Deliverable 2: Solution design +

Functionality preview

**Create a GitHub Wiki entry (NOT an attachment) with the items requested in this template**

All notes containing explanations and examples for filling out the template should be deleted and replaced by the project information.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 1. Architecture & Data

### **1.1. Deployment Diagram** General Description: This diagram illustrates the technological architecture of FarmWay, showing the interaction between different components and how data flows through the system. The platform follows a client-server architecture, with a backend in Flask and a database in MongoDB.

### Main Components:

### Frontend (PC - Web Browser): The user interface with which buyers and sellers interact, developed with HTML and CSS. It communicates with the server through HTTP requests.

### Web Server (Flask Server): Acts as an intermediary between the frontend and the database, processing HTTP requests, managing business logic, and returning appropriate responses to the user interface.

### Database Server (MongoDB): Stores all platform data, including users, products, orders, auctions, payments, and messages.

### Data Flow:

### The user interacts with the web interface from their browser.

### HTTP requests are sent to the Flask server.

### Flask processes the requests, accesses the MongoDB database when necessary, and returns the required information to the user.

### The Flask server can perform read and write operations on the database, such as registering new orders, updating auction statuses, or verifying user credentials.

### This architectural model enables efficient communication between the different system components, ensuring fast and scalable operations.

### 

***1.2. Component Diagram***General Description:  
The *FarmWay* component diagram describes the modular structure of the system, showing different functional modules and how they interact. This diagram is essential for understanding the organization of the software and the distribution of responsibilities within the application.

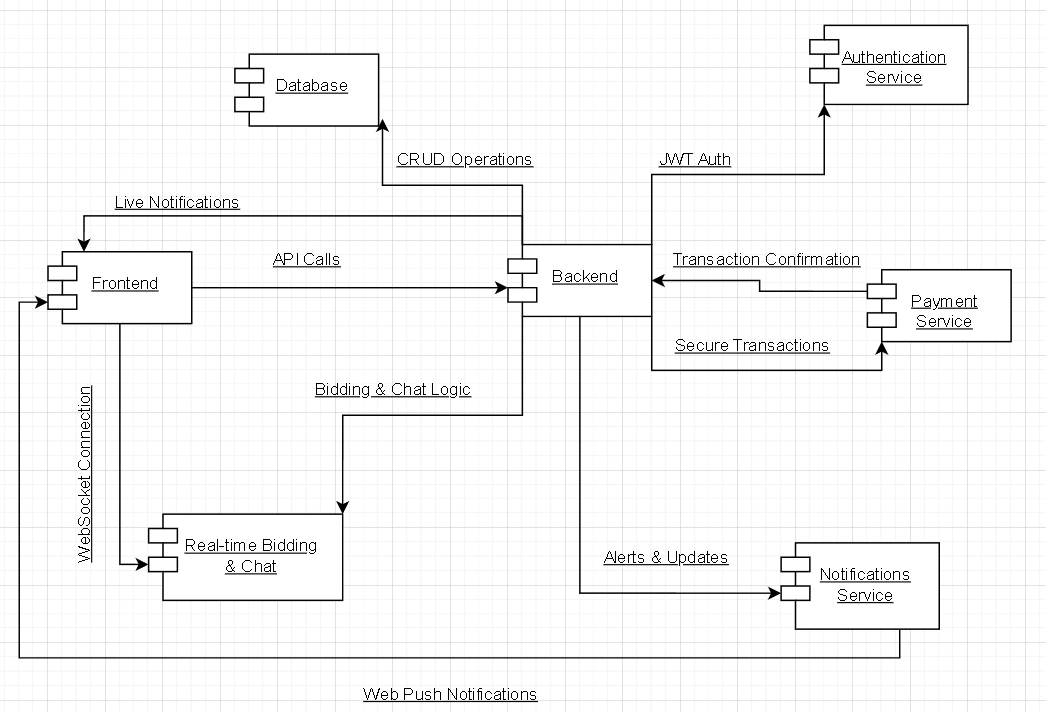
Key Components:

* Frontend: Interacts with users and sends API requests to the backend. It also receives responses that update the user interface with the retrieved data.
* Backend: Processes frontend requests, handles business logic, authenticates users, manages transactions, and communicates with the database.
* Database: Handles all storage, retrieval, and data update operations in MongoDB.
* Authentication Service: Manages user authentication using JSON Web Tokens (JWT), ensuring that only authorized users can access specific system functionalities.
* Payment Service: Processes order payments, verifies transactions, and ensures secure payment handling.
* Notifications Service: Sends real-time alerts and updates to users via WebSockets or push notifications.
* Real-time Bidding & Chat: Implements WebSockets to enable live auction bidding and instant communication between users.

Component Interactions:

1. The user performs an action in the frontend, generating an API request to the backend.
2. The backend verifies authentication and accesses the database or other services as needed.
3. If it is a purchase, the Payment Service processes the transaction.
4. If it is an auction, the Real-time Bidding updates the information live for all participants.
5. The Notifications Service sends alerts to users about relevant events.
6. The processed response is sent back to the frontend for display.

This model enables a modular architecture where each component specializes in a specific function, facilitating scalability and system maintenance.



### **1.3.** **Data Model** General Description: This diagram represents the database structure of FarmWay, including entities, their attributes, and the relationships between them. Its primary purpose is to model the information managed by the platform, ensuring data integrity and allowing the correct execution of system operations.

### Key Elements:

### Users: Stores information about registered users, including name, email, password, address, and role (buyer or seller).

### Products: Contains details of items available in the marketplace, specifying name, category, price, and the user who published them.

### Bids: Represents the offers placed in product or livestock auctions. Each bid is linked to a user and a product being auctioned.

### Orders: Records purchases made on the platform, either from direct purchases or winning bids in an auction.

### Payments: Manages payment-related data, including amount, status (successful or failed), and date.

### Messages: Facilitates communication between users through an internal messaging system.

### Notifications: Manages alerts sent to users regarding important events such as order confirmations, auction updates, or new messages.

### Important Relationships:

### A user can sell multiple products.

### A user can place multiple bids on different auctioned products.

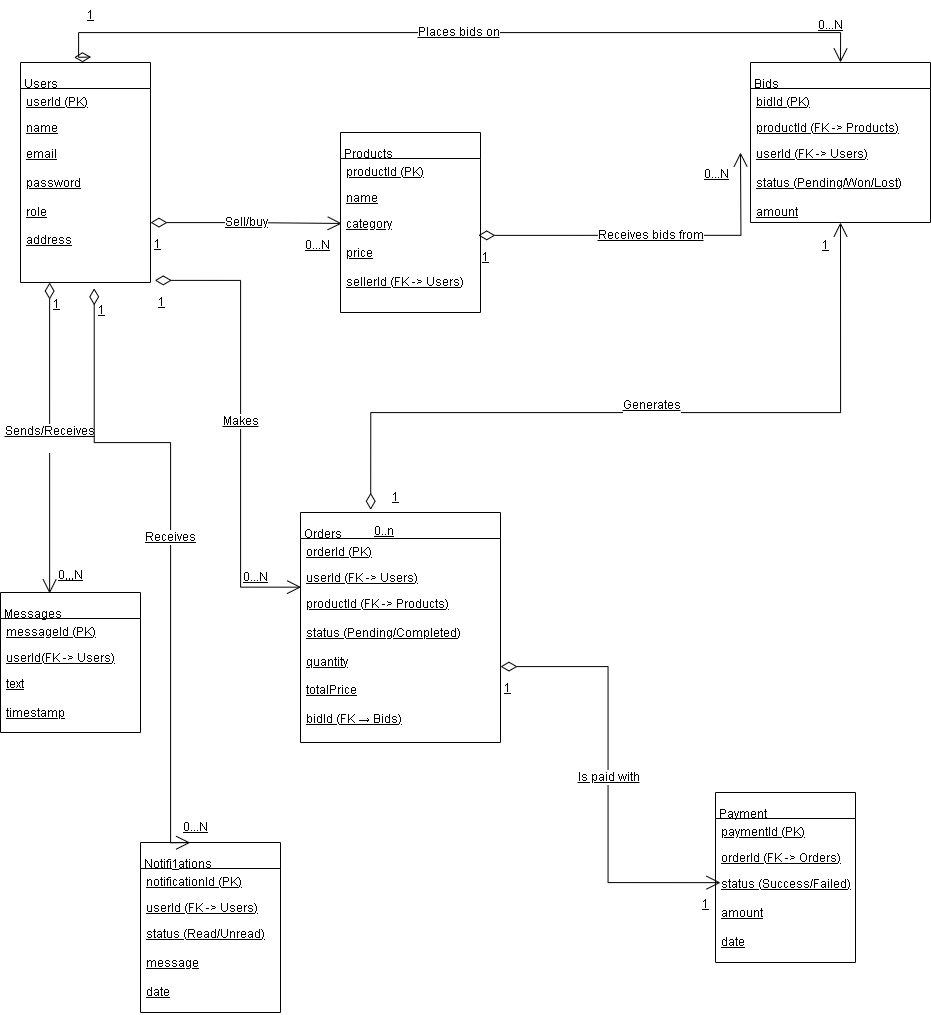
### An order is linked to a user and can contain multiple products.

### An order can generate a payment, which must be associated with a specific transaction.

### A user can send and receive messages from other users within the platform.

### A user can receive multiple notifications related to their activities in the system.

### This database model ensures proper information management in FarmWay, guaranteeing consistency in handling purchases, sales, and auctions.



# 2. Mockups 2.1. Preliminary Mockups



# 

# 2.2. Refined Mockups 3. Repository *Readme*

[*https://github.com/Ssanchezl3/ProyectoIntegrador1-EAFIT/blob/main/README.txt*](https://github.com/Ssanchezl3/ProyectoIntegrador1-EAFIT/blob/main/README.txt)***Requirementets*** *https://github.com/Ssanchezl3/ProyectoIntegrador1-EAFIT/blob/main/Requirements.md*

**4. Video**

https://www.youtube.com/watch?v=2uCvo9pi59g&ab\_channel=joeysalads

**5. Project management**

**Weekly Meeting Record**

Santiago

• What did I do last week?

* Designed the backend structure in Flask and set up the connection with the MongoDB database.
* Created the initial API routes to retrieve product and auction information.
* Started testing data insertion into the database from the backend.

• What will I do this week?

* Adjust the API routes to return data in a format suitable for the frontend.
* Implement the logic for registering new bids in the database.
* Coordinate with Andru on how the frontend should consume backend data.

• Are there any obstacles?

* The integration between the frontend and backend is still unclear and may require adjustments in the data structure.
* We haven't fully defined how user authentication will be handled.

Andru Quiroz  
• What did I do last week?

* Implemented the design of the bid history page in HTML and CSS.
* Started connecting the frontend with the database to display bid history.
* Tested data extraction from MongoDB and its visualization in the interface.

• What will I do this week?

* Adjust the bid history visualization to display the data correctly.
* Improve the frontend code structure to facilitate future integrations.
* Review with Santiago how to retrieve and display active auction data.

• Are there any obstacles?

* There are difficulties in how data is being received from the backend to the frontend, requiring API adjustments.
* We need to define how to handle real-time updates so that bids are reflected automatically.

Sprint Retrospective

• What should we continue doing (best practices)?

* Continue testing with small data sets before implementing full functionality.
* Maintain weekly meetings to review progress and identify issues.
* Keep documenting how each part of the code works to avoid confusion.

• What should we start doing (process improvements)?

* Improve communication between frontend and backend so both follow the same data structure.
* Break tasks into smaller, more achievable goals to avoid delays.
* Use tools like Postman to test API routes before implementing them in the frontend.
* Integrate both parts from the beginning to avoid issues at the end.

• What should we stop doing (process issues and bottlenecks)?

* Stop making changes without first discussing how they will affect the rest of the system.
* Avoid delaying the frontend-backend integration until the end; work in parallel instead.
* Stop working in isolation without syncing progress with the other team member.