Capstone Project: Ft_Transcendence

Group Members' Full Names: Deiaaeldin Hamad,
Mohammed Mohammed, Mohamed Nour, Ahmed Ibrahim, Hisham Eltayb

Group Members' Intra Logins: dehamad,
mohammoh, mohamoha, ahibrahi, heltayb

Date of Submission:

May 1st, 2025

Abstract

The growing challenge of creating secure, interactive web-based gaming applications has led to new approaches combining traditional web frameworks with blockchain technology and modern DevOps practices. Web-based gaming applications face significant challenges in delivering seamless multiplayer experiences while maintaining security, performance, and data integrity. The classic Pong game provides an ideal foundation to explore these challenges, raising the question of how modern web technologies can be leveraged to create a feature-rich, secure, and scalable gaming platform. ft_transcendence is a modern, feature-rich clone of the classic Pong game implemented as a full-stack web application that addresses these challenges. Prior implementations of web-based Pong games have often lacked integration with modern security practices, blockchain technology, and containerized deployment strategies. Many existing solutions fail to address the security vulnerabilities inherent in web applications, particularly when running containerized services as root users. This project demonstrates the integration of real-time gameplay, secure authentication, on-chain tournament recording, and full DevOps automation within a Docker-Compose stack. The backend is powered by Django and PostgreSQL, exposing a clean RESTful API secured with JWT cookies and optional TOTP-based Two-Factor Authentication (2FA). Users can authenticate via 42 OAuth or traditional credentials with proper security measures including non-root container execution. The frontend, crafted using vanilla JavaScript and Bootstrap 5, delivers a responsive single-page application rendered on an HTML5 ¡canvas¿ element for immersive gameplay. Game modes include 1v1, AI opponents, 4-player multiplayer, and tournament brackets. Match and tournament data are stored both in PostgreSQL and on an Ethereum sidechain, ensuring immutability and verifiability. Static content and reverse proxying are handled by Nginx, which also enforces HTTPS using TLS. The system integrates centralized logging and monitoring through the ELK stack, and supports production-grade deployment via Docker and Makefiles. The implementation achieves cross-browser compatibility and accessibility for visually impaired users. This research demonstrates how combining traditional web technologies with blockchain solutions creates secure, verifiable gaming experiences while addressing critical security vulnerabilities through proper container user management. The architecture showcases a framework for building scalable web applications that maintain data integrity while enhancing user engagement through profile customization, friends lists, online status, and detailed match history.

Contents

Abstract i

List of Figures iv

List of Tables v

List of Abbreviations vi

1 Introduction 1

2 Project Goals and Requirements 3

3 Software Development Life Cycle (SDLC) 5
- 3.1 Requirement Analysis . . . 5
- 3.2 Modules Implemented . . . 6
- 3.3 Design the System Architecture . . . 7
- 3.4 Implementation . . . 8
- 3.5 Testing the application and its functionalities . . . 10

4 Elements of SDLC 12
- 4.1 SDLC Methodology . . . 12
- 4.2 Gantt Chart . . . 12
- 4.3 Risk Register and Risk Matrix . . . 14
  - 4.3.1 Risk Register . . . 14
  - 4.3.2 Risk Matrix . . . 16
- 4.4 Functional Requirements . . . 16
- 4.5 Technical Requirements . . . 17
- 4.6 Non-functional Requirements . . . 21
- 4.7 Implementation . . . 23
- 4.8 Testing . . . 24
- 4.9 Evolution . . . 25

5 User Interface and Features 27
- 5.1 Wireframes and User Journey . . . 27

6 Detailed System Design and Implementation 30
- 6.1 Backend Implementation (Django) . . . 30
  - 6.1.1 Project Structure . . . 30

6.1.2 API Design (DRF) . . . 31
6.1.3 Authentication and Authorization . . . 31
6.2 Frontend Implementation (Vanilla JS) . . . 31
6.2.1 SPA Architecture . . . 31
6.2.2 Rendering . . . 32
6.2.3 API Interaction . . . 32
6.3 Database Implementation . . . 32
6.3.1 PostgreSQL . . . 32
6.3.2 Ethereum Sidechain . . . 32
6.4 Real-time Communication . . . 32
6.5 DevOps and Containerization (Docker) . . . 33

7 Testing and Evaluation **34**
7.1 Testing Strategy . . . 34
7.2 Unit Testing . . . 34
7.3 Integration Testing . . . 35
7.4 End-to-End (E2E) Testing . . . 35
7.5 Security Testing . . . 36
7.6 Usability and Accessibility Testing . . . 36
7.7 Evaluation . . . 36

8 Blockchain Integration for Tournament Records **38**
8.1 Rationale for Blockchain Integration . . . 38
8.2 Technology Choices . . . 39
8.3 Smart Contract Design . . . 39
8.4 Backend Interaction (Django) . . . 40
8.5 Security Considerations . . . 40
8.6 Limitations and Alternatives . . . 41

9 Deployment and Operations **42**
9.1 Deployment Strategy . . . 42
9.2 Environment Configuration . . . 43
9.3 Web Server and Reverse Proxy (Nginx) . . . 43
9.4 Monitoring and Logging (ELK Stack) . . . 44
9.5 Maintenance Considerations . . . 44

Conclusion **45**

10 Appendix **46**
References . . . 46

List of Figures

3.1 Workflow 8

3.1 Gantt Chart 13
3.2 Risk Matrix showing probability vs. impact 16

List of Tables

4.1 Risk Register for the ft_transcendence Project 15

List of Abbreviations

2FA	Two-Factor Authentication
CI/CD	Continuous Integration and Continuous Delivery/Deployment
DOM	Document Object Model
DevOps	Development Operations
Git	Global Information Tracker
JWT	JSON Web Tokens
OAuth	Open Authorization
OTP	One-Time Password
SPA	Single Page Application
SDLC	Software Development Life Cycle
SSR	Server-Side Rendering
SQL	Structured Query Language
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
XSS	Cross-Site Scripting

Chapter 1

Introduction

This project, **ft_transcendence**, delivers a modern, feature-rich clone of the classic Pong game implemented as a full-stack web application. The primary goal is to provide a highly interactive and accessible multiplayer gaming experience directly within a web browser, leveraging a Single-Page Application (SPA) architecture. By incorporating features such as user history tracking, matchmaking, and tournament brackets, the platform enhances user engagement through competitive play and detailed gameplay statistics.

The system ensures that user data, match history, and tournament results are securely stored and managed. Standard gameplay data is persisted in a **PostgreSQL** database, while tournament outcomes are additionally recorded immutably on an **Ethereum sidechain**, showcasing a novel approach to data integrity in online gaming. The application is built using modern web technologies and DevOps best practices, aiming for a responsive, scalable, and secure platform where users can engage in real-time contests.

Key technical components include a **Django backend** utilizing the Django Rest Framework (DRF) to expose a clean RESTful API, and a **Vanilla JavaScript frontend** employing ES6 modules and Bootstrap 5 for a responsive user interface rendered on an HTML5 ‘¡canvas¿‘ element. The entire application stack, including the backend, frontend, database, blockchain node, and monitoring tools, is orchestrated using **Docker and Docker-Compose**, facilitated by **Nginx** serving as a reverse proxy and handling HTTPS termination. Centralized logging and monitoring are integrated via the **ELK stack** (Elasticsearch, Kibana).

Standard user management features are robust, offering secure registration and login via **native credentials** or **42 Intra OAuth 2.0**. Authentication relies on **JWT cookies**, with optional **Time-based One-Time Password (TOTP) Two-Factor Authentication (2FA)** for enhanced security. User profiles allow for customization (including avatar uploads and unique display names), display online status, manage friends lists, and provide insights into gameplay statistics such as wins,

losses, and match history. Access to detailed profiles and social features is restricted to logged-in users.

The platform offers diverse gameplay options, including **1-vs-1** matches against other online players, challenging an **AI opponent**, engaging in local **4-player multiplayer** matches, and participating in structured **tournaments** with persistent, on-chain results. Throughout development, emphasis was placed on performance, security, and inclusivity, including considerations for users with visual impairments and ensuring cross-browser compatibility.

This project aims to push the boundaries of web-based gaming applications by integrating real-time gameplay, secure authentication, blockchain technology, and comprehensive DevOps automation into a cohesive and engaging user experience.

Chapter 2

Project Goals and Requirements

The primary objective of the ft_transcendence project is to create a comprehensive and engaging web-based Pong game experience. This involves not only replicating the core gameplay but also integrating modern web technologies, robust security features, and advanced functionalities. The specific goals defined for this project are as follows:

- **Deliver a Single-Page Application (SPA):** The game must be playable directly in the browser without requiring page reloads, offering a seamless user experience.
- **Provide Secure User Management:** Implement a secure system for user registration, login, and profile management. This includes:
  - Native credential authentication (username/password).
  - Integration with the 42 Intra OAuth 2.0 provider.
  - Secure session management using JSON Web Tokens (JWT) stored in cookies.
  - Optional Time-based One-Time Password (TOTP) Two-Factor Authentication (2FA) for enhanced security.
- **Offer Multiple Game Modes:** Cater to different player preferences by providing several ways to play Pong:
  - 1 vs 1 online matches against other registered users.
  - 1 vs AI matches against a computer-controlled opponent.
  - Local 4-player multiplayer on a single screen.
  - Structured tournaments with bracket progression.
- **Persist Data Securely and Reliably:** Store user data, match history, and tournament results using appropriate technologies:

  - Utilize a **PostgreSQL** relational database for general application data, user profiles, and individual match results.
  - Record final tournament results immutably on an **Ethereum sidechain** (using a Proof-of-Authority consensus mechanism) to ensure data integrity and verifiability.

- **Expose a Clean RESTful API:** Develop a well-documented Application Programming Interface (API) using Django Rest Framework (DRF). This allows the frontend SPA to communicate with the backend and enables potential future development of alternative clients (e.g., command-line tools, mobile applications).

- **Implement Production-Grade DevOps Practices:** Ensure the application is easy to deploy, manage, and monitor using industry-standard tools and practices:
  - Containerize all application components (frontend, backend, database, etc.) using **Docker**.
  - Orchestrate the multi-container application stack using **Docker-Compose** for easy setup and deployment.
  - Utilize **Nginx** as a reverse proxy to handle incoming traffic, serve static frontend assets, and manage HTTPS termination.
  - Integrate centralized logging and monitoring using the **ELK stack** (Elasticsearch, Kibana) to provide insights into application performance and potential issues.
  - Provide convenient management commands using a **Makefile**.

These goals collectively aim to produce a polished, secure, and feature-rich web application that showcases proficiency in full-stack development, DevOps, and blockchain integration within the context of a familiar and engaging game.

Chapter 3

Software Development Life Cycle (SDLC)

This chapter outlines our approach to developing the project following the Software Development Life Cycle (SDLC) methodology. We cover the key phases including requirement analysis, design, implementation, testing, and evolution (maintenance). Each phase builds upon the previous one, ensuring a systematic and comprehensive development process.

The Software Development Life Cycle (SDLC) for this project follows a modified Waterfall model, incorporating Agile elements to enhance flexibility and iterative improvements. While the Waterfall approach provided a structured framework for sequential stages like requirement analysis, design, implementation, and testing, we introduced Agile practices such as ongoing feedback and peer-reviewed GitHub pull requests (PRs). This process of PR reviews allowed team members to give feedback on each other's code, identify potential issues early, and ensure quality and consistency across different modules. By integrating peer-reviewed PRs, we facilitated continuous testing, collaborative refinement, and adherence to best practices throughout the development. This hybrid methodology helped us achieve a balance of systematic progression with opportunities for iteration and improvement based on team insights.

3.1 Requirement Analysis

Requirement Analysis is the first and one of the most crucial phases of the Software Development Life Cycle (SDLC). During this phase, the focus is on gathering detailed requirements and defining the project's objectives. This phase helps ensure that the project starts on the right track, with a clear understanding of what needs to be built, why it is being built, and how it will be used.

Proper requirement analysis sets the foundation for all subsequent stages of the software development process. By understanding and addressing potential risks early, such as technical feasibility, resource constraints, or user expectations, the project can avoid delays or even a potential failure. Additionally, proper documentation and agreement on requirements ensure that all members are on the same page, reducing miscommunication and conflicts later. Starting with a solid foundation minimizes misunderstandings and scope changes and sets the stage for successful project execution. In order to properly gather the requirements, it was necessary to choose which modules best fit the team skills and interests. Initially some modules were set, and others were flagged as "interested", that the team would do according to the project development.

3.2 Modules Implemented

After a careful analysis, the following core modules and features were implemented, aligning with the project goals:

1. **Backend Framework:** Utilized Django (Python framework) to build a robust and scalable backend, providing RESTful APIs via Django Rest Framework (DRF).
2. **Frontend Implementation:** Developed a Single-Page Application (SPA) using Vanilla JavaScript (ES6+) for interactivity and HTML5 Canvas for game rendering. Leveraged Bootstrap 5 as a toolkit for responsive design and UI components.
3. **Database Integration:** Employed PostgreSQL as the primary relational database for storing user data, match history, and application settings, integrated via the Django ORM.
4. **User Management and Authentication:** Implemented comprehensive user management features including secure registration, profiles (with avatars and stats), and friend lists. Supported multiple authentication methods: native credentials, 42 Intra OAuth 2.0.
5. **Enhanced Security:** Integrated JWT (JSON Web Tokens) via `djangorestframework-simplejwt` for stateless session management (using secure HttpOnly cookies) and provided optional TOTP-based Two-Factor Authentication (2FA) using `django-otp`.
6. **Core Gameplay (Pong):** Developed the classic Pong game with various modes:
   - 1 vs 1 online matches.
   - 1 vs AI opponent.
   - Local multiplayer (up to 4 players).
   - Structured tournaments with bracket display.

7. **Blockchain for Tournaments:** Integrated an Ethereum sidechain (Proof-of-Authority) to immutably record tournament results using Solidity smart contracts and Web3.py interaction from the backend.

8. **Game Customization:** Provided options for users to customize certain aspects of their game experience (details may vary, e.g., potentially paddle appearance or simple settings).

9. **DevOps and Containerization:** Utilized Docker and Docker Compose to containerize all services (backend, frontend, database, Nginx, ELK, Geth) for consistent deployment and orchestration.

10. **Monitoring:** Integrated the ELK stack (Elasticsearch, Kibana) for centralized logging and application monitoring.

11. **Accessibility Considerations:** Included features to enhance accessibility, such as support for keyboard navigation and considerations for visually impaired users (e.g., high-contrast options).

3.3 Design the System Architecture

The Design Phase is the second major stage in the SDLC, where the focus transitions from defining system requirements to designing how the system will meet those requirements. The project architecture is divided into independent services, each running in Docker containers, to enhance scalability and maintainability.

For the database, `PostgreSQL` is selected to ensure data integrity and seamless integration with `Django`, while `Bootstrap` is utilized for creating a responsive and accessible UI design on the frontend.

In terms of security design, secure login methods, including 42 login or username, with Two-Factor Authentication (2FA) and JSON Web Tokens (JWT) for authentication, are implemented.

During this stage, the initial draft of the website workflow and the structure of page linkages were established in a group meeting. This draft outlined user navigation through the site, detailing the connections between different pages to ensure a coherent user experience. This early planning phase was crucial for aligning development efforts across various teams, such as frontend design, backend functionality, and game integration, setting the foundation for subsequent implementation stages.

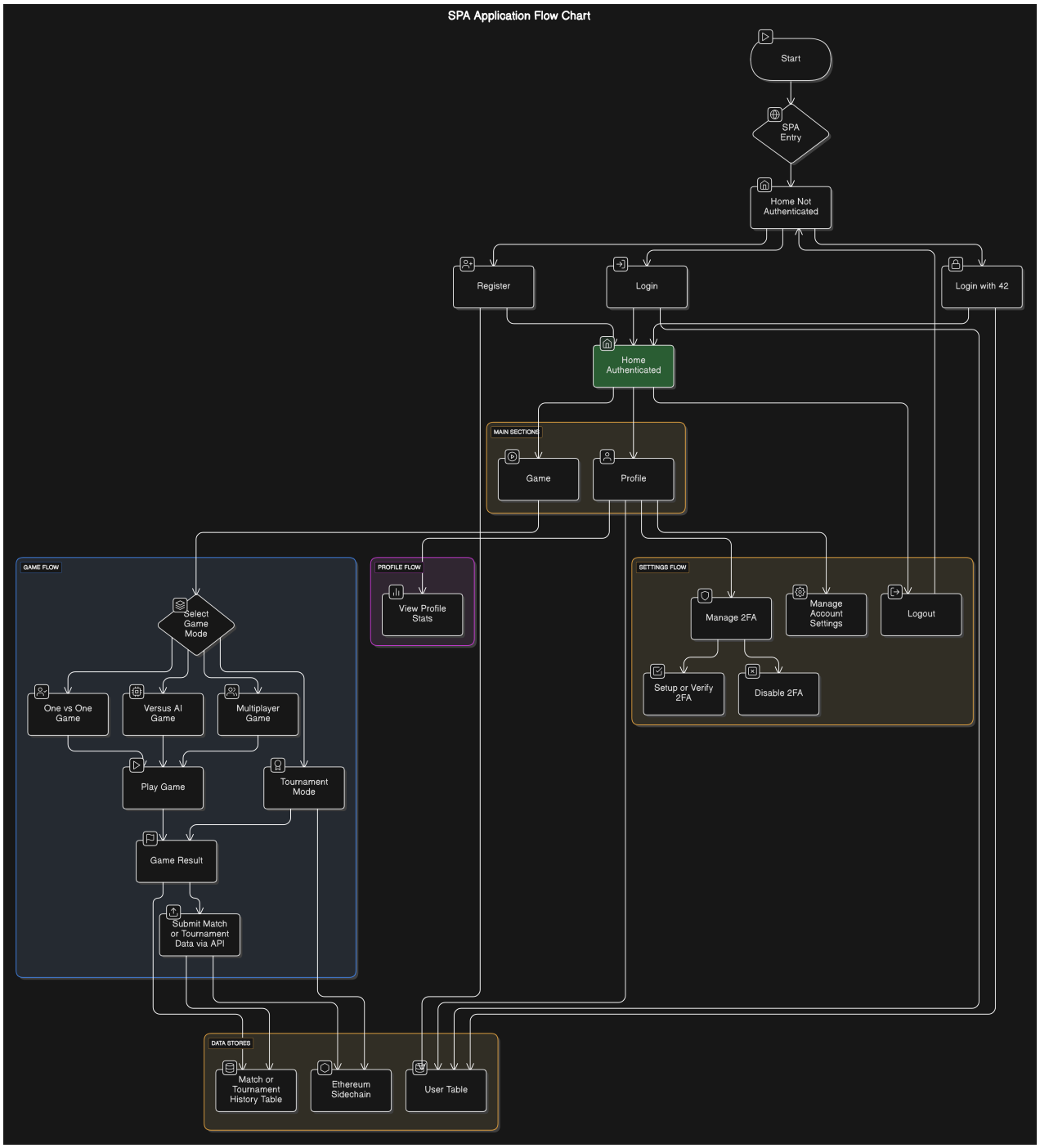

FIGURE 3.1: Workflow

3.4 Implementation

The implementation step involves developing the actual code and functionality based on the design specifications. For this web application project, the implementation step is broken down into the following main aspects:

- **Backend Development:** Development of the core functionality using Django, ensuring the backend supports all game features and security protocols. The backend must also manage user data securely, support multiple game modes, and provide the necessary APIs for frontend interaction. The website is safeguarded against SQL injection and Cross-Site Scripting (XSS) attacks through the inherent security features provided by Django. Specifically, Django mitigates the risk of SQL injection by escaping user inputs and offering secure methods for querying the database. The framework's built-in query functions, such as `.filter()`, `.get()`, and others, incorporate robust mechanisms to prevent the execution of malicious SQL code. Consequently, by using such functions, Django's automatic protection against common security vulnerabilities ensures a safer application environment.

- **Frontend Development:** Build a responsive user interface using Bootstrap, incorporating accessibility features such as high-contrast themes and keyboard navigation using the "tab" key. The frontend should adapt seamlessly across different devices and screen sizes, offering an inclusive experience for users, including those with visual impairments.

- **Game Development:** Implement game mechanics with customization options for different skill levels and settings. This includes developing interactive controls, configuring adjustable game parameters, and ensuring real-time responsiveness. The platform also introduced an additional game, Tic-Tac-Toe, distinct from Pong, to diversify the gaming options available. The game includes user history tracking to record and display individual gameplay statistics.

- **User History and Matchmaking:** Integrate a comprehensive user history tracking system to securely store game history and ensure it is always up to date. These additions aim to enhance user engagement by providing gameplay statistics and facilitating seamless matchmaking for an enjoyable experience.

- **Standard User Management and Authentication:** Implement secure user management features, allowing users to subscribe to the website, log in, and manage their profiles. Users can select unique display names for tournaments, update their information, and upload avatars. Additional features include managing friend lists, viewing online status, and accessing user profiles with stats like wins, losses, and match history. All user management functionalities should comply with best practices for secure authentication.

- To enable simultaneous work on different fronts, such as game development, backend development, and page design, a specific branch was used on GitHub for each task. This approach allowed team members to work independently while ensuring the integrity of the main branch. Before merging any changes into the main branch, a peer review process was conducted to maintain code quality and catch potential issues early. Figure 3.1 provides an illustration of this workflow, covering the period from October 21st to 24th, 2024, to demonstrate how multiple branches were utilized for collaborative development and quality assurance purposes.

3.5 Testing the application and its functionalities

Testing phase in the SDLC ensures that the web application is functional, secure, user-friendly, and meets all requirements. Focusing on Pong game web application, testing can be broken down into several types, each targeting different aspects of the system:

- **Unit Testing:** Test individual components of the application in isolation to ensure their functionality is correct. Examples include verifying user authentication mechanisms, game logic (like ball and paddle movements), and database interactions. Additionally, unit tests were conducted for the Tic-Tac-Toe game, such as validating game rules, user input, and match outcomes.
- **Integration Testing:** After unit testing, it is necessary to test the interactions between multiple components. Integration testing includes verifying the communication between the frontend and backend (e.g., submitting a login request and receiving the correct response, game management, authentication, user profiles), and ensuring data is correctly stored, retrieved, updated, and deleted from the PostgreSQL database. It was also necessary to perform tests to confirm that the user history and matchmaking system of the game were operating seamlessly, and data flows correctly across services.
- **Functional Testing:** The main goal is to verify that the application meets all functional requirements and works as intended. This is done by testing the game functionality (e.g., in the Pong game, issues like ball collision detection, paddle movement, multiplayer synchronization, adjusting ball and paddle speed, difficulty levels, and sound settings should be analyzed). For authentication and security, it is necessary to check if both login methods (42 login and username), 2FA setup, and JWT session management are properly working, as well as if a non-logged user can access the content restricted to a logged user. In addition, the logout option was also tested to ensure that the session was properly finished. Lastly, user management features like user registration, profile updates, password resets, and tournament participation were analyzed.
- **Usability Testing:** This next step evaluates the application's user interface and experience, ensuring it is intuitive and accessible for users. The main tests were focused on testing the web application on different devices (desktop, tablet, mobile) to ensure the UI is responsive and easy to use, and checking that all accessibility features (e.g., high-contrast colors, keyboard navigation) work correctly and comply with accessibility standards. After that, the focus shifted to gathering feedback from users on the game's interface, navigation, and overall experience to identify potential improvements.

- **Compatibility Testing:** The goal is to ensure the application functions consistently across different environments. For this application, the browsers “Chrome” (last available version) and “Firefox” were chosen, and in this sense, all tests were performed on both browsers.

In addition to the tests mentioned above, CI/CD was integrated as an automation tool within the GitHub (Figure 2) workflow to prevent commits that would fail the build. This approach proved highly effective, minimizing rework and ensuring that each merge met the minimum standards required for committing to the main branch.

Chapter 4

Elements of SDLC

4.1 SDLC Methodology

The Software Development Life Cycle (SDLC) for this project followed a hybrid approach, combining the structured framework of the Waterfall model with Agile elements for flexibility. While Waterfall provided sequential stages (requirement analysis, design, implementation, testing), Agile practices like ongoing feedback and peer-reviewed GitHub pull requests (PRs) were integrated. This PR review process allowed team members to assess code, identify issues early, and ensure quality and consistency. This hybrid methodology balanced systematic progression with iterative refinement based on team collaboration and insights.

4.2 Gantt Chart

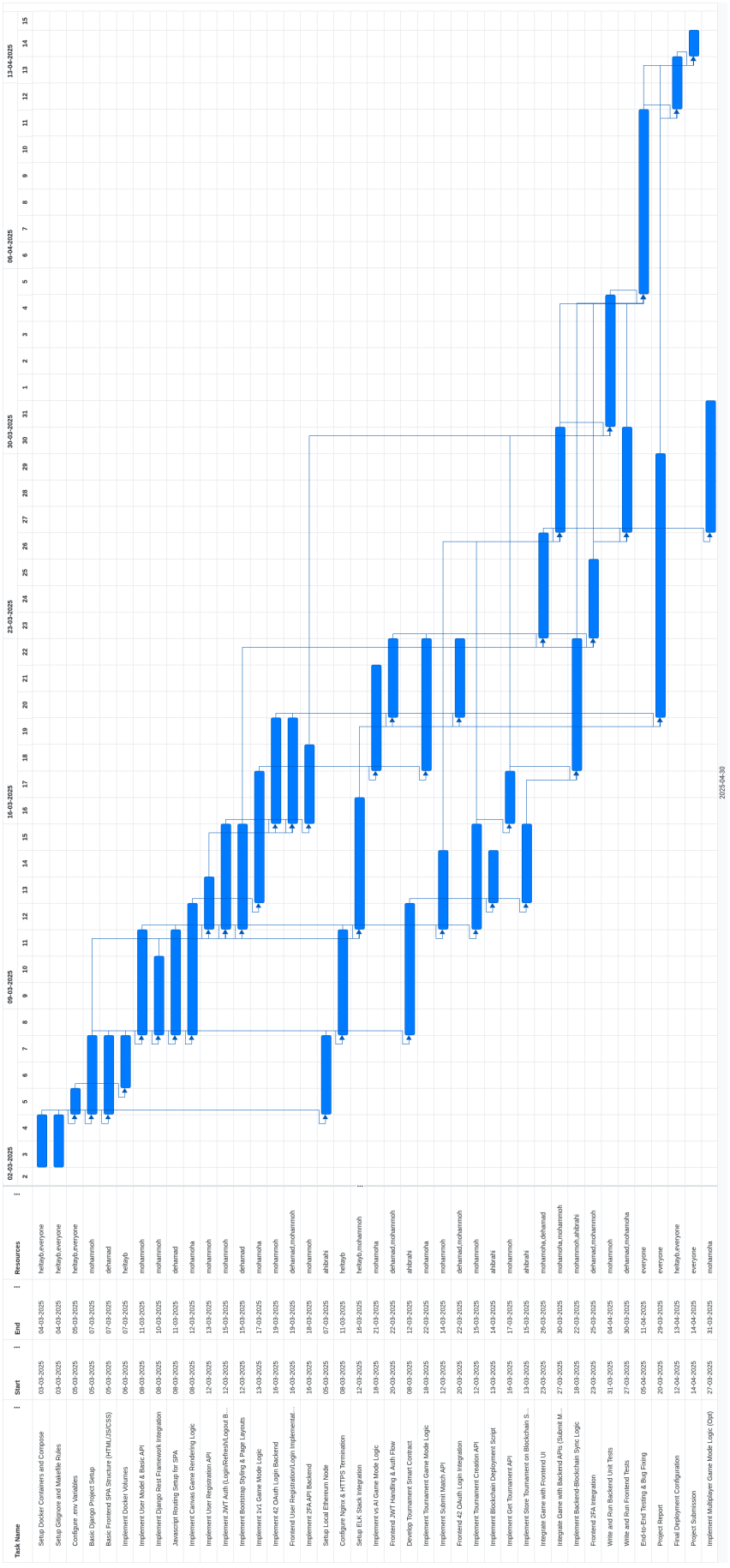

FIGURE 3.1: Gantt Chart

The Gantt Chart (Figure 3.1) delineates the project timeline, outlining tasks, dependencies, and milestones, thereby mapping the project's progression through each phase of the Software Development Life Cycle (SDLC).

This Gantt chart outlines the project timeline for **ft_transcendence**, showing tasks, their durations, and assigned resources. It spans from August to October 2024, covering development milestones like setting up the **Docker environment** with **Docker-Compose**, designing user interfaces using **Bootstrap 5**, and implementing essential features such as user authentication (**Native Credentials, 42 OAuth, JWT, TOTP 2FA**). The project includes building the main Pong game modes (**1v1, vs AI, 4-player local, Tournaments**), integrating **Blockchain** for tournament storage, setting up the **ELK stack** for monitoring, and implementing user features like friend management, match history, and account settings. Key phases involve testing, debugging, and updating, with the planned submission date due by October 29, 2024. Various team members are responsible for specific tasks, ensuring collaborative progress throughout the timeline.

4.3 Risk Register and Risk Matrix

4.3.1 Risk Register

To ensure project success, we identified and managed potential risks through a systematic risk assessment process. The Risk Register documents all identified risks, their impact, probability, and mitigation strategies.

ID	Risk Description	Probability	Impact	Mitigation Strategy	Owner
R1	Technical complexity of integrating blockchain	Medium	High	Early research and prototyping; dedicated team member	Development Team
R2	Real-time gameplay latency issues	High	High	Implement efficient algorithms, stress testing, optimize network code	Game Dev Team
R3	Security vulnerabilities in authentication	Medium	Critical	Code reviews, security best practices, JWT token handling	Security Team
R4	Docker orchestration complexity	Medium	Medium	Early setup, documentation, pair programming	DevOps Team
R5	Scope creep due to additional game features	High	Medium	Clear requirements, regular reviews, prioritization	Project Manager
R6	Integration issues between frontend and backend	Medium	High	API specification first, mock endpoints, continuous integration	Integration Team
R7	Team member unavailability	Low	Medium	Cross-training, documentation, distributed knowledge	Team Lead
R8	Browser compatibility issues	Medium	Medium	Cross-browser testing, progressive enhancement	UI/UX Team
R9	Performance issues with ELK stack	Low	Low	Monitoring, resource allocation, performance tuning	DevOps Team
R10	Incomplete or changing requirements	Medium	High	Regular stakeholder reviews, agile approach	Project Manager

Table 4.1: Risk Register for the ft_transcendence Project

4.3.2 Risk Matrix

The Risk Matrix provides a visual representation of the risks based on their probability and impact, helping prioritize risk mitigation efforts.

RISK MATRIX					
Probability	**Very High**	Low Risk	Moderate Risk	High Risk (R5)	Critical Risk
	High	Low Risk	Moderate Risk	High Risk (R2)	Critical Risk
	Medium	Low Risk (R9)	Moderate Risk (R4)	High Risk (R1, R6, R10)	Critical Risk (R3)
	Low	Low Risk	Low Risk (R7)	Moderate Risk (R8)	High Risk
	Very Low	Low Risk	Low Risk	Low Risk	Moderate Risk
		Low	**Medium**	**High**	**Critical**
		Impact			

FIGURE 3.2: Risk Matrix showing probability vs. impact

4.4 Functional Requirements

User Authentication and Management

- Secure registration, login, and profile management.
- Support for multiple authentication mechanisms: Native username/password, 42 Intra OAuth 2.0, and optional TOTP-based Two-Factor Authentication (2FA).
- User profile management features, including unique display names, avatar uploads, and online status visibility.
- Friends list management (add, remove, view status).

Gameplay and Game Modes

- Core Pong gameplay implemented on HTML5 Canvas.
- Multiple game modes:
  - 1 vs 1: Online match against another registered user.
  - 1 vs AI: Match against a computer-controlled opponent.
  - Local Multiplayer: Up to 4 players on the same screen.
  - Tournament: Knockout bracket competition among registered users.

Match History and Statistics

- Tracking and storing user gameplay statistics, including wins, losses.
- Displaying match history for logged-in users.
- Recording tournament results in PostgreSQL and immutably on the Ethereum sidechain.

Accessibility Features

- Considerations for visually impaired users (e.g., high-contrast options, clear UI elements).
- Keyboard navigation support for gameplay and interface interaction.

4.5 Technical Requirements

Software Specifications

Operating System

- **Server Deployment**: Linux-based distribution (e.g., Debian, Ubuntu) is recommended for hosting Docker containers.
- **Development/Client**: Cross-platform compatibility (Linux, macOS, Windows) for development and playing via supported web browsers.

Programming Languages, Frameworks, and Libraries

- **Backend**: Python 3 with Django 5 framework. Django Rest Framework (DRF) for API development. Libraries include ‘djangorestframework-simplejwt‘ for JWT authentication and ‘django-otp‘ for 2FA.
- **Frontend**: Vanilla JavaScript (ES6+), HTML5, CSS3. Bootstrap 5 for responsive UI components and layout.
- **Blockchain Interaction**: Web3.py library for interacting with the Ethereum sidechain from the backend (or helper scripts).

Database and Data Storage

- **Primary Database**: PostgreSQL (Version 15 specified) for storing user data, match history, game settings, etc.
- **Blockchain**: Local Ethereum Proof-of-Authority (POA) sidechain (using Geth) for immutable storage of tournament results. Smart contracts developed in Solidity (details likely in 'Blockchain/' directory).

Web Server and Reverse Proxy

- Nginx: Used as a reverse proxy to route traffic to the Django backend (via Gunicorn or similar WSGI server), serve static frontend files (JS, CSS, images), and handle TLS/SSL termination for HTTPS.

Containerization and Orchestration

- Docker: To containerize each service (frontend, backend, database, Nginx, ELK, blockchain node).
- Docker-Compose: To define and manage the multi-container application stack.

Authentication and Security Mechanisms

- JSON Web Tokens (JWT): Used for stateless session management via secure HTTP-only cookies.
- Two-Factor Authentication (2FA): TOTP-based 2FA support via 'django-otp'.
- OAuth 2.0: Integration with 42 Intra's OAuth service.
- TLS/SSL: Encryption for all communication via HTTPS, handled by Nginx.

Monitoring and Logging

- ELK Stack: Elasticsearch (Version 7 specified) for log aggregation and indexing, Kibana (Version 8 specified) for visualization and analysis. Log shipping likely handled by Filebeat or integrated Django logging.

Hardware Specifications

Server Requirements (Development/Small Scale)

- Processor: Modern multi-core CPU.
- Memory: Minimum 8 GB RAM (16 GB+ recommended, especially with ELK stack).
- Storage: SSD recommended (minimum 50-100 GB, depending on data volume and logs).
- Network: Standard internet connectivity.

Note: Production requirements would depend heavily on user load.

Client Requirements

- Device: Desktop, laptop with modern web browser.
- Browser: Latest versions of Chrome or Firefox recommended.
- Network: Stable internet connection for real-time gameplay.

System Architecture

Client Layer (Frontend SPA)

The frontend is a Single-Page Application (SPA) responsible for rendering the user interface, handling user interactions, and managing client-side game logic. Technologies:

- Framework/Libraries: Vanilla JavaScript (ES6+), Bootstrap 5.
- Structure: HTML templates, CSS3 for styling.
- Rendering: Client-side rendering. Game logic executed directly in the browser, primarily using the HTML5 Canvas API for Pong gameplay.
- API Communication: Uses Fetch API or similar to interact with the backend REST API for data retrieval, user actions, and submitting game results.
- Authentication Handling: Manages JWT tokens received from the backend (stored securely, likely in HttpOnly cookies handled by the browser).

Application Layer (Backend API)

The backend provides the RESTful API, manages business logic, handles user authentication, interacts with the database and blockchain, and integrates with monitoring systems. Technologies:

- Framework: Django 5 (Python 3).
- API: Django Rest Framework (DRF) for creating API endpoints.
- Authentication: Handles user registration, login (native, OAuth), JWT generation/validation (‘djangorestframework-simplejwt‘), and 2FA logic (‘django-otp‘).
- Database Interaction: Uses Django ORM to communicate with the PostgreSQL database.
- Blockchain Interaction: May trigger helper scripts (‘Web3.py‘) to interact with the Ethereum sidechain for storing tournament data.
- Logging: Integrates with the ELK stack for centralized logging.
- WSGI Server: Typically runs behind Nginx using a WSGI server like Gunicorn (though not explicitly mentioned in README, it’s standard for Django deployment).

Data Layer (Persistence)

The data layer ensures persistent storage of application data using both relational and blockchain technologies. Technologies:

- Relational Database: PostgreSQL 15 for user accounts, profiles, settings, individual match history, etc.
- Blockchain Database: Ethereum sidechain (Geth node running POA) for immutable recording of official tournament outcomes via smart contracts.

Infrastructure Layer (Orchestration and Hosting)

This layer encompasses the tools and services used for deployment, management, and operation of the application stack. Technologies:

- Containerization: Docker packages each component (frontend, backend, Postgres, Nginx, ELK, Geth) into isolated containers.
- Orchestration: Docker-Compose defines the relationships and configurations for running the multi-container application.

- Reverse Proxy / Web Server: Nginx manages incoming HTTPS traffic, routes requests to the appropriate backend service, serves static frontend assets, and handles SSL/TLS termination.
- Monitoring: ELK Stack (Elasticsearch, Kibana) provides infrastructure for log aggregation and analysis.

Communication and Networking

- Internal Communication: Services within the Docker network communicate over a private network defined by Docker-Compose.
- External Communication: All external client communication occurs over HTTPS (port 443), managed by Nginx.
- API Calls: Frontend communicates with the backend via RESTful HTTP requests to the API endpoints.
- Blockchain Calls: Backend (or scripts triggered by it) communicates with the Geth node via RPC calls (typically over HTTP or IPC).

4.6 Non-functional Requirements

Non-functional requirements define the quality attributes and constraints of the system.

Performance

- **Responsiveness**: The SPA frontend should provide a smooth user experience with minimal delays during navigation and interaction. Game rendering on the Canvas should be efficient to maintain playable frame rates.
- **Latency**: Network latency for real-time 1v1 games should be minimized, although the current implementation seems focused on client-side rendering with results submitted post-game. API response times should be reasonably fast.
- **Load Handling**: The system (especially Nginx and the Django backend) should be capable of handling a moderate number of concurrent users, scalable via Docker if needed.

Security

- **Authentication**: Secure mechanisms for login (native, 42 OAuth) and session management (JWT over HTTPS, HttpOnly cookies).

- **Authorization**: Proper checks to ensure users can only access their own data and perform actions they are permitted to.
- **Data Protection**: Use of HTTPS (TLS/SSL) for all external communication. Sensitive data in the database should be handled securely (e.g., password hashing via Django). Input validation to prevent common web vulnerabilities (XSS, SQL Injection) provided by Django/DRF.
- **Infrastructure Security**: Containers should run with non-root users. Network policies within Docker can restrict communication between containers.
- **2FA**: Optional TOTP-based 2FA provides an additional security layer.

Reliability

- **Availability**: The application should be available with minimal downtime. Docker and Nginx help manage service availability.
- **Data Integrity**: PostgreSQL ensures relational data integrity. Blockchain provides immutability for tournament records.
- **Fault Tolerance**: Containerized architecture allows individual services to be restarted if they fail. Error handling should be robust in both frontend and backend.
- **Backup and Recovery**: Regular backups of the PostgreSQL database are crucial (mentioned as a mitigation strategy).

Scalability

- **Component Scaling**: Docker-Compose allows scaling individual services (e.g., adding more backend instances) if necessary, although load balancing setup might need adjustment.
- **Database Scaling**: PostgreSQL offers various scaling options if needed in the future.
- **Stateless Backend**: Using JWT promotes a stateless backend architecture, which generally scales better horizontally.

Maintainability

- **Modularity**: The project is divided into distinct components (Frontend, Backend, Blockchain, Infrastructure) and containerized services, promoting separation of concerns.
- **Code Quality**: Adherence to coding standards and use of frameworks (Django, Bootstrap) aids maintainability.

- **Configuration Management**: Environment variables ('.env' file) are used for configuration, separating code from configuration.
- **DevOps Automation**: Makefile and Docker-Compose simplify build, deployment, and management tasks.

Usability and Accessibility

- **User Interface**: The UI, built with Bootstrap, should be intuitive and easy to navigate.
- **Responsiveness**: The application should adapt to different screen sizes (desktops primarily, given the game type).
- **Accessibility**: Conscious effort to include features for visually impaired users (high contrast, keyboard navigation) as stated in goals.

4.7 Implementation

The implementation phase involved developing the actual code and functionality based on the design specifications. Key aspects included:

- **Backend Development:** Core functionality was built using Django and Django Rest Framework (DRF), providing a secure RESTful API. This included managing user data (PostgreSQL), handling authentication (Native, 42 OAuth, JWT, 2FA), supporting multiple Pong game modes, and ensuring security against common vulnerabilities like SQL injection and XSS through Django's built-in features.
- **Frontend Development:** A responsive Single-Page Application (SPA) interface was built using Vanilla JavaScript, HTML5, and CSS3, leveraging Bootstrap 5 for layout and components. Accessibility features like high-contrast themes and keyboard navigation were incorporated. The frontend interacts with the backend API and uses the HTML5 Canvas for rendering the Pong game dynamically.
- **Game Development:** Pong game mechanics were implemented using JavaScript and Canvas, including ball/paddle physics, scoring, and real-time updates (potentially via WebSockets, although not explicitly detailed in prior context). Game modes (1v1, vs AI, 4-player local, Tournament) were developed.
- **Blockchain Integration:** Smart contracts (Solidity) were developed and deployed to a local Ethereum sidechain (Geth) to immutably record tournament results. Backend interaction was handled using Web3.py.

- **User History and Statistics:** A system was integrated to track and display user match history and statistics (wins/losses), stored primarily in PostgreSQL.
- **User Management and Authentication:** Secure user registration, login (Native, 42 OAuth), profile management (display name, avatar), friends list, and TOTP 2FA were implemented. JWTs, managed via secure HttpOnly cookies, were used for session management.
- **DevOps and Deployment:** The entire application (Backend, Frontend, PostgreSQL, Nginx, ELK, Geth node) was containerized using Docker and orchestrated with Docker Compose. Nginx served as a reverse proxy and static file server. The ELK stack was configured for centralized logging and monitoring.
- **Collaborative Workflow:** GitHub branches were used for parallel development (e.g., backend, frontend, game features). Peer reviews were conducted on Pull Requests before merging to the main branch to maintain code quality.

4.8 Testing

The testing phase ensured the application's functionality, security, usability, and adherence to requirements. Key testing types included:

- **Unit Testing:** Verifying individual components like authentication logic, specific API endpoints, Canvas rendering functions, and game rule implementations (e.g., ball collision, scoring) in isolation.
- **Integration Testing:** Testing interactions between components, such as frontend API calls to the backend for login/registration, fetching user profiles, submitting game results, database interactions (PostgreSQL CRUD operations), and backend communication with the Ethereum sidechain.
- **Functional Testing:** Validating end-to-end features against requirements. This involved testing all Pong game modes, user registration/login flows (Native, OAuth, 2FA), profile updates, friend management, tournament creation/participation, and ensuring restricted content access controls worked correctly.
- **Usability Testing:** Evaluating the user interface and experience for intuitiveness and accessibility. Testing responsiveness across different screen sizes (desktop, tablet, mobile) using Bootstrap, checking keyboard navigation, and verifying accessibility features for visually impaired users.
- **Compatibility Testing:** Ensuring consistent functionality across target browsers (latest Chrome and Firefox).

- **Security Testing:** Reviewing implementations for common web vulnerabilities (though specific penetration testing might not have been performed), verifying JWT handling, 2FA enforcement, and general adherence to security best practices mentioned in mitigation strategies.
- **CI/CD Integration:** Using GitHub Actions to automate checks (e.g., linters, basic tests) to prevent merging failing code into the main branch.

4.9 Evolution

This phase focuses on maintaining and enhancing the software post-initial deployment.

Backend Development

- **Initial Phase:** Focused on core Django/DRF setup, basic user auth, and foundational API endpoints for Pong.
- **Intermediate Phase:** Expanded to support multiple game modes, integrated 42 OAuth and 2FA, developed blockchain interaction logic (Web3.py), and refined APIs based on frontend needs.
- **Current Phase:** Provides a comprehensive API managing users, game logic, statistics, tournaments (with blockchain logging), and authentication, running within a Dockerized environment integrated with ELK.

Frontend Development

- **Initial Phase:** Basic SPA structure with Vanilla JS, Bootstrap layout, initial Canvas implementation for Pong.
- **Intermediate Phase:** Refined UI/UX based on feedback, improved responsiveness, enhanced Canvas rendering, integrated API calls for dynamic data, added accessibility features.
- **Current Phase:** Delivers an interactive SPA experience with real-time updates, customizable settings, smooth navigation, robust game interface via Canvas, and accessibility considerations.

Game Development

- **Initial Phase:** Developed core Pong mechanics on Canvas (ball/paddle movement, collision, scoring).

- **Intermediate Phase:** Added different game modes (1v1, AI, 4-player, Tournament), improved real-time responsiveness, added basic customization.
- **Current Phase:** Offers multiple Pong game modes integrated seamlessly within the SPA, with results feeding into the user history and tournament system.

User History and Statistics

- **Initial Phase:** Basic recording of game outcomes to PostgreSQL.
- **Intermediate Phase:** Expanded tracking to include more details (opponents, scores, dates), developed API endpoints for retrieval.
- **Current Phase:** Provides comprehensive user statistics and match history accessible via the user profile, with tournament results also logged immutably on the blockchain.

User Management and Authentication

- **Initial Phase:** Implemented native registration/login, basic profile management.
- **Intermediate Phase:** Added 42 OAuth integration, JWT implementation, friend list functionality, and initial 2FA setup.
- **Current Phase:** Offers a robust system with multiple auth methods (Native, OAuth, 2FA), secure session management (JWT via HttpOnly cookies), comprehensive profile features (avatar, stats), and friend management.

Chapter 5

User Interface and Features

5.1 Wireframes and User Journey

Wireframes provide a visual outline of the project's user interface structure and play a key role in mapping the user journey. They enable early visualization of layout and user interactions, offering insight into user navigation within the system well before the detailed design and coding phases.
In the **ft_transcendence** project, wireframes and key website screenshots illustrate the user journey. The initial screen allows the user to log in using registered credentials or via their 42 Intra account. New users can register by providing a unique username, a valid email address, and a password. Basic input validation is performed during registration.

Once authenticated (either via native login or 42 OAuth), the user is directed to the main dashboard or game selection screen. From here, the user can choose from several Pong game modes: playing a **1 vs 1** match against another online user, playing against an **AI opponent**, engaging in a **4-player local multiplayer** match, or participating in a **Tournament**. Users can also access their profile and manage friends.

Selecting a game mode like 1 vs 1 might involve matchmaking or inviting an opponent. Upon starting a match, the Pong game interface is displayed. Player names (or identifiers) are shown, often near indicators for their keyboard controls (e.g., W/S for one player, Up/Down arrows for another). The interface typically includes buttons to return to the main menu ('home'), toggle accessibility options (like a visual impairment mode), access game settings, and view game rules.

If the user activates the visual impairment mode, the display adjusts accordingly, often using high-contrast colors and potentially adapting the appearance of the ball, paddles, and interface elements for better visibility.

When a match concludes (e.g., a player reaches the target score), a game over screen is displayed, indicating the winner. Options are typically presented to return to the main menu or play again (rematch).

The 4-player local mode would adapt the interface to accommodate four paddles and potentially different control schemes.

Game settings might be accessible via an icon (e.g., a gear). These settings could potentially include options like adjusting AI difficulty (for 'vs AI' mode) or toggling sound effects.

A help or rules icon might display a popup explaining the basic rules of Pong.

For **Tournaments**, the interface would guide users through the bracket. This might involve an initial screen to sign up or view the tournament structure. As matches are played, the bracket view would update to show winners advancing. The final tournament results are recorded in the PostgreSQL database and also immutably on the Ethereum sidechain.

Back on the main dashboard or via a dedicated section, users can manage their **Friends List**. This involves searching for other users by username and sending friend requests. Appropriate feedback is given if a user does not exist.

When a user receives a friend request, they are notified (e.g., on their dashboard or friends page) and can choose to accept or reject the request.

Users can access their **Profile** page, typically via an icon in the header. The profile displays gameplay statistics such as wins and losses, and a history of recent matches played. Users can also edit their profile information, which may include changing their display name or uploading a new avatar.

Account Settings provide options for managing security and account details. Users can enable or disable Two-Factor Authentication (2FA) and change their account password. Setting up 2FA typically involves scanning a QR code with an authenticator app (like Google Authenticator) and verifying a TOTP code.

Finally, an **About** page may provide information about the ft_transcendence project, the game of Pong, and the development team, potentially including links to their GitHub profiles.

Chapter 6

Detailed System Design and Implementation

This chapter delves into the specific implementation details of the core components of the **ft_transcendence** application, expanding on the architectural overview provided previously. It covers the backend logic, frontend structure, data persistence mechanisms, real-time communication, and the containerization strategy.

6.1 Backend Implementation (Django)

The backend is built using the Django framework (version 5), leveraging its robust features for web development and the Django Rest Framework (DRF) for creating the RESTful API.

6.1.1 Project Structure

The Django project follows a standard layout, organized into distinct apps for modularity:

- `users`: Handles user registration, authentication (native credentials, 42 OAuth), profile management, and friend relationships.
- `game`: Contains the logic for Pong gameplay, including game state management, rules enforcement, and matchmaking (for 1v1 online).
- `tournaments`: Manages the creation, progression, and recording of tournament brackets.
- `chat`: (If implemented) Would handle real-time chat functionalities.

6.1.2 API Design (DRF)

The API provides endpoints for frontend interaction. DRF's ViewSets and Serializers are used to handle data validation, database interaction, and JSON response generation. Key endpoints likely include:

- User authentication and registration.
- Profile viewing and editing (including avatar upload).
- Friend request management.
- Game initiation and state updates.
- Tournament creation and status retrieval.
- Match history access.

6.1.3 Authentication and Authorization

Security is managed through multiple layers:

- **Native Credentials:** Standard username/password login.
- **42 OAuth:** Integration with the 42 Intra's OAuth2 system for seamless login for 42 students.
- **JWT (JSON Web Tokens):** Simple-JWT library is used to issue JWTs stored in secure HTTP-only cookies upon successful login, authenticating subsequent API requests.
- **2FA (TOTP):** Optional Time-based One-Time Password (TOTP) using `django-otp` provides an additional security layer.

6.2 Frontend Implementation (Vanilla JS)

The frontend is a Single Page Application (SPA) built entirely with vanilla JavaScript (ES6+), HTML5, and styled with Bootstrap 5. This approach avoids reliance on heavy frontend frameworks.

6.2.1 SPA Architecture

JavaScript manages routing (likely using the History API or hash-based routing) to dynamically load content and views without full page reloads. State management might be handled through simple JavaScript objects or potentially a lightweight custom solution.

6.2.2 Rendering

- **Game Interface:** The core Pong gameplay (paddles, ball, score) is rendered dynamically on an HTML5 `<canvas>` element, providing fine-grained control over animations and interactions.
- **UI Elements:** Other interface components (dashboard, login forms, profile page, menus) are built using standard HTML elements manipulated via the Document Object Model (DOM) by JavaScript. Bootstrap provides styling and layout components.

6.2.3 API Interaction

The frontend communicates with the Django backend via asynchronous JavaScript (e.g., using the `fetch` API) to send requests to the RESTful API endpoints and receive data in JSON format. Real-time updates (discussed below) are handled separately.

6.3 Database Implementation

Data persistence relies on two distinct systems:

6.3.1 PostgreSQL

A PostgreSQL (version 15) relational database serves as the primary datastore for user accounts, profiles, friend relationships, game settings, and detailed match history. Django's ORM (Object-Relational Mapper) facilitates interaction with the database.

6.3.2 Ethereum Sidechain

For tournament results, an Ethereum-compatible sidechain (potentially using Proof of Authority) is employed to ensure immutable and verifiable record-keeping. The Django backend interacts with a deployed smart contract via the `Web3.py` library to record the winners and potentially other key tournament data.

6.4 Real-time Communication

Real-time functionality, crucial for interactive gameplay and potentially chat, is likely implemented using **WebSockets**. Django Channels is the probable framework used within Django to handle

WebSocket connections, allowing bidirectional communication between the server and connected clients for instant game state synchronization and message passing.

6.5 DevOps and Containerization (Docker)

The entire application stack is containerized using Docker and orchestrated with Docker Compose, facilitating consistent development, testing, and deployment environments.

The `docker-compose.yml` file defines the services:

- **Backend:** Runs the Django application (likely using Gunicorn or Uvicorn).
- **Frontend/Nginx:** An Nginx container serves the static frontend files (HTML, CSS, JS) and acts as a reverse proxy for the Django backend. It also handles TLS termination for HTTPS.
- **Database:** A PostgreSQL container manages the relational data.
- **ELK Stack:** Separate containers for Elasticsearch and Kibana handle centralized logging and monitoring.

Makefiles are likely used to streamline common Docker operations like building images, starting/stopping containers, and running management commands.

Chapter 7

Testing and Evaluation

Ensuring the quality, reliability, and security of the **ft_transcendence** application is paramount. This chapter details the testing strategies employed throughout the development lifecycle and provides an evaluation of the project against its initial objectives.

7.1 Testing Strategy

A multi-layered testing approach was adopted to verify different aspects of the application:

- **Unit Testing:** Focused on verifying the correctness of individual components (functions, classes) in isolation.
- **Integration Testing:** Examined the interactions between different modules and services (e.g., backend API with database, frontend with backend).
- **End-to-End (E2E) Testing:** Validated complete user workflows from the user's perspective.
- **Security Testing:** Assessed the application's resilience against common web vulnerabilities and ensured security features function correctly.
- **Usability and Accessibility Testing:** Evaluated the user-friendliness of the interface and the effectiveness of accessibility features.

7.2 Unit Testing

Unit tests formed the foundation of the testing pyramid, providing rapid feedback on code changes.

- **Backend (Django):** Django's built-in testing framework was likely utilized to write tests for models (database logic), views (API endpoint logic), serializers (data validation), and utility functions. Mocking was used where necessary to isolate units from external dependencies like the database or external APIs (e.g., 42 OAuth during testing).
- **Frontend (Vanilla JS):** Unit testing vanilla JavaScript can be challenging. While specific tools might not have been mandated, critical utility functions or logic components might have been tested using a framework like Jest or through manual verification during development.

7.3 Integration Testing

Integration tests focused on verifying the correct communication and data flow between different parts of the system.

- **API Level:** Tests were conducted to ensure that frontend requests to the Django REST Framework API endpoints resulted in the correct backend actions (e.g., database updates) and generated the expected responses (status codes, data formats).
- **Database Interaction:** Verified that the Django ORM correctly interacted with the PostgreSQL database, ensuring data integrity and correct query execution.
- **Blockchain Interaction:** Tested the backend's ability to correctly interact with the Ethereum sidechain smart contract via Web3.py for recording tournament results.

7.4 End-to-End (E2E) Testing

E2E tests simulated real user scenarios across the entire application stack. Due to the complexity of setting up automated browser tests, particularly for canvas-based games, E2E testing likely relied heavily on structured **manual testing**. Key workflows tested included:

- User registration (native and 42 OAuth) and login.
- Navigating the SPA and accessing different sections.
- Initiating and completing various Pong game modes (1v1, vs AI, 4-player).
- Participating in and completing a tournament.
- Managing profile information and avatar uploads.
- Adding and accepting friend requests.
- Enabling and using 2FA.

7.5 Security Testing

Given the focus on security, specific tests and checks were performed:

- **Authentication/Authorization:** Verified that only authenticated users could access protected endpoints and that authorization rules (e.g., editing own profile) were enforced. Tested JWT handling (cookie security, expiration) and 2FA logic.
- **Input Validation:** Checked for proper handling of user inputs to prevent common vulnerabilities like Cross-Site Scripting (XSS) and SQL Injection. Django's ORM and DRF serializers provide significant built-in protection.
- **Dependency Scanning:** Tools might have been used to scan project dependencies for known vulnerabilities.
- **HTTPS Enforcement:** Verified that Nginx correctly enforces HTTPS connections.

7.6 Usability and Accessibility Testing

- **Usability:** Informal usability testing was likely conducted throughout development, gathering feedback on the ease of navigation, clarity of instructions, and overall user experience.
- **Accessibility:** The visual impairment mode was specifically tested to ensure it provided sufficient contrast and visibility improvements for users with visual challenges. General accessibility principles (e.g., keyboard navigation, semantic HTML where applicable outside the canvas) were considered.

7.7 Evaluation

Evaluating the ft_transcendence project against the goals defined in Chapter 2:
Successes:

- The core goal of delivering a functional Pong game as an SPA was achieved.
- Secure user management with native credentials, 42 OAuth, and 2FA was successfully implemented.
- Multiple game modes (1v1, vs AI, 4-player, Tournaments) were developed.
- Data persistence using both PostgreSQL and an Ethereum sidechain was realized.

- A RESTful API using DRF was exposed.
- Production-grade DevOps practices using Docker, Nginx, and ELK were implemented, demonstrating containerization and monitoring capabilities.
- Key features like profiles, friends list, and match history enhance user engagement.
- Accessibility considerations were included.

Potential Areas for Improvement/Future Work:

- Implementation of automated E2E testing for more comprehensive regression testing.
- More extensive performance testing under load.
- Potential addition of features like real-time chat (if not fully implemented).
- Further refinement of the AI opponent's difficulty levels or strategies.

Overall, the project successfully demonstrates the integration of diverse technologies to create a modern, secure, and feature-rich web application, fulfilling the primary requirements outlined.

Chapter 8

Blockchain Integration for Tournament Records

An innovative feature of the **ft_transcendence** project is the integration of blockchain technology to immutably record the outcomes of official Pong tournaments. This chapter details the rationale, design, and implementation of this integration.

8.1 Rationale for Blockchain Integration

While the primary database (PostgreSQL) stores comprehensive match and tournament data, utilizing blockchain offers distinct advantages for tournament results:

- **Immutability:** Once recorded on the blockchain, tournament results become extremely difficult to alter or delete, providing a tamper-proof historical record.
- **Verifiability:** Anyone with access to the blockchain network (even if it's a private or consortium sidechain) can potentially verify the recorded results independently, enhancing transparency.
- **Decentralization (Conceptual):** Although potentially deployed on a controlled sidechain, it introduces the concept of decentralized record-keeping, contrasting with the centralized nature of the primary database.

This serves as both a technical showcase and a method to guarantee the integrity of high-stakes tournament outcomes within the game's ecosystem.

8.2 Technology Choices

- **Ethereum Sidechain:** An Ethereum-compatible sidechain was chosen rather than the mainnet, likely due to cost (gas fees) and performance considerations. A Proof of Authority (PoA) consensus mechanism might be used for controlled environments, offering faster transaction times and lower energy consumption compared to Proof of Work.
- **Smart Contract (Solidity):** The logic for recording and potentially retrieving tournament data is encapsulated in a smart contract written in Solidity, the standard language for Ethereum-based development.
- **Web3.py:** The Django backend interacts with the deployed smart contract on the sidechain using the Python library `Web3.py`. This library allows the backend to connect to an Ethereum node, load the contract's Application Binary Interface (ABI), and call its functions.

8.3 Smart Contract Design

The smart contract is designed to be simple yet effective for its specific purpose. Key aspects include:

- **State Variables:** Stores essential information, likely mapping a tournament identifier to the winner's user ID and perhaps a timestamp. Example: `mapping(uint256 => TournamentResult) public tournamentResults;`
- **Structs:** A struct (e.g., `TournamentResult`) might be used to group related data: `struct TournamentResult  uint256 winnerUserId; uint256 timestamp;`
- **Functions:**
  - `recordTournament(uint256 tournamentId, uint256 winnerUserId)`: A function callable only by an authorized address (likely the backend server's wallet) to record the outcome of a completed tournament. It would populate the state variables.
  - `getTournamentWinner(uint256 tournamentId) returns (uint256)`: A public view function to retrieve the winner of a specific tournament ID from the stored data.
- **Events:** An event (e.g., `event TournamentRecorded(uint256 indexed tournamentId, uint256 indexed winnerUserId);`) is likely emitted when a tournament is recorded, allowing off-chain applications or indexers to easily track new records.

8.4 Backend Interaction (Django)

The Django backend orchestrates the interaction with the smart contract:

1. **Connection Setup:** Using `Web3.py`, the backend connects to an accessible node of the Ethereum sidechain (specified via its RPC URL).
2. **Contract Loading:** The backend loads the smart contract's ABI and address.
3. **Transaction Trigger:** Upon the confirmed conclusion of a tournament within the application logic, the backend prepares to call the `recordTournament` function.
4. **Signing and Sending:** The backend uses its configured private key to sign the transaction and sends it to the sidechain network via `Web3.py`. This requires the backend's wallet address to have sufficient funds (native sidechain currency) to cover any transaction gas fees, even if minimal on a PoA chain.
5. **Confirmation Handling:** The backend might wait for transaction confirmation or handle it asynchronously, potentially updating the local PostgreSQL database status once the blockchain record is confirmed.
6. **Reading Data (Optional):** The backend could also use `Web3.py` to call view functions like `getTournamentWinner` if needed, though primary data retrieval likely still relies on PostgreSQL for efficiency.

8.5 Security Considerations

Implementing blockchain integration requires careful security management:

- **Smart Contract Security:** The Solidity code needs auditing for common vulnerabilities (e.g., reentrancy, integer overflow/underflow), although the contract's simplicity reduces the attack surface. Access control on functions like `recordTournament` is critical, ensuring only the authorized backend wallet can call it.
- **Private Key Management:** The private key for the backend's wallet, used to sign transactions, must be stored securely (e.g., using environment variables or a secrets management system) and never exposed in the codebase. Compromise of this key would allow fraudulent tournament records.
- **Node Access:** Secure communication (e.g., HTTPS or WSS) with the sidechain node is necessary.

- **Gas Management:** While likely low on a sidechain, the backend must handle potential gas costs and ensure its wallet maintains a sufficient balance.

8.6 Limitations and Alternatives

- **Complexity:** Adds significant technical complexity compared to solely using PostgreSQL.
- **Cost:** While sidechains reduce costs, there are still infrastructure and maintenance overheads associated with running or connecting to the sidechain node.
- **Speed:** Blockchain transactions are inherently slower than direct database writes.
- **Alternatives:** A cryptographically signed log stored in a conventional database or distributed file system could offer some degree of tamper evidence without the full overhead of a blockchain.

Despite the limitations, the blockchain integration serves as a valuable demonstration of applying this technology to ensure the integrity of specific, high-value data points within the application.

Chapter 9

Deployment and Operations

Deploying and operating the **ft_transcendence** application involves leveraging the containerization and orchestration tools defined in the project architecture. This chapter outlines the deployment strategy, configuration management, web server setup, monitoring practices, and basic maintenance considerations.

9.1 Deployment Strategy

The primary deployment mechanism relies on Docker and Docker Compose, ensuring consistency across different environments (development, testing, production).

- **Docker Compose:** The `docker-compose.yml` file defines and configures all the necessary services: the Django backend, the Nginx web server/reverse proxy, the PostgreSQL database, and the ELK stack components (Elasticsearch, Kibana). It manages container builds, network connections between services, and volume mounts for persistent data.
- **Makefile:** A `Makefile` simplifies common deployment and management tasks by providing commands (e.g., `make up`, `make down`, `make build`, `make logs`) that abstract away the underlying Docker Compose commands, making the process more user-friendly and less error-prone.

Deployment typically involves cloning the repository onto the target server, configuring environment variables, and running a `make` command to build and start all services.

9.2 Environment Configuration

Managing configuration, especially sensitive data like API keys, database passwords, and the backend's blockchain wallet private key, is crucial for security. This is typically handled through environment variables, often sourced from a `.env` file located in the project root.

- `.env` **File:** This file (which should be included in `.gitignore` and never committed to version control) stores key-value pairs for settings like `SECRET_KEY`, `POSTGRES_PASSWORD`, `ETH_NODE_URL`, `BACKEND_WALLET_PK`, etc.
- **Docker Compose Integration:** The `docker-compose.yml` file is configured to load variables from the `.env` file and inject them into the respective service containers' environments at runtime.

This approach keeps sensitive information separate from the codebase.

9.3 Web Server and Reverse Proxy (Nginx)

Nginx plays a critical role in the deployed architecture:

- **Static File Serving:** It efficiently serves the static frontend assets (HTML, CSS, JavaScript, images) directly to users, reducing the load on the Django backend.
- **Reverse Proxy:** It acts as a reverse proxy, receiving incoming HTTP/HTTPS requests and forwarding appropriate requests (e.g., API calls to `/api/...`) to the Django backend application (likely running with Gunicorn or Uvicorn inside its container).
- **HTTPS/TLS Termination:** Nginx is configured to handle SSL/TLS certificates (e.g., obtained via Let's Encrypt) to enforce HTTPS, encrypting traffic between clients and the server. It terminates the TLS connection and communicates with the backend over the internal Docker network, simplifying the backend configuration.
- **Load Balancing (Optional):** While not explicitly stated for the base project, Nginx could potentially be configured to load balance requests across multiple instances of the backend container if scaling becomes necessary.

Nginx configuration files define these behaviors, specifying server blocks, locations, proxy settings, and SSL parameters.

9.4 Monitoring and Logging (ELK Stack)

Centralized logging and monitoring are handled by the ELK stack (Elasticsearch, Kibana):

- **Log Collection:** Application logs (from Django, Nginx, potentially others) are configured to be shipped to Elasticsearch. This might involve container logging drivers configured in Docker Compose or agents like Filebeat/Logstash (though Logstash wasn't explicitly required, implying a simpler setup might be used).
- **Elasticsearch:** Stores and indexes the log data efficiently, enabling fast searching and aggregation.
- **Kibana:** Provides a web interface for visualizing the log data stored in Elasticsearch. Developers and operators can use Kibana to search logs, create dashboards to monitor application health (e.g., error rates, request times), and troubleshoot issues.

This centralized system provides crucial operational visibility into the running application.

9.5 Maintenance Considerations

Ongoing operation requires routine maintenance:

- **Updates:** Regularly updating base Docker images, operating systems within containers, and application dependencies (Django, Python packages, JS libraries) is essential for security and performance.
- **Backups:** Implementing a strategy for backing up the PostgreSQL database volume is critical to prevent data loss. Blockchain data is inherently persistent on the sidechain itself.
- **Monitoring Checks:** Regularly reviewing logs and dashboards in Kibana helps proactively identify and address potential issues.
- **Certificate Renewal:** Ensuring SSL/TLS certificates are renewed before expiration is vital for maintaining HTTPS.

The containerized nature of the application simplifies some maintenance tasks, as updates can often be managed by rebuilding Docker images and redeploying containers.

Conclusion

In conclusion, the **ft_transcendence** project successfully achieved its objective of developing a modern, full-stack web application centered around the classic game of Pong. It delivers a feature-rich Single Page Application (SPA) providing multiple gameplay modes, including 1 vs 1 online matches, contests against an AI opponent, local 4-player games, and structured tournaments. Key features enhancing user engagement include user profiles with avatars, match history, and a friends system.

The project demonstrates proficiency across a range of modern web technologies and practices. The backend, built with Django and Django Rest Framework, provides a secure RESTful API. User authentication is robust, offering native credentials, 42 OAuth integration, JWT-based session management via secure cookies, and optional TOTP 2FA. The frontend utilizes vanilla JavaScript and the HTML5 Canvas for dynamic game rendering, ensuring a responsive experience without reliance on large frameworks. Data persistence is handled innovatively through PostgreSQL for primary application data and an Ethereum sidechain for immutable tournament record-keeping, interacted with via Web3.py. The entire application is containerized using Docker and orchestrated with Docker Compose, including Nginx for web serving/reverse proxying and the ELK stack for centralized logging and monitoring, showcasing production-grade DevOps practices.

Adherence to the Software Development Life Cycle (SDLC), including requirements definition, architectural design, implementation, and multi-faceted testing (unit, integration, manual E2E, security, usability), ensured a systematic approach to development and quality assurance. The project successfully addressed technical challenges related to real-time interactions, security implementation, and cross-component integration within a containerized environment.

Future work could focus on areas identified during evaluation, such as implementing comprehensive automated End-to-End testing, conducting more extensive performance and load testing, potentially adding a real-time chat feature, and further refining the AI opponent's capabilities. This project serves as a strong demonstration of integrating diverse technologies to build a secure, scalable, and engaging web-based gaming platform.

Chapter 10

Appendix

Network graph of GitHub, representing the main branch (black), fixes on the game (green), improvement of design of the pages (purple), and features of user management (yellow and blue).

Continuous Integration and Continuous Deployment workflow used in the project. CI/CD integration on GitHub.

The visual impairment mode of the project was inspired by the study Co-designed mini-games for children with visual impairment: A pilot study on their usability by Battistin et al. (2022), which explored accessible game design principles tailored for visually impaired children to enhance usability and engagement.

References

[1] 42 (n.d.). Intra API documentation. Retrieved October 20, 2024, from https://api.intra.42.fr/apidoc/guides/

[2] Battistin, T., Dalla Pozza, N., Trentin, S., Volpin, G., Franceschini, A., & Rodà, A. (2022). Co-designed mini-games for children with visual impairment: A pilot study on their usability. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-022-13665-7

[3] Bootstrap. (n.d.). Bootstrap documentation. Retrieved September 15, 2024, from https://getbootstrap.com/docs

[4] Django Software Foundation. (n.d.). Django documentation (version 5.0). Retrieved September 15, 2024, from https://docs.djangoproject.com/en/5.0/

[5] Django OTP contributors. (n.d.). Django OTP documentation. Retrieved October 28, 2024, from https://django-otp-official.readthedocs.io/en/latest/

[6] Encode OSS Ltd. (n.d.). Django REST framework documentation. Retrieved October 28, 2024, from https://www.django-rest-framework.org/

[7] Docker Inc. (n.d.). Docker documentation. Retrieved October 28, 2024, from https://docs.docker.com/

[8] Ecma International. (n.d.). ECMAScript language specification. Retrieved September 25, 2024, from https://www.ecma-international.org/publications-and-standards/

[9] Elastic. (n.d.). Elastic Stack and Product Documentation. Retrieved October 28, 2024, from https://www.elastic.co/guide/index.html

[10] GeeksforGeeks. (2024). What is single page application? Available at: https://www.geeksforgeeks.org/what-is-single-page-application/

[11] Guanabara, G. (2019). Curso de JavaScript [Video]. YouTube. https://www.youtube.com/watch?v=1-w1RfGIov4

[12] Guanabara, G. (2020). Curso de HTML e CSS [Video]. YouTube. https://www.youtube.com/watch?v=1-w1RfGIov4

[13] Programming with Mosh. (2021). Python Django tutorial for beginners [Video]. YouTube. https://www.youtube.com/watch?v=rHux0gMZ3Eg

[14] Nginx Inc. (n.d.). Nginx documentation. Retrieved October 28, 2024, from https://nginx.org/en/docs/

[15] The PostgreSQL Global Development Group. (n.d.). PostgreSQL documentation. Retrieved October 26, 2024, from https://www.postgresql.org/docs/current/

[16] Simple JWT. (n.d.). Simple JWT documentation. Retrieved October 18, 2024, from https://django-rest-framework-simplejwt.readthedocs.io/en/latest/

[17] Ethereum Foundation. (n.d.). Solidity documentation. Retrieved October 28, 2024, from https://docs.soliditylang.org/

[18] Stakeholdermap.com. (n.d.). Risk assessment matrix - Simple 3x3. Retrieved October 26, 2024, from https://www.stakeholdermap.com/risk/risk-assessment-matrix-simple-3x3.html

[19] Web3.py contributors. (n.d.). Web3.py documentation. Retrieved October 28, 2024, from https://web3py.readthedocs.io/en/stable/

[20] Wikipedia contributors. (n.d.). Single-page application. Wikipedia, The Free Encyclopedia. Retrieved September 8, 2024, from https://en.wikipedia.org/wiki/Single-page_application