CEN 308 SOFTWARE ENGINEERING

PROJECT DOCUMENTATION

Online Pharmacy

Prepared by:
Naida Fatić
Mirza Krupić

Proposed to:
Nermina Durmić, Assist. Prof. Dr.
Aldin Kovačević, Teaching Assistant

19.06.2022

Table of Content

1. Introduction

1.1. About the Project

Online Pharmacy is online store for selling, buying medicines. This will provide users ability to buy medicines from their homes without leaving the house.

This application is aimed for people that cannot buy medicines from them self or in times when people cannot leave their homes like in quarantine.

The application will give users ability to see all medicines and enter their information in order to purchase medicines.

Also, the admin user of the application can see all activities in the application and add or change existing medicines.

The link to application is here.

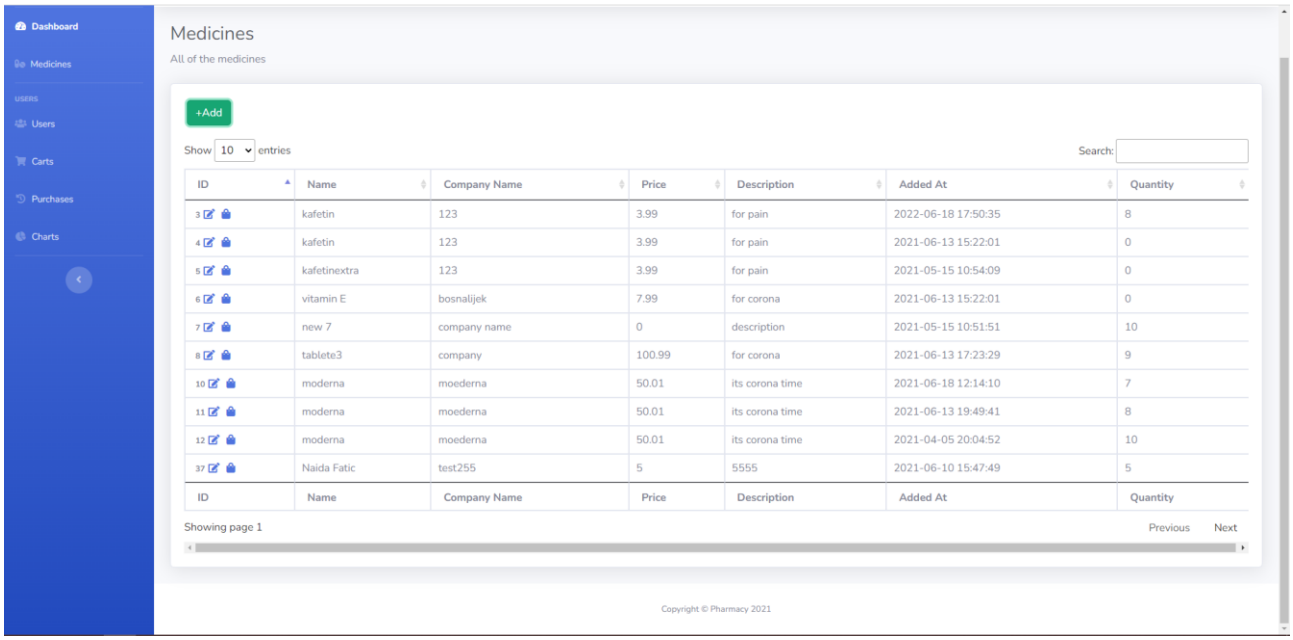
1.2. Project Functionalities and Screenshots

Describe or list the main features of the application and provide a few screenshots of your project.

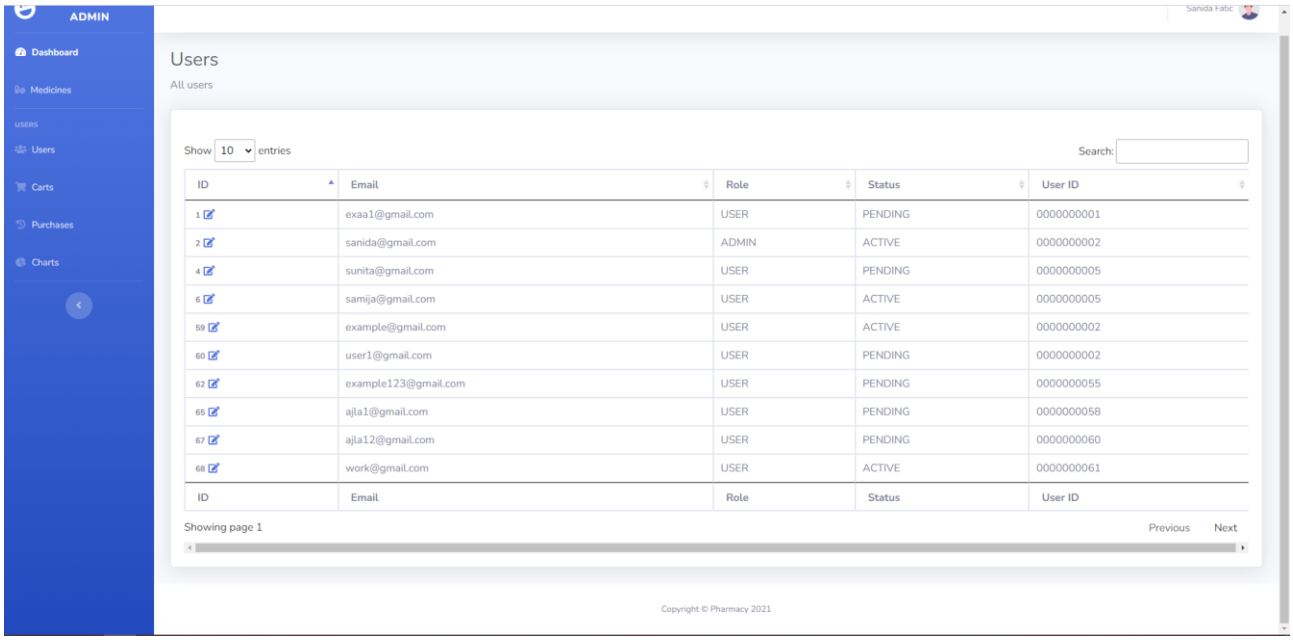
Application is consisted of two parts admin and user side. Both type of users shares some basic functionalities like login, register, forgot password.

Admin side is only accessible to the admin and she/he can:

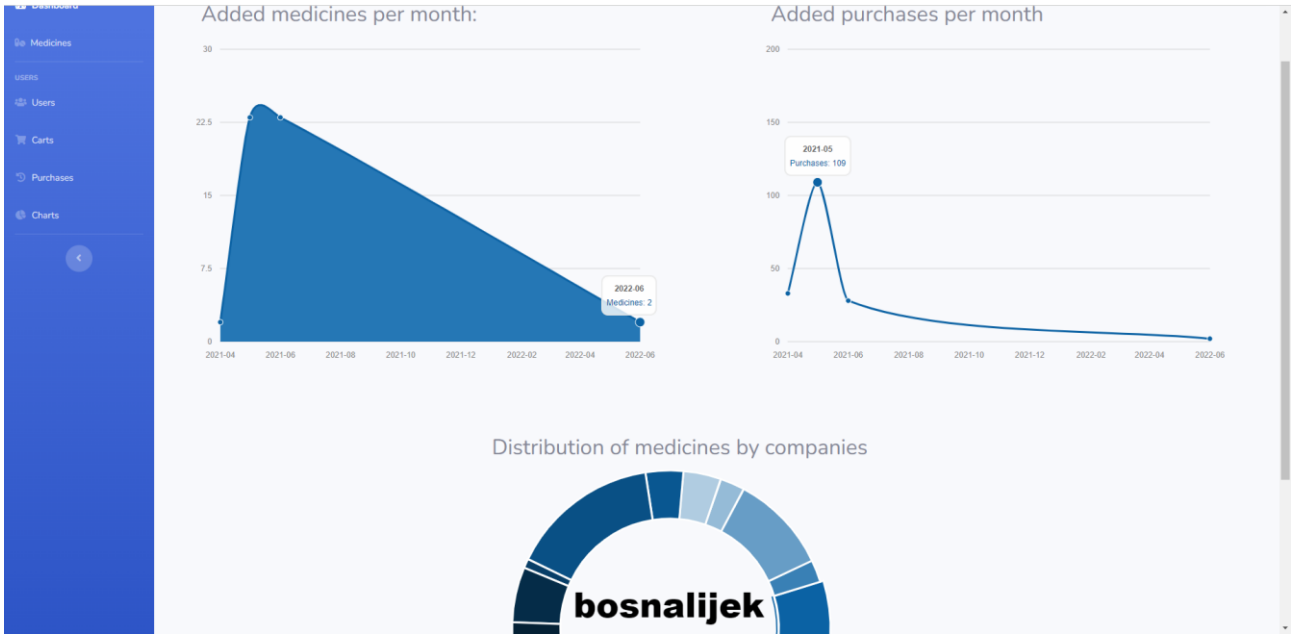
- add or update medicines,



- see and update all users,



- see all purchases,
- see all medicines in carts,
- see graph representation of the medicines that is being added, graph representation of added medicines into database over time and graph representation of all purchase made over time

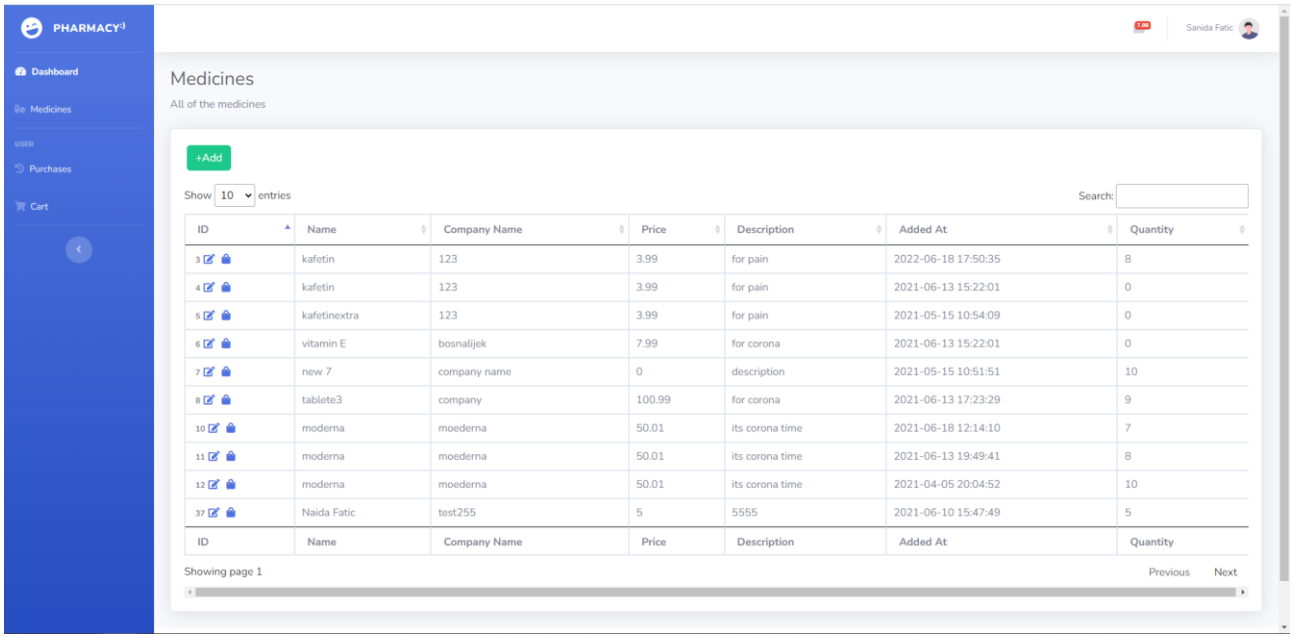


To access admin side user has to have privileges of admin and go to admin page on top left corner.

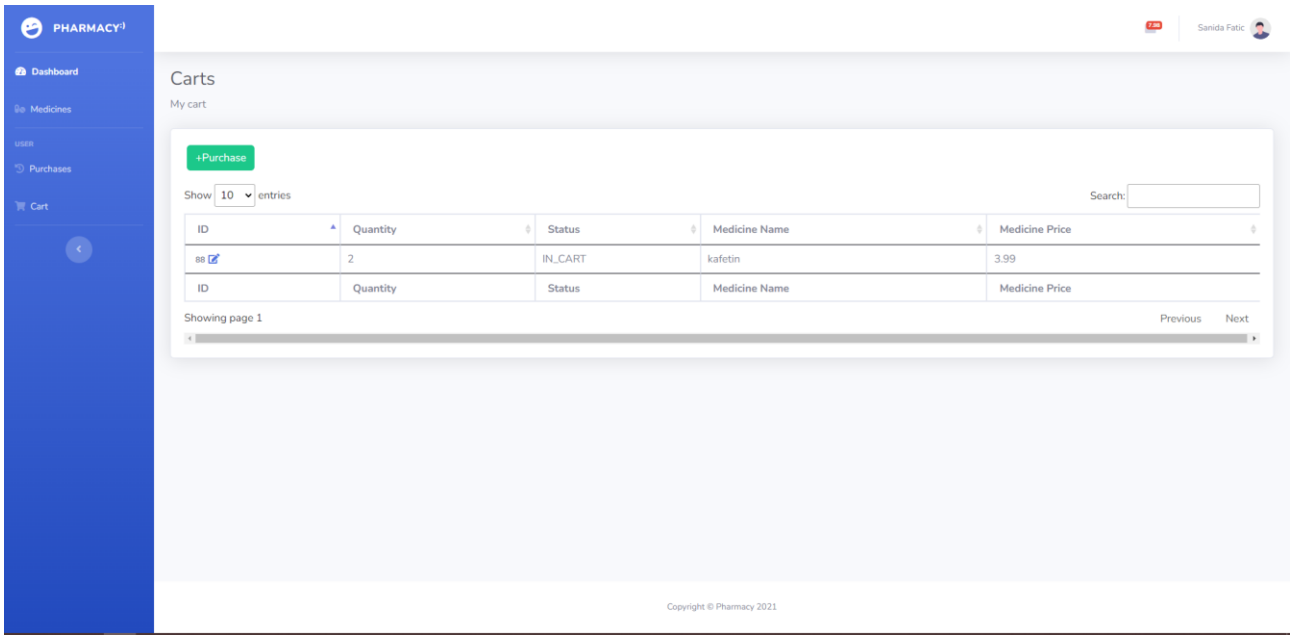
For regular users there will be only profile page with their information.

On the users side the functionalities are:

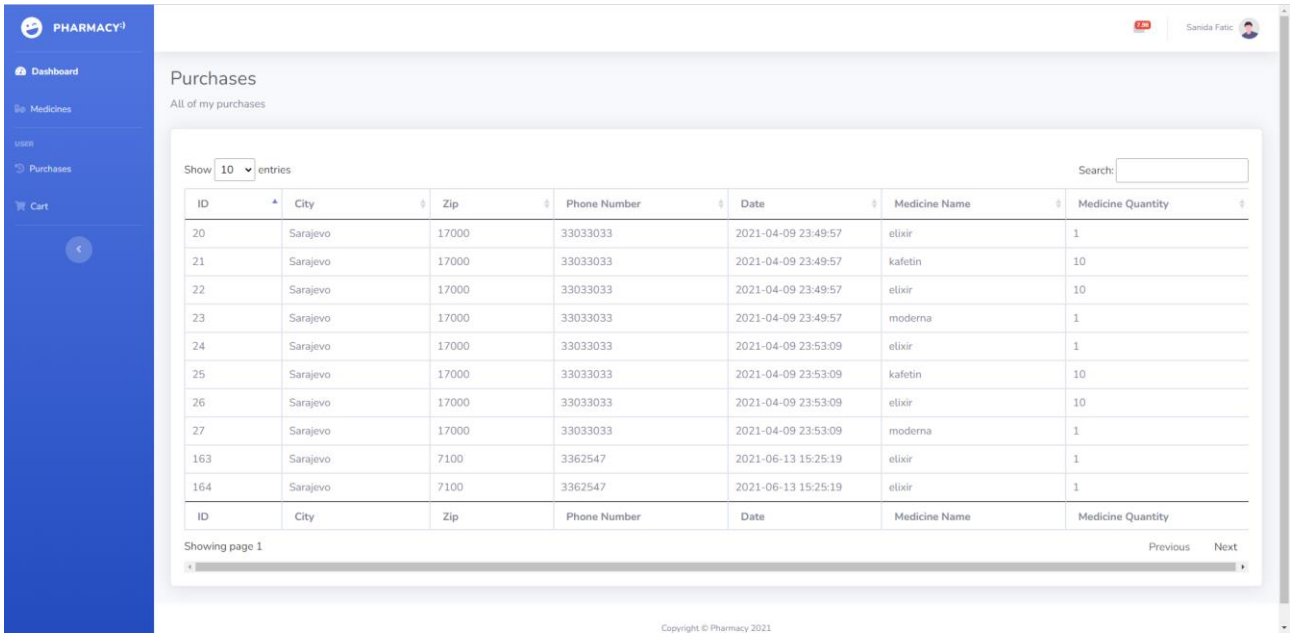
- search and add medicines



- see what is in cart, change the medicine in cart and make purchase



- see all purchase that user made



2. Project Structure

2.1. Technologies

We used PHP v8.1, HTML, CSS, SCSS, JS for programming languages.

We also used Bootstrap for frontend coding, FlightPHP for backend routing, Composer for installing dependencies and Swagger for representing routes until we build frontend.

For database we used SQL and hosted it on remotemysql.com site (we are sorry if database is slow or not working at the time of grading, it may have some limitations).

The coding standard we used in backend is PSR-12.

2.2. Database Entities

Tables in our database are:

- Accounts → stores data about users and their privileges
- Purchase → stores data about purchasing medicines
- Cart → stores data about users' cart
- Medicines → stores data about all medicines
- Users → stores data about all users and their basic information

2.3. Architectural Pattern

In this application we use Layered Architectural Pattern (n-tier pattern).

We separated out application in backend (api folder) and frontend folders (views and assets folder).

On the backend there is data access layer (DAO), business logic layer and application layer.
In the DAO we have classes for connecting to the database and creating SQL queries for each table, in the business(services) layer we implemented logic.
The application(routes) layer we have implemented routes for each table in our database and connected it to the corresponding services layer.
All three layers are connected in a form that DAO is connected to the service layer and service layer is connected to the routes layer. The layers are connected only to the layer above. If the higher level(routes) wants to fetch data from database it needs to go through the service and DAO layer.

We also have the index.php file that connect all of these layers and make them work on frontend.

The frontend folders (assets and views folders) can represent presentation layer where we fetch API and presented in HTML form.
Fetching API is done in the assets folder and presenting it is done in views folder.

2.4. Design Patterns

In this application we used patterns:

- Creational Patterns
  - Abstract factory: we used it in backend on DAO layer(api/dao/BaseDao.class.php) and business(api/services/BaseService.class.php) layer.
    We have one base class that we used to extend to other abstract classes.
    In DAO layer we used BaseDao class to define methods that are frequently repeated and used it on other DAO classes. The DAO produces generic methods that we can modify to specific table and reuse them.
    We used same method on service layer where we had base service.
    We used this pattern in order to reduce redundant code and to have more cleaner and secure code.
  - Singleton: we used this pattern in database connection(api/dao/BaseDao.class.php). We used this pattern because we want to set up one connection per user and used only that one for all transactions. We have achieved that by setting connection with database in BaseDao.class.php in the constructor method that will form only once when class is created.
- Structural Patterns:
  - Façade: we used this one in the frontend (assets\js\restClient.js, assets\js\utils.js). We use RestClient class which we create all API call in form of methods and when API is needed, we just call this class and methods.
    Also, in Utils class we have hidden methods that do complicated work like transforming HTML form data into json form. These methods are frequently called and we made it easier by calling a method that will hide all of the code and also reduce code complexity.
- Behavior Patterns
  - Command: We were not sure if the index.php can represent command pattern but it does part of the command pattern job. It connects service and route layers and all of the request has to come thought the index.php. In index.php we have error controller which will control the representation of the error in the frontend. Also, all requests form frontend first has to pass the index.php which will modify results into what we want.

3. Conclusion

We were satisfied with the application. We realized how many patterns we use but we did not recognize at the moment. We followed good practices for coding and at the end concluded that some of our code is patterns. We were a bit shocked because we just followed our logic and basic knowledge of object-oriented programming.

We could maybe improve our code in a sense that we could separate our project into backend and frontend and host bout parts on different URL or maybe just have two main folders frontend and backend where we will divide our code. This way the code can be even cleaner.

The most difficult part of this project was to host this application. The problem that we had was the outdated packages that were included in .json file. Because we implemented this application a year ago some packages were outdated for PHP8.1 version (which was used when deployed) and we had to change a code or find some solution. Eventually we had to change the version of the package to new version to make package compatible with the PHP version.