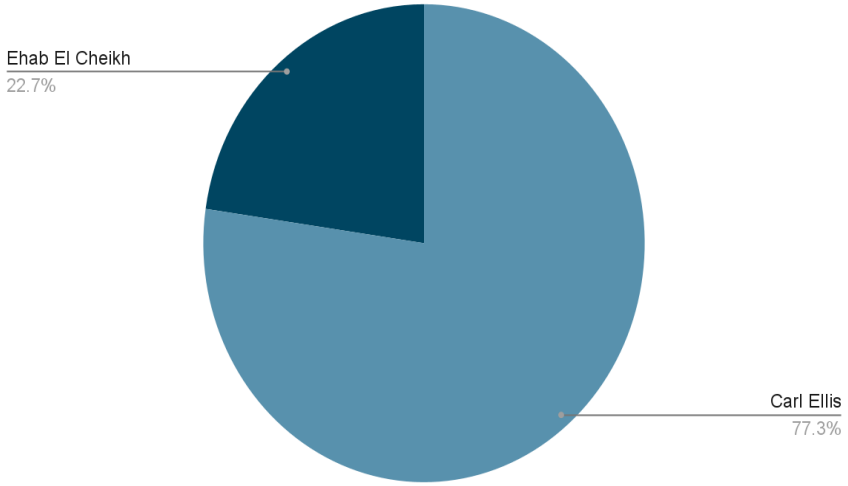
ELEC5619 Final Report

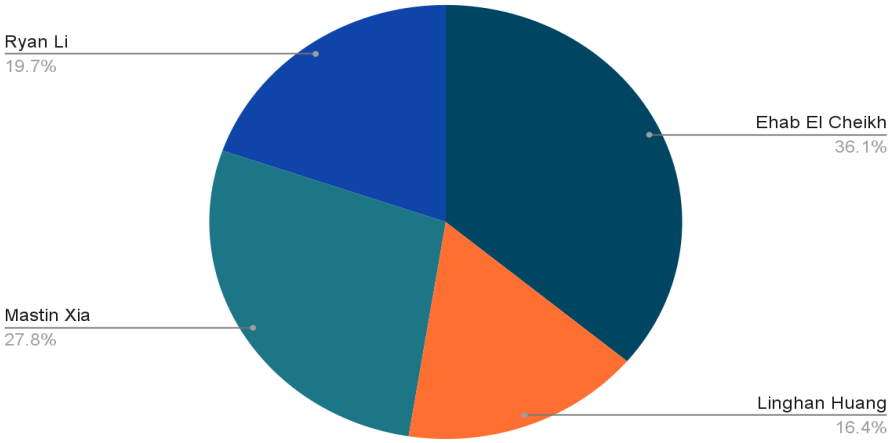
Contributions

Name	SID	Contribution	Lines of Code (Front)	Lines of Code (Back)	Testing
Carl Ellis	500706646	Front End Development	5,481	0	0
Ehab El Cheikh	490434606	Front End Development and Backend development	~ 1 5 0 0 (Used non enterprise account)	948 added 291 removed	0
Linghan Huang	500055832	Backend development and testing	0	261 added	345 added
xuhan li	490029365	Backend development and testing	0	364 added	127 added
Rui Xia	500077834	Backend development and testing	0	515 added 208 removed	63 added

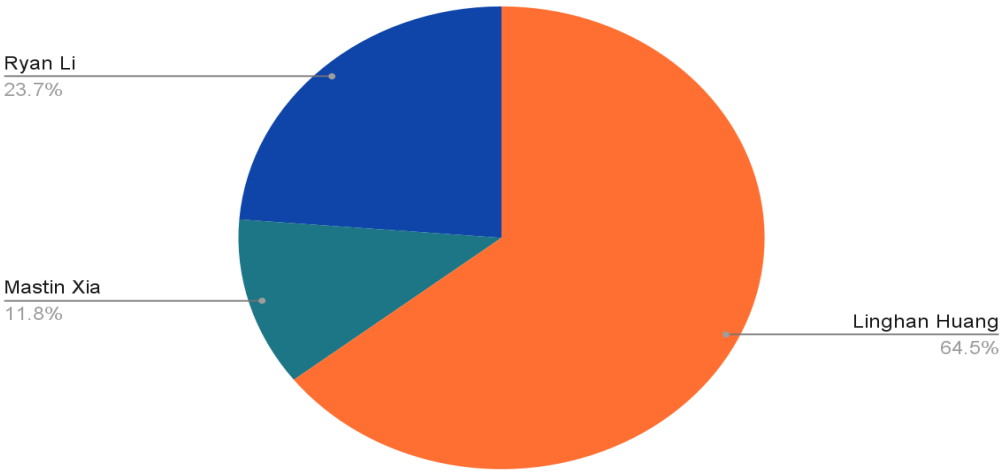
Front End (Lines Of Code)



Back End (Lines Of Code)



Testing



1. Introduction

1.1 Overview

The “Play with Pro” website aims to connect everyday players with professionals in order to enhance their gaming experience. This website allows players to learn from the best and provide them a space to communicate and learn from experienced players. This website is available to all gamers with the intention of playing with professional players meaning it has a wide range for target users. It will allow users to register a new account on the website. After registering and logging in users will be able to view a selection of games and pro players on the home page where they can choose who they want to team up with. If the user wishes to pick a different game, they are able to do so by clicking on one of the game cards featured on the top of the page which will then display a new selection of pro players who are playing these games. The user can then click on a button that appears on the pros card to purchase a session so that they may play together. Users are also able to perform searches and filter results based on their preferences. On the user’s profile they can save their contact link where the pro can find them in order to proceed with the session the user had paid for. The user can manage their purchases via the cart screen in case they wish to remove anything from their cart or update the amount of time they wish to play with the pro. Here they can complete their purchase where the pro will then be informed of the purchase so they may contact the player. Both pros and users have their own profiles showcasing information about themselves where pro profiles will have more information such as games they play and information about themselves. This project will have a Spring backend with the front end being made in flutter which will be deployed as a web application.

1.2 Aims

The “Play with Pro” website is committed to providing a full-quality gaming community and providing a high-quality income platform for young people to earn their first money through their advantages in the gaming field. In addition, the platform also provides a good environment for the gaming community and a community for all gamers to make friends.

1.3 Project Scope

Main functions of the project:

Profile management features for users and professional players

Shopping cart function
User registration and login functions
Show list of professional players
Game filtering and search capabilities

Technical limitations:
The backend uses the Spring framework
The front-end uses the Flutter framework and is deployed as a web app lication

1.4 Project Scenarios
New User Registration:
Scenario description: The user clicked the register button and filled in all the necessary information (email, username, password). After r egistering, he was able to log in to his account immediately.

Shopping cart management:
Scenario description: The user adds all three players to the shopping cart. But he decided to cancel one of the players. So he goes to the cart, removes a player from the cart, and completes checkout.

Browse and select professional players:
Scenario description: The user logs into the website and wants to fin d a professional League of Legends player. He clicked on the "League of Legends" game card and saw a series of professional players.

1.5 Audiences and Primary Users
Normal Players:
Requirements: Be able to choose professional players, buy time with professional players to play with them.
Desire: To have a high-quality gaming experience that will impr ove his gaming skills.

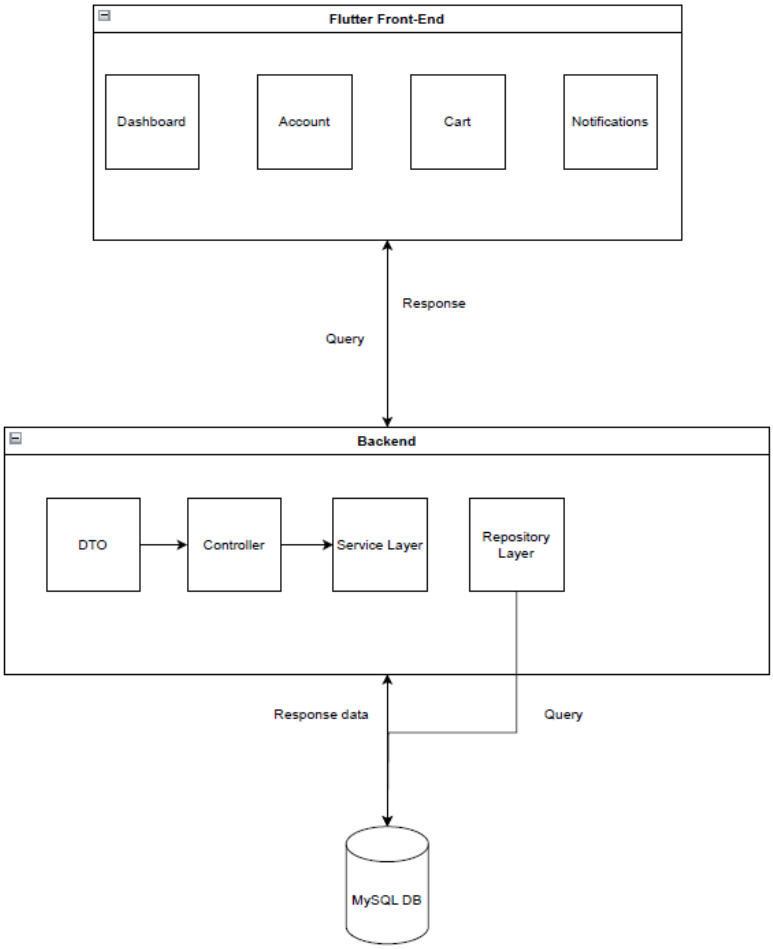
Pros:
Requirements: able to create and manage his own pro profile, as well as connect with regular players and schedule game times wi th them.
Expectations: Make money through the site and interact with a v ariety of players.

2. Project Architecture

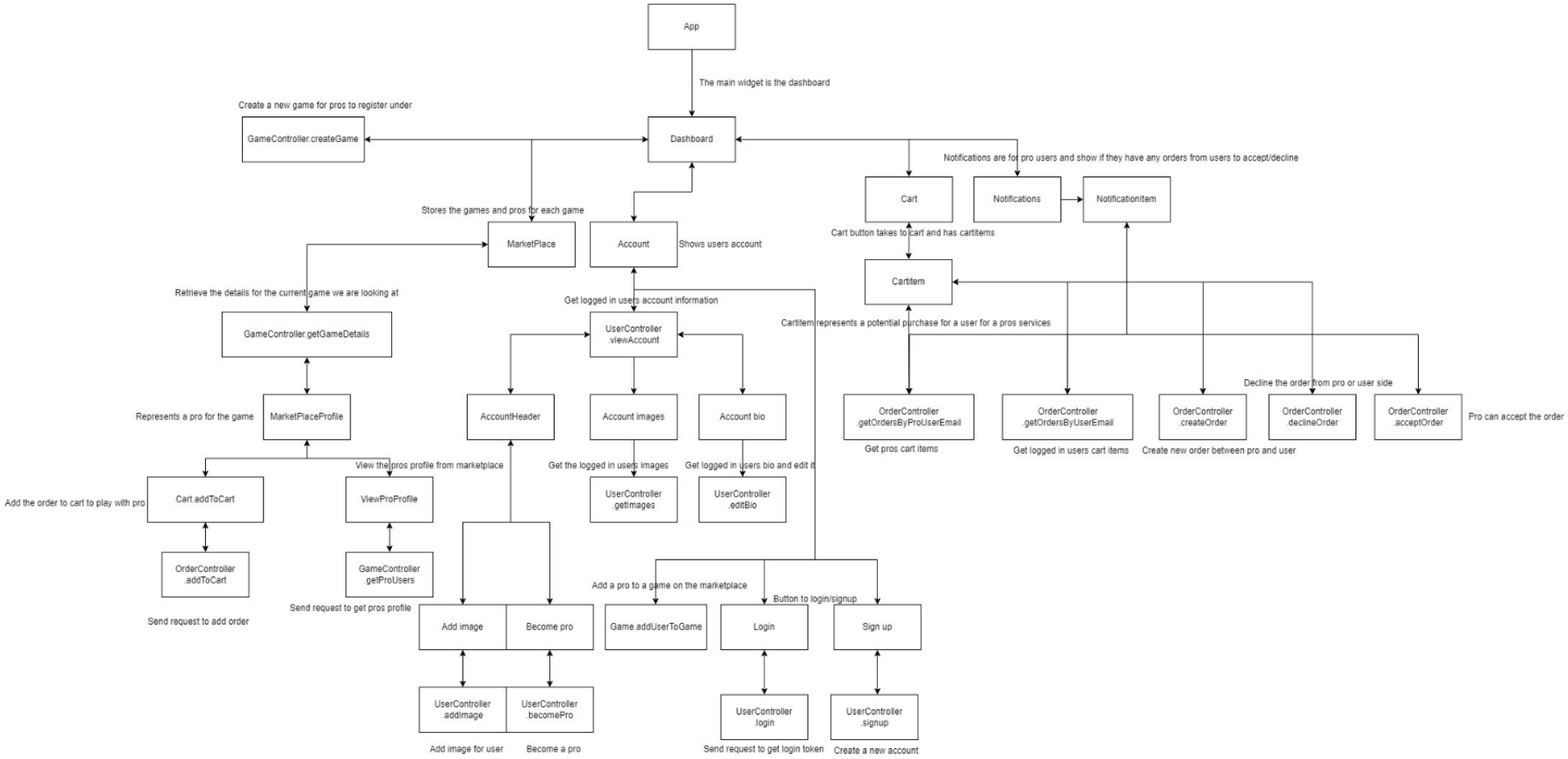
2.1 System Architecture

Simplified diagram

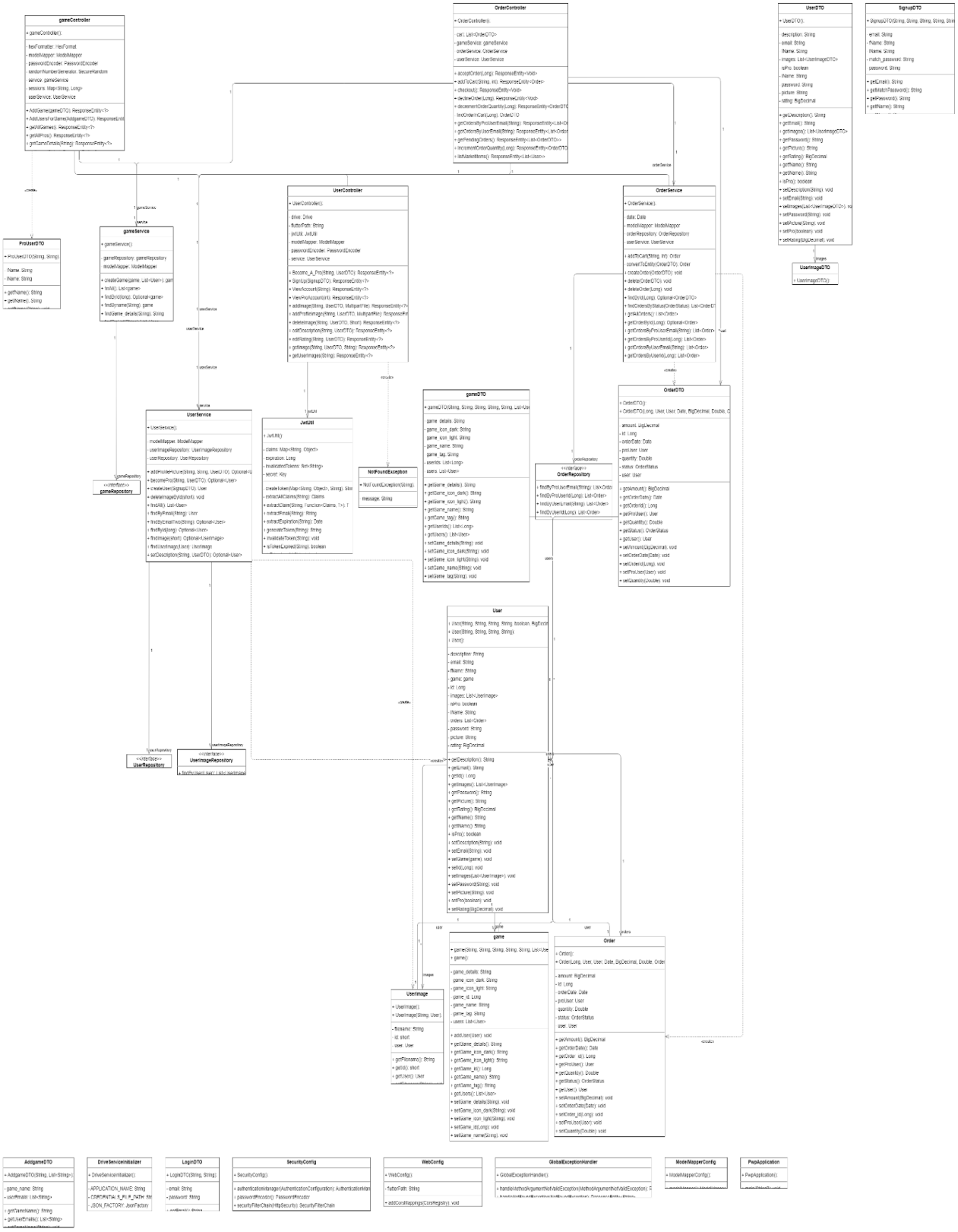
Four major sections are displayed on the front-end page, and there are corresponding functions in each section. When the front-end receives a request from the user, the request is sent to the back-end. The backend controller accepts the request and processes the logic, and sends the data service request to the DTO. DTO will use *the service* to process database data.



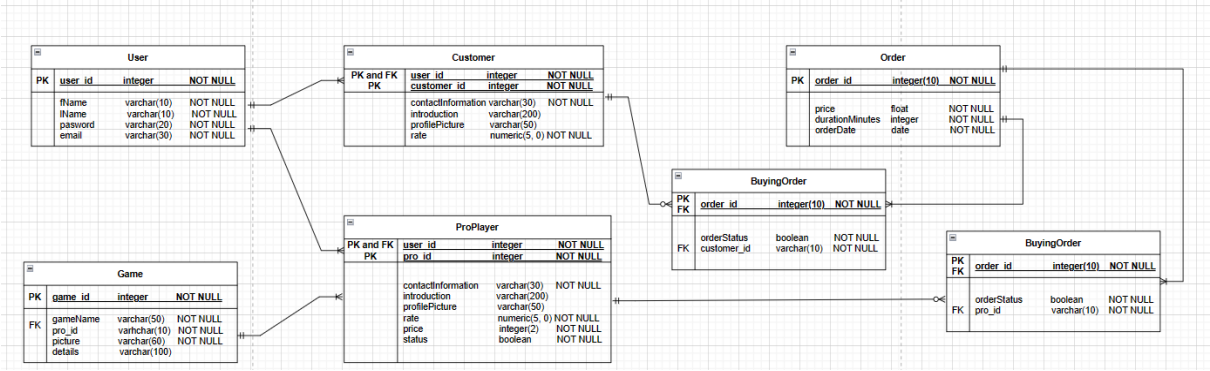
Front end API calls



Backend architecture



2.2 System data model (ERD)



3. Functionalities

3.1 Project functionalities

1. New users can enter their email, username and password to register.
   This feature allows all users to access our platform

2. Users can log in via username and password
   This feature allows all users to access our platform

3. Users can view all basic information about professional players (prices, ratings, introductions, pictures)
   This function allows users to fully understand the information of pro player

4. The user clicks the button to add the professional player to the shopping cart
   This function allows users to place orders for pro players

5. Users can check the order status on the "Orders" page.
   This feature allows users to track orders

6. Users can choose to be verified as a professional player on the profile page
   This feature allows all users to become pro players on the platform, providing an opportunity for young people to make money.

7. Users can upload professional player verification pictures on the profile page

This feature allows users to provide proof of their qualificati ons to become a pro player

8. Users can edit their personal introduction on the profile page
   This feature allows users to edit their own personal homepage

9. Users can delete orders from their shopping cart.
   This feature allows users to adjust the order price and order q uantity

10. The user clicks on the game icon to get a list of professional players
    This function allows users to clearly see the game classificati on of the platform

11. User can view the top 10 games in the website
    This feature allows users to view popular gaming trends

12. User can log out
13. pro players can edit prices
    This function allows all pro players to modify the price accord ing to their own capabilities

14. Users can edit contact information on the profile page
    Users get in touch with the pro player by providing their conta ct information so they can start order service.

15. Users can use this website on different devices, including iPad s, mobile phones, and computers.

3.2 Satisfied non-functional requirements

Security Requirements:

Databases need encryption for sensitive data

User passwords are stored securely using strong encryption technique s.

Maintainability:

The code for the front and back ends remain properly readable and tes table.

Design the platform with a modular architecture, where components are independent and updated or replaced without affecting the entire syst em.

Use version control systems like github to manage changes to the code base.

Scalability:

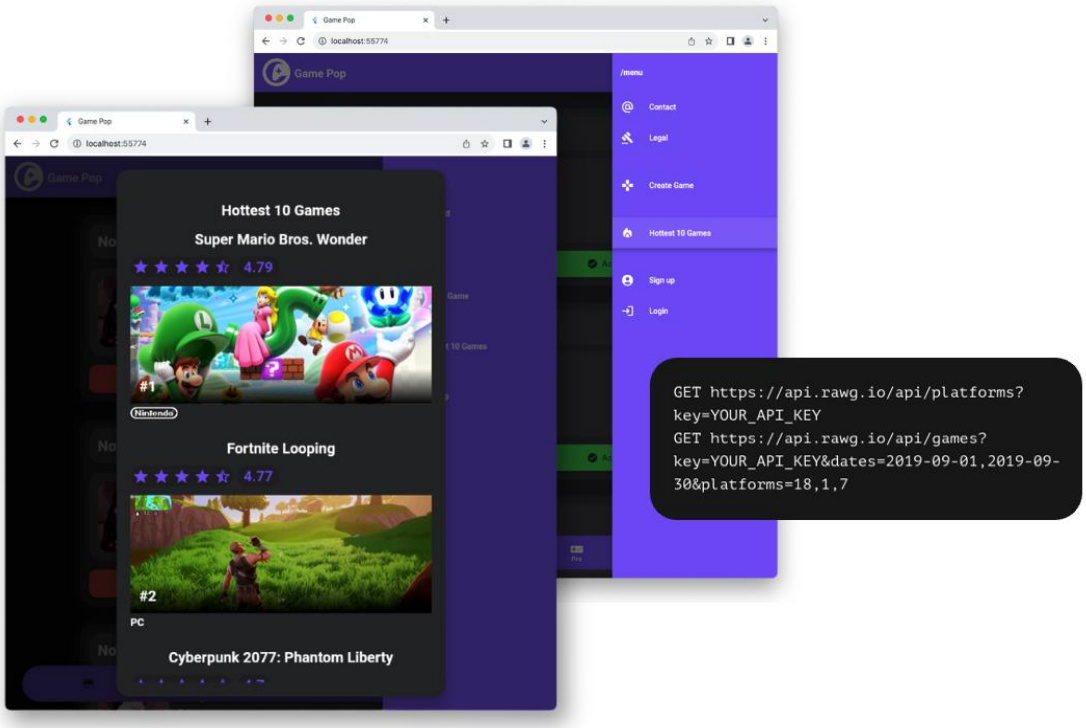
The functional iteration of the website developed gradually and used to meet future market demands. The platform's architecture is designe d to handle increasing numbers of users without significant degradati on in performance.

4. APIs and Spring

4.1 External APIs

The External API was added to enhance user experience with the applic ation, we integrated an external API that shows the user the top 10 g ames of the day. This API is sourced from rawg.io, which is the large st video game database and game discovery service.

`



To use the raw g API the api key is needed which was provided free on sign up to the raw g website. The API key was then stored in the fron t end of the application and when the “Hottest 10 Games” button was clicked by the user a GET request was then sent to the raw g API whic h then returned a JSON object. The JSON object consisted of the top 1 0 ranked games based on the current date. Each game contained its ran king which the application displayed with a 5 star ranking system, a game image url, consoles the game could be played on, and of course t he game title.

```
class GetPopularGames{
  final PopularGamesModelValue callBackPopularGames;
  final IntValue callBackError;
  GetPopularGames({
    required this.callBackPopularGames,
    required this.callBackError,
  }){
    getPopularGames();
  }
  Future<void> getPopularGames() async{
    try{
      final request = http.MultipartRequest(
          AppRequests.get,
          Uri.parse("${AppApi.rawgApiBase}${AppApi.rawgDates2023}${AppApi.rawgOderByRating}&key=${AppApi.rawgApiKey}&page_size=10"));
      request.headers['Content-type'] = 'application/json; charset=UTF-8';

      final streamedResponse = await request.send();
      var response = await http.Response.fromStream(streamedResponse);
      PopularGamesModel popularGamesModel = PopularGamesModel.fromJson(json.decode(response.body));
      if(response.statusCode == 200){
        callBackPopularGames(popularGamesModel);
        return;
      }
      else{
        callBackError(response.statusCode);
        return;
      }
    }
    catch(err){
      if (kDebugMode) {
        print(err);
      }
    }
  }
}
```

4.2 Spring

There are mainly three spring controllers in our backend in terms of user controller, game controller, and order controller. Just like the names of each controller, the user controller is responsible for hand ling users’ behaviors like login, log out, edit profile, etc. The ga me controller is used to display games and all the pro players linked to the games. And the order controller is dedicated to process order s.

1. User Controller

   Functionalities of the user controller:

   SignUp: This function allows a new user to create a new account by providing first name, last name, email address, and passwor d. After successful registration, the user can access all the f eatures of the website.

   LogIn: Registered users can use email address and password to l og into the website. Successful authentication grants them acce ss to various functionalities, including their account profile and additional features.

   LogOut: Users can choose to log out whenever they want to exit the website securely and prevent unauthorized access of their a ccounts.

   ViewAccount: Once users log into the website, they are able to see their profiles shown on the website.

   EditProfile: Users are able to customize their profiles by edit ing description, deleting or adding pictures, and choosing to b ecome a pro player.

```java
@PostMapping("/login")
public ResponseEntity<String> login(@Valid @ModelAttribute LoginDTO data) {
    User user = service.findByEmail(data.getEmail());
    if(user == null)
        return ResponseEntity.status(HttpStatus.NO_CONTENT).body("Entered email is incorrect.\n");

    if (passwordEncoder.matches(data.getPassword(), user.getPassword())) {
        String token = jwtUtil.generateToken(user.getEmail());

        return ResponseEntity.status(HttpStatus.OK).header("Authorization", token).body("User logged in successfully.\n");
    } else {
        return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body("Entered password is incorrect.");
    }
}

no usages
@GetMapping("/logout")
public ResponseEntity<String> logout(@RequestHeader("Authorization") String jwtToken,
                                     @Valid @ModelAttribute UserDTO data){
    User userMapped = modelMapper.map(data, User.class);
    if (jwtUtil.validateToken(jwtToken, userMapped))
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Token is already expired.");

    // Invalidate the token
    jwtUtil.invalidateToken(jwtToken);

    return ResponseEntity.status(HttpStatus.OK).body("User logged out successfully.\n");
}
```

2. Game Controller

   Functionalities of the game controller:

Display Games: the website can show all the pro players under a certain game, the picture and description of a certain game.

Add games: the website is able to add more games for both norma l player and pro player.

```java
@GetMapping("{gameName}/List_Pro_users")
public ResponseEntity<?> return_Pro_users(@PathVariable String gameName) {

    if (service.findProList(gameName) == null)
        return new ResponseEntity<>("no pro_user exist!", HttpStatus.BAD_REQUEST);

    List<User> users = service.findProList(gameName);

    List<ProUserDTO> usersNames = new ArrayList<>();
    for (User user : users) {
        usersNames.add(new ProUserDTO(user.getfName(), user.getlName()));
    }

    return new ResponseEntity<>(usersNames, HttpStatus.OK);
}


no usages
@GetMapping("{gameName}/Details")
public ResponseEntity<?> getGameDetails(@PathVariable String gameName) {
    if (service.findGame_details(gameName) == null) {
        return new ResponseEntity<>("Game not found", HttpStatus.NOT_FOUND);
    }

    String gameDetails = service.findGame_details(gameName);
    game game = service.findByname(gameName);

    return new ResponseEntity<>(game, HttpStatus.OK);
}
```

3. Order Controller
   Functionalities in the order controller:

   Notification: once the order is generated, both pro player and normal player will receive a notification of the new order info rmation.

   Accept or decline: pro player can choose to accept or decline t he new order

Cart: normal users can choose a certain amount of duration to play with the pro player and add the order into the cart. Both pro players and normal players will see orders in their carts separately.

```java
@GetMapping("/notification")
public ResponseEntity<List<OrderDTO>> getPendingOrders() {
    List<OrderDTO> pendingOrders = orderService.findOrdersByStatus(Order.OrderStatus.PENDING);
    return new ResponseEntity<>(pendingOrders, HttpStatus.OK);
}

no usages
@PostMapping("/order/accept/{orderId}")
public ResponseEntity<Void> acceptOrder(@PathVariable Long orderId) {
    Optional<OrderDTO> optionalOrder = orderService.findById(orderId);

    if (!optionalOrder.isPresent()) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    OrderDTO order = optionalOrder.get();
    order.setStatus(Order.OrderStatus.ACCEPTED);
    orderService.save(order);
    return new ResponseEntity<>(HttpStatus.OK);
}
```

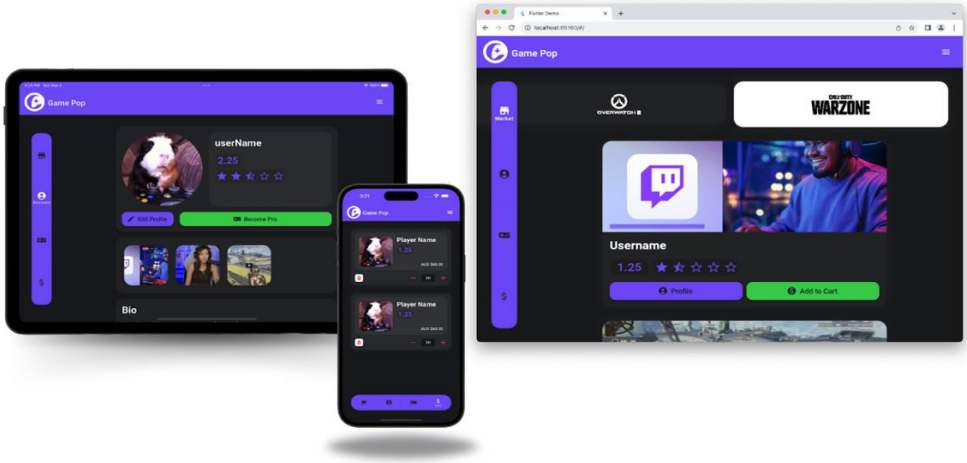
5. Presentation Layer

The presentation layer of this project is an important layer that will determine how users interact with the application itself, a poor user interface no matter how well the application is designed will result in poor performance and less users.

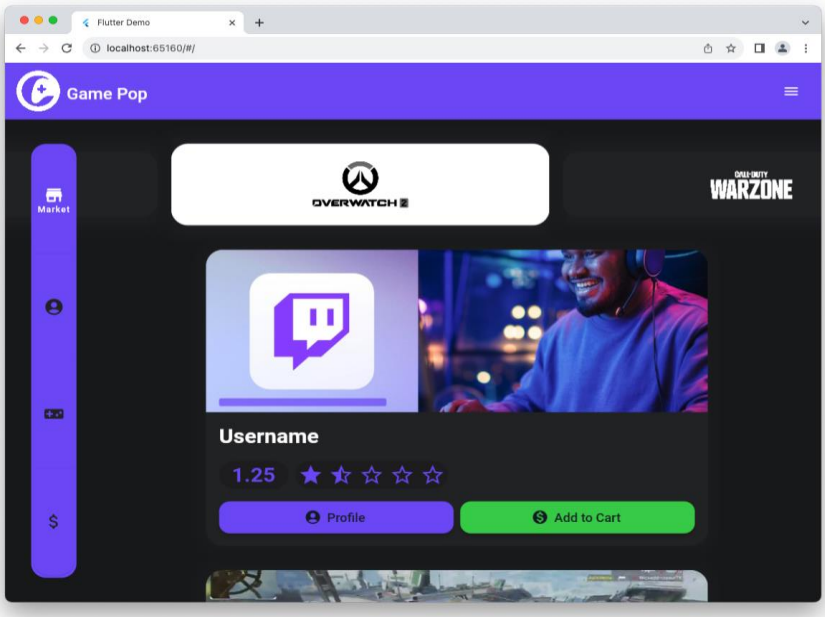
To develop the presentation layer or the UI (User Interface) of this application many frameworks were evaluated, each with their own pros and cons. Factors included things such as performance, development velocity, device compatibility and more.

Our chosen framework was Flutter developed by Google, a non native framework known for its scalability which is also able to deliver a high quality user experience.

There are many reasons why Flutter is a good choice, firstly, Flutters development speed is impressive, which is crucial when time is of the essence. Flutter also offers cross platform compatibility which means only one piece of code needs to be developed which is then supported on a wide range of platforms including, IOS mobile devices, Android mobiles, and desktop web browsers, which is a significant advantage. This means our application will maintain a consistent look and feel across all platforms, providing a seamless user experience.

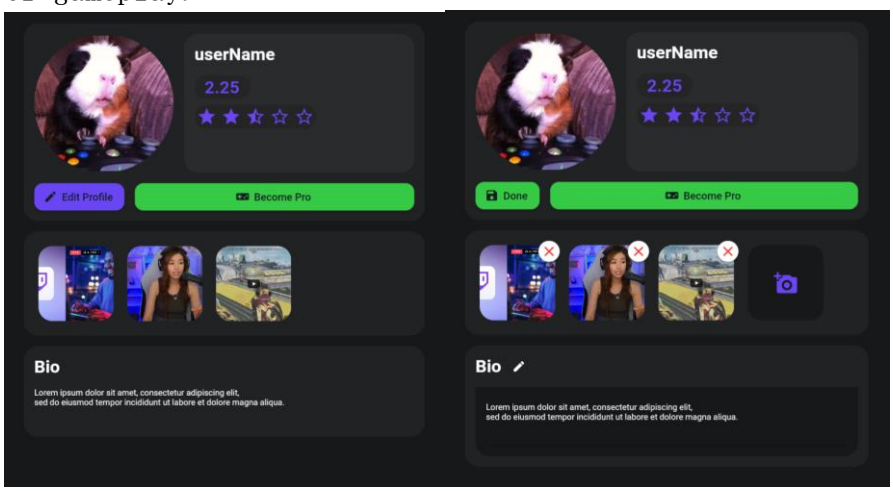


Flutter uses dart which is a client optimized language. Flutter is able to create web applications by using a source to source compiler to javaScript making Flutter web applications available for use in all major browsers including Google, Firefox and Safari. The dart to javascript compiler used is able to create javaScript code that is faster than equivalent code written in just JavaScript.

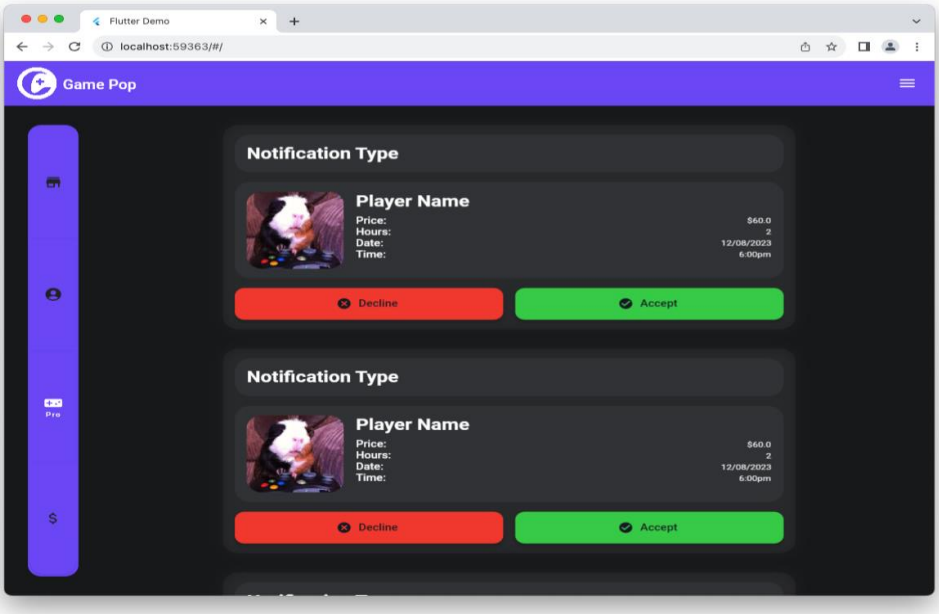


Once the development framework was selected it was time to start creating the front end which would include four main pages available from the navigation bar which include the Market, Account, Notifications and Cart where payments are made. A side menu was also created which included account sign up / in and sign out buttons.

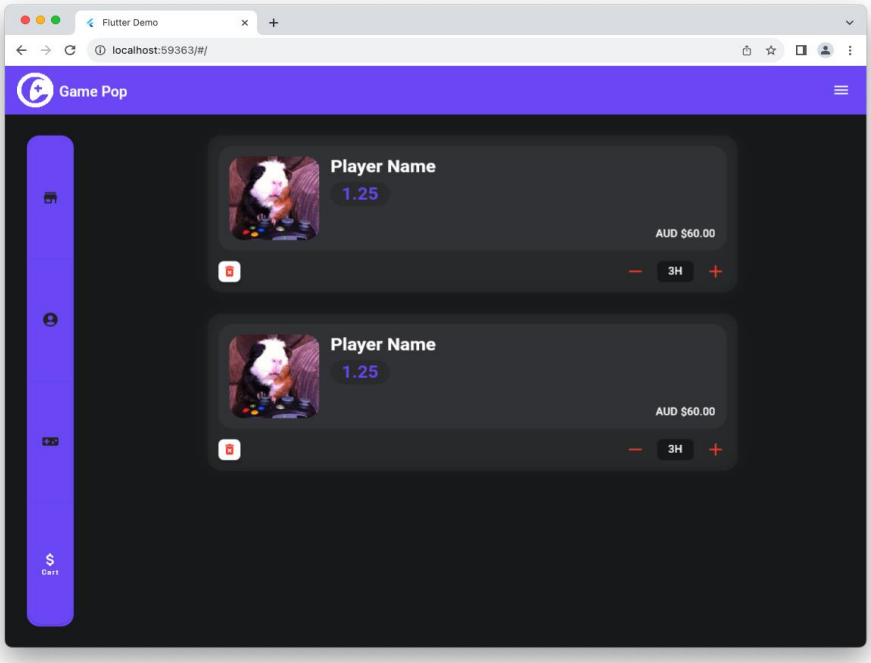
The marketplace page shown above is where users are able to play video games against different pro players, these pro players are shown in a scrollable list which are all pro at the game selected by the user, in this case the Overwatch is selected. From the Marketplace the user is able to view pro players profiles and add pro players to their cart so that the user can purchase hours of gameplay.



The Account is where users can upload their own profile images via an edit profile button available in the user header. Standard users also have the option to become pro players and earn their own income via this application.

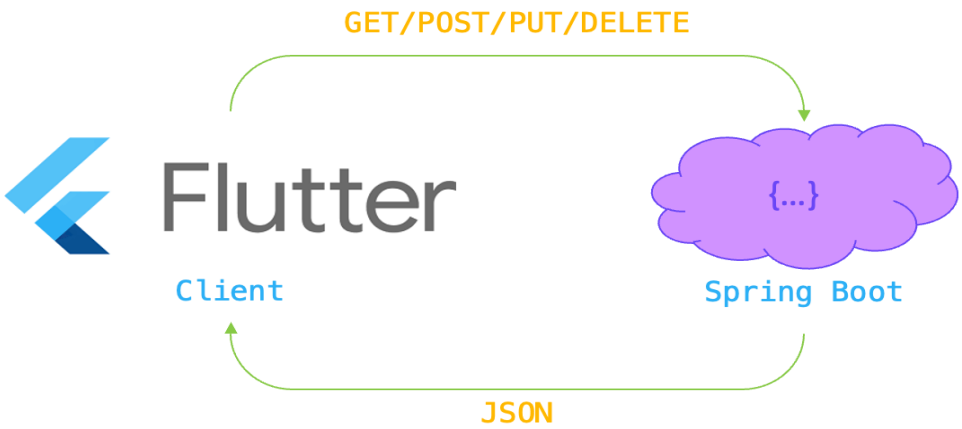


The Notification page, shown above, serves as a way of communication for users who have made a purchase. it allows them to interact with the professional players from whom they have bought playing hours. This feature allows easy coordination between the users and the professional players and helping them to plan their gaming sessions effectively/



The Cart page, as shown above, is the feature that allows users to purchase gaming hours from professional players. Each professional player sets their own hourly rate for gaming sessions. This feature allows users to play video games with professionals, and enhance their gaming experience.

Once the Web application has been fully developed using the Flutter framework the application will then need to communicate with the spring boot backend API.



To do this endpoints will be used. Endpoints are the URL routes where the backend Spring Boot services are accessed. Spring boot provides a set of RESTful API’s that the Flutter front end can interact with. These APIs allow the front end to use CRUD (Create, Read, Update, Delete) operations.

Http requests are used to communicate between the Flutter frontend and the Spring boot backend using the main types of HTTP requests shown in the table below:

GET	Retrieve data from server
POST	Create new data on the server
PUT	Update data on the server
DELETE	Delete data on the server

In our application these HTTP requests are used in various ways. For example, when a user views a pro players profile, a GET request is sent to the Spring boot backend to retrieve the players information.

```
typedef GameModelValue = void Function(GameModel);
class GetGamePro {
  String gameName;
  final GameModelValue callBackGamePro;
  final IntValue callBackError;

  GetGamePro({
    required this.gameName,
    required this.callBackGamePro,
    required this.callBackError,
  }) {
    getUserAccountInfo();
  }
  Future<String> getUserAccountInfo() async {
    try{
      final request = http.MultipartRequest(
          AppRequests.get, Uri.parse("${AppUrls.base}${"pwf/$gameName/Details"}"));
      final streamedResponse = await request.send();
      var response = await http.Response.fromStream(streamedResponse);
      GameModel gameModel = GameModel.fromJson(json.decode(response.body));
      if (response.statusCode == 200) {
        callBackGamePro(gameModel);
        return response.body;
      }
    } catch(error){
      return "Failed";
    }
    return "Failed";
  }
}
```

The HTTP requests are made using the Flutter http package, which allows the application to manage and send HTTP requests. The responses from the requests which if successful have the status code 200. Thes JSON object responses are then converted into a dart model which are then used to update the state of our Flutter application.

6. Testing

During the development process, to ensure software quality and system stability, we tested the project. The goal of testing is to verify the functionality of each method and component and the response of the system. We used unit testing and mock testing

unit test

1. Purpose: Our main goal is to ensure that every method in the project works properly and returns the expected results.
2. Tools: Use the JUnit framework for testing.
3. Implementation:
    a. Test every method in the project.
    b. Use assertions to verify that the output of a method is as expected.
    c. Ensure that each function can be called and used normally.

Mock test

1. Purpose: Since the project uses the Spring framework, we need to verify whether the system can correctly handle HTTP requests.
2. Tools: Use the Spring framework's MockMvc tool for testing.
3. Implementation:
    a. Test the controller in the Spring module.
    b. Use Mock to simulate sending HTTP requests.
    c. Verify that the system can accept HTTP requests and give correct responses.

Test Results

The parts we tested passed the test very well, and no errors or exceptions were found; through the Mock test, the system successfully processed all simulated HTTP requests and returned the expected response.

Generally speaking, after our unit testing and Mock testing, we are confident that the system can work normally in the actual environment.

Element	Class, %	Method, %	Line, %
usyd.elec5619.demo.USER	55% (5/9)	21% (19/87)	12% (30/241)
User	100% (1/1)	50% (12/24)	45% (16/35)
LoginDTO	100% (1/1)	100% (3/3)	100% (5/5)
UserImage	100% (1/1)	14% (1/7)	11% (1/9)
UserController	100% (1/1)	15% (2/13)	5% (6/105)
DriveServiceInitializer	100% (1/1)	50% (1/2)	25% (2/8)
UserRepository	100% (0/0)	100% (0/0)	100% (0/0)
UserImageRepository	100% (0/0)	100% (0/0)	100% (0/0)

Element	Class, %	Method, %	Line, %
usyd.elec5619.demo.ORDER	75% (3/4)	67% (33/49)	43% (43/100)
Order	100% (2/2)	94% (17/18)	96% (27/28)
OrderDTO	100% (1/1)	100% (16/16)	66% (16/24)
OrderRepository	100% (0/0)	100% (0/0)	100% (0/0)