Developer Documentation (Designs & Processes)

*This document provides a record of the **product & process documentation including designs, development processes, ideas outlined & overall product vision** that was continuously defined throughout the first sprint. The documentation describes the product System & Architecture, Prototype Features & Functions, UI/UX & Interface, Kanban Development Process, Testing Strategy as well as provides all Meeting Notes recorded and the Development Log. The process commentary & designs of all of these have been recorded. In addition to this document, a Sprint 1 Documentation document was also produced which provides a Reflection of the First Sprint.*

System & Architecture Overview (Sprint 1)

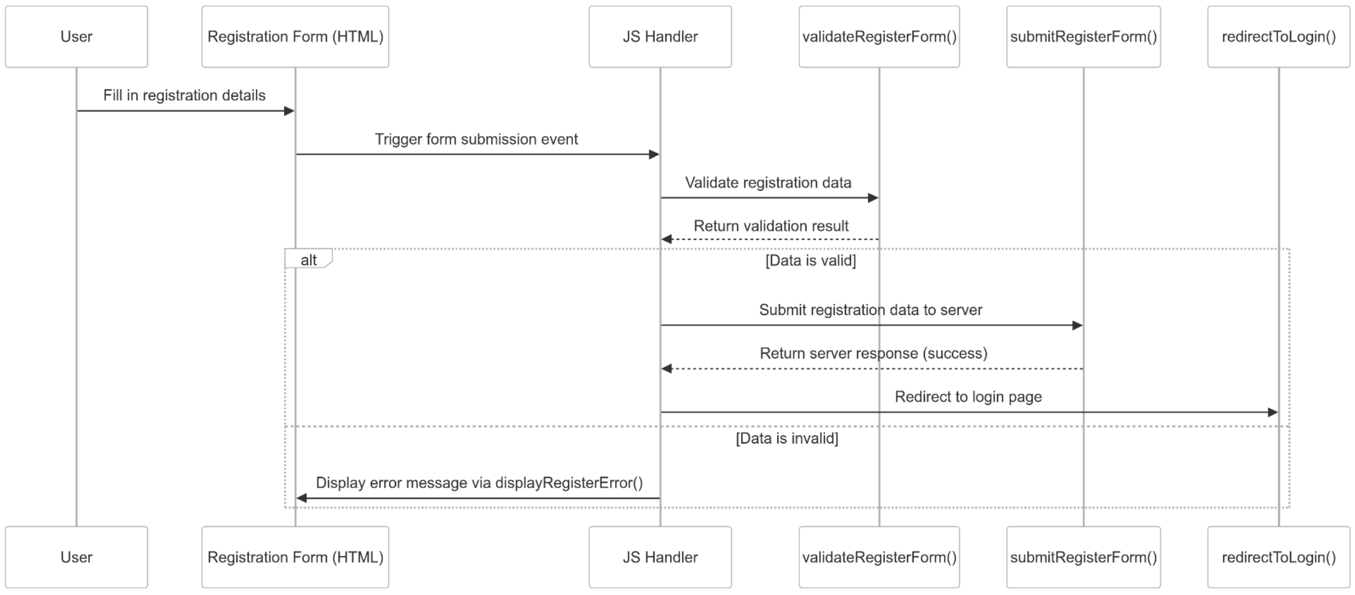
Front-End

The project plan involves the use of HTML, JS & CSS in combination with Bootstrap 5.3.3 assets to provide the following pages within out Front-End Development:

- Register Page
- Login Page
- Landing Page
- User Inventory Pages

The Register Page allows enrollment of new-users into EcoGos' application provided an end-point connecting to the back-end (database) storing these details. This integration as well as functions of the database are explained further into this document. To implement a successful registration of users, the following designs are implemented:

Design - Registration Page

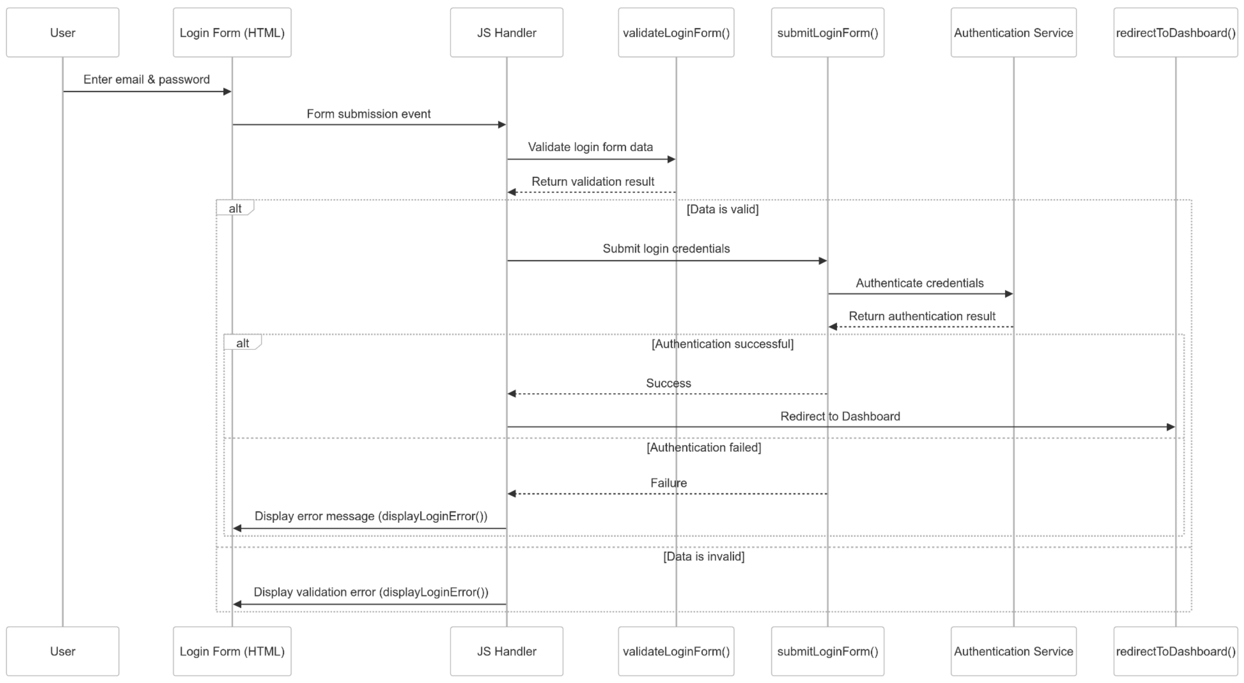

User
Registration Form (HTML)
JS Handler
validateRegisterForm()
submitRegisterForm()
redirectToLogin()
Fill in registration details
Trigger form submission event
Validate registration data
Return validation result
alt
[Data is valid]
Submit registration data to server
Return server response (success)
Redirect to login page
[Data is invalid]
Display error message via displayRegisterError()

Method	**Purpose**
validateRegisterForm()	Validation of registration inputs.

submitRegisterForm()	Sends validated registration data to the backend for account creation
Promise/Response Object	Triggered after successful validation post-executing the form
displayRegisterError(msg)	Displaying registration field issues
redirectToLogin()	Invoked upon successful registration

The Login Page interacts with the database to check the validity of all fields that were input in the LoginForm. To implement a successful authentication of user login, the following designs were implemented:

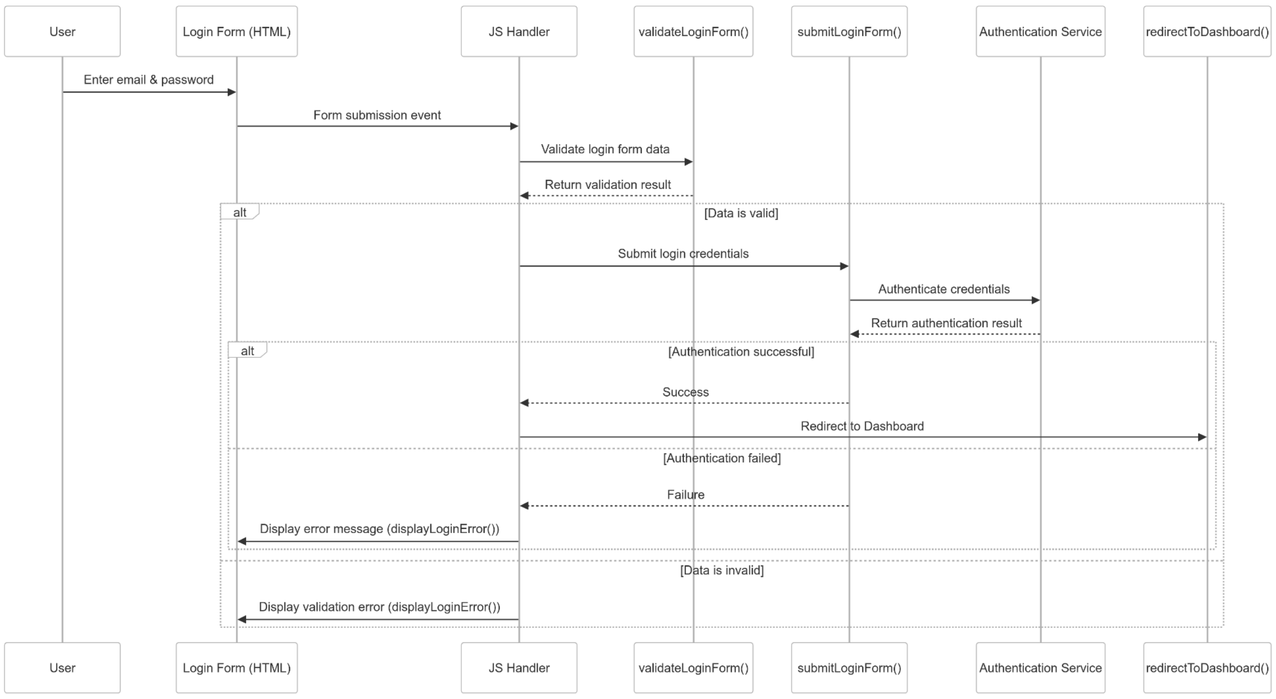
Design - Login Page



Method	**Purpose**
validateLoginForm()	Validation of login form inputs.
submitLoginForm()	Sends login credentials to the database for authentication.
displayLoginError(msg)	Shows an error message if the login fails.
togglePasswordVisibility()	Changes the password fields between masked and unmasked.

The Landing Page is the initial call-to-action page of the EcoGo project, that provides an outline of the application as well as provides a direction for the Login and Register functions. The following designs were implemented to guide the development of this page:

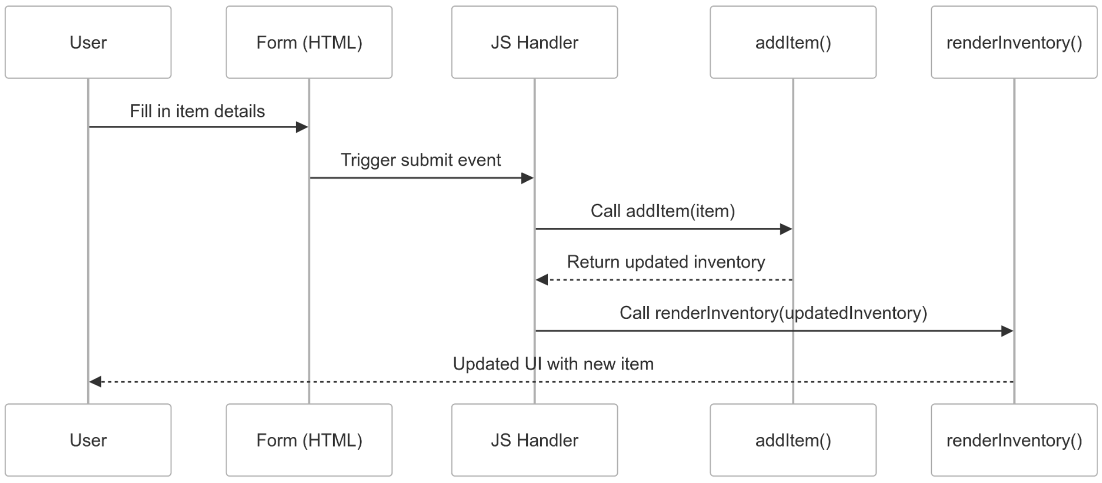
Design - Landing Page



Method	**Purpose**
renderLandingPage()	Displayers primary landing page UI.
initLandingComponents()	Initialises interactive elements (sliders, pages).
navigateToRegister()	Link to the Registration Page.
navigateToLogin()	Link to the Login Page.

The <u>User Inventory Pages</u> (front-end and html pages) allow users to view, add, update, and delete inventory items, ensuring their inventory data is dynamically managed. The following designed were prepared for the implementation of this page:

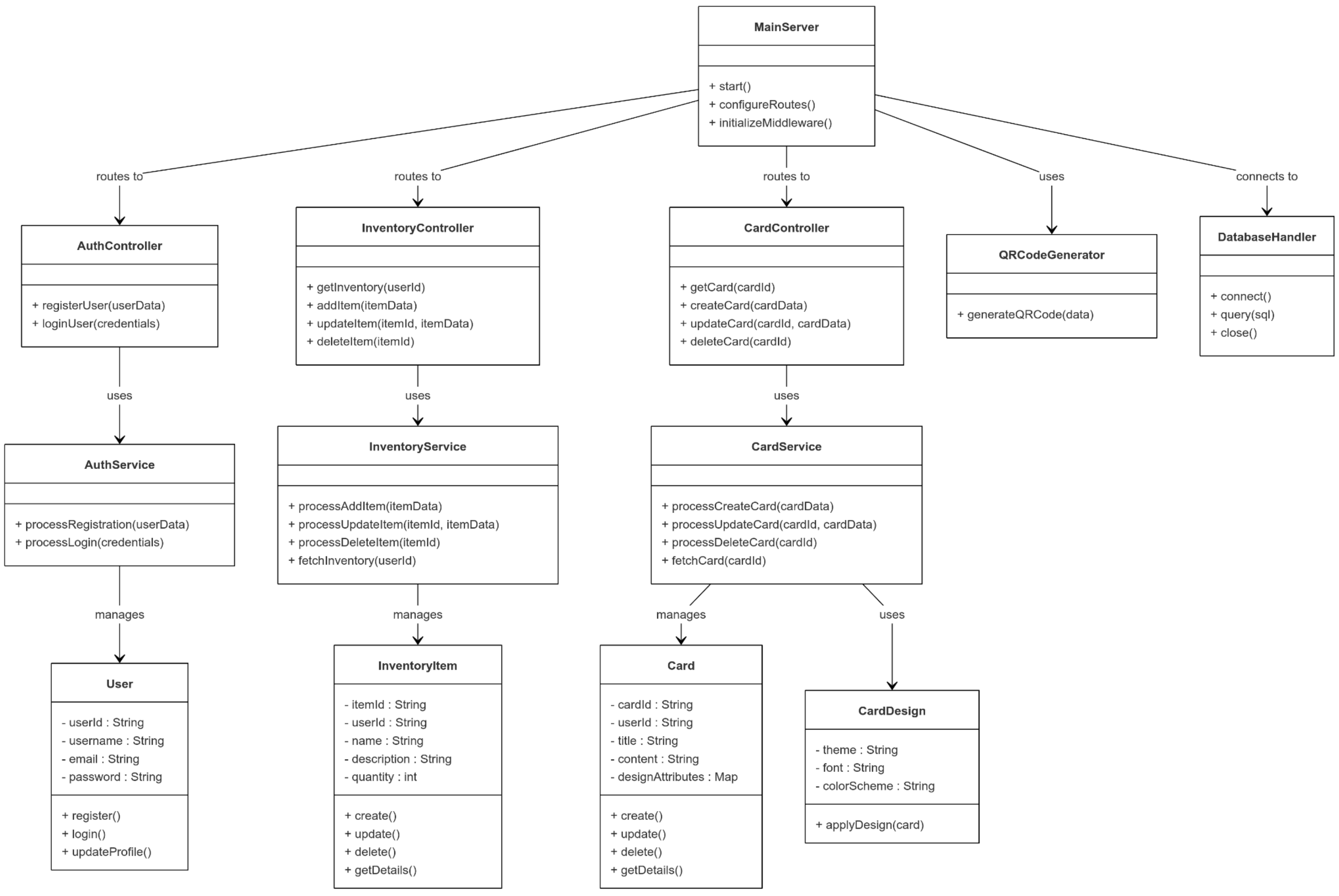
Design: Add User Inventory



Method	Purpose
initializeInventory()	Sets up initial inventory data structure on page load.
renderInventory()	Renders appropriate inventory to the DOI, updates UI.
addItem(), deleteItem(), updateItem()	Adds, Removes & Changes Items in the inventory respectively.
searchItems()	Filters inventory based on a search string and returns a list of the relevant items.

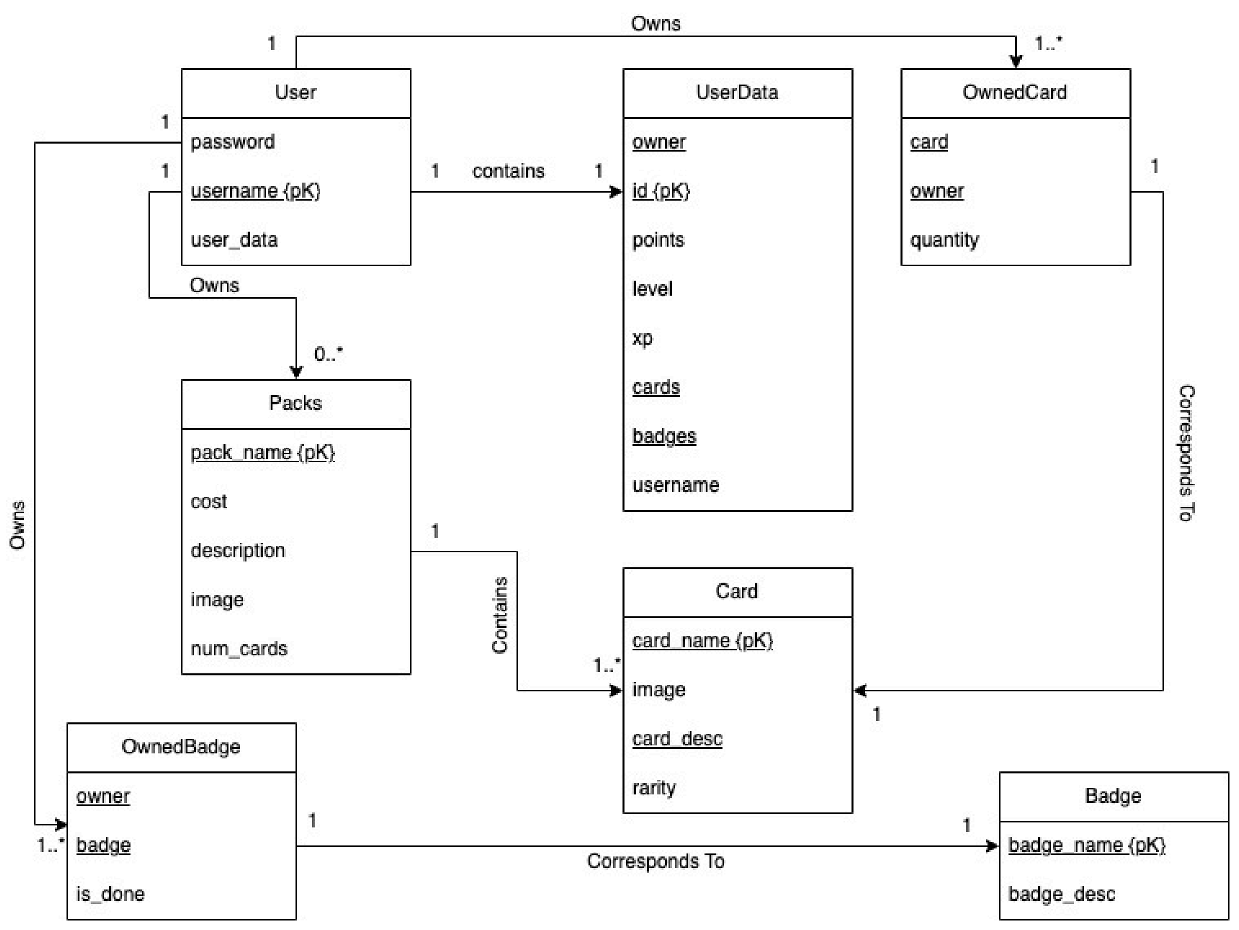
Back-End
The Back-End is a modular implementation supporting user authentication, inventory management, custom card handling with an integration QR code functionality. The Main server, initialises the application and routes all requested to the relevant class. These were designed initially using UML Design followed by a description outlining the need for all the classes implemented.

Design - (UML Diagrams: Functionalities & Modules Implemented)



Class/Module	Purpose
MainServer	Serves as the application's entry point, by configuring routes, middleware, and initialising all modules.
User	Represents a user account with credentials and profile data. Supports registration, login and displaying of user inventory.
AuthController & AuthService	Manage user registration and authentication. Accepts registration details and calls AuthService. Verifies credentials and returns an authentication token or error.
InventoryItem	Represents an individual inventory object.
InventoryController & InventoryService	Handles requests for managing inventory, adding, removing or updating items in the inventory.
Card	Models a card entity.
QRCodeGenerator	Dynamic Generation of QR codes based on URL input data.
DatabaseHandler	DB Connectivity & Query execution.

Database



Entity	**Relational Schema**	**Constraints**
User	User (password, <u>username</u>, user_data)	User-OwnedCard: One-To-Many User-Packs: One-To-Many User-OwnedBadge: One-To-Many
Packs	Packs (<u>pack_name</u>, cost, description, image, num_card)	Packs-Cards: One-To-Many
OwnedBage	OwnedBage (<u>owner</u>, <u>badge</u>, is_done)	OwnedBadge-Badge: One-ToOne
Card	Card (<u>card_name</u>, image, <u>card_desc</u>, rarity)	N/A
UserData	UserData (<u>owner</u>, <u>id</u>, point, level, xp, <u>cards</u>, <u>badges</u>, username)	N/A
OwnedCard	OwnedCard (<u>card</u>, <u>owner</u>, quantity)	OwnedCard-Card: One-To-One
Badge	Badge (<u>badge_name</u>, badge_desc)	N/A

Description of Prototype
EcoGo is primarily aimed towards raising awareness about sustainability through a gamified approach to engage users. It is implemented as a Web Application allowing users to collect cards, packs by scanning QR codes placed at sustainable locations around campus. These could entail:

- Recycling Bins
- Public Transit Stops
- Bottle Refill Stations
- Plastic Cutlery Stations
- Grocery Stores

This strategic positioning nudges users to engage with our application for a competitive benefit of gaining points, xp and level through these cards. Furthermore, by obtaining a card, a sustainability fact is revealed, making it more educative for users. Users are also able to trade cards amongst other users, increasing engagement, and participate in battles using their cards, allowing for an added level of user experience within gamification.

The features and functionalities of our project are defined below, however, since this is a prototype, not all functionalities have been included yet, and an overall function of the application has been implemented.

System & Architecture Overview (Sprint 2)

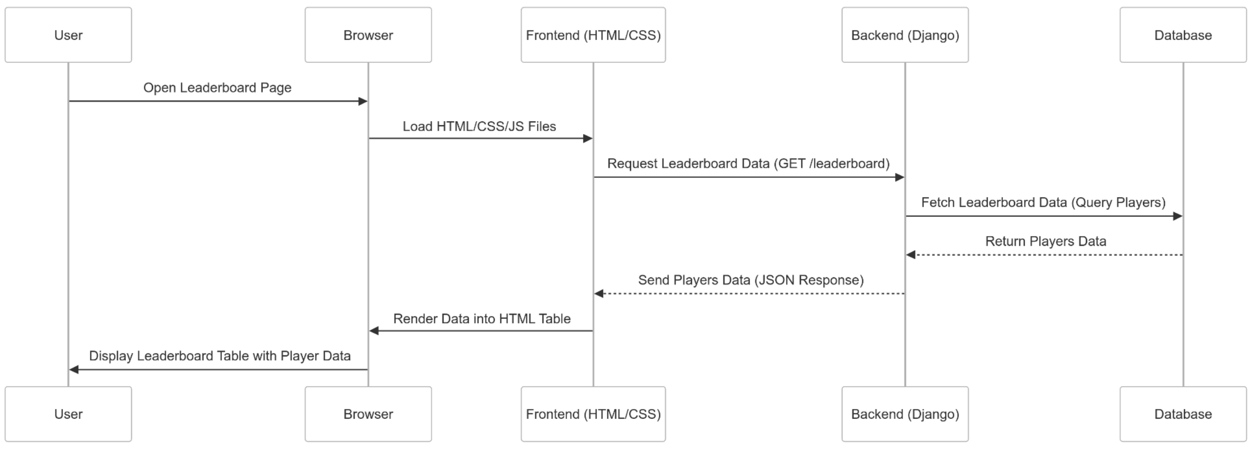
Front-End

Extending the Front-End development from Sprint 1, the Leaderboard, GeoLocations and Trading functionalities were the key inclusions within our Sprint 2 development framework. The following Front-End pages were implemented as a part of this deliverable:

- Leaderboard Page
- GeoLocation Page
- Trading Page

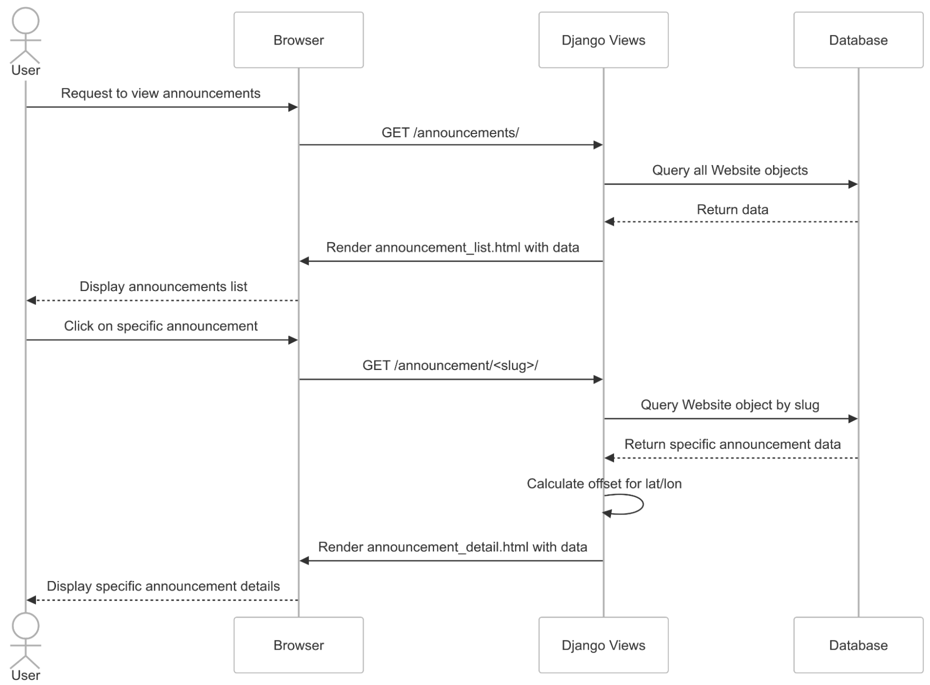
The Leaderboard Frontend displays a list of the Top 10 users filtered by levels, which have been reached by gaining XP. This demonstrates how “sustainably conscious” the user has been and thus, gained the adequate rewards for it.

Design - Leaderboard


User
Browser
Frontend (HTML/CSS)
Backend (Django)
Database
Open Leaderboard Page
Load HTML/CSS/JS Files
Request Leaderboard Data (GET /leaderboard)
Fetch Leaderboard Data (Query Players)
Return Players Data
Send Players Data (JSON Response)
Render Data into HTML Table
Display Leaderboard Table with Player Data

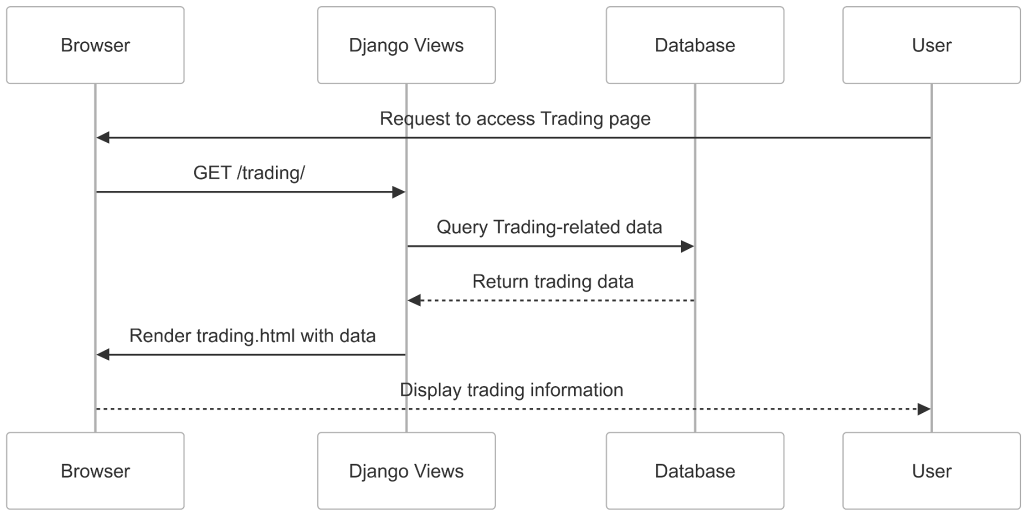
The GeoLocations Front-End contains an announcement list page displaying the full announcement details: title, description, date, and location on a map. It also includes an interactive map using Leaflet or Google Maps for geolocation display.

Design - GeoLocations

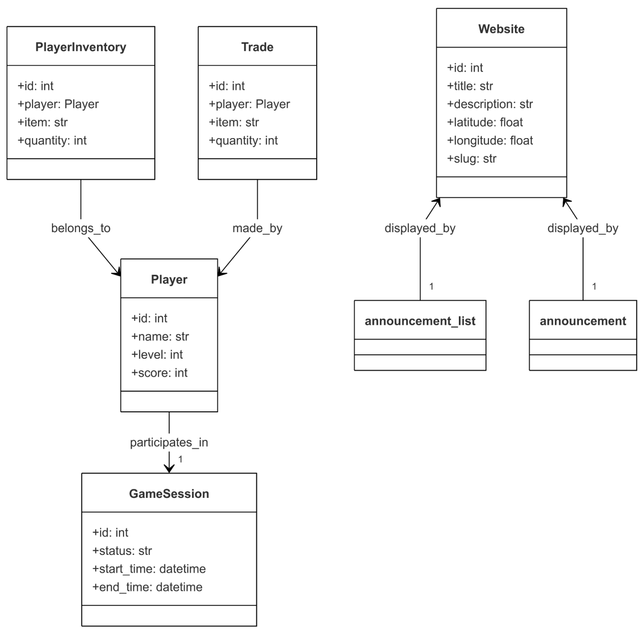

User
Browser
Django Views
Database
Request to view announcements
GET /announcements/
Query all Website objects
Return data
Render announcement_list.html with data
Display announcements list
Click on specific announcement
GET /announcement/<slug>/
Query Website object by slug
Return specific announcement data
Calculate offset for lat/lon
Render announcement_detail.html with data
Display specific announcement details

The Trading Frontend features an intuitive inventory-style UI that displays available items for trade in a grid or list format. It includes a seamless drag-and-drop or click-based interface, allowing players to easily select and exchange items. When a trade is initiated, a sleek confirmation modal or pop-up appears to confirm the transaction. The page also showcases the player’s current inventory and trade history, providing a clear overview of available resources and past trades.

Design - Trading


Browser
Django Views
Database
User
Request to access Trading page
GET /trading/
Query Trading-related data
Return trading data
Render trading.html with data
Display trading information
Browser
Django Views
Database
User

Back-End
The backend for the Geolocations, Game Master, and Trading pages is done in Django, which handles data retrieval, processing, and rendering. For the Geolocations page, the announcement_list and announcement views fetch data from the Website model to display a list of announcements and individual details, including location data with calculated latitude/longitude offsets. The Game Master page interacts with game models such as GameSession and Player to keep track of player progress, game state, and update in real time through WebSockets or AJAX. The Trading page manages trade logic through the Trade and PlayerInventory models, refreshing inventory and trade history. The backend processes data, applies business logic, and returns dynamic templates through Django's templating system.


PlayerInventory
+id: int
+player: Player
+item: str
+quantity: int
Trade
+id: int
+player: Player
+item: str
+quantity: int
Website
+id: int
+title: str
+description: str
+latitude: float
+longitude: float
+slug: str
belongs_to
made_by
displayed_by
displayed_by
1
1
Player
+id: int
+name: str
+level: int
+score: int
announcement_list
announcement
participates_in
1
GameSession
+id: int
+status: str
+start_time: datetime
+end_time: datetime