FunChase

INF-2900 Software Engineering

By group 4:

Ali Ehsani-Yeganeh

Adrian Løberg Moen

Antonio Manuel LUQUE MOLINA

Jørgen Ingebrigt Trondsen

Morten Ditlev-Simonsen Jansen

Contents

1 Introduction

This report describes the design and implementation of our game and software application called "FunChase".

2 The Project and the Vision

2.1 The Idea

In the beginning we brainstormed various ideas, like a social credit app, podcast recommendations, type-racer and quiz game. We also tried combining some of the ideas into one single application. The most important factor was that everybody got their ideas listened to and that it would be something that keeps everybody motivated. Eventually we decided to make a game where players join a lobby and complete challenges while in different gamemodes. The game should be inclusive so that people with different backgrounds and ages categories can enjoy our game.

2.1.1 Product Vision

FOR social gatherings and fun experiences with friends and family **WHO** desire a fun and different way to connect and compete in an interactive game. FunChase is a social game **THAT** makes gatherings into enjoyable adventures by offering different game modes and challenges, ensuring laughter and exitement for everybody involved. **UNLIKE** traditional party games or other software applications **OUR PRODUCT** prioritizes inclusivity and simplicity, making sure users of all ages, beliefs and abilities can participate and have a good time. With a user-friendly design and straightforward gameplay, FunChase brings people together, creating genuine moments of joy and togetherness for everyone involved. [1]

2.2 Description of the Project

FunChase is a social game designed for gatherings where players can join lobbies and participate in three different game modes; "Night out", "Family friendly" and "Mountain hike". Players gather in these lobbies by using unique invite codes. After everybody has joined, the game master initiates the game by spinning a wheel that selects a player who must do the given challenge; different challenges give different amount of points. These challenges can be anything from funny to skill-based, which makes every game session exciting. After each challenge players vote on whether it was completed successfully. If the majority votes yes, points are awarded, which adds to the competitive atmosphere and encourages players to attempt the challenges.

Our project was built using Django for the backend and React for the frontend, with CSS handling the styling. We developed and edited our code using Visual Studio Code and hosted the application on the university server for accessibility. The codebase was managed using Github and Git.

2.3 Goals

We knew the given timeframe and surrounding circumstances wouldn't be enough to add all the features that should have been added, considering the potential of our game. Our main goal was to create a fun and social game that is user-friendly, easy to play and mobile based. The game itself should be the focus and not include any design or feature that is disturbing towards the game-play. For the codebase itself we focused on having a modular design and high quality documentation so that working during the time of the project and in the future would be as hassle-free as possible.

3 The Development Process

3.1 Methodologies - Agile Development

Agile software development is an iterative approach that focuses on breaking the project into different phases and delivering incremental releases of working software, which allows for continuous improvement and adaptation to changing requirements. This method helps in estimating the amount of work that can be done in each development phase, which is important when needed to estimate a release date for a product. In our development process for FunChase we have used the "Scrum framework" which is a specific type of agile development that focuses on short, time-boxed sprints. Each sprint lasts for three weeks and starts with a meeting where we go through the tasks from our backlog and add more if we need. Each sprint we also choose a new scrum master whos responsibility is to ensure shared responsibility and project growth. At the end of each sprint we held retrospective meetings to evaluate or performance, identify areas for improvement and make adjustments for the upcoming sprint.

Specially in the start phase and the last sprints we used extreme programming (XP), because it targets speed and simplicity without sacrificing great software quality. We had to do some of these heavier sessions when we assigned more development in some sprints, than others.

Although refactoring was a continuous process throughout the project when improvements were found it was particularly emphasized during the last sprint. Our attention shifted towards enhancing the code readability and structure. In the beginning our priority was to develop a MVP and ensure functionality. For instance our gamelobby component consisted of multiple features like the leaderboard, end-game screen and and roulette wheel, which we later divided into distinct components to enhance code clarity.

Pair programming is another agile software development technique where two developers share a single workstation and work together to develop a feature. We have benefited from this style a lot and have had mostly good experiences. It leads to fewer conflicts in development, because people are writing and testing features simultaneously. It is also easier to obtain a common understanding of what design of the project should be, which leads to less misunderstandings. We employed pair programming for most of our development for this project. We had group meetings at least two times a week, where we set up a live share(using VScode) for people to connect and spent that day developing the project together.

It was also important to realize we all have our own strength and specific interests so the skills of each member should be recognized and exploited. While everybody got to work more or less in both frontend and backend we discussed between us and which parts we wanted to have responsibility with, whether it was making frontend components, design, setup database design or backend communication.

3.2 Test-Driven Development

Test driven development (TDD) is an approach where tests are written before writing the actual code. Its a way of showing the presence of errors in a program, not that there are no remaining faults. In the development of FunChase, testing was done mostly on a development level, which involved running the code after adding each feature and stress-test against different edge cases. We also conducted smaller levels of user-testing by engaging friends and classmates to play the game and provide feedback.

An important acknowledgement is that it was difficult to follow this principle accurately. TDD was a new development method for all of us and we weren't sure what the tests would accomplish. Even though the features were clear it was not always obvious on the direction we were going with the implementation. The tests were made towards the end of the project when we were already testing the code manually. The important part is the takeaway which is the purpose of TDD and writing the tests themselves, as well as the fact that we now understand what test driven development is and have made an attempt at it. [2]

3.3 Backlog Planning

The backlog was organized using Notion which is a tool that helps keep track of tasks and ideas. In the first week we had several brainstorming sessions trying to add lots of features and tasks for our game. Then we organized them in different categories such as; "Must have", "Should have", "Nice to have" and "Could have". The backlog was necessary in creating a clear path for reaching our goal, but still during the development process we continuously added new tasks or split existing tasks into sub-tasks as the workload was more than expected. We also changed the priority of tasks and removed based on new ideas and the current state of the game. To organize tasks effectively we used the personas and user stories we created earlier. These helped us understand our users better and map out different features and tasks. For example one of our user stories was a player who loved competition so the idea of a leaderboard feature came to life.

3.4 Github Workflow

For this project we shared a GitHub repository where all our code was stored and managed collaboratively. Considering we spent most of the time using live share coding when developing, we didn't encounter too many merge conflicts. We also tried using branches when starting on new features separately during a work session, if the case of bad merges we could trace back to a branch. Before pushing any code we usually had review processes to try and identify mistakes before they were pushed to Github. This helped maintain code consistency, version control and traceability throughout the development process. Looking back we learned that improving our commit messages would have been beneficial. Clear commit messages would have helped in understanding the changes made and correcting mistakes efficiently.

3.5 Retrospective meetings

The retrospective meetings we had with the TA and within the group served as valuable checkpoints to evaluate whether we had achieved our sprint goals. It was an important part of the agile development process, because in these meetings we discussed what went well, what could be improved and additional insights for future sprints. Even though we were collaborating during the sprints, the retrospective meetings allowed us to identify our strengths and weaknesses over a time period and not only in the moment.

4 Design

4.1 Frontend structure

The frontend is one of the two major directories in the codebase, and here we use react as a framework. Though this directory share mostly the same structure as the basic react framework, we have made some additional folders to support our design.

- **Assets:** This folder contains mostly images used for styling our frontend components, some sound effects and fonts.
- **Components:** This folder contains all the javascript files used for actually creating the behavior of the frontend, in which the user will interact with. Most of these files are URL pages such as: Gamelobby, CreateGame, Profile, etc. Some of the files are also just functions displayed in other URL pages, such as: Roulettewheel and question container are both used by Gamme lobby.
- **Services:** Here we have made files that are responsible for making requests to the backend. These are called by the different components in the frontend. This cleans the code up a bit, and gathers all the requests to the backend on a single point. The main purpose of this is to make deployment easier, as we want a base API url for all these requests.
- **Styles:** The styles folder contains all the css files we have made for the frontend components.
- **Utils** The utils folder was planned to contain more utility files, but now only contains one: authUtils. This folder is supposed to contain functions that are often repeated in order to shorten the and increase clarity of code.

4.1.1 UI-design and color

The UI is designed to be relaxing to look at and have clear and intuitive navigation, which is one of our main goals for this project. For this it is important to keep distractions on a minimum and only display necessary functionality. There are no nonessential buttons or text. Upon entering the application the user is asked to create a user, log in and then join a game by code or create a game. User can go to the "about us” for precise instructions and after logging in have the possibility to check out their profile. For the design we wanted to follow a theme and our main color is purple. Purple is pleasing to look at, unique compared to similar applications and therefore helps in creating a memorable brand identity.

4.1.2 Flowchart

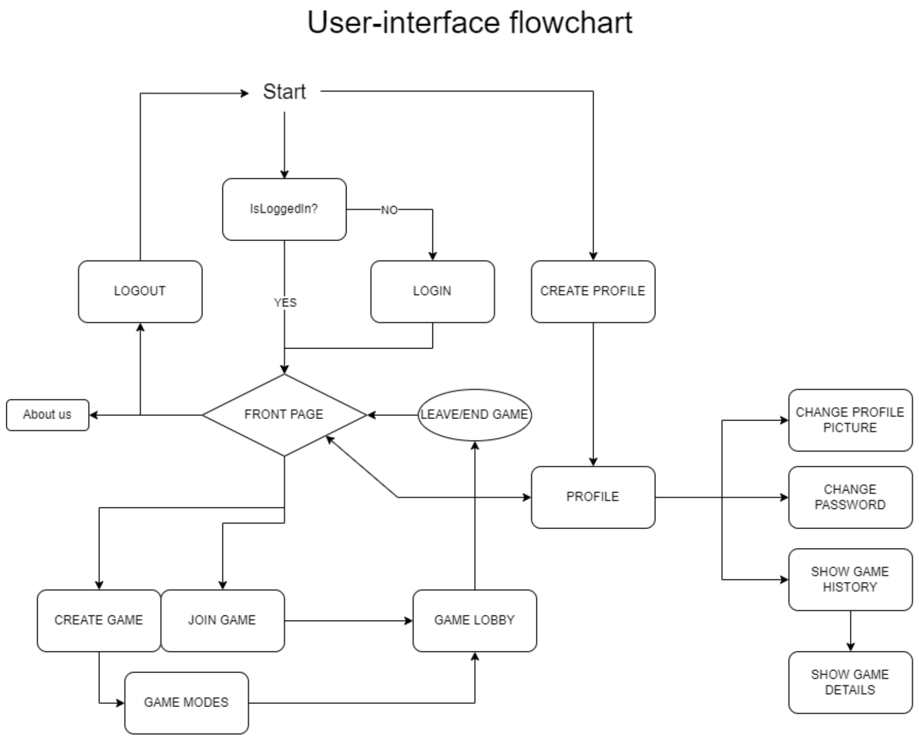


Figure 1: User-interface flowchart

4.2 Backend Structure

The back-end of the Funchase game is built on a structured relational database that supports the game’s interactive features and complex mechanics. The database used is SQLite3 which is a lightweight and simple database, well suited for small projects and applications. As mentioned earlier we are also using Django, which means that the backend is formatted as such with its utilities aswell.

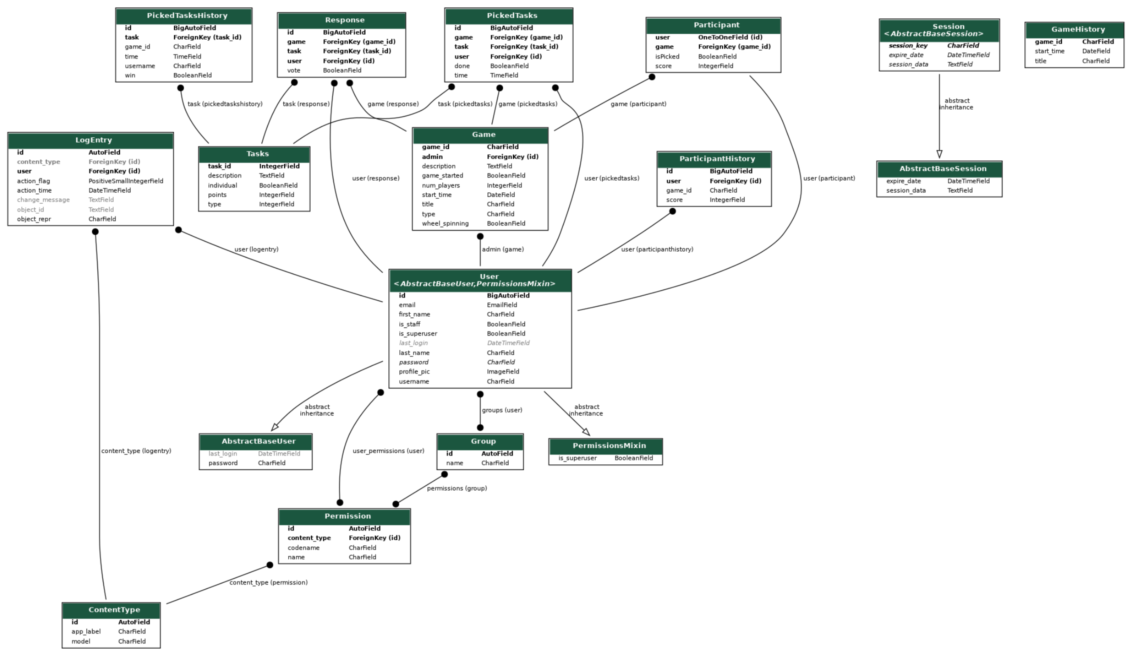


Figure 2: Database Schema of FUNchase

4.2.1 Database schemas

Our database is carefully designed to ensure all tables are interconnected through foreign keys, which help maintain data accuracy and organization. The database is composed of several essential tables, each fulfilling a specific function:

- **User:** This table is used for managing users. It stores important information like email addresses, usernames, and securely hashed passwords, along with personal details such as profile pictures.
- **Game:** This table tracks all ongoing instances of active games. The table holds information such as gameID, title, type and other metadata which is needed by other tables and used in game logic.
- **Tasks:** This contains all the challenges within the games, detailing what each task is about and how many points it's worth, it also holds a "type" attribute which signifies what kind of task it is(Used when querying different gameModes)
- **Participant:** Here we log the details about each user participating in an ongoing game. This is used when handling their score, and picking players for a challenge.
- **PickedTasks:** This table keeps track of which tasks has been done/ongoing and who has been given the task. We also need this table in order to not pick the same tasks over again, essentially ensuring that each task fetched is new.
- **Response:** This table keeps track of the different votes each player has given to the specific task during that game.
- **History tables:** The history tables includes: GameHistory, pickedTasksHistory and participantHistory. These three tables are needed when we want to display a proper history for each game, such as the tasks that happened that game and what players participated(scoreboard).

4.2.2 Backend files

The backend directory contains many files with the django structure, but we will be looking at the ones of most relevance for the implementation of this project.

- **models.py**: This file contains the implementation of our database schemas. Here we use the django module to create the different schemas, and define it's attributes.
- **urls.py**: This file is split into several files such as urls_auth, urls_game, urls_profile etc. Reason for this is because it makes the url routing more clear. These files contain the path to each of the function calls in the views.py file.
- **views.py**: This file contains all the functions that perform necessary actions such as authentication, generation of token and most importantly querying the database. The functions in this file is called by accepting axios requests from frontend, and these requests are subsequently routed through the urls file and to the specific functions.
- **consumers.py**: This file contains the implementation of our websocket. The websocket is setup to receieve incoming connection from all players within the instance of a game. Once connection is established, all players within that stream will be receiving updates from that backend. The purpose of the websocket is to give live updates to all the players within the given game. This is needed give a synchronous experience to the players in the given game.
- **tasks.py**: This file is a small function that sets the wheel spinning to false in the Game table, and notifies the websocket that it should display the next task to players. This function is called using the celery module in websocket, that schedules this function to run when the wheel has stopped spinning
- **celery.py**: The celery file is used to configure and integrate celery into the django project. We use celery to schedule tasks, and specifically the end_wheel_spin in the tasks.py file. [3]
- **settings.py**: The settings file holds all the settings for our django application, such as activated apps(celery/redis-server), database configuration(sqlite3), secretKey, session cookie age etc. Essentially metadata for our server.

In summary, the back-end structure of Funchase is crafted to be robust and adaptable, supporting the complex needs of managing user interactions and game play effectively, ensuring that players have a smooth and engaging experience.

4.3 Flowchart

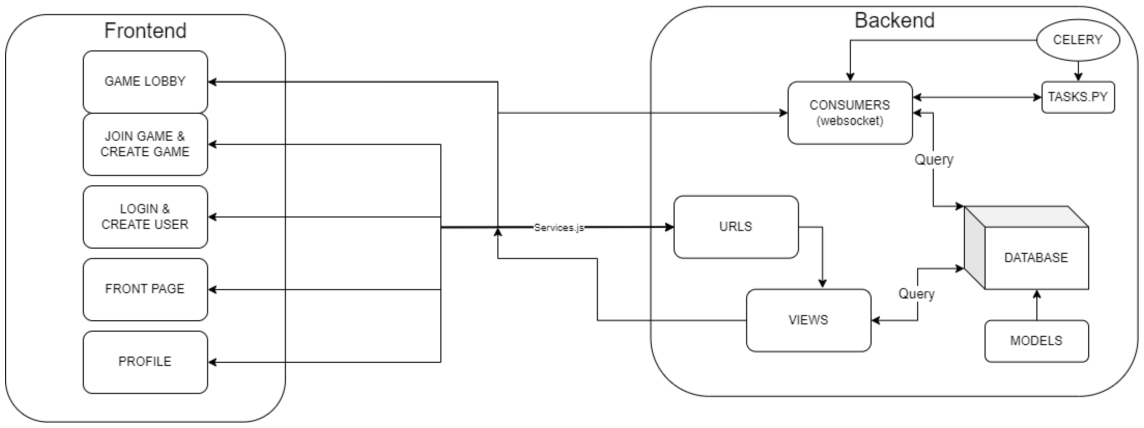


Figure 3: Flowchart of communication between frontend and backend

4.4 Security

The application integrates several advanced security mechanisms tailored to mitigate specific types of cyber-security threats and vulnerabilities.

4.4.1 Authentication and Session Management

- **JSON Web Tokens (JWT):** FUNchase employs JWT for authenticating users, which helps manage user sessions securely. JWTs are created on the server-side and are configured as HTTP-only cookies to prevent them from being accessed by JavaScript running in the browser. This design choice protects against:
  - **Cross-Site Scripting (XSS) attacks:** By preventing the token from being accessed by client-side scripts, the risk of malicious scripts capturing the token and using it to impersonate the user is greatly reduced.
- **Session IDs:** The application utilizes Django's built-in session management, which stores session IDs in secure, HTTP-only cookies. This approach enhances security by:
  - **Minimizing the exposure of session identifiers:** Keeping the session ID in HTTP-only cookies prevents client-side scripts from accessing them, thus protecting against session hijacking through XSS.

4.4.2 Cross-Origin Resource Sharing and Cross-Site Request Forgery Protection

- **CORS Settings:** FUNchase configures CORS to define which cross-origin requests are permissible. This helps prevent:
  - **Data leaks:** CORS restricts which domains can interact with your application, preventing the browser from sending sensitive information to any unauthorized origins.
- **CSRF Protection:** Using both JWT and Django's built-in CSRF protection mechanisms provides robust defense against CSRF attacks, where an attacker tricks a user into executing unwanted actions on a web application where they are authenticated. These mechanisms ensure that:
  - **Every state-changing request must be accompanied by a valid CSRF token or JWT,** which must be explicitly provided by the client, thus ensuring that only intended user interactions are executed.

4.4.3 Password Handling and DOS attacks

- **Secure Password Storage:** Passwords are stored using strong hashing algorithms provided by Django's authentication framework (`django.contrib.auth.authenticate`), ensuring:
  - **Protection against brute force and rainbow table attacks:** Hashing passwords converts them into a fixed-size string of characters that is practically irreversible. This means even if the password hash data is exposed, it cannot be easily converted back into the original password.
- **DOS attacks:** Using django's built in ratelimiter we are also able to prevent DOS attacks. This is done by limiting the rate at which a client can send requests to the backend.

In summary, Funchase's security framework employs a combination of state-of-the-art authentication protocols, session management techniques, and proactive defenses against prevalent web vulnerabilities. Although we are using session timeout as a preventative measure for session hijacking, we would prefer to use traffic encryption such as https/TLS. This is because we don't actually want to timeout our players too often, and as https/TLS is a strong security measure for any application.

5 Implementation

5.1 User authentication

5.1.1 Create user

Beginning in the frontend, for creating a user there is a simple form where users can input their information such as first name, last name, username, email and password. When the user submits the form, an Axios

POST request is triggered which sends the user data to the backend. The backend processes the request, validates the data and a new user instance is created and stored in the database. Finally a JSON response is sent back to the frontend to indicate the success or failure of the user creation process. There also exists error handling for already existing username and email.

5.1.2 Login

On the frontend side the login component takes username/email and password input from the user. When the user submits this information an Axios POST request is sent to the backend with the entered credentials. If the login attempt is successful the user is considered logged in and the frontend updates the state accordingly, setting the user as logged in. A successful login navigates the user to the home page giving the option to create or join a game.

5.1.3 Backend

On the backend side the username/email and password is recieved from the request body and authenticated with Djangos built-in authentication system. A successful request generates a "JSON Web Token" for the user. This token is returned as part of the response data. The JWT and a HTTP cookie is then set, which serves as a digital signature. This token holds information about the users identity and data needed for authentication. This way the backend can verify the users identity and authenticate their actions without requiring the user to repeatedly log in for each request. When logging out we simply enter the backend and call djangos built in logout function to remove the users auth ID.

5.2 Frontpage

After logging in/logged in users are navigated to frontpage, which welcomes the user and provides the option to either join or create a game. It provides an input field for entering the game code when the join option is selected and a title when game is created. If join game is initiated, we will send the request to backend and look for any existing games with that code. If we are creating game, then we will be routed to the create-game url with the title we made.

When joining the game, the backend will be called to look for the game in the database. If found it will try and add this user to the participant table, and if successful the player should be navigated to the game lobby.

5.3 Create game

The create game component displays the game modes next to each other together with a descriptive picture and explanation. When selecting a game, a random game ID is generated and a POST request is made to the backend API. The request contains the generated game ID, user ID and game description.

When the request arrives at backend, do some checks to evalute the request data and user making the requests. If no errors occur, then we will use the data received from front end to add this game to the database, aswell as adding the creator of the game as a participant to this game.

5.4 Navbar

The navbar provides simple navigation between the necessary pages. When logged out, users can either log in, create user or go to the about us page for an introduction to the game. When logged in users can access their profiles as well as logging out.

5.5 Profile

The frontend UI is very simple showing the name, lastname, username and email. Also there is a possibility to change password as well as editing of profile picture. When the profile page is acceseed it sends requests to the backend API endpoints to fetch user data and profile pictures. The frontend listens for user interactions such as editing profile fields, uploading profile pictures, deleting profile pictures and selecting profile pictures from the gallery. During these interactions additional API requests are made to the backend.

The backend ensures that only authenticated users can perform profile-related actions by checking for authentication tokens or session cookies sent with the requests. The backend interacts with the database to persist user data, including profile information and profile pictures. Images are validated to be in the right format, sent to the appropriate location and all images belonging to the user are returned as an list of URLs. The profile picture is updated based on the chosen URL and images are also when chosen to.

5.5.1 Game history

In profile we also have the option of browsing the history of games we have played. The game history information is fetched the same time as we fetch the profile information from the database. The games are then displayed as buttons, and if pressed we will send a request to backend to gather further information about that game. This information includes: all players who participated in the game and the different tasks that occurred during that game as well.

5.6 Gamelobby

The game lobby is a core component and where the game takes place. It includes other components like the leaderboard, question container and the spinning wheel. It starts by initializing various states about game related information like the list of players, game ID, task details, game status and more. Then the web socket connection is established with the server to allow live updates and communication. It fetches game-related data from the backend using Axios requests. This includes fetching the list of participants, current game state and active tasks. It renders different components based on the game state. For example the leaderboard component is rendered to display player scores when the game is ongoing or that the question container always displays the current task. Buttons provide the possibility for actions like ending the game, leaving the game, starting the game, fetching the next task and voting. When an admin ends the game or a player leave they are redirected to an endgame screen that displays ”Game over!” and a button that navigates to frontpage.

5.6.1 Websocket(backend)

Websockets are crucial for maintaining responsive, real-time updates for players in the game lobby, ensuring all players have the correct data at all times. The backend server only facilitates one websocket, using djangos channels extension. Since JWT tokens are added as httponly upon login, and is the only thing required for authentication, there is no need to append data any to the URL. Connecting to websocket via its url, the server relies only on user authentication to connect the player to the specific game group. General information transfer is done via messages, and msg_type which tells the websocket and/or players exactly what must be done. [4]

5.6.2 Question container

The question container component is responsible for displaying challenges. It contains the task it self, the player assigned to it, amount of points and voting buttons. If waiting for a spin the container will display a message indicating that the wheel is currently selecting a player. The player that is not picked will be provided with buttons. The buttons allow the user to vote ”Yes”, ”No”, or ”Skip” on whether the picked player successfully completed the task. The question data is fetched by pressing the start game/next challenge button(only visible to admins). This will then send a message to the backends database using the websocket. This way every player connected to the channel, will receive the same task in real time from the socket. The same goes for the voting buttons, each player sends their vote to the backend where there will be performed voting logic(majority vote wins) and votes are then displayed to each player in the game concurrently.

5.6.3 Leaderboard

The Leaderboard component displays a list of players and their scores in a game. It includes a button to show or hide the leaderboard. When the button is clicked, it changes the visibility of the leaderboard. The player list is mapped over to display each players username and score. The leaderboard also uses the websocket to give live updates of points for each challenge completed, aswell as updating the amount of players displayed on the leaderboard(leaving/joining).

5.6.4 Roulettewheel

The RouletteWheel component implements a spinning wheel where participants names or profile pictures are displayed on segments of the wheel. When the wheel spins, it randomly selects a participant, and the wheel stops after a set duration. There are several mechanics to ensure smooth operation, such as checking if wheel is spinning when a player joins mid spin. If players refresh or joins mid spin, they will bypass this spinning effect. To fix this we use celery to schedule a task in the websocket. This way players will not have displayed the next task until a timer of twelve seconds has passed, and then the websocket will send out a message to all players.

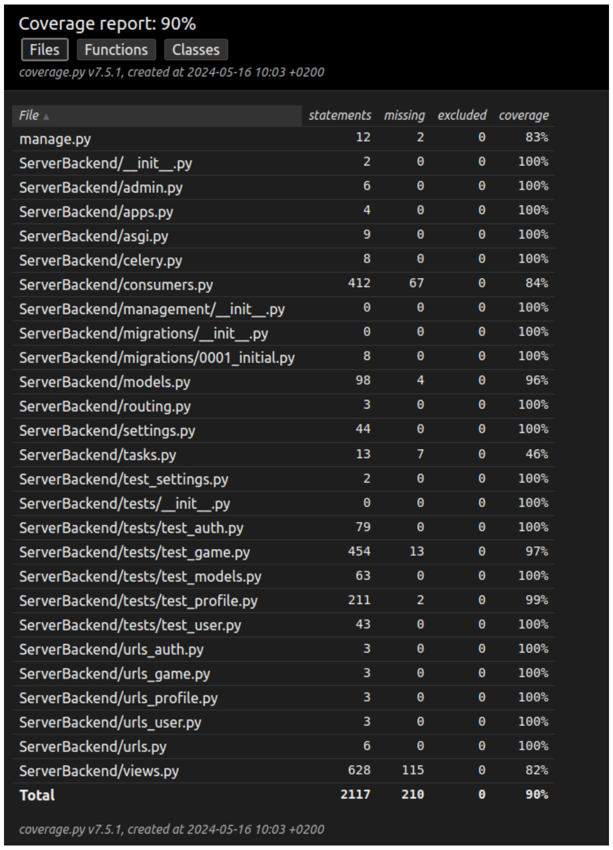
5.6.5 End/Leave Game

Depending on whether you are the creator of the game or you are a joining player, you will have displayed different kinds of buttons. As an admin you will have the option of ending the game, while as a player you will have the leave game button. These buttons perform different operations in the backend, where end game will delete all information of the instance of the current game, and store the relevant information in the history tables. Leave game however, only removes the player from the active participant table and add to the history table. When ending the game, we are provided with an endgame screen which displays the scoreboard of the game, and a button to navgite to front page. Should also mention that when joining a game we have left, we will fetch the player data from the history table to retain the points in the game.

5.7 Testing

Despite not employing test driven development, tests have still been implemented. The tests are mainly done to prevent and test for unauthorized actions and user errors, such as accessing other players files or deleting game when not admin. Structurally they are split into five different files, each testing their own 'module' or area; authentication, user, profile, models, and game.

The tests are all done with djangos unittests, and we have used py coverage to benchmark code coverage. As the figure below 4 below shows, 90% of the backend code base is tested. A coverage of 100% is nearly impossible, mostly due to the fact that not all code can run when using tests, as they are safeguarded at a higher level, or they simply do not happen. [5]


Coverage report: 90%

coverage.py v7.5.1, created at 2024-05-16 10:03 +0200

File ▲	statements	missing	excluded	coverage
manage.py	12	2	0	83%
ServerBackend/__init__.py	2	0	0	100%
ServerBackend/admin.py	6	0	0	100%
ServerBackend/apps.py	4	0	0	100%
ServerBackend/asgi.py	9	0	0	100%
ServerBackend/celery.py	8	0	0	100%
ServerBackend/consumers.py	412	67	0	84%
ServerBackend/management/__init__.py	0	0	0	100%
ServerBackend/migrations/__init__.py	0	0	0	100%
ServerBackend/migrations/0001_initial.py	8	0	0	100%
ServerBackend/models.py	98	4	0	96%
ServerBackend/routing.py	3	0	0	100%
ServerBackend/settings.py	44	0	0	100%
ServerBackend/tasks.py	13	7	0	46%
ServerBackend/test_settings.py	2	0	0	100%
ServerBackend/tests/__init__.py	0	0	0	100%
ServerBackend/tests/test_auth.py	79	0	0	100%
ServerBackend/tests/test_game.py	454	13	0	97%
ServerBackend/tests/test_models.py	63	0	0	100%
ServerBackend/tests/test_profile.py	211	2	0	99%
ServerBackend/tests/test_user.py	43	0	0	100%
ServerBackend/urls_auth.py	3	0	0	100%
ServerBackend/urls_game.py	3	0	0	100%
ServerBackend/urls_profile.py	3	0	0	100%
ServerBackend/urls_user.py	3	0	0	100%
ServerBackend/urls.py	6	0	0	100%
ServerBackend/views.py	628	115	0	82%
Total	**2117**	**210**	**0**	**90%**

coverage.py v7.5.1, created at 2024-05-16 10:03 +0200

Figure 4: Coverage of unittest

5.8 Non-functional attributes

Right now the application isn't hosted on an external server so its not available for other users and for that reason only been tested locally. [6]

Responsiveness: The application uses websockets during game play to prompt challenges, update the leaderboard and facilitate other user actions in a reasonable time.

Security: The passwords are not hashed or encrypted in the frontend, however they are stored as hashed values. Better security measures would involve encrypting them in transit by switching to HTTPS instead of HTTP.

Reliability: Means that the system futures performs as expected by users and developers. This hasn't been thoroughly tested due to the short lifespan of the application and the application has not been deployed for users to test.

Availability: Only for developers at this time, but locally the game perform on reasonable time.

Usability: The design prioritizes user-friendliness and provides only necessary information. When entering the website, users are immediately prompted to log in with simple graphics. Simple error handling is added to guide the user on the next step in case of failure.

Maintainability: is important for long-term viability and scalability. The system has good code readability, documentation and modular design. The game has a lot of potential and could simply be updated and repaired over time.

Resilience: The application lacks extensive backup systems in case of failure. If a single component fails, depending on which it is, the game could still be playable. For example the game could still be played without a working leaderboard, but if the game lobby component isn't working it would be impossible to play.

6 End Product

6.1 Log in

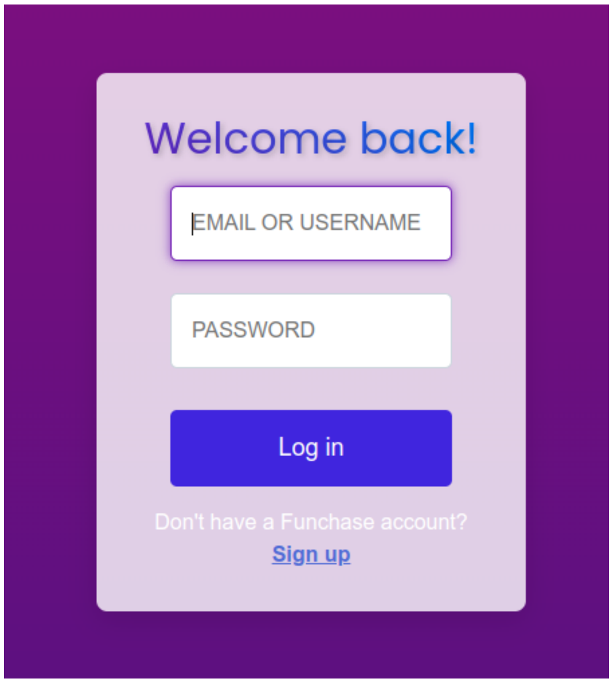
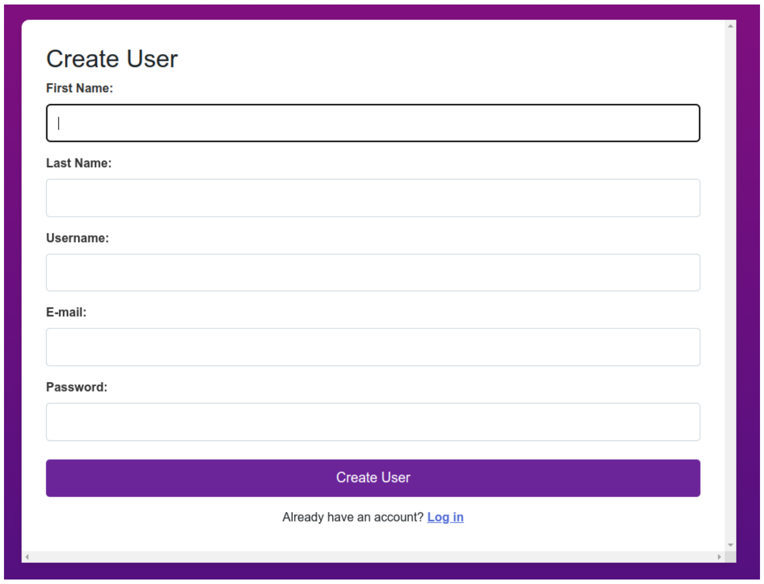


Figure 5: Login Page

6.2 Create user


Create User

First Name:

Last Name:

Username:

E-mail:

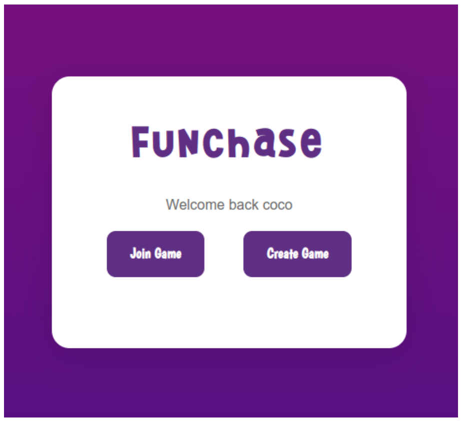
Password:

Create User

Already have an account? Log in

Figure 6: Create User Page

6.3 Index Site


FUNChase

Welcome back coco

Figure 7: Index Page

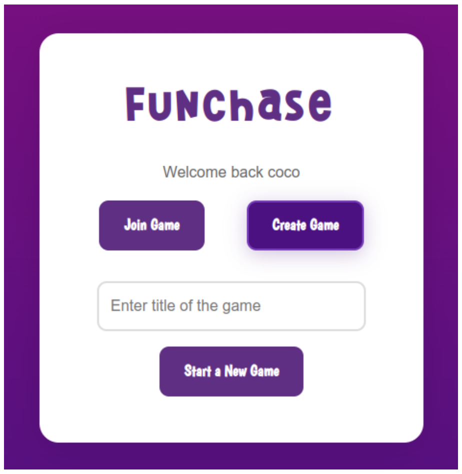


Figure 8: Index Page after pressing Create Game button

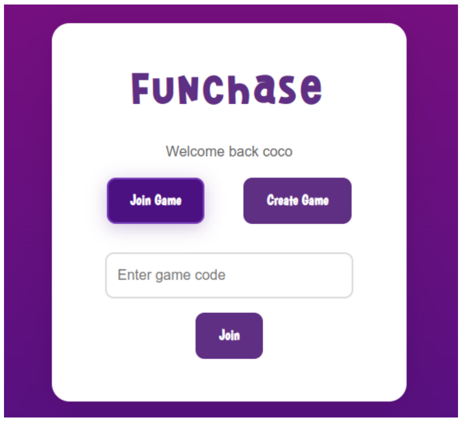


Figure 9: Index Page after pressing Join Game button

6.4 Create Game Site

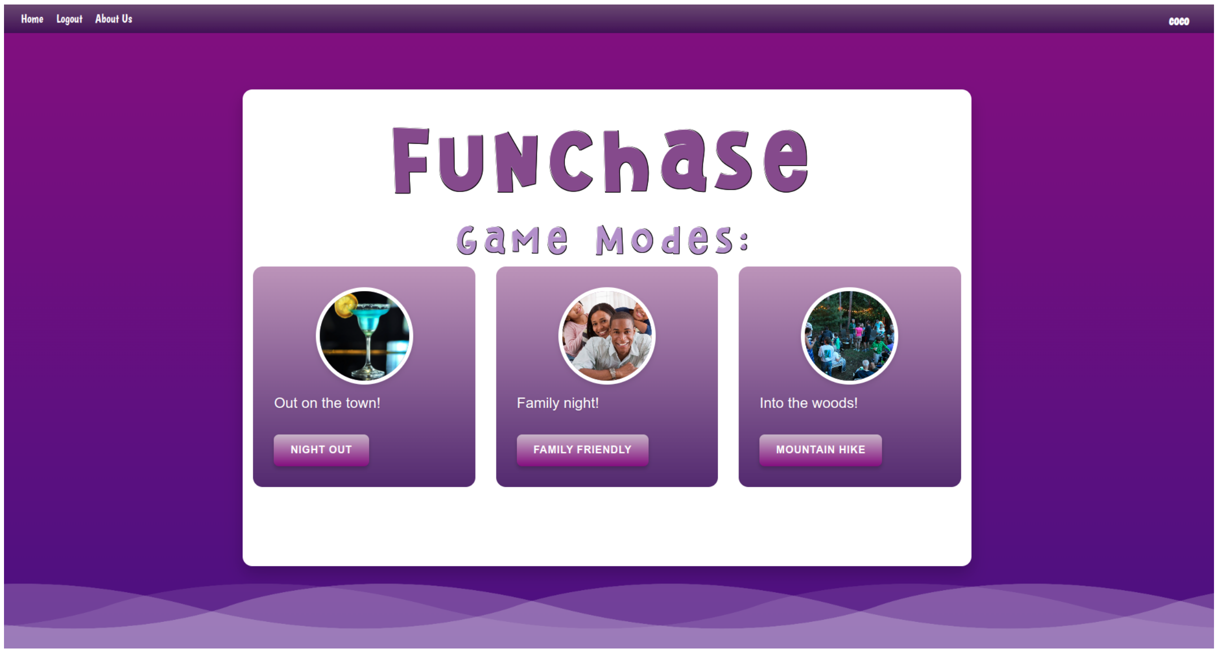
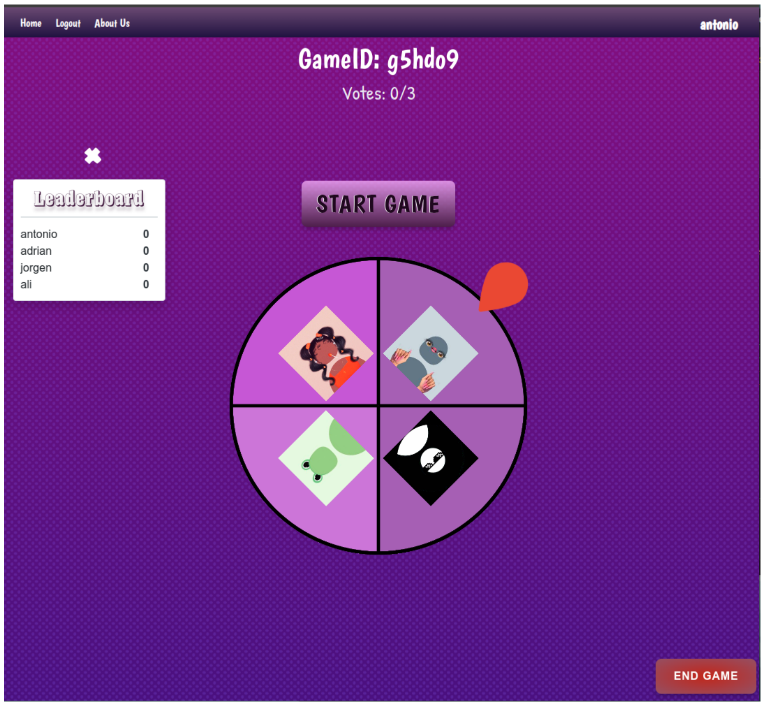


Figure 10: Create Game Page

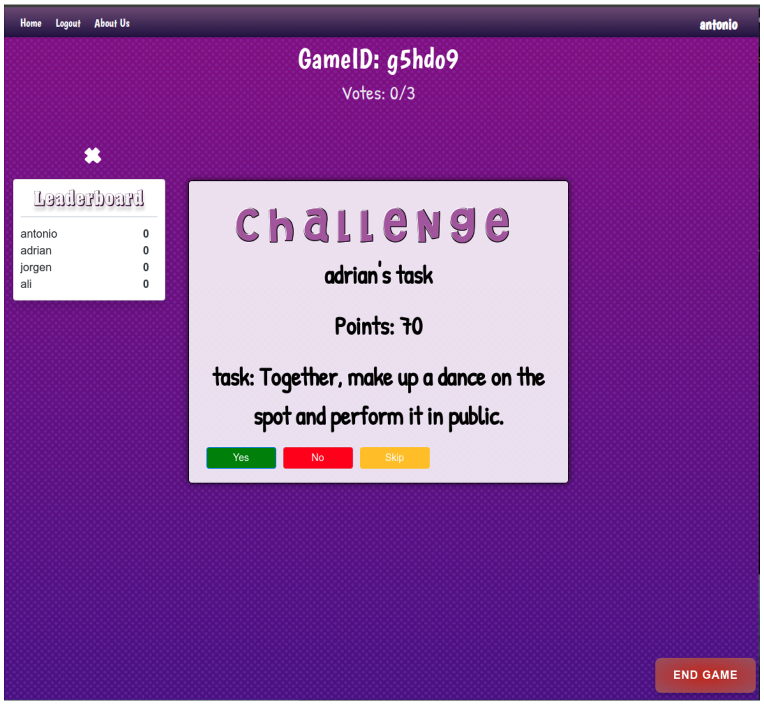
6.5 Game Lobby Site



Leaderboard

antonio	0
adrian	0
jorgen	0
ali	0

Figure 11: Game Lobby Page



GameID: g5hdo9

Votes: 0/3

Leaderboard

antonio	0
adrian	0
jorgen	0
ali	0

Challenge

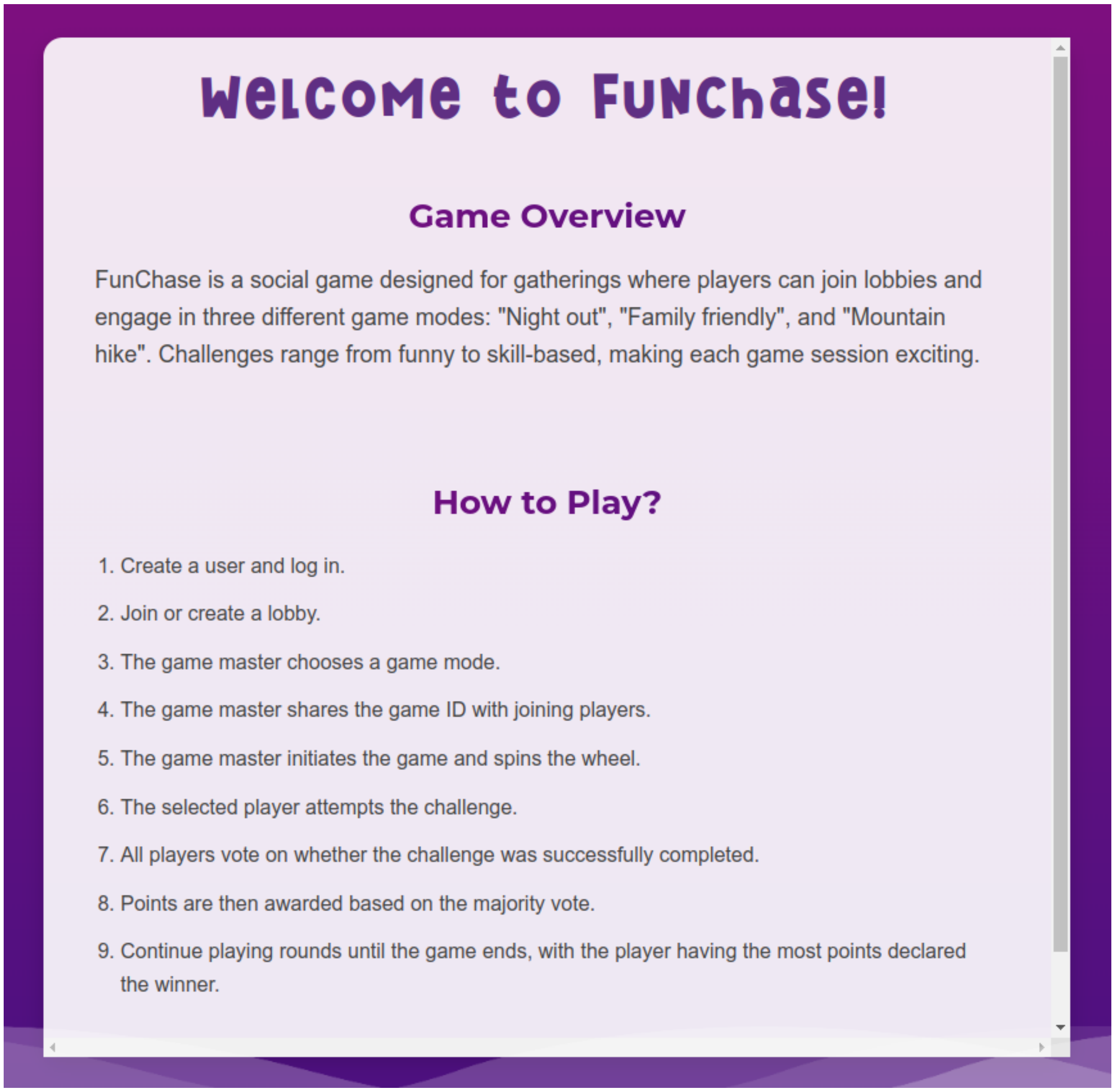
adrian's task

Points: 70

task: Together, make up a dance on the spot and perform it in public.

Figure 12: Challenges inside GameLobby Page

6.6 About Site



Welcome to FunChase!

Game Overview

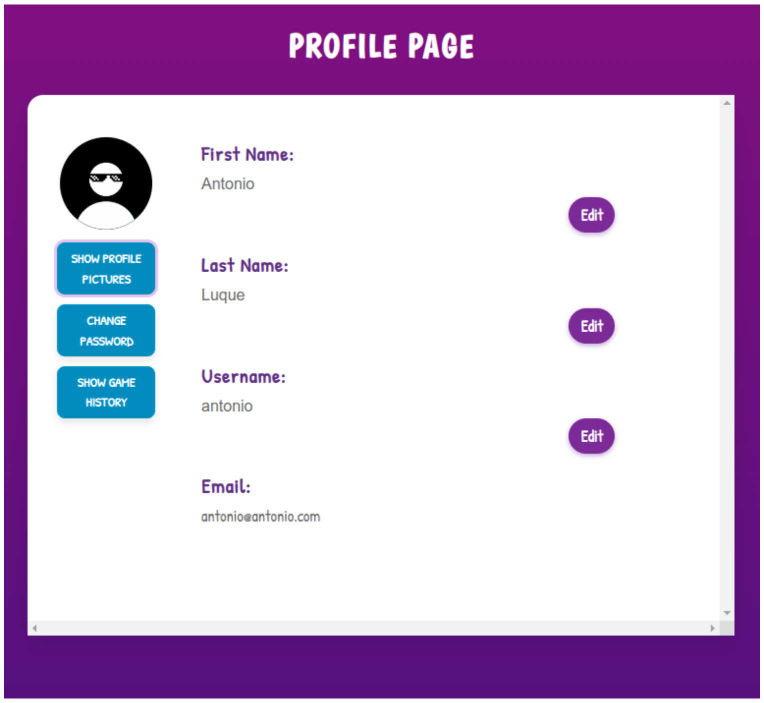
FunChase is a social game designed for gatherings where players can join lobbies and engage in three different game modes: "Night out", "Family friendly", and "Mountain hike". Challenges range from funny to skill-based, making each game session exciting.

How to Play?

1. Create a user and log in.
2. Join or create a lobby.
3. The game master chooses a game mode.
4. The game master shares the game ID with joining players.
5. The game master initiates the game and spins the wheel.
6. The selected player attempts the challenge.
7. All players vote on whether the challenge was successfully completed.
8. Points are then awarded based on the majority vote.
9. Continue playing rounds until the game ends, with the player having the most points declared the winner.

Figure 13: About Page

6.7 Profile Site



First Name:
Antonio

Last Name:
Luque

Username:
antonio

Email:
antonio@antonio.com

Figure 14: Profile Page

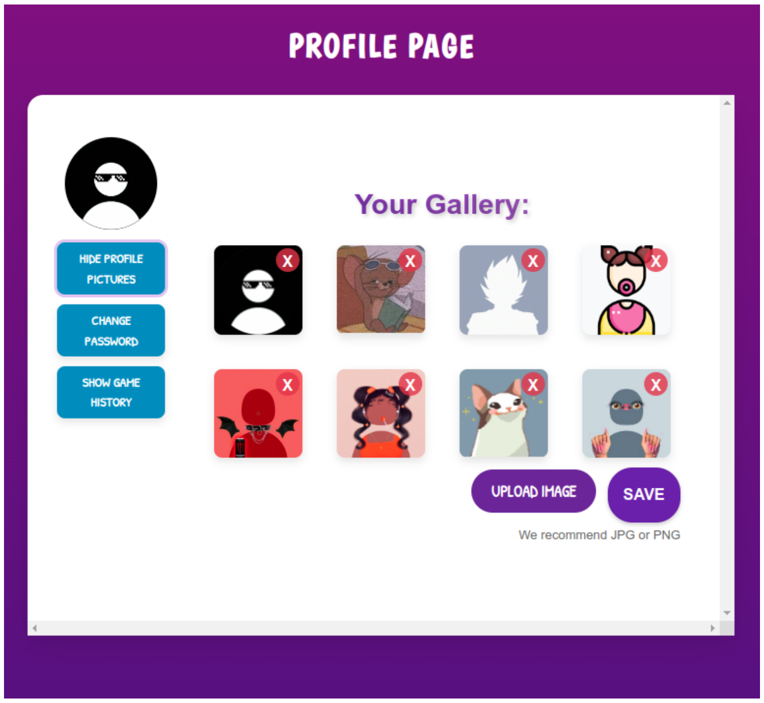


Figure 15: Gallery inside Profile Page

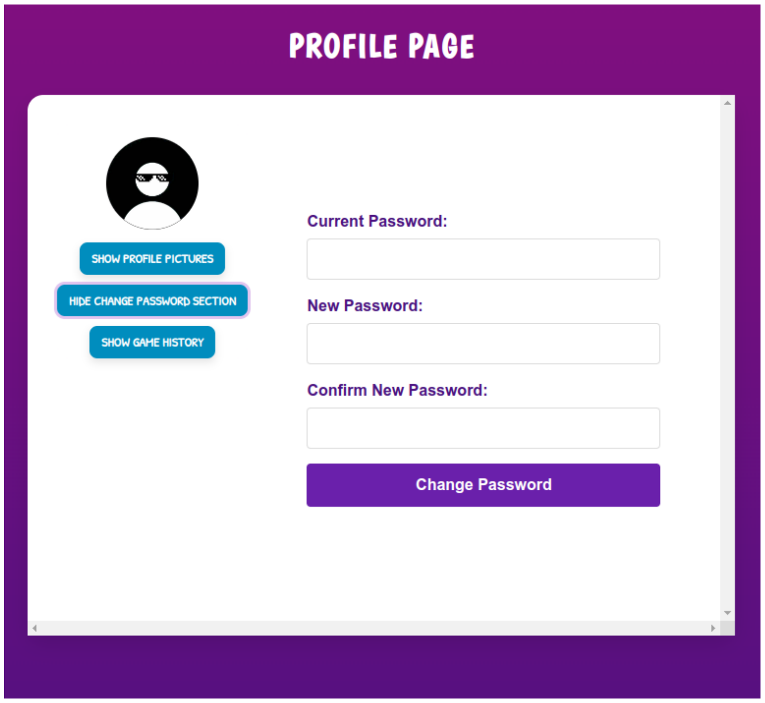


Figure 16: Change Password section inside Profile Page

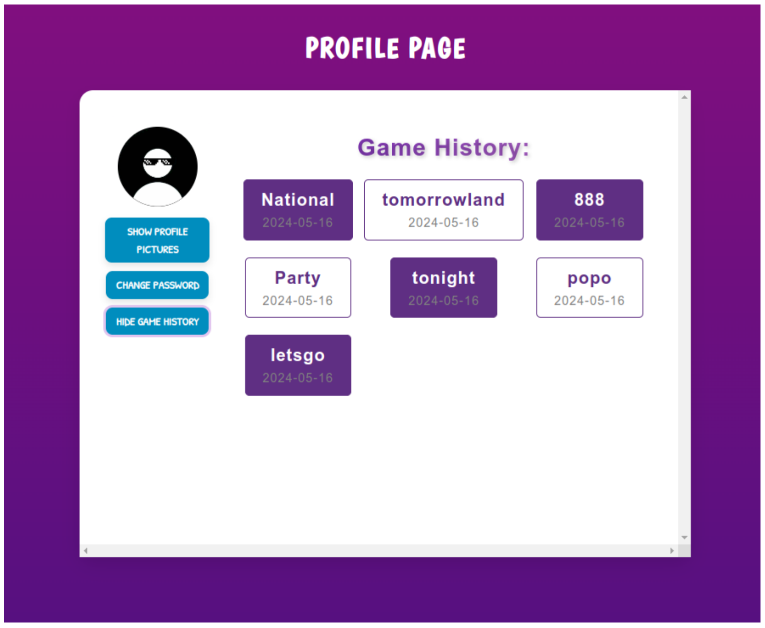
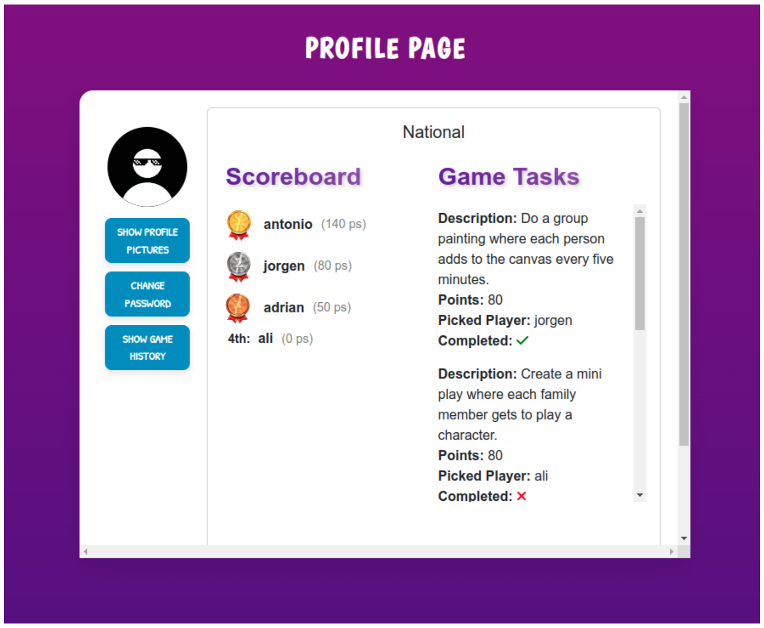


Figure 17: Game History inside Profile Page



National

Scoreboard

antonio (140 ps)
jorgen (80 ps)
adrian (50 ps)
4th: ali (0 ps)

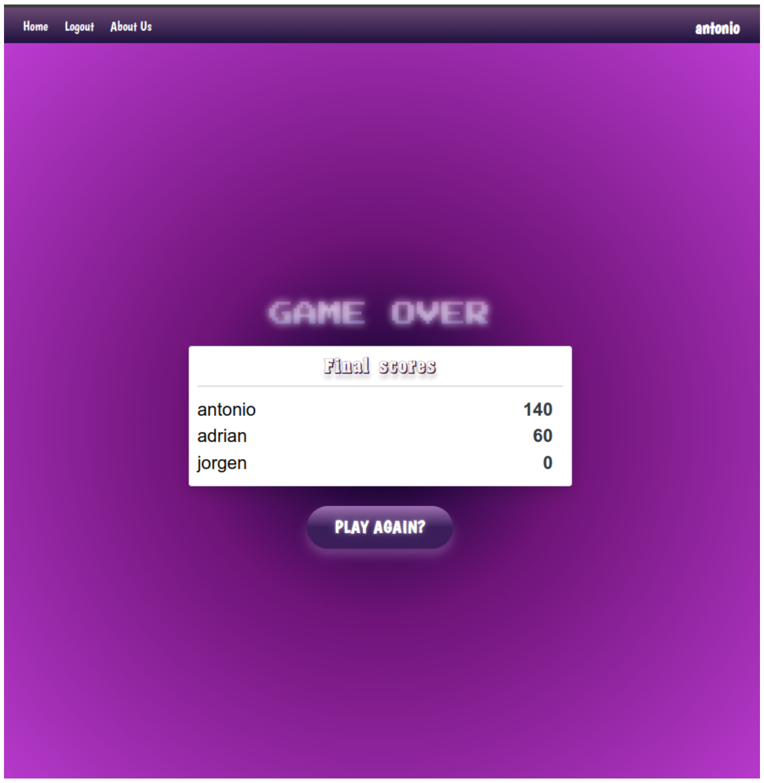
Game Tasks

Description: Do a group painting where each person adds to the canvas every five minutes.
Points: 80
Picked Player: jorgen
Completed: ✓

Description: Create a mini play where each family member gets to play a character.
Points: 80
Picked Player: ali
Completed: ✗

Figure 18: Details of the "National" game inside Game History of Profile Page

6.8 End Game Site



Final scores

antonio	140
adrian	60
jorgen	0

Figure 19: End of the Game Page with final leaderboard

7 Discussion

7.1 Product Development Learnings

During this project we combined the knowledge and skills acquired from our previous experiences with programming, communication, security, database management and more into the development of our own application. We worked with technologies like Django, React, CSS and SQLite, and followed agile working methodologies like using scrum, extreme programming, pair programming, test driven development and conducted weekly meetings. By following the scrum framework we learned the benefits of organizing our work into sprints, allowing us to focus on specific tasks and deliver incremental updates to our application.

7.2 Challenges and Solutions

A big reason as to why we used pair programming, is that the features in our project were very dependent on each other and it was difficult to sometimes work independently. We also never had a great amount of tasks to be distributed at a given point, because future features need prior features to be developed. The times we worked independently development was going slower, because like in a relay you had to wait for the person before to finish on a task/feature, before you could continue.

When problems appeared, they could be discussed and fixed faster when working simultaneously. When working in pairs, or more, together the motivation was higher, more people were involved and communication was better within the group, because development was being constantly discussed. Pair programming was done specifically by connecting one of our computers to a common big screen and with VS-code extension that allowed several us to work in the same document.

7.2.1 Testing

While testing with djangos TestCase class was relatively easy, tesing the asynchronous nature of the websocket was not. TestCase was thrown out for a few different unittest classes in the beginning, but ended with ChannelsLiveServerTestCase, paired with djangos http and websocket communicators. An example of the challenges faced with this is maintaining a session. Cookies had to be manually extracted from login responses decoded, formatted correctly, encoded, and put into other requests.

This was solved by creating helper functions, f.ex. one for the http communicator that dynamically constructed the proper cookie header to be passed, allowing us to use a single function to make all http communicator requests. Another interesting helper function is a function that logged in, and stored all the relevant tokens (JWT, session, csrf) in their respective files at specific player indices.

7.3 Code Structure

The code structure is organized to be simple, maintainable and scalable. The codebase is divided into two main folders: the backend and the frontend. The backend folder contains all the server-side code written in Django which includes models, views, websocket and API endpoints. It consists of the core logic and functionality of our application, such as user authentication, game mechanics and database interactions. The frontend folder contains all the client-side code written in React which includes components and utility functions. This folder represents the user interface layer of our application and is responsible for rendering the user interface, handling user interactions and communicating with the backend server via API requests.

Within the backend and frontend folders we used a modular design approach to break down our codebase into smaller, reusable components. Each component is designed to facilitate a specific piece of functionality or UI element. This makes it easier to understand, maintain and update. This also allows for greater flexibility and agility in development, since we are able to work on different components concurrently without worrying about conflicts or dependencies.

7.4 Branches

When writing code is was important to work in branches using Git. Every time a feature or task was added it was done in a specific branch with a clear and descriptive name. Looking back there should have been more descriptive commit messages, which would have made it easier to identify mistakes and go back to previous states.

7.5 Reflections

For the app, we chose to use the Django framework for the backend. Django was a good framework to work with, it gave us a lot of functionality out of the box and made features such as authentication, database implementation and overall architecture.

The frontend framework we chose for the app was react, which has worked well for us. It is very widely used, so there are many modules developed for it, making it easier to implement sophisticated functionality. However, some of the components we developed handles more logic than necessary and could be split into more components for improved modularity and readability.

Too much logic in one component can also cause issues when there are many states handled, as changing a state in react results in a re-render of the component.

7.6 Lessons Learned

A major takeaway from this project was the critical role of communication. The times we were meeting and were regularly involved over our group chat were the times we got the most work done. Sometimes we weren't able to reach our sprint goals, because we weren't checking the chat and meeting often enough which led to having to compensate in the sprint after. During the second half of the project this improved a lot and we started more often notifying team members when tasks were completed and communicating effectively about ongoing work to avoid duplication of effort. This improved coordination and ensured that everyone was working towards the same objectives.

We learned that we should have had smaller goals within the sprints and divide tasks into smaller and more manageable parts. Having shorter time periods goals instead of big sprint goals lasting three weeks would have allowed us to make progress more frequently. To clarify the sprint itself can be three weeks, but have smaller task goals that should be completed in two or three days. For example if the sprint goal was to implement a new feature that allows to upload images to their profiles, the tasks connected to this feature should have had deadlines of their own. Additionally, this might have helped is in identifying and address issues sooner.

We have learned and gained lots of software development experience. By using Django, React, Git and following agile development we have become more proficient in developing applications. Django taught us backend skills, React improved our frontend abilities and Git helped with collaboration. Working in an agile way, like with Scrum and TDD, gave us structure. This hands-on experience has made us better developers for future projects.

7.7 Alternative approach - DevOps

Since FunChase has not yet been deployed, it has not followed all the steps of the DevOps cycle. The app has currently only been in the development stages, the next step is to release it and gain new knowledge from operation.

First, it is crucial to ensure well-structured Code Management to make every part of the code understandable and accessible for the whole team. Github will be used for code transfer, version control and retrieval, merging, branching and recovery. Next, it is important to reduce the time and costs for integration, deployment and delivery as well as making these processes more reliable and reproducible. For that we will follow the aspects of DevOps automation:

- **Continuous Integration**: A core principle of DevOps is to automate tasks that can be automated, so it would be natural to use a CI tool to automatically build and test the app on every commit.

- **Continuous Delivery and Deployment**: The next step is to use a CD pipeline to automatically deploy changes after passing tests. For both CI/CD, this can be accomplished with tools like GitHub Actions and Selenium for testing.
- **Containerization**: We should containerize the app to ensure a consistent environment across development, testing, and production using tools like Docker and Kubernetes.
- **Infrastructure as Code**: We should define infrastructure using IaC tools like AWS CloudFormation and manage it through code with tools such as Puppet and Chef that can automatically install software and services on servers according to the infrastructure definition.

Once these steps are completed, we will have adopted automated operational aspects of DevOps. However, it is important to continuously improve our DevOps processes to achieve faster deployment of better-quality software, we could achieve that following these DevOps Measurements:

- **Process Measurement**: Monitoring process metrics such as deployment frequency, mean time to recovery, change volume and lead time from development to deployment.
- **Service Measurement**: Investigating service metrics such as performance, availability, number of customer complaints and percentage increase in customer numbers.
- **Usage Measurement**: Measure active users, session duration, feature usage and user retention.
- **Business Success Measurement**: Tracking revenue, conversion rates, Customer Acquisition Cost (CAC), Customer Lifetime Value (CLTV). (All of that after having a business plan.)

By following all the above steps, FunChase would follow the principles of the DevOps method. [7]

7.8 Ethics

The initial idea was to have a drinking game called "BoozeChase" and keep implementing this, but due to certain ethical considerations we expanded our idea into "FunChase" instead. The shift from a drinking-focused game to a family-friendly reflects our commitment to inclusivity for various age groups and cultural backgrounds. It was important for us to promote healthy choices and alternative forms of entertainment to avoid encouraging risky behaviors or excessive alcohol consumption. This led to the creation of different game modes so our game is more all-rounded and suitable for everyone.

8 Future Work

8.1 Unimplemented Functionality from the Backlog

- **Custom game modes:** One of the features we wanted to expand on was the possibility of creating custom game modes and have a more diverse set of game modes. This way users could make their own questions and challenges and tailor the game experience to a specific occasion. Also, adding functionality to choose which components to include in a game mode would further increase the possibilities for the user and make the game broader and easier to tailor for each users needs.
- **Chat functionality:** This would enable players to communicate while being in-game. The chat functionality is nice to have if the game is played while the participants are not near each other or to have a private discussion before voting. Overall this feature would have improved the overall user experience by allowing better coordination and interaction during gameplay while also keeping users on the platform.
- **Interactive user responses:** Interactive responses would allow users to respond with pictures and text, which makes the game more engaging and dynamic. A challenge could require users to upload a photo as proof of task completion or respond to a question in text format.

8.2 Ideas for Further Development

- **Continuous development:** By continuously developing based on seasonal events we could keep the game fresh and engaging. For example we could have a christmas or easter theme and limited-time game modes.
- **Mobile app based:** A mobile based application will make the game more accessible and flexible for various settings.
- **Achievements:** Achievements would incentivize users to play the game more and reward regular users. It would also make it possible for us to steer the users on a path, which benefits the platform by giving achievements, for example creating a game mode or playing 100 games.
- **Better security:** Right now the codebase is using HTTP for requests between browser and server which transfers data in plaintext. We should go over to HTTPS, because it enhances security by encrypting data while transferring.
- **More modular design:** As of right now, we have to two files that perform the logic in the backend. All our code is written in views.py and consumers.py seperately, even though they perform many of the same tasks. Here we could have made another file that contains operations made on the database. This way if we want to add more files in the backend we would simply call the functions from this operations file. And reduce the amount of overall code.

9 Conclusion

This report has discussed the development of our game "FunChase", and the implementation of its features, design and challenges. The goal of this development process was implementing a simple and fun social game using a Django backend and React frontend, in which we have been successful. There are still room for improvements, like the unimplemented functionality from the backlog and our ideas for further development. There has also been challenges to overcome, like integrating web sockets to make a live action game and combing our different conceptual ideas for how the game should behave. A major takeaway from this project was learning about and experience agile development and the impact it has to software development. This includes concepts such as scrum, test driven development, pair programming and extreme programming. Overall it has been a valuable learning experience, combining different technologies and agile methodologies to develop a functional and engaging application.

10 User Stories

10.1 User story 1

As a party host I want to offer a fun activity game night for my friends when they are visiting.

10.2 User story 2

As a player I want to have my own profile where I can change profile information and track progress.

10.3 User story 3

As an outdoor person I want to participate in challenges in parks and nature.

10.4 User story 4

As a player I want to play a game with clear instructions so I can quickly understand how to play and enjoy the game without feeling overwhelmed.

10.5 User story 5

As a family man I want to enjoy wholesome entertainment with my loved ones, disregarding age and beliefs, without any inappropriate content.

10.6 User story 6

As a competitive person I want to view leaderboards to see how my performance compares to other players.

11 Personas

11.1 Personas 1 - Social Sabrina:

Sabrina is a 23-year-old social butterfly who loves to connect with friends and meet new people. She is an only child who was born in New York and have lived here all here life with her parents who are both college professors. Her mom is a law professor and her father is a business and marketing professor. She has after inspiration from her father gotten a bachelors degree in business and is going to take a masters degree specializing in finance. Since she is moving out the coming semester into her own apartment she wants to look for resources that can help her in organizing exciting social gatherings and parties.

Sabrina often struggles to come up with fresh and engaging ideas for social gatherings. Since she plans to be hosting a lot and often invite new people its important that the game or activities she is having are fun and simple for both new people and her friends to socialize over. She is very tech-savvy aswell and since seeing that FunChase has a party/drinking mode it could be just the right platform for her to entertain people on social gatherings.

11.2 Personas 2 - Family-Friendly Frank

Frank is a 42 year old father of two children ages 12 and 14, who values spending quality time with his family. Frank has a masters degree in computer science from Harvard university and now works as a Java full-stack developer. He now lives in the suburbs with his wife and two kids.

They have a mountain cabin two hours away from their home where they enjoy outdoor activities like hiking, camping and fishing. Frank also wants to find a fun and wholesome game they can enjoy while indoors. It is important that the game is family-friendly, suitable for all age groups and appeal to both adults and children.

Recently while browsing the app-store he stumbled upon FunChase which tics all his boxes. He believes that with the simplicity and all the game modes of the application, countless hours of boredom will be saved.

11.3 Personas 3 - Adventurous Adam

Adam is a 31 year old outdoor enthusiast who loves adventures and exploring the outdoors. Adam works as an infleuncer and travel blogger. He is passionate about nature, wildlife and outdoor sports like rock climbing, surfing and mountain biking. For cheaper travels Adam often stay at hostels.

Since he is always traveling and spends lots of time outside he wants to connect with like-minded individuals who share his passion for outdoor exploration and fun activities. FunChase appeals to Adam because it offers a simple platform to start engaging with other outdoor enthusiasts and constantly be prompted with new and fun challenges.

12 Scenarios

12.1 Scenario 1

Sabrina is having a birthday party and invites her own friends and their friends. She wants to arrange a fun activity that includes everyone and makes sure everybody get to know each other. From her IPhone she visits the FunChase website, logs in and creates a ”nights out” game lobby.

Her guests visit the website and create a user that takes them to the frontpage where they can easily join the game lobby with the invite code from Sabrina. As the game master Sabrina starts the game and the spinning wheel selects her friend. The challenge that is prompted is: "do a freestyle dance in front of everybody that lasts for 20 seconds". Everyone watches eagerly as her friend awkwardly dances around the room. After the performance the other party members cast their votes. With the majority approval her friend earns 100 points and the game continues with more fun and friendly competition.

12.2 Scenario 2

Peter is hosting a gathering for his football team and wants to add some excitement to the night. Instead of the usual board games or movie nights he introduces his friends to FunChase. Peter logs into FunChase from his computer and creates a "night out" game mode. He displays the invite code for the others on the big screen he is connected to.

After everybody has joined, John starts the game and the spinning wheel selects his friend Mark to attempt a drinking challenge; 3 sips of beer in under 10 seconds. Peter then starts a timer on his phone and Mark attempts the challenge. After the 10 seconds everybody votes and this time most people voted that he didn't complete the challenge in time.

Mark gained 0 points from this round the the wheel spins again to select a new player for a challenge. Mark will get new opportunities later.

12.3 Scenario 3

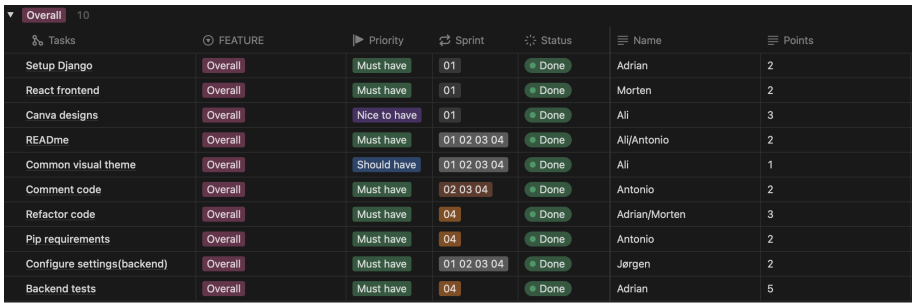
Frank is taking his wife and two children to the park where they will be meeting some of their friends and their children for a family outing. The kids start playing around, while the adults are chatting and preparing the food. Shortly after they gathered around a carpet on the grass field and ate dinner.

After dinner Frank suggests playing a game to keep everyone entertained. Remembering the FunChase app he recently discovered, Frank takes out his smartphone and opens the app. He tells everybody to take out their phones as well and navigate to the frontpage. Before choosing a game mode, he asks everybody if they want to play "family friendly" or "mountain hike". The children yell "mountain hike".

Everybody eagerly join in and Frank starts the spinning wheel. Players are picked and challenges are prompted one after another making them explore the park and nature nearby. [8]

13 Backlog

13.1 Overall

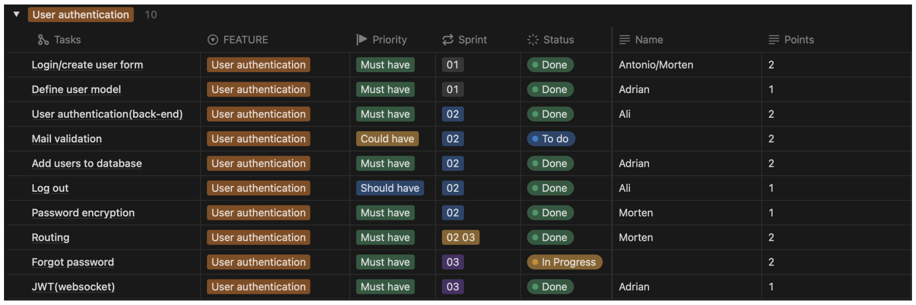


Overall 10

Tasks	FEATURE	Priority	Sprint	Status	Name	Points
Setup Django	Overall	Must have	01	Done	Adrian	2
React frontend	Overall	Must have	01	Done	Morten	2
Canva designs	Overall	Nice to have	01	Done	Ali	3
READme	Overall	Must have	01 02 03 04	Done	Ali/Antonio	2
Common visual theme	Overall	Should have	01 02 03 04	Done	Ali	1
Comment code	Overall	Must have	02 03 04	Done	Antonio	2
Refactor code	Overall	Must have	04	Done	Adrian/Morten	3
Pip requirements	Overall	Must have	04	Done	Antonio	2
Configure settings(backend)	Overall	Must have	01 02 03 04	Done	Jørgen	2
Backend tests	Overall	Must have	04	Done	Adrian	5

Figure 20: Overall backlog

13.2 User authentication

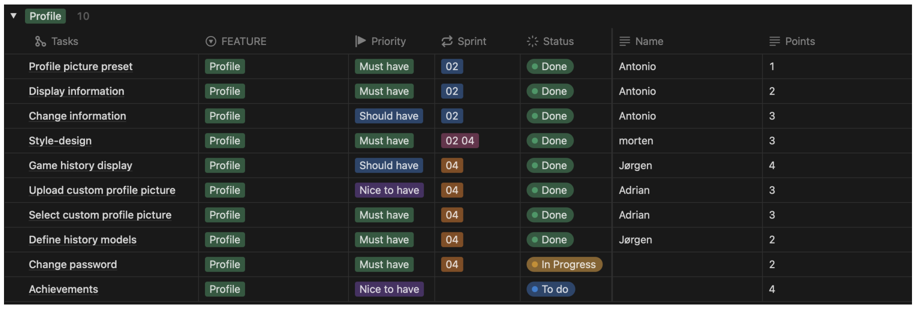


User authentication 10

Tasks	FEATURE	Priority	Sprint	Status	Name	Points
Login/create user form	User authentication	Must have	01	Done	Antonio/Morten	2
Define user model	User authentication	Must have	01	Done	Adrian	1
User authentication(back-end)	User authentication	Must have	02	Done	Ali	2
Mail validation	User authentication	Could have	02	To do		2
Add users to database	User authentication	Must have	02	Done	Adrian	2
Log out	User authentication	Should have	02	Done	Ali	1
Password encryption	User authentication	Must have	02	Done	Morten	1
Routing	User authentication	Must have	02 03	Done	Morten	2
Forgot password	User authentication	Must have	03	In Progress		2
JWT(websocket)	User authentication	Must have	03	Done	Adrian	1

Figure 21: User authentication backlog

13.3 Profile

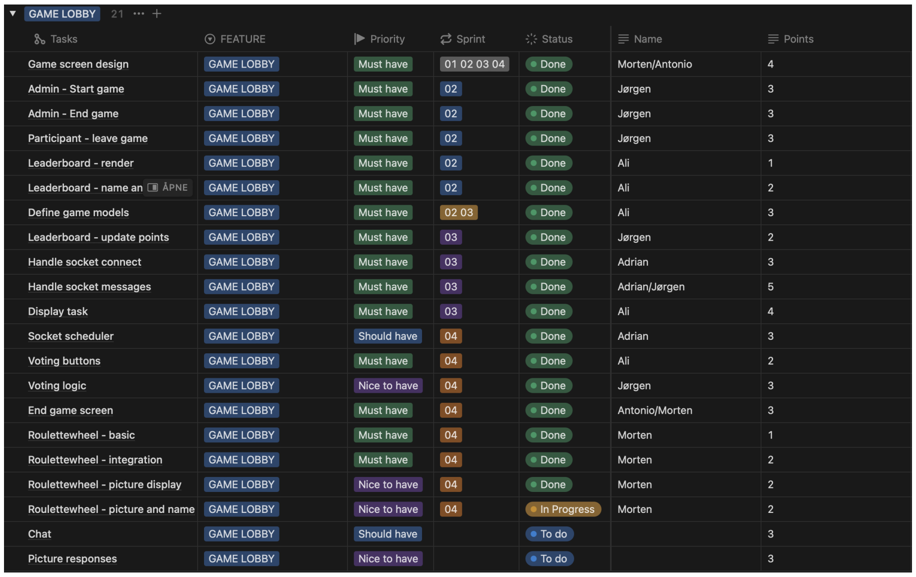


Profile 10

Tasks	FEATURE	Priority	Sprint	Status	Name	Points
Profile picture preset	Profile	Must have	02	Done	Antonio	1
Display information	Profile	Must have	02	Done	Antonio	2
Change information	Profile	Should have	02	Done	Antonio	3
Style-design	Profile	Must have	02 04	Done	morten	3
Game history display	Profile	Should have	04	Done	Jørgen	4
Upload custom profile picture	Profile	Nice to have	04	Done	Adrian	3
Select custom profile picture	Profile	Must have	04	Done	Adrian	3
Define history models	Profile	Must have	04	Done	Jørgen	2
Change password	Profile	Must have	04	In Progress		2
Achievements	Profile	Nice to have		To do		4

Figure 22: Profile backlog

13.4 Game lobby

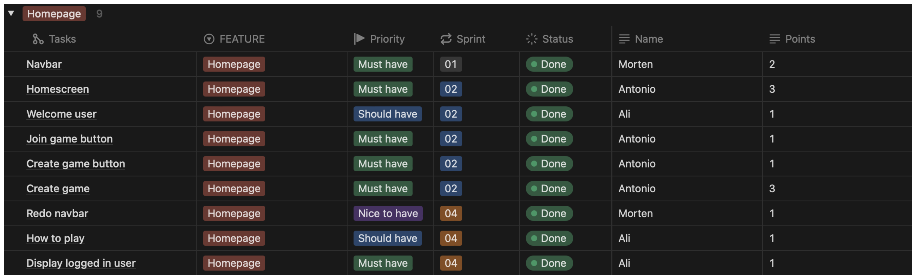


GAME LOBBY 21

Tasks	FEATURE	Priority	Sprint	Status	Name	Points
Game screen design	GAME LOBBY	Must have	01 02 03 04	Done	Morten/Antonio	4
Admin - Start game	GAME LOBBY	Must have	02	Done	Jørgen	3
Admin - End game	GAME LOBBY	Must have	02	Done	Jørgen	3
Participant - leave game	GAME LOBBY	Must have	02	Done	Jørgen	3
Leaderboard - render	GAME LOBBY	Must have	02	Done	Ali	1
Leaderboard - name an ÅPNE	GAME LOBBY	Must have	02	Done	Ali	2
Define game models	GAME LOBBY	Must have	02 03	Done	Ali	3
Leaderboard - update points	GAME LOBBY	Must have	03	Done	Jørgen	2
Handle socket connect	GAME LOBBY	Must have	03	Done	Adrian	3
Handle socket messages	GAME LOBBY	Must have	03	Done	Adrian/Jørgen	5
Display task	GAME LOBBY	Must have	03	Done	Ali	4
Socket scheduler	GAME LOBBY	Should have	04	Done	Adrian	3
Voting buttons	GAME LOBBY	Must have	04	Done	Ali	2
Voting logic	GAME LOBBY	Nice to have	04	Done	Jørgen	3
End game screen	GAME LOBBY	Must have	04	Done	Antonio/Morten	3
Roulettewheel - basic	GAME LOBBY	Must have	04	Done	Morten	1
Roulettewheel - integration	GAME LOBBY	Must have	04	Done	Morten	2
Roulettewheel - picture display	GAME LOBBY	Nice to have	04	Done	Morten	2
Roulettewheel - picture and name	GAME LOBBY	Nice to have	04	In Progress	Morten	2
Chat	GAME LOBBY	Should have		To do		3
Picture responses	GAME LOBBY	Nice to have		To do		3

Figure 23: GameLobby backlog

13.5 Homepage

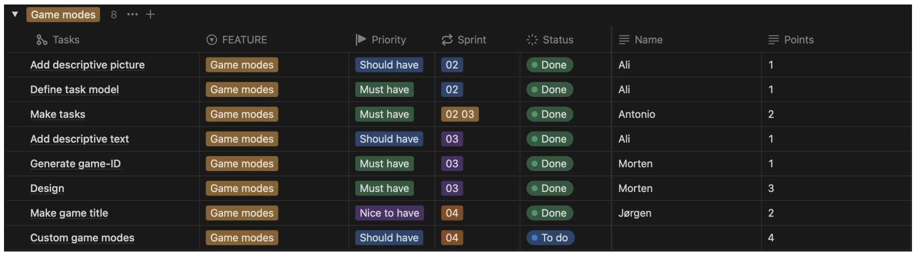


Homepage 9

Tasks	FEATURE	Priority	Sprint	Status	Name	Points
Navbar	Homepage	Must have	01	Done	Morten	2
Homescreen	Homepage	Must have	02	Done	Antonio	3
Welcome user	Homepage	Should have	02	Done	Ali	1
Join game button	Homepage	Must have	02	Done	Antonio	1
Create game button	Homepage	Must have	02	Done	Antonio	1
Create game	Homepage	Must have	02	Done	Antonio	3
Redo navbar	Homepage	Nice to have	04	Done	Morten	1
How to play	Homepage	Should have	04	Done	Ali	1
Display logged in user	Homepage	Must have	04	Done	Ali	1

Figure 24: Homepage backlog

13.6 Game modes



Game modes 8

Tasks	FEATURE	Priority	Sprint	Status	Name	Points
Add descriptive picture	Game modes	Should have	02	Done	Ali	1
Define task model	Game modes	Must have	02	Done	Ali	1
Make tasks	Game modes	Must have	02 03	Done	Antonio	2
Add descriptive text	Game modes	Should have	03	Done	Ali	1
Generate game-ID	Game modes	Must have	03	Done	Morten	1
Design	Game modes	Must have	03	Done	Morten	3
Make game title	Game modes	Nice to have	04	Done	Jørgen	2
Custom game modes	Game modes	Should have	04	To do		4

Figure 25: Gamemodes backlog

14 Meeting Notes

14.1 January

14.1.1 29.01.23

- Set up a new repository for the project.
- Brainstormed ideas with the Teaching Assistant and team members.
- Should Implement user authentication with varying access levels.
- Start working on personas, scenarios and user stories.
- Look at each others code before pushing changes.
- Established a backlog using Notion.
- Explored ideas such as social credit app, podcast recommendations, multi-streaming podcasts, quizzes, and games.
- Make a useful product.
- Rescheduled meetings to Tuesday from 11-12.
- Should be able to use application on mobile.

14.2 February

14.2.1 6.02.23

- Brainstormed several ideas for the project, including: A game where players join a lobby and complete challenges as they move around town.
- A ski park recommendation application.
- A ”WhatToDo” application suggesting activities based on user preferences.
- ”BroPoints,” a favor system within the application.
- A TypeRacer-style game

14.2.2 11.02.23 - Extra Group Meeting

- Develop a minimum reliable product
- Give points before a task and then you get a point after tasks are done. All members should do the same amount of points.
- Look and develop tests.

14.2.3 13.02.23

- Initialized a React project for frontend development.
- Initialized Django for backend development.
- Merged our previous repository with the new official repository created by the TA.
- Began the design process for the application interface using tools like Canva.
- Planned for the application to be browser-based but compatible with mobile devices.
- Explored the possibility of integrating LLM to generate tasks dynamically.

14.2.4 20.02.23

- Core components: creating user, making a room/group where you can invite others (and one is an admin who can delegate tasks, edit the group etc.), have a score system
- 2 weeks sprint: maybe 1-2 features, 20.02 – 05.03
- Some ideas for features: Maybe guest users as an extra feature and different modes for the challenges
- Scrum master: Ali

14.2.5 27.02.23

- Organize a better backlog
- Should be together more often when programming - pair programming

14.3 March

14.3.1 05.03.23

- Login and user authentication - hashing for passwords
- Retrospective meeting
- Some cornerstones setup
- Should work more consistently
- Be better at taking initiative
- Goal for next sprint: more frontend components, session setup
- Scrum master: Jørgen

14.3.2 12.03.23

- Nothing new, keep working with tasks from the backlog

14.3.3 19.03.23

- A lot better frontend styling and components
- More progress when working on tasks together then working on tasks indiviidually
- Have demo of product - navigate between components
- Have a register user page and a login page, and sessions are working now
- Scrum master: Adrian

14.3.4 21.03.23 - Extra Group Meeting

- Easter holiday - add more tasks to backlog and delegate who does what.

14.4 April

14.4.1 10.04.23

- Got a lot of work done
- Better communication
- Spent a lot of time together
- Improve telling eachother when we are done with a task
- Added multiple modes: for family, mountain hike and for going out with friends
- Can create game and join the lobby from other users
- With a player list (leaderboard)
- Both individual and group challenges
- Only the game master can end the game, but every participant can leave the game
- Have a profile page where they can edit the profile settings etc.
- Are going to add a log database for game history, and a button for completing the task and giving points
- 80 percent done on the MVP.

14.4.2 17.04.23

- Online meeting - several members sick
- Start refactoring the code a bit
- Keep working on backlog tasks
- Scrum master: Antonio

14.4.3 23.04.23

- Only group tasks, so no individual tasks anymore
- The person with majority votes get points
- Are going to fix a game history and some frontend

14.5 May

14.5.1 30.04.23

- Have been figuring out how they want the final result to be
- Tried to accomplish the requirements for the project
- Have been good at defining tasks and following them (at different pace)
- Got a lot of things done
- Worked well this month
- Testing can be improved
- Should have been better at setting deadlines and actually following them
- Start on the report
- Have a MVP

15 References

References

[1] Sommerville, Ian (2019). *Engineering Software Products: An Introduction to Modern Software Engineering*. Global Edition, chapter 1 - Product vision.

[2] Sommerville, Ian (2019). *Engineering Software Products: An Introduction to Modern Software Engineering*. Global Edition, chapter 2 - Agile Software Engineering.

[3] Celery (2023). *Celery Documentation: First Steps with Django*. Version 5.4.0. Retrieved from `https://docs.celeryq.dev/en/stable/django/first-steps-with-django.html`

[4] Django (2022). *Django Channels Documentation*. Retrieved from `https://channels.readthedocs.io/en/latest/`

[5] Django. (2024). *Testing in Django*. Retrieved from `https://docs.djangoproject.com/en/5.0/topics/testing/`

[6] Sommerville, Ian (2019). *Engineering Software Products: An Introduction to Modern Software Engineering*. Global Edition, chapter 4.1 - Non functional attributes.

[7] Sommerville, Ian (2019). *Engineering Software Products: An Introduction to Modern Software Engineering*. Global Edition, chapter 10 - DevOps.

[8] Sommerville, Ian (2019). *Engineering Software Products: An Introduction to Modern Software Engineering*. Global Edition, chapter 3 - Features, Scenarios, and Stories.