OWCA Advanced FitchFork Installation Manual

1. Introduction

This document provides detailed instructions for setting up the Advanced FitchFork project on a local development machine. The manual is intended for developers, system administrators, and technical users who need to clone, configure, build, and run the application from the source code.

Advanced FitchFork is comprised of three primary components that must be configured correctly:

- Backend API: A server built with Rust that handles all core logic, database interactions, and user authentication.
- Frontend Application: A client-side user interface built with React and TypeScript.
- Code Manager: A Dockerized environment used by the backend to securely and safely execute student-submitted code.

Following these steps will result in a fully functional local instance of the application.

2. Prerequisites

Before you begin, ensure your system has the following software installed. Please use the recommended versions to avoid potential compatibility issues.

Software	Recommended Version	Installation Notes
Git	Latest	Required for cloning the source code.
Node.js	20.x (LTS)	Used to run the frontend application.
Rust	Latest stable	Required for building and running the backend API. Install via **rustup**.
Docker	Latest	Required for the containerised code execution.
cargo-make	Latest	A cargo subcommand used to automate build tasks.

Installing cargo-make

After installing Rust, you must install cargo-make. Open your terminal or command prompt and run:

```
cargo install cargo-make
```

3. System Installation and Setup

Follow these steps sequentially to configure the project.

Step 1: Clone the Repository

First, clone the project repository from GitHub to your local machine.

```
git clone https://github.com/COS301-SE-2025/Advanced-FitchFork.git
cd Advanced-FitchFork
```

This will create a directory named Advanced-FitchFork containing the entire project structure.

Step 2: Configure the Code Manager (Docker)

The backend relies on a custom Docker image to run code submissions. This image must be built before starting the backend. Navigate to the image directory:

```
cd backend/code_manager/images
```

Build the Docker image. This command creates an image named universal-runner, which the system will use.

```
docker build -t universal-runner .
```

Note: Ensure your Docker daemon is running before executing this command. This process may take a few minutes.

Return to the project's root directory:

```
cd ../../../
```

Step 3: Configure and Run the Backend API

The backend requires environment variables and a database setup before it can run.
Navigate to the backend directory:

```
cd backend
```

Create an environment configuration file by copying the example template:

```
cargo make fresh
```

For local development, most of the default values in the .env file are sufficient; however, users will have to source their API keys for the various services that are used, such as Gemini and MOSS

Set up the database and run migrations. The cargo make fresh command will automatically create the SQLite database file and apply the necessary schema.

```
cargo make fresh_seed
```

Start the backend API server:

```
cargo make api
```

Upon successful startup, your terminal should display output indicating that the server is running and listening for connections, typically on http://127.0.0.1:8000.

Keep this terminal window open.

Start the backend code manager:

```
cargo make code
```

Upon successful startup, your terminal should display output indicating that the server is running and listening for connections, typically on http://127.0.0.1:3000.

Keep this terminal window open, and **make sure that Docker is running**.

Step 4: Configure and Run the Frontend Application
The frontend connects to the backend API to display and manage data.
Open a new terminal window and navigate to the frontend directory from the project root:

```
cd frontend
```

Install all the required Node.js packages:

```
npm install
```

Start the frontend development server:

```
npm run dev
```

Your terminal will display a local URL, typically http://localhost:5173. Open this URL in your web browser to access the Advanced FitchFork application.

4. Running Tests

To ensure the integrity of the system and validate new changes, you can run the project's automated tests.

Frontend Tests
The frontend uses Playwright for end-to-end testing.
Navigate to the frontend directory:

```
cd frontend
```

Run the tests:

```
npm run e2e
```

Backend Tests
The backend uses Rust's built-in testing framework.
Navigate to the backend directory:

```
cd backend
```

Run the tests:

```
cargo test
```

5. Troubleshooting Common Issues

Issue: cargo make command not found.
Solution: Ensure you have installed cargo-make globally by running cargo install cargo-make.

Issue: Docker command fails with a "daemon not running" error.
Solution: Make sure Docker Desktop (or the Docker daemon on Linux) is running before you try to build the universal-runner image.

Issue: Port conflict on 8000 or 5173.
Solution: Another application on your system is using one of the required ports. Stop the other application or configure the port in the respective .env (backend) or vite.config.ts (frontend) file.

Issue: Frontend shows network errors or fails to log in.
Solution: Confirm that the backend API server is running in its own terminal window and that you did not close it. Ensure there were no errors during its startup.