used to show that there is some relationship between classes (for example, between two classes). Usually, using it on a class diagram, they show that one class uses the functionality of another class.

2. Dependency relationship
A dependency relationship is used to show that changing one class requires changing another class.

3. Inheritance relationship
An inheritance relation is used to show that one class is the parent (base class or superclass) of another class (child, derived class).

4. Aggregation relationship
An aggregation relationship between two classes indicates that one of them includes the other class as a constituent part. In this case, the class-part can exist separately from the class-whole (later we will reveal the meaning of this phrase).
(Booch, Rumbaugh, & Jacobson, 1999)

2. behavior diagram

a. use case diagram

Use case diagrams describe relationships and dependencies between groups of use cases and actors involved in a process.
It is important to understand that use case diagrams are not meant to represent a design and cannot describe the internals of a system. Use case diagrams are intended to facilitate interaction with future users of the system, with customers, and are especially useful for

determining the necessary characteristics of the system. In other words, use case diagrams tell you what the system should do without specifying the methods themselves.

Actor

An actor is an external source (not a system element) that interacts with the system through a use case. Actors can be real person (for example, user of the system) or other computer systems or external events.

Actors do not represent physical people or systems, but their roles. This means that when a person interacts with the system in various ways (assuming different roles), he is displayed by several actors. For example, a person who works in the service desk and takes orders from customers will appear in the system as a "support member" and "sales member".

Actors can have two types of use case relationships:

- Simple association - represented by a line between the actor and the use case (no arrow). Reflects the relationship between an actor and a use case.

- A directed association is the same as a simple association, but indicates that the use case is being initialized by an actor.

    (Booch, Rumbaugh, & Jacobson, 1999)

3.4 ISO/IEC 9126 software quality standards

The ISO/IEC 9126 standard has been developed to address software quality issues. It specifies software product quality characteristics and sub-characteristics and proposes metrics for their evaluation. It is generic, and can be applied to any type of software product by being tailored to a specific purpose.

Quality model defines six characteristics which, with minimal duplication, describe the quality of the software:

-functionality (suitability; correctness; interoperability; consistency; security);

-reliability (stability; resistance to error; recoverability);

- practicality (understandability; learnability; ease of use);

-Usability (character of change in time; nature of change of resources);

-maintainability (analyzability; changeability; stability; testability);

-portability (adaptability; ease of implementation; conformity; interchangeability).

These characteristics form the basis for further refinement and software quality descriptions.

(Islam, R., & Khan, M. 2010)

4 Practical Part

4.1 UML diagrams

4.1.1 Class diagram
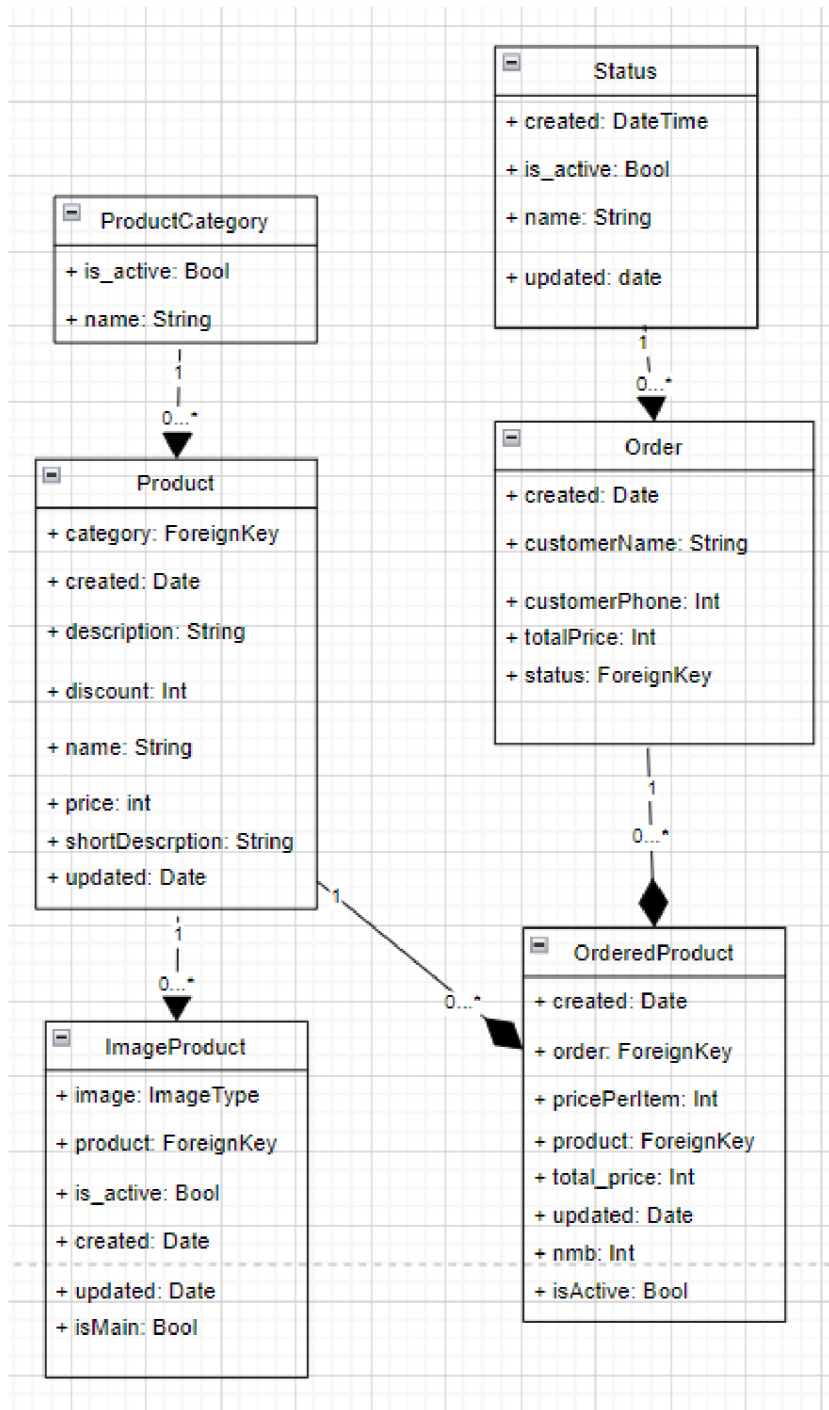
Figure 4 Class diagram

## 4.1.2   Use case Diagram
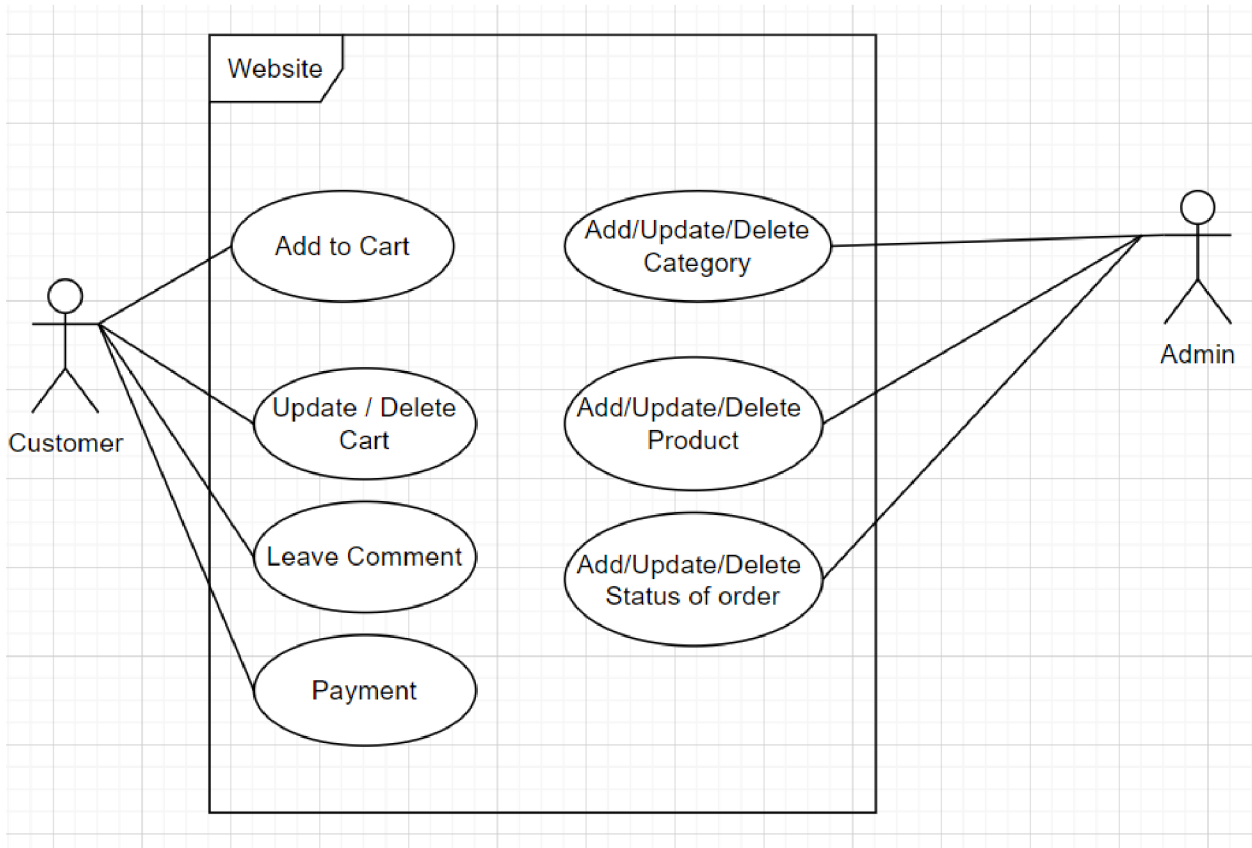
*Figure 5 Use case diagram*

4.1.3   Activity diagram
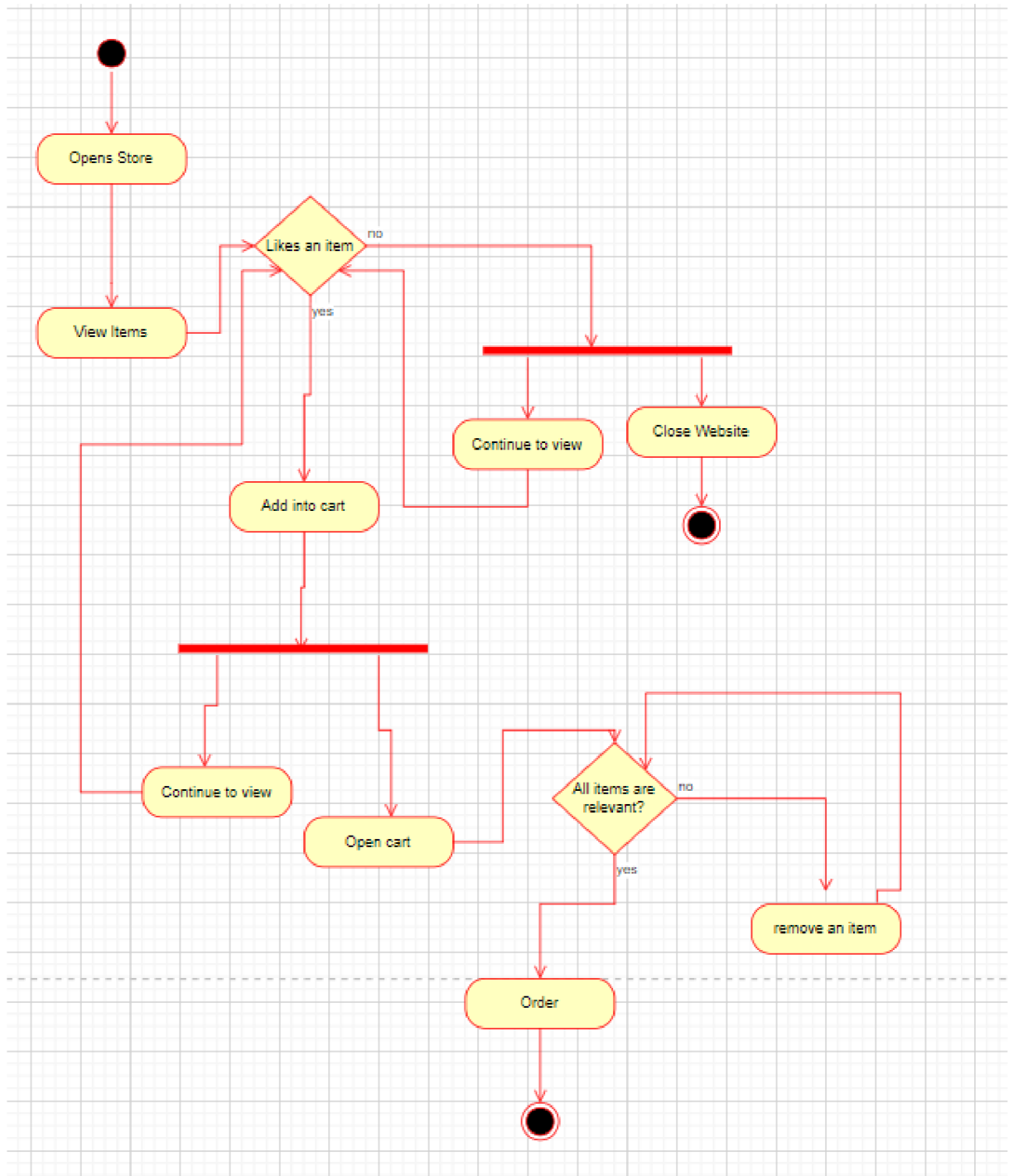
Figure 6 activity diagram

4.2 Development process

4.2.1 Frond-end

For a designing and creating visible part of the website, will be taken free template and the help of the boostrap will be used. Also, some modifications in code will be made by HTML, CSS and JavaScript as the free templates are not always well-written.

4.2.1.1 Hierarchy of the website
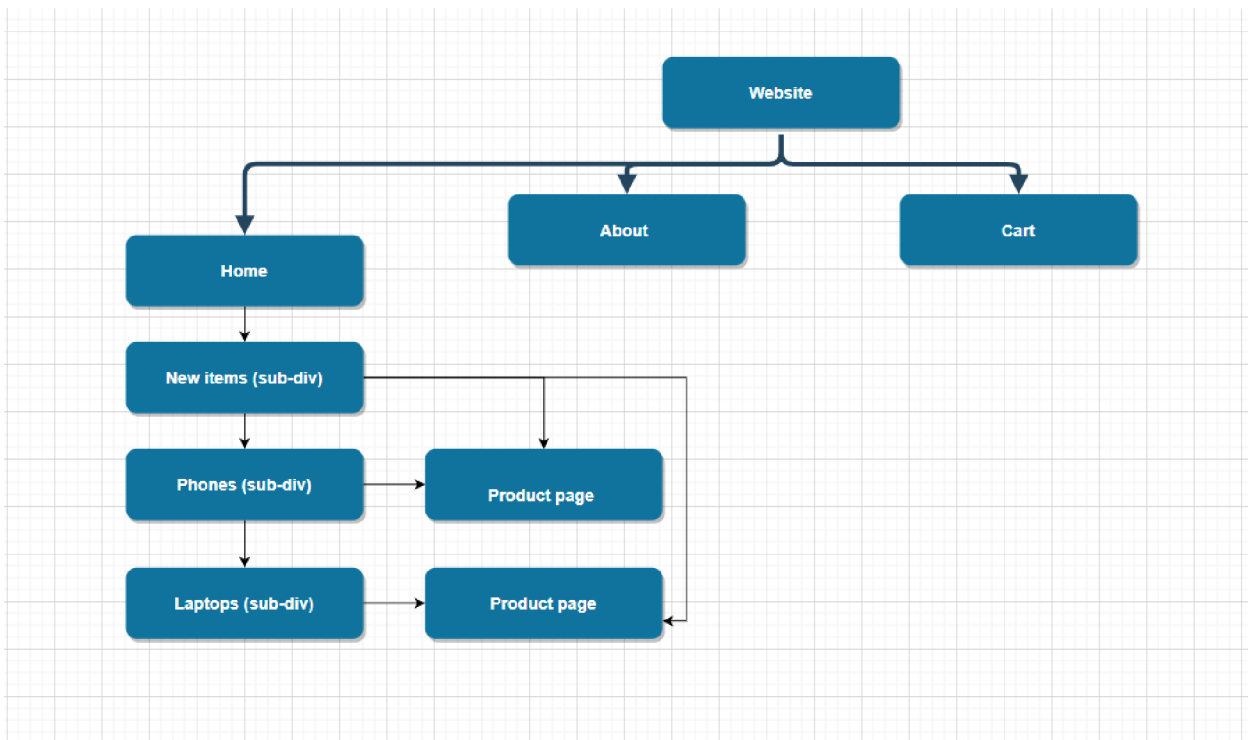
Figure 7Hierarchy of the website

The website includes three main pages and sub-pages.

- Home page

  The page where all items of the products are located, from this page a customer is able to view the items and go to the product's page for more details.

- About page

  The basic page which include objective of the website.

- Cart page

  The page includes all selected items from the cart and also there a customer is able to create the order.

4.2.2 Back-end

4.2.2.1 Creating the project

As virtual environment already installed, the project can be created by command "*django-admin startproject website*". After the execution of the command, it creates the files in structure below:
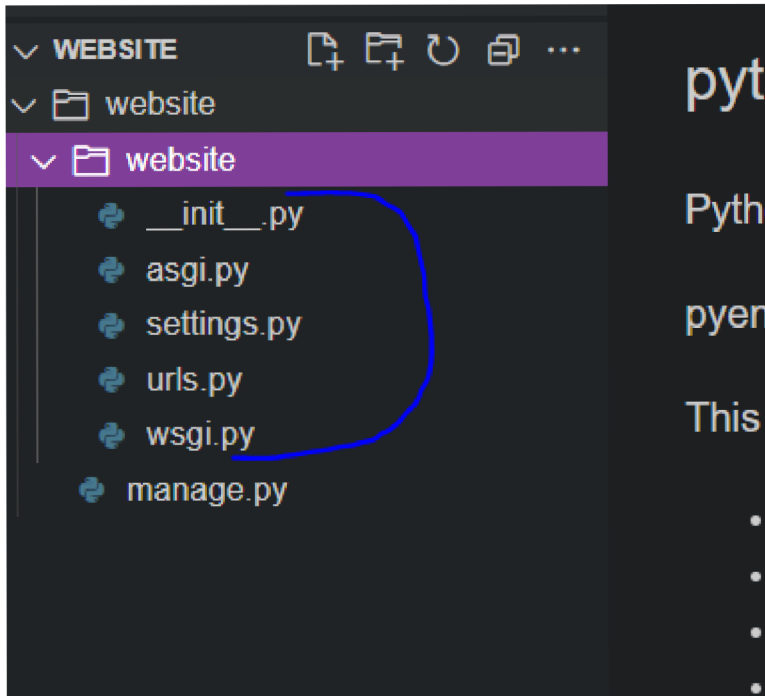

Figure 8 Structure of the project at the beginning

4.2.2.2 Creating the applications

After defining the classes in section 4.1, they can be applied in the project. By using command "$ python manage.py **products/orders** polls", Django created all neccesseary files that are needed for the apps. Below are shown the created structure:
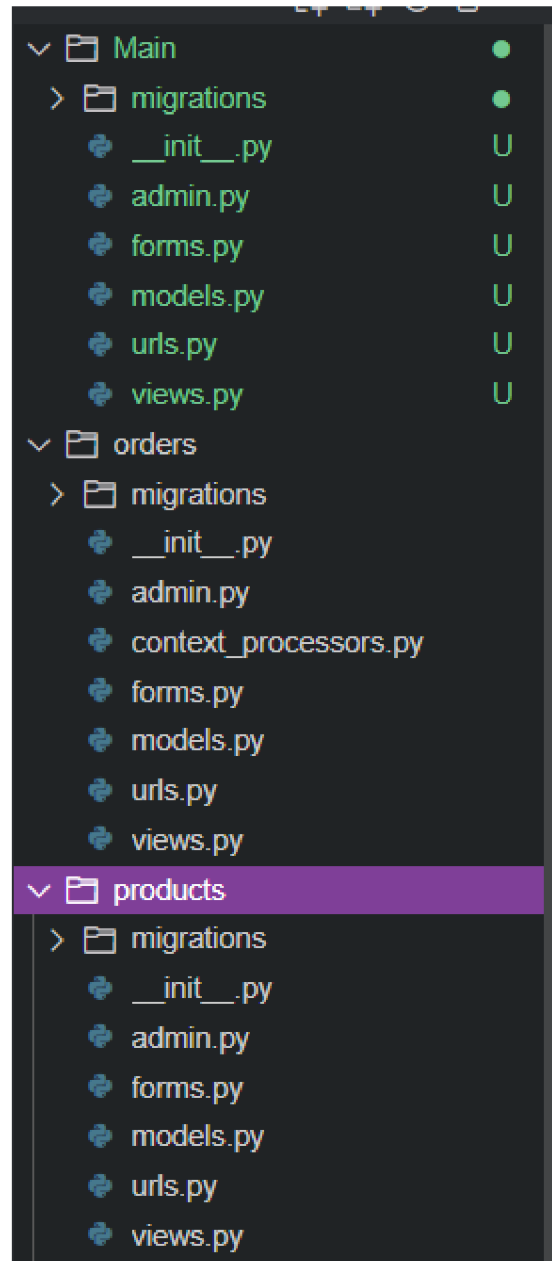
4.2.2.2.1   Products

Models

As according to the diagram from the section 4.1, we create parent class Product and two childs classes ProductImage and ProductCategory

```python
class Product(models.Model):
    name = models.CharField(max_length=64, blank=True, null=True, default=None)
    price = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    discount = models.IntegerField(default=0)
    category = models.ForeignKey(ProductCategory, blank=True, null=True, default=None, on_delete=models.CASCADE)
    short_description = models.TextField(blank=True, null=True, default=None)
    description = models.TextField(blank=True, null=True, default=None)
    is_active = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True, auto_now=False)
    updated = models.DateTimeField(auto_now_add=False, auto_now=True)
```

Figure 10 Class Product

```python
class ProductImage(models.Model):
    product = models.ForeignKey(Product, blank=True, null=True, default=None, on_delete=models.CASCADE)
    image = models.ImageField(upload_to='products_images/')
    is_main = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True, auto_now=False)
    updated = models.DateTimeField(auto_now_add=False, auto_now=True)
```

Figure 11 class ProductImage

```python
class ProductCategory(models.Model):
    name = models.CharField(max_length=64, blank=True, null=True, default=None)
    is_active = models.BooleanField(default=True)
```

Figure 12 Class ProductCategory

View

The function below helps to get the product ID and also to request session key

```python
def product(request, product_id):
    product = Product.objects.get(id=product_id)


    session_key = request.session.session_key
    if not session_key:
        request.session.cycle_key()


    print(request.session.session_key)


    return render(request, 'products/product.html', locals())
```

Figure 13 Function for getting product and session key

Forms

Not used for the app.

URL

Below code helps to map view and the corresonding URL. As according to the documentation all URL should be mapped.

```
urlpatterns = [

    url(r'^product/(?P<product_id>\w+)/$', views.product, name='product'),
]
```

Figure 14 Mapping of the view and url

4.2.2.2.2 Orders

Models

As we defined from the class diagram that we have parent class Product and two sub classes ProductInOrder and ProductInBasket the below the realization of it.

```
class ProductInOrder(models.Model):
    order = models.ForeignKey(Order, blank=True, null=True, default=None, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, blank=True, null=True, default=None, on_delete=models.CASCADE)
    nmb = models.IntegerField(default=1)
    price_per_item = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    total_price = models.DecimalField(max_digits=10, decimal_places=2, default=0)#price*nmb
    is_active = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True, auto_now=False)
    updated = models.DateTimeField(auto_now_add=False, auto_now=True)
```

Figure 15 Class Product in Order

```
class Order(models.Model):
    user = models.ForeignKey(User, blank=True, null=True, default=None, on_delete=models.CASCADE)
    total_price = models.DecimalField(max_digits=10, decimal_places=2, default=0)#total price for all produc
    customer_name = models.CharField(max_length=64, blank=True, null=True, default=None)
    customer_email = models.EmailField(blank=True, null=True, default=None)
    customer_phone = models.CharField(max_length=48, blank=True, null=True, default=None)
    customer_address = models.CharField(max_length=128, blank=True, null=True, default=None)
    comments = models.TextField(blank=True, null=True, default=None)
    status = models.ForeignKey(Status, on_delete=models.CASCADE)
    created = models.DateTimeField(auto_now_add=True, auto_now=False)
    updated = models.DateTimeField(auto_now_add=False, auto_now=True)
```

Figure 16 Class Order

```
class Status(models.Model):
    name = models.CharField(max_length=24, blank=True, null=True, default=None)
    is_active = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True, auto_now=False)
    updated = models.DateTimeField(auto_now_add=False, auto_now=True)
```

Figure 17 Class Status

```
class Orderedproduct(models.Model):
    order = models.ForeignKey(Order, blank=True, null=True, default=None, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, blank=True, null=True, default=None, on_delete=models.CASCADE)
    nmb = models.IntegerField(default=1)
    price_per_item = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    total_price = models.DecimalField(max_digits=10, decimal_places=2, default=0)#price*nmb
    is_active = models.BooleanField(default=True)
    created = models.DateTimeField(auto_now_add=True, auto_now=False)
    updated = models.DateTimeField(auto_now_add=False, auto_now=True)
```

Figure 18 Class OrderedProduct

URLS

The code below helps to map the view and URLs

```
urlpatterns = [

    url(r'^basket_adding/$', views.basket_adding, name='basket_adding'),
    url(r'^checkout/$', views.checkout, name='checkout'),

]
```

Figure 19 Mapping view and URL

Forms

As for Checkout we need some information from a customer, we created a form to get the name and phone, below the code for it.

```
class CheckoutContactForm(forms.Form):
    name = forms.CharField(required=True)
    phone = forms.CharField(required=True)
```

Figure 20 Form for checkout

View

For creating the business logic for the app two main functions are created. Basket_adding is used to put the selected an item into the basket and checkout for creating the order.

```
def basket_adding(request):
    return_dict = dict()
    session_key = request.session.session_key
    print_(request.POST)
    data = request.POST
    product_id = data.get("product_id")
    nmb = data.get("nmb")
    is_delete = data.get("is_delete")

    if is_delete == 'true':
        ProductInBasket.objects.filter(id=product_id).update(is_active=False)
    else:
```

Figure 21 function for adding item into the basek

```
def checkout(request):
    session_key = request.session.session_key
    products_in_basket = ProductInBasket.objects.filter(session_key=session_key, is_active=True, order__isnull=True)
    print_(products_in_basket)
    for item in products_in_basket:
        print(item.order)


    form = CheckoutContactForm(request.POST or None)
    if request.POST:
        print(request.POST)
        if form.is_valid():
            print("yes")
```

Figure 22 function checkout

4.2.2.2.3 Registration of the apps

As according to the documentation, all created apps should be registered in the file *settings.py,* by adding the names into ISTALLED_APPS.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',


    'main',
    'products',
    'orders',
]
```

Figure 23 Registration of the apps

4.2.3 Templates

To store all created HTML files a folder "templates" is created. It also includes sub-folders for HTML files to have it in good order.

As Django provides a function that helps to avoid the repeating of the writing code. it was decided to create separately HTML files for footer, navbar and head.
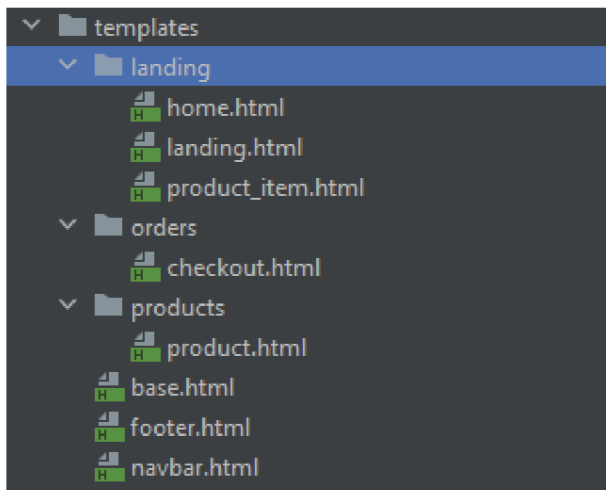
Figure 24 Structure of the folder Templates

After creating additional mentioned HTML files, we can simple include or extend them to other pages by command {$ include footer.html $} etc.

4.2.4 Filling the Database

After all procedures from the previous sections, we can fill the database by the products. Django provides a perfect admin panel where can do it without the writing SQL code, what is just need is to creare super user by command it manage.py createsuperuser. After creation we need to open admin panel and simply fill the database.

Figure 25 Admin Panel

- Products

  Example of the creation:

Figure 26 Creating the object

38

After filling all neccesseary information (attributes) for the object and saving it, we are able to see all products in the database.

Figure 27 Overview of the database

- Orders

The objects will be created automatically after clicking the button „order", only what was created is the status of the orders "New", "Processed", "Cancelled", "Done". As according to diagram in the section 4.2, an admin needs to update the status of the order.

4.2.5   Static folder

After creation the HTML files and Filling the databeses, we need to add styles (CSS) for HTML files, and store somewhere media Files from database, as according to the documentaiton, all these files will be stored in static folder.
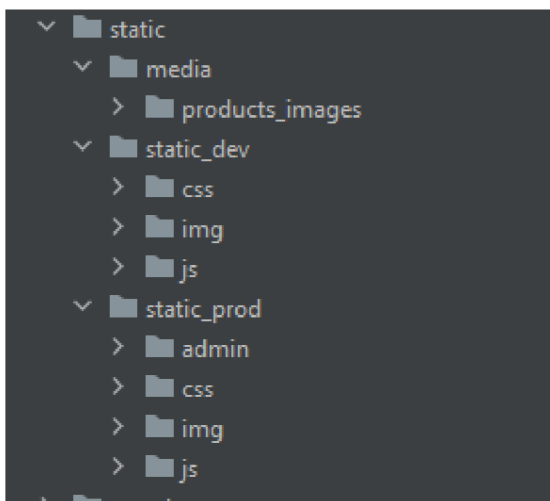
Figure 28 Structure of Static Folder

- Sub folder Media
  Includes all media files from Database for products

- Sub folder static_dev and static_prod
  Include CSS files, media files for HTML files and JavaScript file.

Static folder should be set up, that's why the file settings.py is modified as according below:

```
STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "static", "static_dev"),
)



STATIC_ROOT = os.path.join(BASE_DIR, "static", "static_prod")

MEDIA_URL = '/media/'

MEDIA_ROOT = os.path.join(BASE_DIR, "static", "media")
```

Figure 29 changes in settings.py

4.2.6   Testing of the website

After all procedures, it should be checked If our website works properly,that's why we simulate a behavior of the customer that wants to buy something.

- Selecting the first item and putting into the basket:
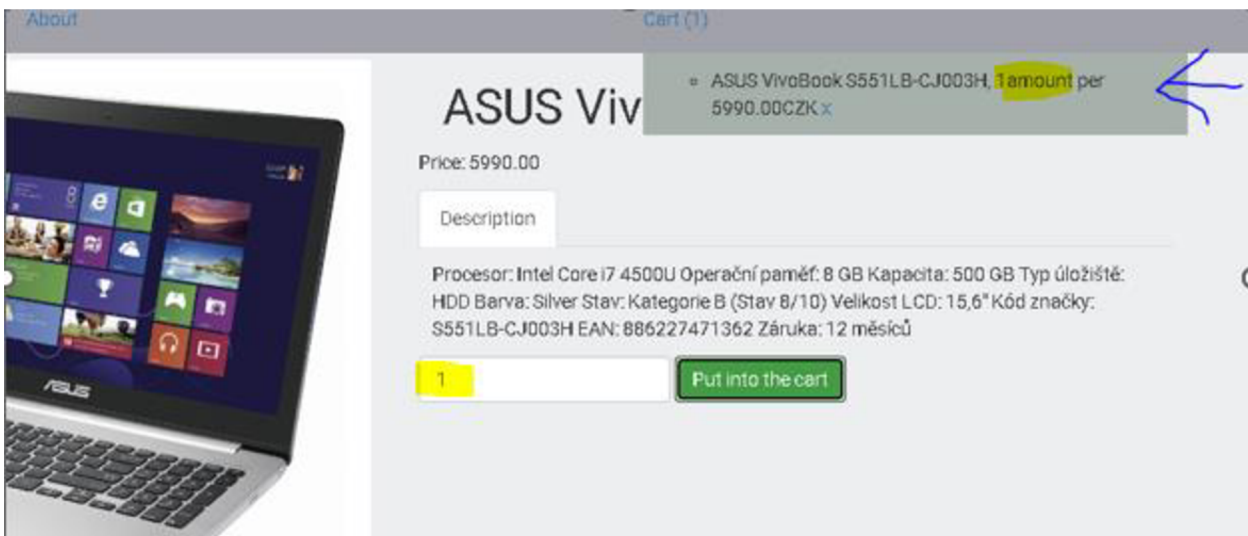
Figure 30 Ordering an item

Terminal:

40

12/Mar/2023 01:10:07] "GET /product/20/ HTTP/1.1" 200 10450
QueryDict: {'product_id': ['20'], 'nmb': ['1'], 'csrfmiddlewaretoken': ['FoR7WjjI2HrVpyGgOkXu7IsZ0q4Cwr7vZ0hEqfedN2idUAKwnfv1Qjiuji19eEHc']}>
12/Mar/2023 01:10:13] "POST /basket_adding/ HTTP/1.1" 200 129

*Figure 31 response in terminal*

- Selecting the second item
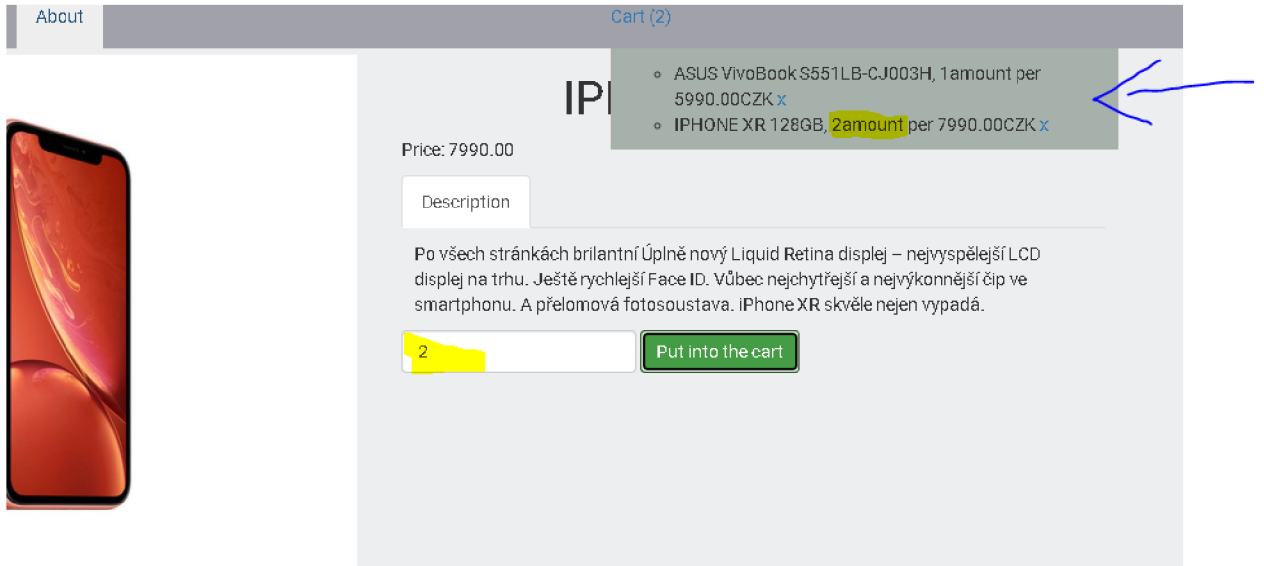
Figure 32Ordering an item

Terminal:

[12/Mar/2023 01:10:59] "POST /basket_adding/ HTTP/1.1" 200 207
<QueryDict: {'product_id': ['16'], 'nmb': ['2'], 'csrfmiddlewaretoken': ['xvZon1Q87zVj5wyEqtqnbfUvaEdni3E4R7pVRXLDSUMBAyCUZoYUUQKOtwaUOgeL']}>

Figure 33 Response in Terminal

- Openning The basket page:
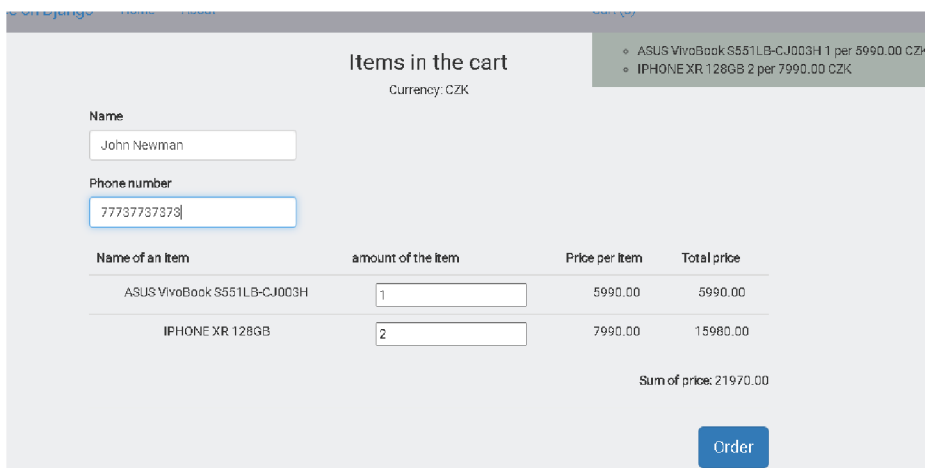
Figure 34 Purchasing

41

- Clicking the button <u>Order</u>

  <u>Terminal</u>:

Figure 35 response in Terminal

- Checking the order in admin panel

Figure 36 Created order after simulation

After simulation the customer's behavior, no issues were found, the items are added into the basket, and then after clicking the button information is posted and the order is created.

4.3 ISO/IEC 9126 software quality standards

The defined advantages of the framework by using ISO/IEC 9126 software quality standards

Table 1 Evaluationof Django

Portability	• Built-in development server that helps to test applications on platforms; • Flexible and Modular architecture that helps to scale applications;
Functional Suitability	• Dynamic pages; • Extending and Including HTML files for avoiding repeating of writing code; • A high number of libraries that can be imported for additional functionality;

	• Caching system;
Reliability	• ORM system helps to ensure integrity and consistency; • Caching system helps that reduce loading of webpage and improve response; • Built-in support for finding errors; • Good default security settings;
Practicality	• Built-in admin interface; • Powerful ORM system; • A good ecosystem; • Flexible architecture; • Different additional libraries;
Usability	• High level of flexibility; • A good documentation; • High level of functional correctness;
Maintainability	• High level of Modular system that makes easy to organize the codebase of applications; • A good separation of concerns between different parts of applications; • A good built-in testing system that makes easy to write automated tests;

4.4 Disadvantages

After the creation of the website, we found some disadvantages for the framework.

• Monolithic

The architecture MVC makes any project to much monolthic, that's why in some cases it was difficult to reach the needs as it was difficult to customize Django.

- Deploying all components together

  As Django monolithic, all components are deployed together.

- ORM limitation

  Although, ORM system is user-friendly and easy to use, but it is limited in the functions, that's why in some cases we will have to write some SQL queries.
  .

- High level of Knowledge

  To reach a high level of performance for Django and use all opportunities, a developer is demanded to have expertise in web development and database design

5 Results and Discussion

In general Django provides a good built-in development server that can be used to run Django applications locally during the development process. This can help to improve the portability of Django applications by making it easy to test and develop them on different platforms and environments. In addition, Django provides a flexible and modular architecture that makes it easy to deploy and scale Django applications in different environments. For example, Django applications can be deployed as standalone applications, or they can be integrated into larger web application frameworks or microservices architectures.

The degree of the functions provided by Django is stated and implied on a high level. With the help of the framework, it was created a dynamic website and the forms that work in Django interesting with the help of it; the data was read from the models, and then generating HTML pages were occurred with the necessary data. In case of the sudden closing of the page, the information in the cart has not been lost due to the help of Caching. Also, it is essential to notice the idea of the admin panel as the centric interface is flexible and easily understandable, reducing time for managing the website's content. Also, as Django based on python language, that's why it is easy to import some additional libraries that be used in the project in the future. The key advantage is its built-in admin interface, which provided a powerful and easy-to-use interface for managing the content and data of a web application. It includes a powerful ORM (Object-Relational Mapping) system that provides a high-level abstraction of the database, making it easy to interact with the database without writing complex SQL queries, but due to limitations, for high level queries we still need to use it. Overall, Django is a highly usable web framework that provides developers with the tools and features they need to build highly usable and user-friendly web applications. Its built-in features and functionalities, flexible architecture, and rich ecosystem of third-party packages can be a popular choice.

6 Conclusion

In conclusion, we can claim that the Django framework is a scalable option for developers looking to create an e-commerce website. With built-in different features and a huge amount of functionality, including an admin panel or reporting system, Django can make easier the development process.

The main benefit which was found during the practical part of using Django is the Model-View-Controller (MVC) architecture, which promotes modular, maintainable code by separating the presentation layer from the business logic and data models, but also, due to this architecture, it makes any project too much monolithic, which brings one additional issue, that all components, in this case, are deployed together. Also, using the Admin panel makes filling the database much more easier, and we can avoid using SQL language as responsibility takes the mentioned built-in system. Only the point is that we should write some logic to set up the admin panel. Indeed, Django provides opportunities for creating dynamic pages that help us to avoid repeating code and correspond to the needs of e-commerce websites.
Following the evaluation of the Django framework using software quality metrics, it can be concluded that it performs at a high level across all six characteristics

As Django based on the python language, that's why a huge amount of third-party packages can be imported that can make integration of essential e-commerce components.

In the end, Django is a top-tier framework for creating e-commerce websites. Although, its many advantages, developing an e-commerce site with Django remains a complex undertaking that demands expertise in web development and database design. However, it is critical to have a comprehensive understanding of its capabilities and limitations before commencing such a project.

7 References

7.1 Bibliography

Karthik, P. (2018). Web Application Using JSP. International Journal of Innovative Research in Technology, 4(10), 95-97.

Hix, D., & Hartson, R. (1993). Developing User Interfaces: Ensuring Usability Through Product & Process. John Wiley & Sons, Inc.

Calongne, C. (2004). Designing for web site usability. Education and Information Technologies, 9(3), 235-248. doi: 10.1023/B:EDUC.0000038144.94559.8c

James, R., Rumbaugh, I., Jacobson, I., & Booch, G. (1998). Advanced Praise for The Unified Modeling Language Reference Manual, Second Edition. Addison-Wesley.

Tariq M., (2019). SEO: A unique approach to enhance the site rank by implementing Efficient Keywords Scheme Corresponding

Islam, R., & Khan, M. (2010). Code quality evaluation methodology using the ISO/IEC 9126 standard. International Journal of Software Engineering & Applications (IJSEA), 1(3), 17-28. DOI: 10.5121/ijsea.2010.1302

James, McGaw. (2009). Beginning Django E-Commerce (Expert's Voice in Web Development) 1st ed. Edition. ISBN 1430225351.

Lutz, M. (2011). Programming Python. Boston: O´Reilly Media. ISBN 978-0-596-15810-1.

Matthes, E. (2019). Python Crash Course: A Hands-On, Project-Based Introduction to Programming. San Francisco: No Starch Press. ISBN 978-1-59327-928-8.

George, N. (2020). Mastering Django. ISBN 0648884414.

Holovaty, A., & Kaplan-Moss, J. (2010). The Definitive Guide to Django: Web Development Done Right (2nd ed.). Apress. https://doi.org/10.1007/978-1-4302-1937-8

7.2 Online resources

Django official documentation: https://docs.djangoproject.com/
Django for Professionals: https://djangoforprofessionals.com/
Django Packages: https://djangopackages.org/

Morris F. (2017). Frontend vs Backend. Retrieved from
https://www.corewebprogramming.com/frontend-vs-backend/

Big, N. (2016, September 29). Beginner Lesson 1: Why Django? Retrieved from
https://masteringdjango.com/django-tutorials/beginner-lesson-1-why-django/

Juviler, J. (2022). Static vs. Dynamic Websites: Here's the Difference. Retrieved from
https://www.wix.com/blog/2022/02/static-vs-dynamic-websites/

Jeen B., (2023). What Is the Semantic Web? Retrieved from
https://www.ontotext.com/knowledgehub/fundamentals/semantic-web/

JavaTpoint. (2021, September 28). Django Models. JavaTpoint. Retrieved from
https://www.javatpoint.com/django-model

Dhulam, A. (2019, June 19). Django Framework. Medium. Retrieved from
https://medium.com/@ajitdhulam/django-framework-11ccf1614cb5

8 List of pictures, tables, graphs and abbreviations

8.1 List of pictures

8.2 List of tables

Appendix

E-commerce website.zip

- Visible part of website