

**CEN 308 SOFTWARE ENGINEERING**

PROJECT DOCUMENTATION

Sneaker Station

Prepared by:

**Lejla Breščić**

**Amna Ganić**

Proposed to:

**Nermina Durmić, Assist. Prof. Dr.**

**Aldin Kovačević, Teaching Assistant**

21.6.2023.

**Table of contents**

[1. Introduction 3](#_Toc137803288)

[1.1. About the Project 3](#_Toc137803289)

[1.2. Project Functionalities and Screenshots 3](#_Toc137803290)

[2. Project Structure 6](#_Toc137803291)

[2.1. Technologies 6](#_Toc137803292)

[2.2. Database Entities 7](#_Toc137803293)

[2.3. Architectural Pattern 7](#_Toc137803294)

[2.4. Design Patterns 8](#_Toc137803295)

[3. Conclusion 8](#_Toc137803296)

# 1. Introduction

Presented below is our solution to the project and its requirements for CEN 308 Software Engineering course on the International Burch University. The project requirements can be found on the following link [[1]](#footnote-1). We tried to adhere to every requirement that it was presented upon us.

In each section and its sub-sections, you will find the detailed analysis of the project done.

## Application Logo



## 1.1. About the Project

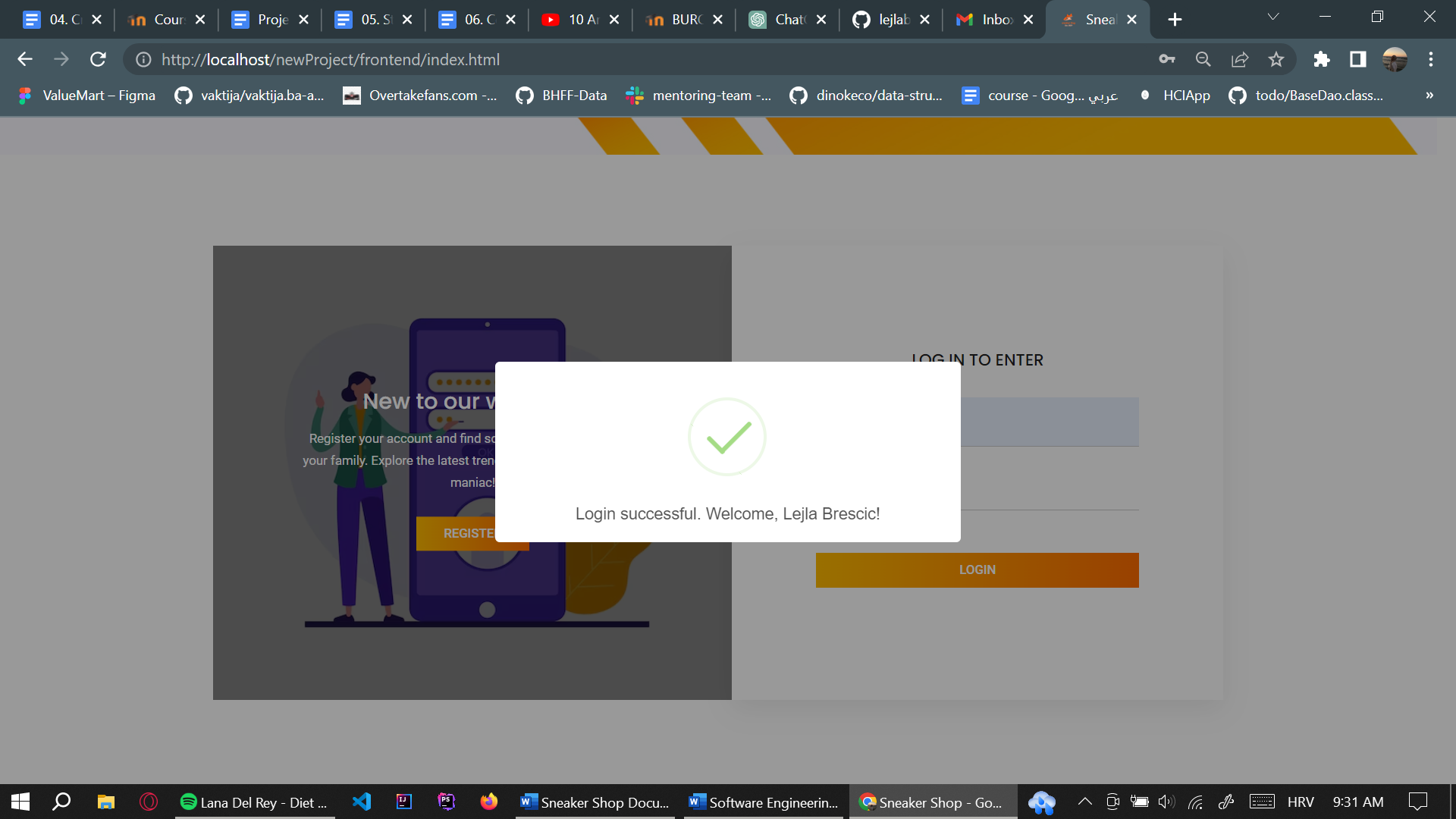
The Sneaker Shop is the web application for selling sneakers. It gives insight into latest footwear trends from a various companies and manufacturers. It offers different functionalities to users in order to retrieve and find products in much easier way. It gives companies the opportunity to exhibit their products and for the customers to buy them online and not being obligated to go in store. Regarding our application’s roles, we decided to not go with the initial plan that we proposed in the project proposal. Our application has two roles: registered user & admin. We will discuss their roles in the sections below. Our user interface is responsive, user-friendly, as we tried to maximize user experience. Link to our application: <https://starfish-app-fd6z8.ondigitalocean.app/>

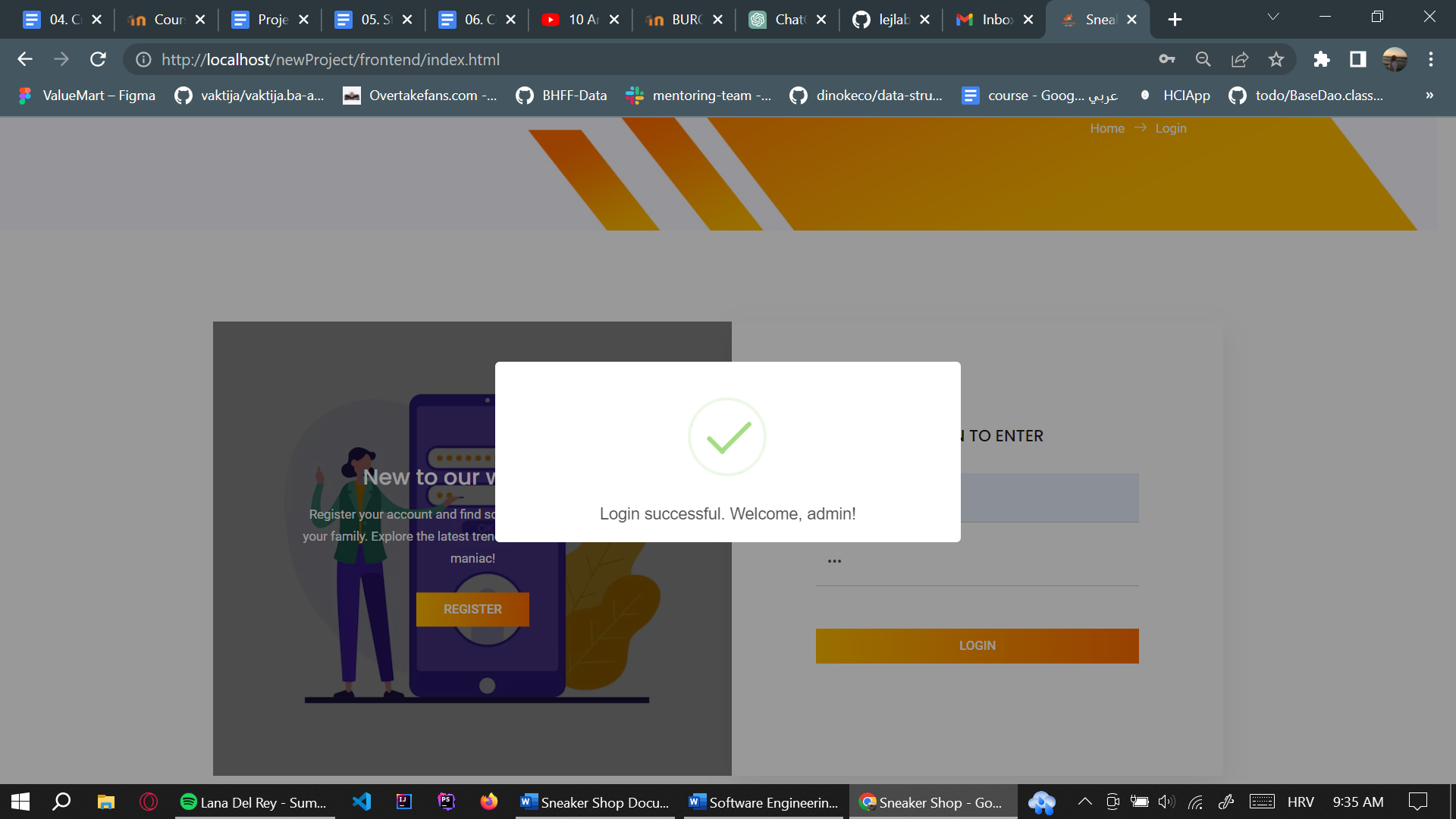
## 1.2. Project Functionalities and Screenshots

Some of the main features of application include the following:

* User can easily be retrieved from database if they exist. By inputting email address and password, the user backend is called to check if the user’s credentials are in the database.

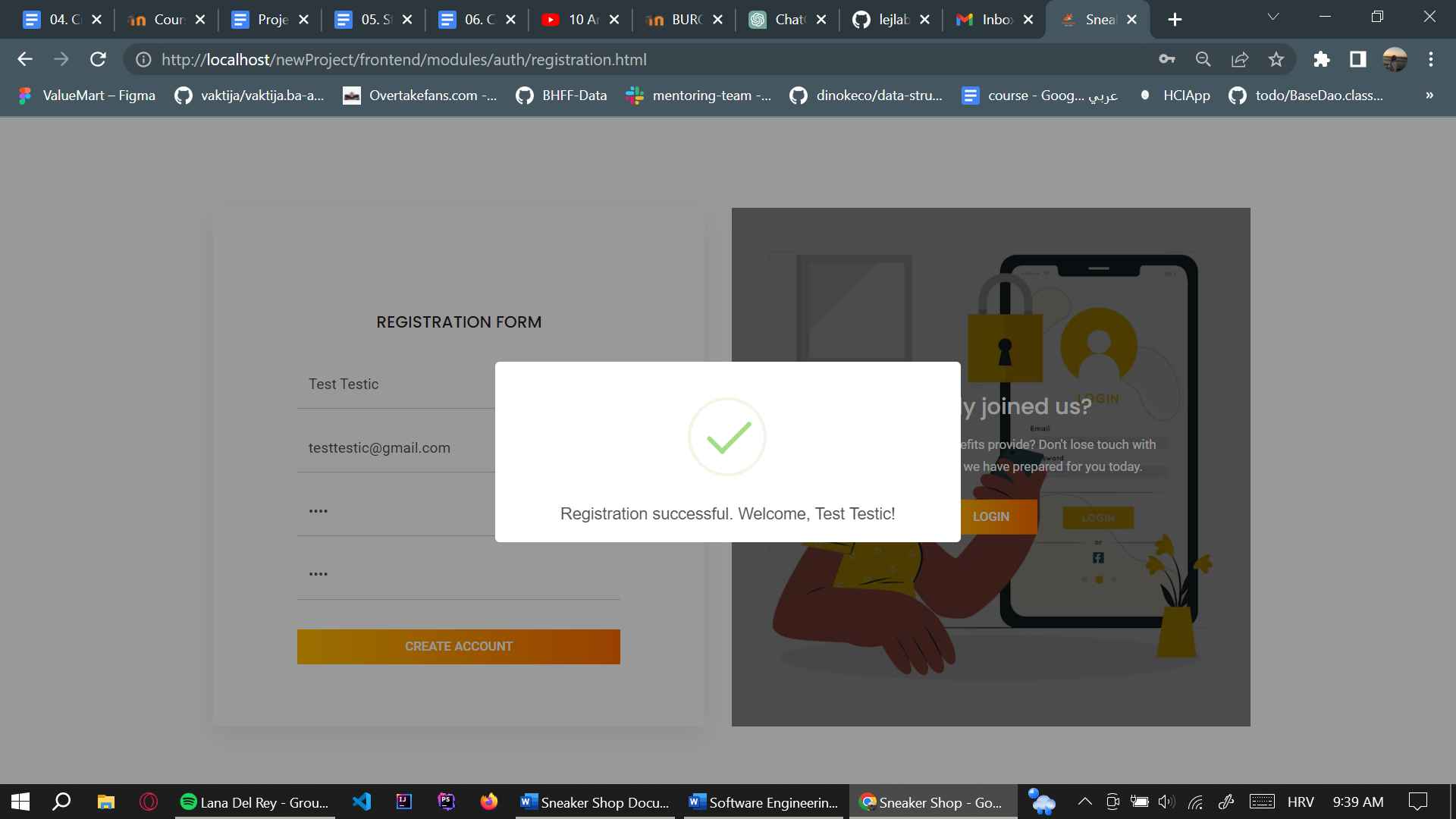
If that is the case, user is redirected to the home page, if their role is user. If their role is admin, they are redirected to the admin dashboard. If user doesn’t have an account, by clicking the *Register* button, user can easily make the account[[2]](#footnote-2).





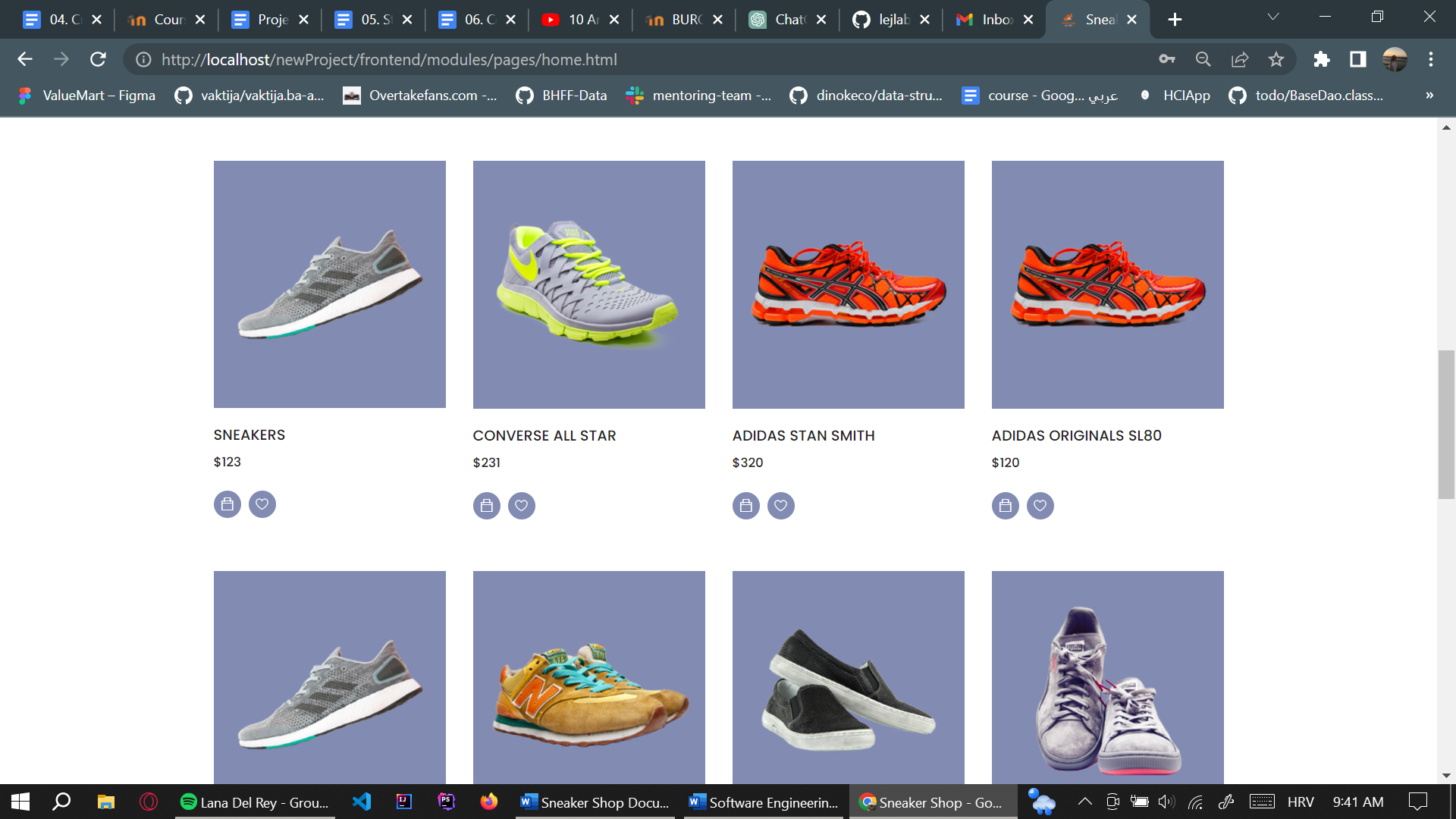
* Register

As stated above, user can easily make the account by inputting username, email address, password and once again to confirm the password. User cannot leave any field empty, nor the username can have whitespaces, as we coded backend input validations. No more than one user can have the same email address. After registration, user must go to the login page again in order to proceed with our application.



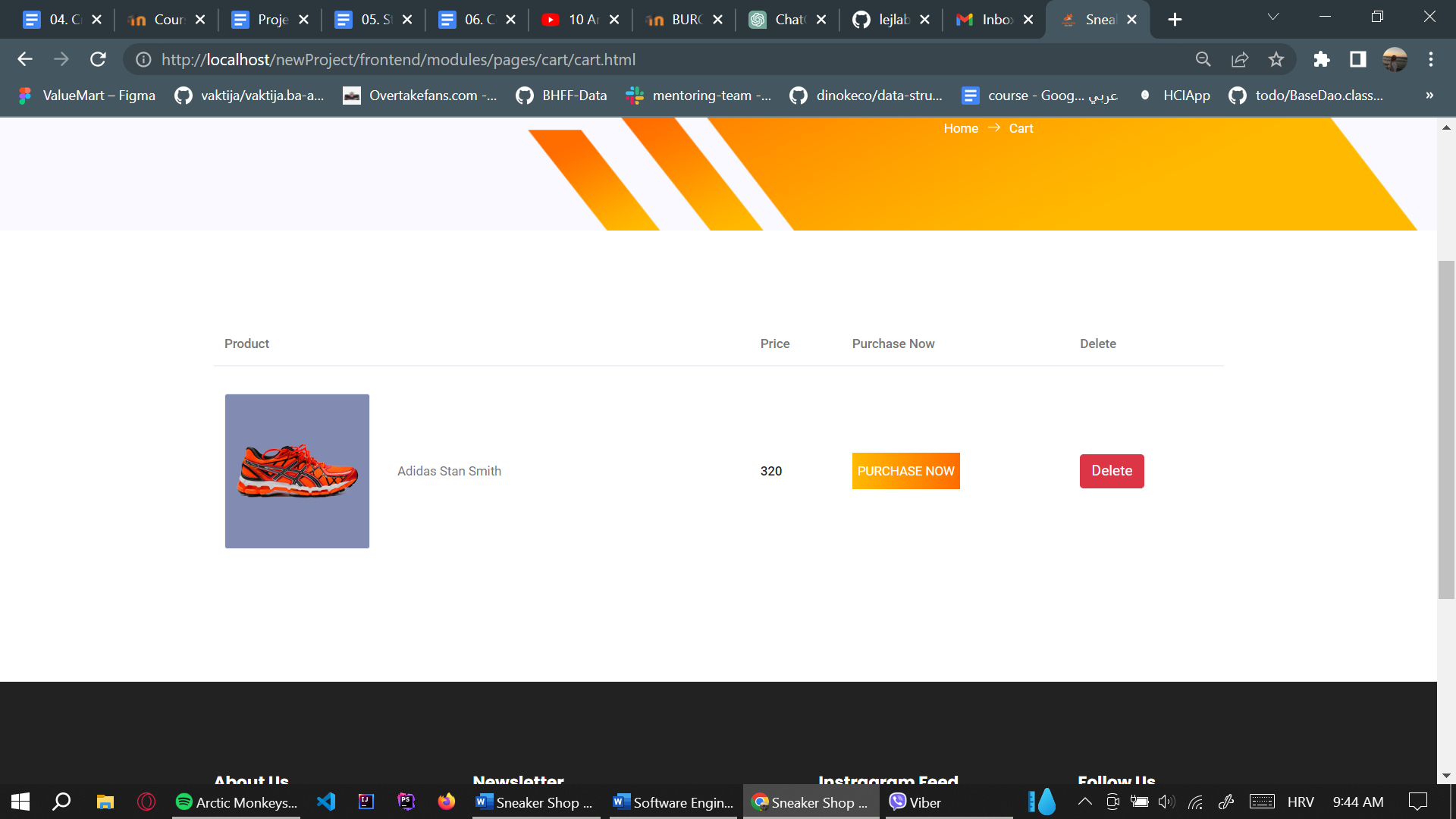
* Product retrieval

Products are retrieved from database; their price, name, alongside with their image.



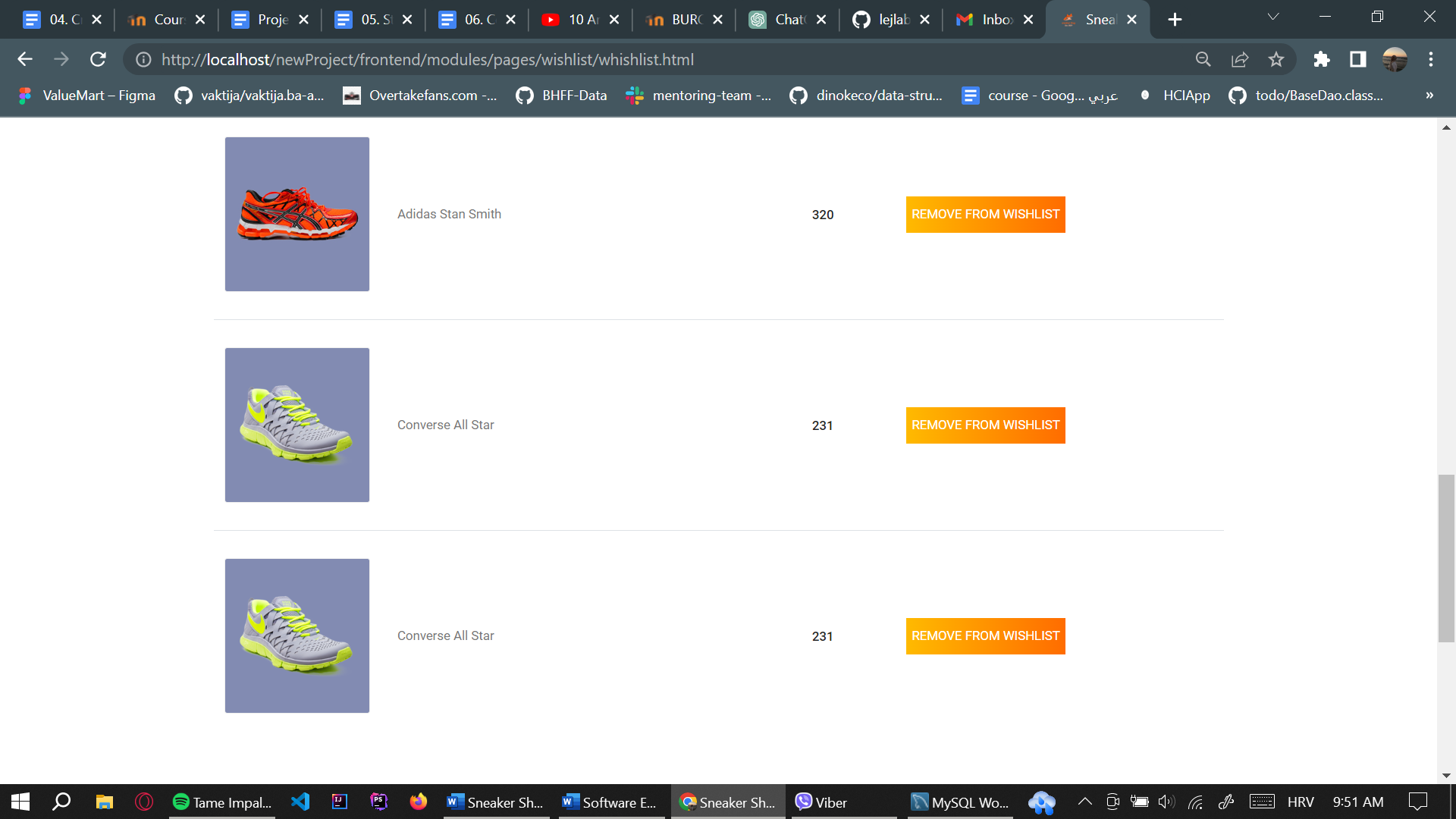
* Adding to cart & Purchasing

Every product can be added to the cart, by clicking on the *Bag* icon. It is important to say that every item added to cart stays there even after logging out. It can be removed from the cart by clicking the *Delete* button. Every item that is added is stored in the *Cart* table in our database. To be more precise, userId, productId and other product details are added. By clicking *Purchase now* button, the order is stored in *Purchase* table.



* Adding to wishlist

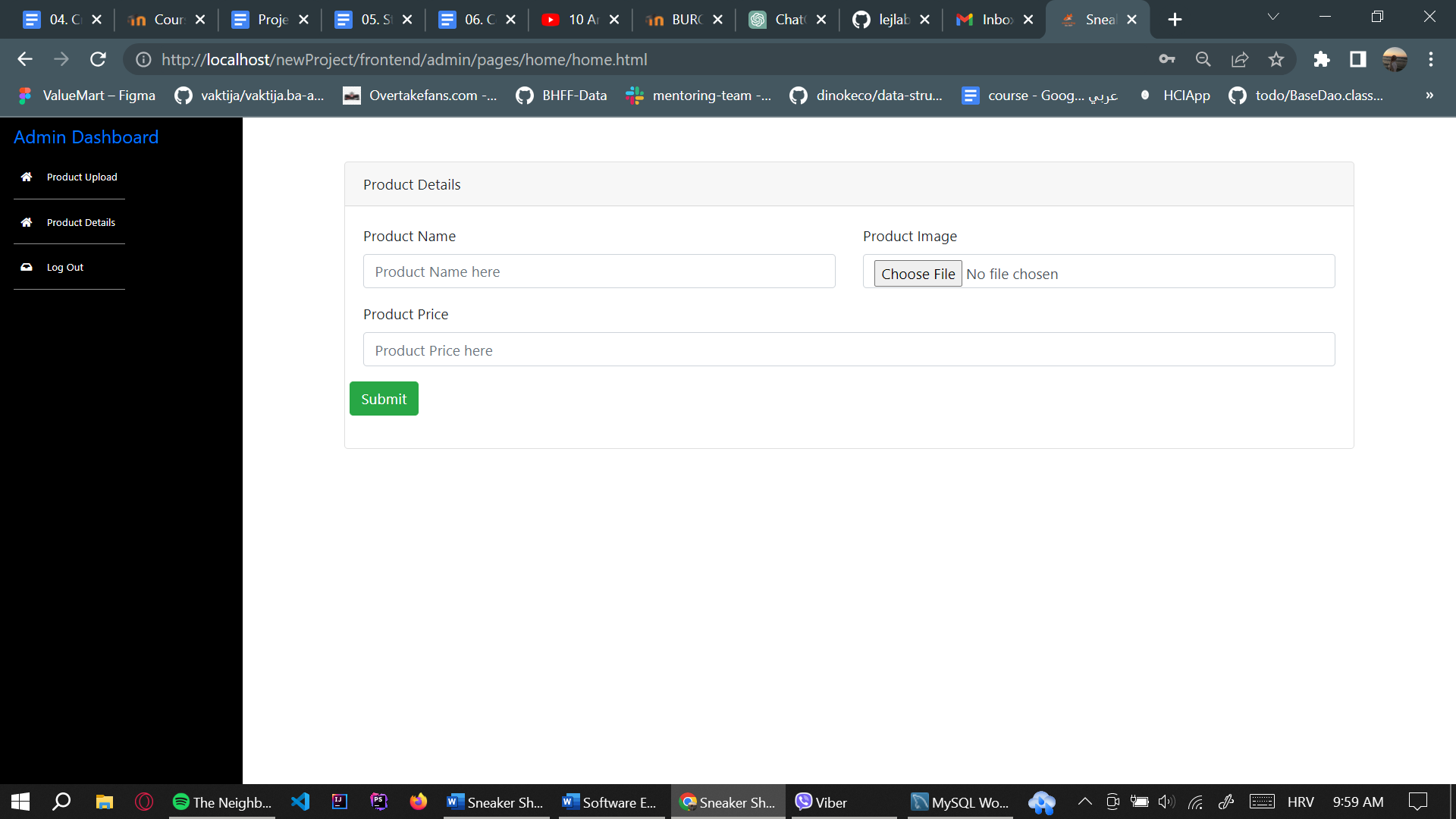
Products can be added to wishlist by clicking *Wishlist* icon. To see all wishlist products, we can click on the *Wishlist* icon in the navbar. The same thing works for the wishlist as for adding to cart. These items stay for each user even after logging out. Wishlist items can be removed by clicking *Remove from wishlist* button. Every product added to wishlist is automatically stored in *Wishlist* table in database.



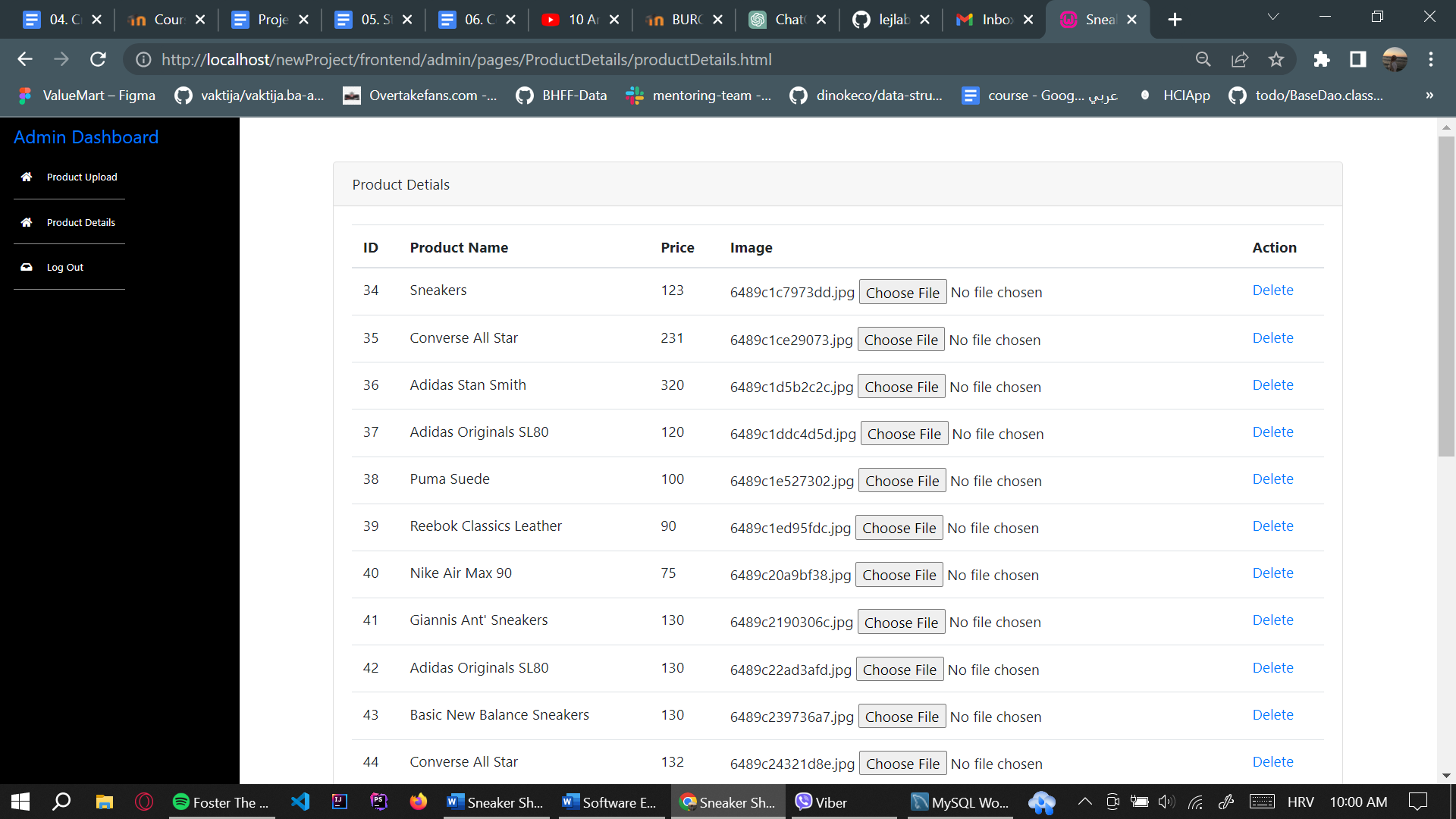
* Admin functionalities

This is the feature that both of us are most proud of. Together we found a cool tutorial to follow in order to make the admin dashboard. After successfully creating and finishing the functionalities, the process of adding new products, changing their details became automated.

On the picture below, we can see the product upload page. After submitting details and adding product image, product is added in *ProductDetails* table. It is important to mention that pictures chosen are saved in upload folder in assets folder of this project.

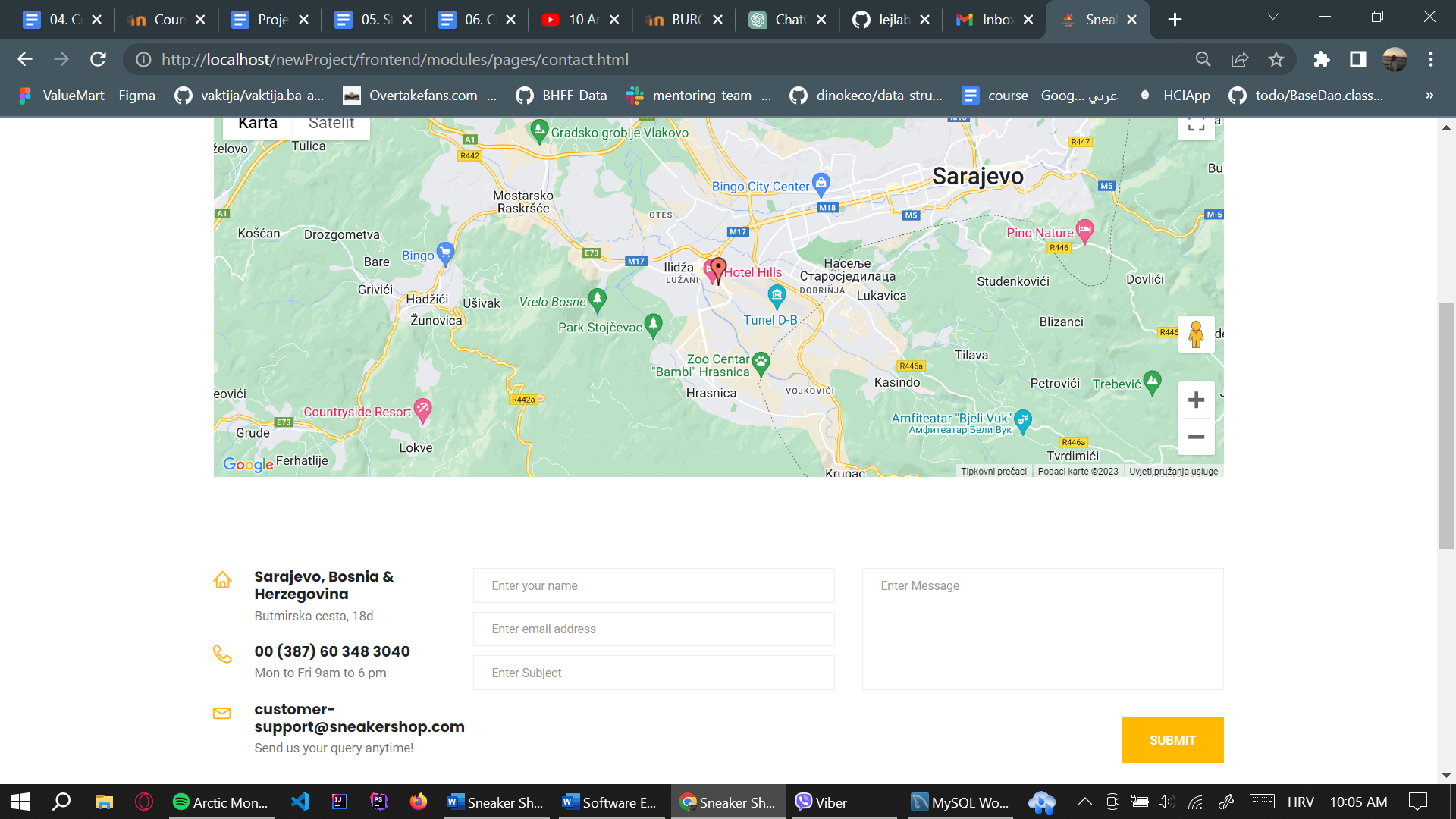


Product details page is presented below. Admin can update product name, price or choose a new product image. After clicking logout, the index page is shown.



* Contact page

In order to create accessible and user-friendly application, we created *Contact* page that gives details about our store and possibility to send query regarding some issues. The map is made using Google API and the query is sent using Google SMTP service.



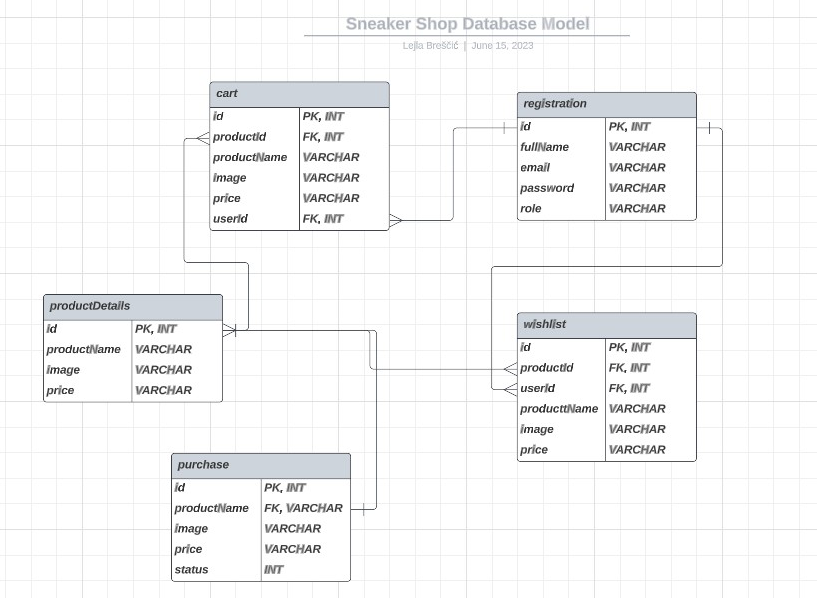
# 2. Project Structure

## 2.1. Technologies

For this project, regarding the frontend, we decided to use HTML, CSS and JavaScript (jQuery). For the backend, we decided to go with PHP (FlightPHP), as this is something we are learning on our Web Programming course. For the database, we used MySQL database. Both database and application are deployed on Digital Ocean. We also used Google API for the map and Google SMTP server for query sending.

For the PHP, we used PSR-12 coding standard. For the JavaScript, we used AirBnB JavaScript Style Guide standard.

## 2.2. Database Entities

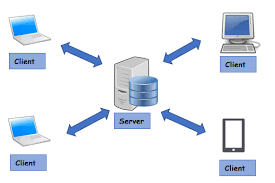


On the picture below, the initial project schema is shown. The tables and their functionalities work as follows:

* registration 🡪 saves details about users
* productdetails 🡪 stores information about products
* wishlist 🡪 stores user and product details from wishlist
* cart 🡪 stores user and product details from cart
* purchase 🡪 stores user and product details for the purchase

## 2.3. Architectural Pattern

Regarding the architectural pattern, we decided to go with **client-server** architectural pattern. The reason for that is because it provides scalable and efficient options. Client, using a device, is communicating with server, that is hosting data and business logic. Server handles authentication, purchasing, order processing and others, while the client handles user experience and interaction with computer. The picture explanation can be seen below.

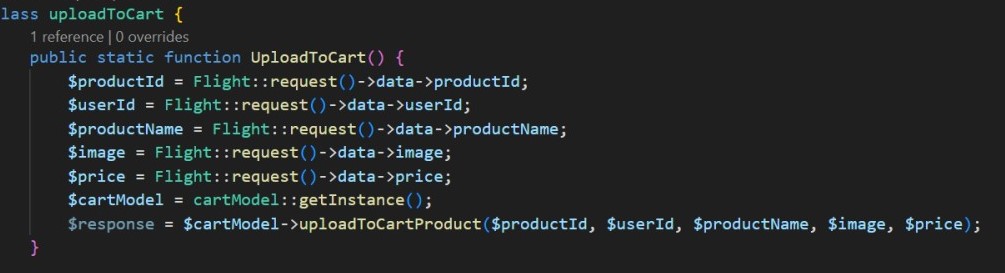


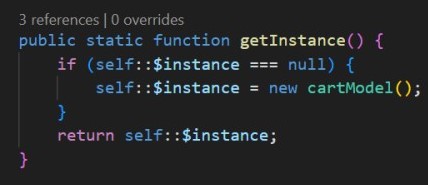
## 2.4. Design Patterns

As we will discuss in the conclusion part of the project, we had the most difficulties in the project while working and choosing the right design patterns. The three patterns that we have chosen are listed below.

- **Singleton pattern**

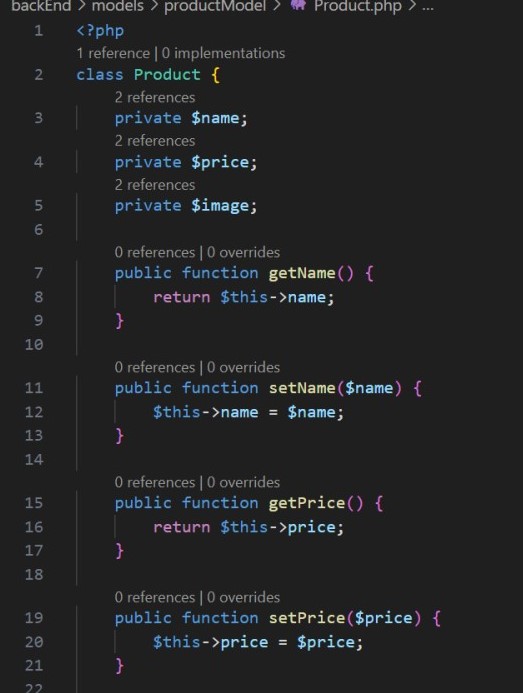
Singleton pattern ensures that a class has only one instance, and provides a global point to it. In our project, we implemented singleton design pattern in addToCartController.php and cartModel.php. The examples of singleton and its usage can be seen below.

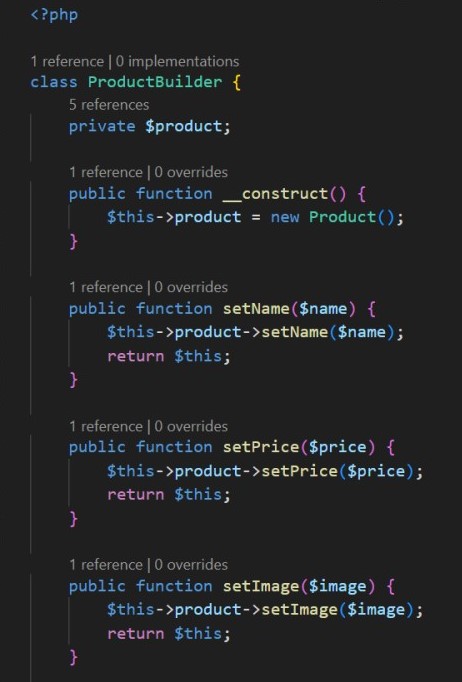


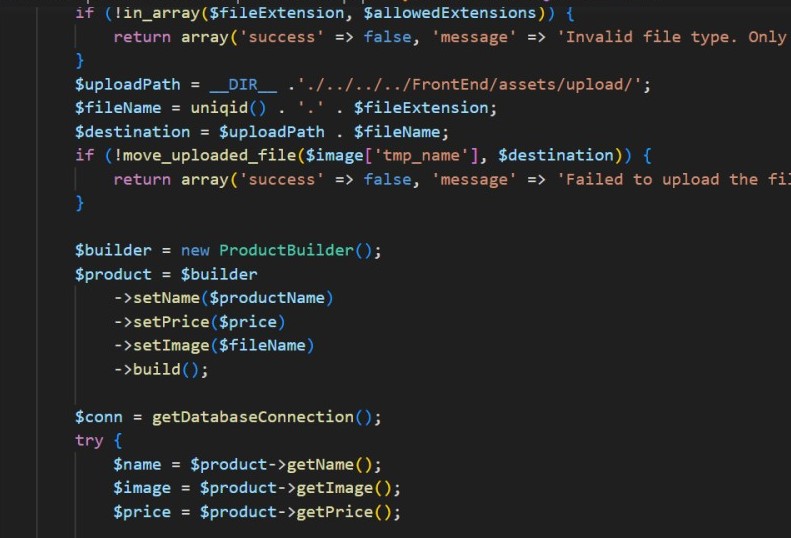


- **Builder pattern**

**Builder** is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code. The examples on how we used this pattern is shown on the images below.

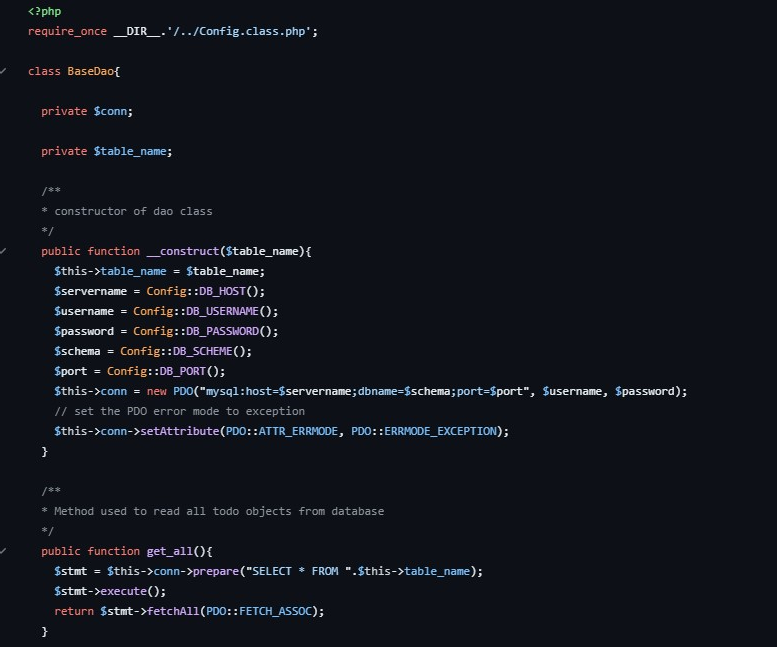


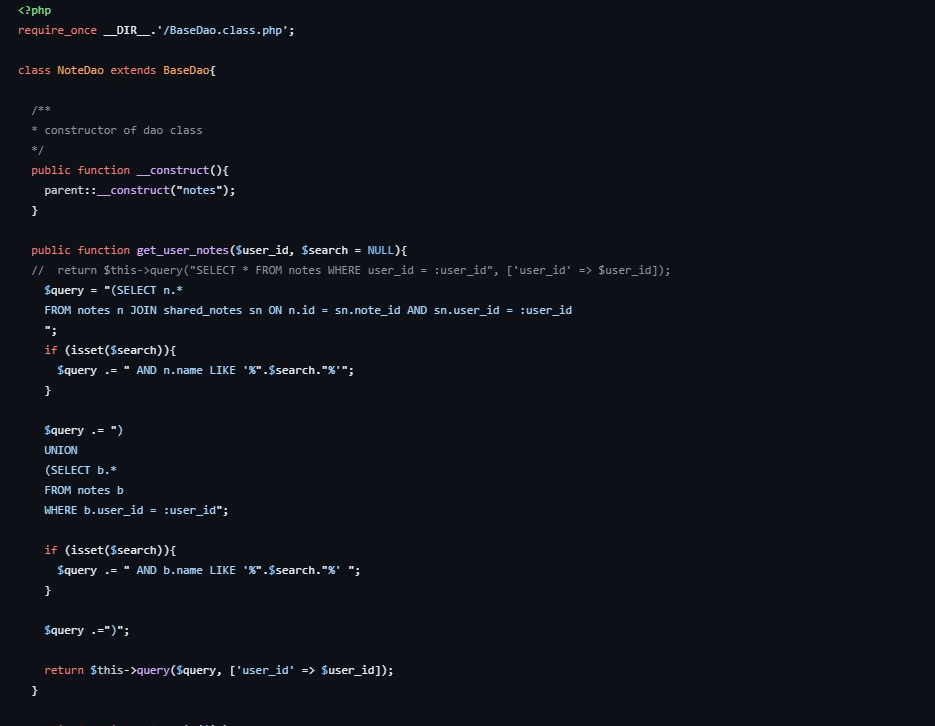




- **Factory pattern**

Because of lack of time, we did not implement this pattern, but we will provide the whole plan in general on how the implementation would look like. Factory pattern defines an interface or abstract class for creating an object, but let the subclasses decide which class to instantiate. Since we are also on the Introduction to Web Programming course, we learned on how to separate backend logic into DAO, routes and services. What we could’ve done in our project is to separate the database connection and basic functionalities and put them in BaseDao class and after let every other Dao only extend this class and to add its own functionalities. Also, we could have put all routes for classes in their appropriate Routes files, instead of putting them in index.php file.   
The example of this pattern is shown below.





# 3. Conclusion

To be honest, since we are having Web Programming course at the same time as this course, it was interesting, but sometimes hard to learn many things at the same time. We realized the importance of this subject while doing this project and how much of a knowledge we can use later in work. Personally, the most difficult thing to do was to understand design patterns and which one to use on our project. Maybe the reason for that is because I firstly coded the whole application and then decided to see how I can refactor the code to adhere to some design pattern.  
Also, we had problems with using Selenium for PHP. It was much easier to setup Selenium for Java, at least in our opinion, so that is the reason why we decided to go with pure PHPUnit and not use Selenium, even though is definitely a more interesting option.

Apart from everything, we enjoyed working on this project. We think it is really important to do as it prepares for the career of software engineer, as we are doing every aspect of work while creating a new software.

1. <https://docs.google.com/document/d/1e26z23pBHTyIJYXeYw9F7G6P375fBRej_BKIX7aaAok/edit> [↑](#footnote-ref-1)
2. Some test data -> admin credentials : [admin@gmail.com](mailto:admin@gmail.com) , 123

   regular user credentials : [test@yahoo.com](mailto:test@yahoo.com), test [↑](#footnote-ref-2)