Bil 372

Software Layers

Mustafa Berke İmamoğlu

Ahmet Yasin Aydın

Kaan Güneş

Table Of Contents

Programming Language

Python

- **Rapid Development**: Python's simplicity and readability allowed us to develop complex backend functionalities quickly, which was essential given our tight project timeline.
- **Django Compatibility**: Python is the language for Django, our chosen web framework, facilitating seamless integration and efficient development.
- **Extensive Libraries**: Python offers a rich ecosystem of libraries for data manipulation, web development, and scientific computing, aiding in various aspects of the project.

UI Technology

React

- **Component-Based Architecture**: React's modular components allow for reusability and easier maintenance, reducing development time and ensuring consistency across the application.
- **Virtual DOM**: Enhances performance by minimizing direct manipulations of the DOM, providing a smoother user experience.
- **Rich Ecosystem**: Access to a wide range of libraries and tools, such as Redux for state management and React Router for client-side routing.
- **Developer Tools**: React Developer Tools aid in debugging and optimizing the application by providing insights into component hierarchies and state management.

ORM or Database Access Technology

Django ORM

- **Object-Relational Mapping (ORM)**: Abstracts database interactions, allowing us to work with Python objects instead of writing raw SQL queries, thereby increasing productivity and reducing errors.
- **Database Agnosticism**: Supports multiple relational databases, offering flexibility to switch databases with minimal code changes.
- **Automatic Schema Migrations**: Django's migration framework handles database schema changes efficiently, maintaining consistency and facilitating collaboration.
- **Data Validation and Integrity**: Enforces data validation at the model level, ensuring only valid data is saved, which is crucial for maintaining the integrity of the gaming platform's data.

Challenges Faced

1. Simulating Large-Scale Data

- **Issue**: Demonstrating the system's capability to handle large datasets (e.g., 10,000 players) without access to real users.
- **Solution**: Developed data generation scripts to create synthetic data, populating the database with realistic player profiles and activities.
- **Challenge**: Ensuring that the synthetic data accurately represented real-world usage patterns to validate system scalability effectively.

2. Tight Development Timeline

- **Issue**: Limited time to develop and integrate both frontend and backend components while meeting all project requirements.
- **Solution**: Prioritized essential features and utilized rapid development tools and frameworks to accelerate the development process.
- **Challenge**: Balancing the depth and quality of features implemented within the constrained timeframe.

3. Visualizing the Matchmaking Process

- **Issue**: Needed to demonstrate the dynamic process of players entering matchmaking queues and being assigned to game sessions.
- **Solution**: Implemented frontend visualizations that dynamically update to reflect the matchmaking process, including players being added to queues and game sessions.
- **Challenge**: Handling real-time data updates efficiently without causing performance issues or overwhelming the frontend.

4. Ensuring Data Integrity and Consistency

- **Issue**: Managing complex relationships and ensuring data consistency across multiple tables and relationships.
- **Solution**: Leveraged Django ORM's features for defining clear model relationships and constraints, and used transactions to maintain data integrity.
- **Challenge**: Dealing with concurrent access and potential race conditions when multiple operations occur simultaneously.

Lessons Learned

1. Importance of Robust Planning and Design

- **Effective Schema Design**: Investing time in designing the database schema upfront is crucial. A well-thought-out schema simplifies development and maintenance and helps avoid potential issues later.
- **Alignment with Requirements**: Continuously referring back to the project requirements ensured that all necessary features were implemented and met the specified criteria.

2. Leveraging Frameworks and Tools

- **Django and React Efficiency**: Utilizing these frameworks allowed us to focus on implementing business logic rather than building foundational features from scratch.
- **Third-Party Libraries**: Incorporating existing libraries accelerated development and added functionality, but also required careful consideration to ensure they met our needs without introducing unnecessary complexity.

3. Effective Team Collaboration

- **Communication**: Regular team meetings and updates were essential for coordinating tasks and ensuring everyone was aligned.
- **Version Control with GitHub**: Maintaining a centralized repository facilitated collaboration and kept the project organized.

4. Handling Real-Time Data and Scalability

- **State Management**: Gained valuable experience in managing application state effectively, especially when dealing with real-time data updates in React.
- **Performance Optimization**: Recognized the importance of optimizing database queries and using bulk operations to handle large datasets efficiently.

5. Adaptability and Problem-Solving

- **Overcoming Technical Challenges**: Faced with various technical hurdles, we learned to research and implement solutions effectively.
- **Time Management**: Prioritized tasks and features to ensure that critical components were completed within the project timeline.

Choice of ORM

We chose the Django ORM for several key reasons:

- **Integration with Django Framework**: Since our backend was built using Django, using its native ORM ensured seamless integration with other components like views, templates, and the admin interface.
- **Productivity**: The ORM's high-level abstraction allowed us to define database models as Python classes, significantly speeding up development by eliminating the need to write raw SQL queries for common operations.
- **Database Agnosticism**: Django ORM supports multiple relational databases (e.g., PostgreSQL, MySQL, SQLite). This flexibility allowed us to develop without being tightly coupled to a specific database backend.

API Documentation

Endpoint	Method	Description	Request Body	Response Example
/api/login	POST	Authenticates the user and returns a userID	{ "username": "user", "password": "pass" }	{ "userID": "12345" }
/api/register	POST	Registers a new user in the system.	{ "username": "user", "email": "email@example.com", "password": "pass" }	{ "message": "User registered successfully" }
api/players/:userID	POST	Sets the preferred game mode for the user.	{ "gamemode": "GM001" }	{ "message": "Game mode updated" }
/api/queue/	GET	Retrieves the list of players currently in the queue.	None	[{ "queueid": "123", "playerid": "userID", "status": "waiting", "preferredgamemodelid": "GM001" }, ...]
/api/accept-game	POST	Accepts a matched game for the user and returns game session details.	{ "userID": "12345" }	{ "gamesessionid": "GS001", "players": [{ "id": "1", "name": "Player 1" }, ...] }
/api/reject-game	POST	Rejects a matched game for the user.	{ "userID": "12345" }	{ "message": "Game rejected" }
/api/gamesession/:sessionID	PUT	Ends the current game session and returns a summary.	{ "userID": "12345" }	{ "summary": [{ "id": "1", "name": "Player 1", "points": 100 }, ...] }

Supported Interfaces in the Application

Login Page

Description: Allows users to authenticate into the application.

Features:

- Username and Password input fields.
- Login button with API integration to /api/login.
- Error message display for failed authentication.
- “Not registered yet?” link redirects to the Register page.

UI Elements:

- Username Input
- Password Input
- Login Button
- Link to Register Page

Register Page

Description: Allows new users to register in the system.

Features:

- Fields for Username, Email, and Password.
- Submit button with API integration to /api/register.
- Error handling for failed registration (e.g., duplicate email or username).

UI Elements:

- Username Input
- Email Input
- Password Input
- Register Button
- Link to Login Page

Dashboard

Description: Main page after login, allowing users to start/join a game or view game-related details.

Features:

- Displays the current status (in queue, in-game).
- Game mode selector dropdown with options for different modes.
- Join Game button integrated with /api/players/:userID and /api/queue/.
- Displays queued players in a scrollable table.
- Popup for accepting or rejecting matched games.
- Accept and Reject buttons integrated with /api/accept-game and /api/reject-game.
- End Game button integrated with /api/gamesession/:sessionId.

UI Elements:

- Game Mode Selector
- Join Game Button
- Queue Player Table
- Accept/Reject Popup
- End Game Button

First Page (Control Center)

Description: Provides administrative controls and monitoring for the application.

Features:

- Buttons to fetch and display different types of data (e.g., players, blacklists, game sessions).
- Tables for displaying fetched data.

UI Elements:

- Data Fetch Buttons
- Paginated Tables for Players, Blacklists, and Game Sessions