MASTER TEAM PROJECT SS2025

HAND2HAND

Team 1

(Global Team)

NAME	EMAIL	ROLE	MATRICULATION NUMBER	GITHUB USERNAME
Rachna Tiwari	rachna-vivek.tiwari@informatik.hs-fulda.de	Team Lead, Document Master	1568192	rachna-hs-fulda
Shambhavi Santosh Hupare	shambhavi-santosh.hupare@informatik.hs-fulda.de	Fronted Lead	1534137	Shambhavi-0506
Yapa Puwakdandawage Ridma Kaveendra	ridma-kaveendra.yapa-puwakdandawage@informatik.hs-fulda.de	Backend Lead	1533862	ridmakaveendra7
Aishwarya Ravi	aishwarya.ravi@informatik.hs-fulda.de	GitHub Master, Backend Developer	1540215	aishwaryaravi1302
Bindu Laksmanappa	bindu.lakshmanappa@informatik.hs-fulda.de	FrontEnd	1511297	BinduLakshmanappa

MILESTONE 2

Document created on : 11.5.2025

Date submitted for review	14.5.2025
Date revised for feedback	

INDEX

1. FUNCTIONAL REQUIREMENTS

1.1 Common Features

1. **Register and Authenticate user - (P1)**
   The system allows university users to register with a verified campus email and log in securely. Guest access is allowed for limited browsing.
2. **Manage User Profile - (P1)**
   Users can manage their profiles, including viewing, editing personal info, uploading a profile picture, and displaying badges, reviews, and activity stats and finally delete the profile after use.
3. **Create, Edit and Delete Post - (P1)**
   Users can create, edit, or delete listings for products. Listings include photos, descriptions, and optional auction settings.
4. **Search Product - (P1)**
   Users can search by keywords and filter by category, rating, price and availability.
5. **Create Wanted Post - (P1)**
   Users must be able to create wanted posts specifying the item they are looking for, including details like category, description, budget and urgency. These posts should be saved to the system and visible to relevant sellers or service providers.
6. **Match Wanted Item - (P2)**
   The system must automatically match wanted posts with available listings or sellers based on relevance, category, and location. Users should receive notifications when potential matches are found.
7. **Message Buyers and Sellers - (P1)**
   Registered users can message each other to negotiate sales, schedule calls for discussion, and coordinate deliveries.
8. **Place Order and Checkout - (P1)**
   Users can place orders. Orders include product details, total price, and notes.
9. **Book and Manage Delivery- (P1)**
   The system must allow users to select from multiple configurable delivery modes (e.g., standard, express), with pricing, estimated delivery time and availability based on the user's location and notify users of the selected delivery details.
10. **Match Delivery Request - (P1)**
    Users can request delivery at checkout. The system matches available delivery agents based on time, capacity, and delivery type.
11. **Track Delivery- (P2)**
    Delivery agents accept jobs, view pickup/drop-off info, and mark deliveries complete with proof of delivery.
12. **Rate User - (P1)**
    Users must be able to rate each other on a standard scale (e.g 1 to 5 stars) after a completed transaction. These ratings should be stored securely and factored into the user's overall reputation score on the platform.

13. **Review User - (P1)**
    Users must be able to write descriptive reviews about their transaction experience, including comments on the product or service quality. Reviews should be publicly visible on user profiles.
14. **Add Badge for User - (P1)**
    Users earn points, badges, and levels for platform activity like sales, deliveries, and positive reviews.
15. **Notify User - (P2)**
    Users receive alerts for messages, bids, orders, delivery updates, reviews and admin actions via in-app notifications.
16. **Manage Admin Settings - (P1)**
    Admins can monitor users, manage content, approve verification, handle reports, and oversee the platform's operation. The admin has to approve each post that created by any user for selling or wanted page.
17. **Report Post - (P2)**
    Users can report inappropriate content or behavior. Admins review reports and take necessary moderation actions.
18. **Guest Access - (P1)**
    Guests can browse listings, categories, and public reviews but cannot post, message or purchase.
19. **Log Transaction - (P2)**
    All transactions are mocked and displayed as "payment successful" when user clicks on Complete Payment button.

1.2 Unique Features

1. **Bid for Items - (P3)**
   Users must be able to place real-time bids on products, with bid status updated instantly, and receive notifications for bid status changes. The auction should automatically close when the set duration ends.
2. **Post Auction - (P3)**
   Users must be able to post products for auction, including setting a starting price, auction duration, and product details. The system should ensure the posted auction is visible to potential bidders.
3. **Translate Text - (P1)**
   The application shall support switching the interface language for core navigation, buttons, and labels to accommodate users of different linguistic backgrounds limited to German and English Language.
4. **Notify for Matching Listings - (P2)**
   Users are notified when a new product listing matches their Wanted post criteria.
5. **Match Delivery Partners - (P1)**
   System shall match users with available delivery agents based on item type, time slot, and location.

2. LIST OF MAIN DATA ITEMS AND ENTITIES

1. **Product**
   - ➢ **Meaning:** An item being sold on the platform.
   - ➢ **Fields:** *ProductID, Name, Description, Price, Condition, Images, SellerID, Category, Stock, CreatedAt, UpdatedAt, Status.*
   - ➢ **Usage:** Viewable by users, searchable, and filterable.

2. **UserProfile**
   - ➢ **Meaning:** Profile of a student or staff user.
   - ➢ **Fields:** *UserID, LastName, FirstName, Email, UserType (Student/Staff), isVerified, Reviews, isDeliveryAgent, Badge, SellCount, BuyCount Level, Bio, JoinedDate, ProfilePicUrl, address_id, role_id*
   - ➢ **Usage:** Displays the user's details, activity, reputation, and badges.

3. **Role**
   - ➢ **Meaning:** Role of a user. (Admin/Guest/User)
   - ➢ **Fields:** *RoleId, RoleName*
   - ➢ **Usage:** Define different role types used in the system.

4. **DeliveryAgent**
   - ➢ **Meaning:** Profile of a student or staff user.
   - ➢ **Fields:** *UserID, JoinedDate, Category, Price, isAvailable, PhoneNumber, Reviews, DeliveriesCompleted.*
   - ➢ **Usage:** Displays the user's details, activity, reputation, and badges.

5. **Order**
   - ➢ **Meaning:** Represents a finalized checkout by a user, including products or services.
   - ➢ **Fields:** *OrderID, UserID, ProductIDs, ServiceIDs, TotalAmount, Status, PaymentStatus, CreatedAt, ShippingAddressID, Notes*
   - ➢ **Usage:** Tracks confirmed purchases and bridges between Cart and Transaction.

6. **Category**
   - ➢ **Meaning:** Classification used to organize products and services.
   - ➢ **Fields:** *CategoryID, CategoryName*
   - ➢ **Usage:** Helps users filter, search, and explore listings.

7. **DeliveryRequest**
   - ➢ **Meaning:** A request for someone to deliver a purchased product.

- ➢ **Fields:** *RequestID, OrderID, AssignedDeliveryAgentID, PickupLocation, DropLocation, Status*.
- ➢ **Usage:** Managed between the buyer, seller, and delivery agent.

8. **Review**
   - ➢ **Meaning:** Feedback mechanism for transactions.
   - ➢ **Fields:** *ReviewID, ReviewerID, RevieweeID, ProductID, Rating, Comment, Timestamp*.
   - ➢ **Usage:** Builds trust and reputation for buyers, sellers, and service providers.
9. **Address**
   - ➢ **Meaning:** Stores delivery or pickup addresses for users or orders.
   - ➢ **Fields:** *AddressID, UserID, HouseNumber, Street, City, ZipCode*
   - ➢ **Usage:** Used for delivery coordination when requesting or fulfilling orders.
10. **Message**
    - ➢ **Meaning:** Communication between users.
    - ➢ **Fields:** *MessageID, SenderID, ReceiverID, Text, Timestamp, IsRead*.
    - ➢ **Usage:** Enables private negotiation and communication.
11. **Report**
    - ➢ **Meaning:** Used to report inappropriate content or users.
    - ➢ **Fields:** *ReportID, ReporterID, TargetType (User/Product/Service), Reason, Timestamp, Status*.
    - ➢ **Usage:** Sent to admin for review and action.
12. **Notification**
    - ➢ **Meaning:** Alerts to user about relevant actions.
    - ➢ **Fields:** *NotificationID, UserID, Type, Message, ReadStatus, Timestamp*.
    - ➢ **Usage:** Keeps the user updated on transactions, delivery, and replies.
13. **Badge**
    - ➢ **Meaning:** Virtual achievements earned by users.
    - ➢ **Fields:** *BadgeID, Name, Criteria, Description*.
    - ➢ **Usage:** Enhances profile with achievements.

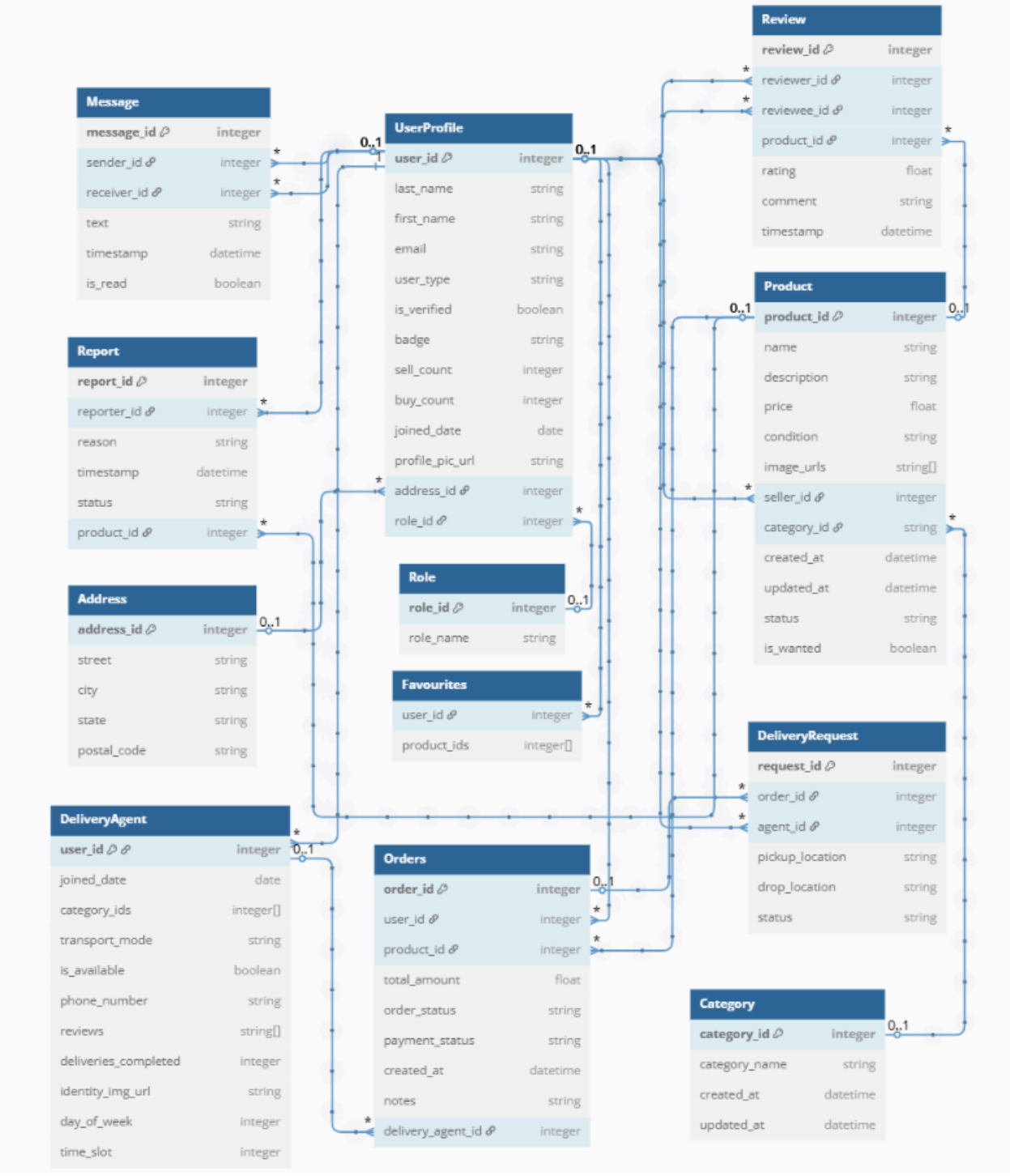
3. UI MOCKUPS AND STORYBOARDS

The UI mockups and the storyboards as per use cases can be found on the miro board link access: https://miro.com/app/board/uXjVI2AMjxc=/?share_link_id=643164930821

4. HIGH-LEVEL ARCHITECTURE AND DATABASE ORGANISATION

- **DB Organisation**

  The Hand2Hand application's database is organized around core entities such as UserProfile, Products, Orders, and Messages, with clear relationships between buyers, sellers, and delivery agents. It follows a relational schema optimized for e-commerce, supporting features like delivery, messaging, and multilingual interactions while ensuring data integrity and scalability.



- **Media storage**

  Users can upload images in JPEG and PNG formats (profile pictures, product images, ..). Uploaded images in the Hand2Hand app will be stored in Amazon S3, with file URLs saved in the database rather than using BLOBs. This approach ensures efficient storage and scalability, especially for product listings and user profile pictures.

- **Search/filter architecture and implementation**

  The search functionality uses SQL with the *%LIKE%* operator to match keywords against indexed columns, ensuring efficient searches.

  Search Algorithm Steps:

  1. User Input: The user enters keywords in the search bar. Keywords are sent as query parameters with the API request.
  2. SQL Query Construction: The backend constructs an SQL query using *%LIKE%* to match the keywords with indexed columns (e.g., product_name, description).
  3. Query Execution: The query is executed, and matching results are fetched from the database.
  4. Display Results: The results are formatted and sent to the front-end, where they are displayed to the user.

  Optimizations:

  - Indexes: Improve search performance by using indexed columns to avoid full table scans.
  - Advanced Filters: Additional filters (e.g., price, category) can be applied in the SQL query.
  - Relevance Ranking: Results can be ranked by relevance based on the query.
  - Users can sort results by various fields (e.g., name, price, date added). This is implemented by adding an *ORDER BY* clause to the SQL query.

- **Our APIs**

❖ Authentication

1. POST /api/auth/register/
    - Register a new user.
    - Request Body: {UserProfile}
    - Response: 201 Created

2. POST /api/auth/login/
    - Log in as a user.
    - Request Body: {email, password}
    - Response: 200 OK

❖ User

1. GET /api/user/{id}
    - Get the user profile.
    - Response: 200 OK [UserProfile]

2. PUT /api/user/{id}
    - Update user profile.
    - Request Body: {UserProfile}
    - Response: 200 OK [UserProfile]

3. DELETE /api/user/{id}
    - Delete the user profile.
    - Response: 200 OK

4. GET /api/user/{id}/badges/
    - Fetch earned badges.
    - Response: 200 OK [List[Badges]]

5. GET /api/user/{id}/reviews/
    - Get the user's reviews.
    - Response: 200 OK [List[Reviews]]

❖ Products

1. GET /api/products/
    - List all products with optional filters.
    - Response: 200 OK [List[Product]]

2. POST /api/products/
    - Create a new product.
    - Request Body: {Product}
    - Response: 201 Created

3. GET /api/products/{id}/
    - Get product details.
    - Response: 200 OK [Product]

4. PUT /api/products/{id}/
    - Edit product.
    - Request Body: {Product}
    - Response: 200 OK [Product]

5. DELETE /api/products/{id}/
    - Delete product.
    - Response: 200 OK

6. GET /api/products/wanted/
    - View all wanted product posts.
    - Response: 200 OK [List[Product]]

❖ **Delivery**

1. POST /api/delivery/
    - Create a delivery request.
    - Request Body: {DeliveryRequest}
    - Response: 201 Created

2. GET /api/delivery/
    - View delivery requests.
    - Response: 200 OK [List[DeliveryRequest]]

3. POST /api/delivery/accept/
    - The delivery agent accepts a delivery request.
    - Request Body: {request_id, delivery_agent_id}
    - Response: 200 OK

4. GET /api/delivery/agents/
    - View available delivery agents.
    - Response: 200 OK [List[DeliveryAgents]]

❖ **Messaging**

1. GET /message-threads/{thread_id}/
    - Get all the current user's message threads.
    - Response: 200 OK [List[Message]]

2. POST /message-threads/

- Returns an existing thread if it already exists between user_id and peer_id; if not, creates a new one.
- Request Body: {user_id, peer_id}
- Response: 201/200 {Message}

3. PUT /message-threads/{thread_id}/
    - Send a message in a thread.
    - Request Body: {content, sent_at}
    - Response: 200 OK [Message]

4. POST /api/messages/translate/
    - Translate a message.
    - Request Body: {text, target_language}
    - Response: 200 OK [TranslatedText]

❖ Gamification

1. GET /api/engagement/
    - Get the user engagement profile.
    - Response: 200 OK [EngagementData]

❖ Reviews & Reports

1. POST /api/reviews/
    - Submit a review.
    - Request Body: {Review}
    - Response: 201 Created

2. GET /api/reports/
    - Get all reports (Admin).
    - Response: 200 OK [List[Report]]
3. GET /api/reports/{id}
    - Get a specific report (Admin).
    - Response: 200 OK [Report]

4. POST /api/reports/
    - Report a product or user.
    - Request Body: {Report}
    - Response: 201 Created

5. DELETE /api/reports/{id}/
    - Delete a false report.Aishu
    - Response: 200 OK

❖ **Upload/Download**

1. POST /api/upload/images
    - Upload multiple images.
    - Request Body: {images[]}
    - Response: 200 OK {image_ids[]}

2. GET /api/download/{file_id}
    - Download a file by ID.
    - Response: 200 OK [File]

Significant Non-Trivial Algorithms and Processes

The **Hand2Hand Buy and Sell Application** goes far beyond basic listing functionality. It integrates several intelligent, non-trivial processes and algorithms designed to improve discoverability, trust, engagement, and user satisfaction across buyers, sellers, and delivery agents. These include smart ranking systems, gamification mechanics, matching algorithms, and more.

1. Product & Seller Rating Algorithm

To highlight trustworthy sellers and products, we use a **multi-factor ranking system** that calculates a dynamic score based on:

- **Average Rating**
- **Review Count / Sales Count**
- **Recency of Reviews or Sales**

Product Rating Formula

Product Score=(w1×Average Rating) + (w2×Review Count) + (w3×Recent Reviews [Last 30 Days])

Seller Rating Formula

Seller Score = (w1×Average Rating) + (w2×Total Sales) + (w3×Recent Sales [Last 30 Days])

Where: $w1$, $w2$, $w3$ are the weights assigned to each factor, respectively.

Example Calculation:

Assume a Product has the following metrics:

Average Rating: 4.5
Review Count: 40
Number of Reviews in the Last 30 Days: 8

w1=0.5
w2=0.3
w3=0.2

Product Rating = (0.5×4.5) + (0.3×40) + (0.2×8) = 2.25+12+1.6 = 15.85

SQL Implementation

```
-- Product Rating
-- Product Rating
SELECT
    product_id,
    (0.5 * AVG(rating)) +
    (0.3 * COUNT(review_id)) +
    (0.2 * SUM(CASE WHEN DATEDIFF(CURRENT_DATE, review_date)
<= 30 THEN 1 ELSE 0 END)) AS recent_reviews_score
FROM Reviews
GROUP BY product_id
ORDER BY recent_reviews_score DESC;

-- Seller Rating
-- Seller Rating
SELECT
    seller_id,
    (0.5 * AVG(rating)) +   -- Average rating
    (0.3 * COUNT(transaction_id)) +   -- Total sales count
    (0.2 * SUM(CASE WHEN DATEDIFF(CURRENT_DATE,
transaction_date) <= 30 THEN 1 ELSE 0 END)) AS recent_sales_score
FROM Transactions
```

```
GROUP BY seller_id
ORDER BY recent_sales_score DESC;
```

2. Auction Bidding & Auto-Closing Algorithm

For auction-based listings, the system implements a real-time bidding engine that:

- Accepts user bids
- Tracks the **highest bid**
- Automatically closes the auction when the **end time** is reached
- Assigns the **winning bidder** to the product
- Restricts Repeat Bidding by Same User

SQL Snippet: Get Winning Bid

```
SELECT bidder_id, MAX(bid_amount) AS highest_bid
FROM Bids
WHERE auction_id = 'AUC_123'
GROUP BY auction_id;
```

The system automatically assigns the winning bidder and triggers an in-app notification once the auction concludes.

3. Delivery Agent Matching Algorithm

When a buyer selects delivery, the system assigns an available delivery agent using:

- **Agent Availability**
- **Time Proximity**
- **Product Category Match**

SQL Matching Sample

```
SELECT user_id
FROM DeliveryAgents
```

```
WHERE isAvailable = 1
ORDER BY ABS(TIMESTAMPDIFF(MINUTE, availability_time,
CURRENT_TIMESTAMP))
LIMIT 1;
```

This ensures quick appropriate assignments.

4. Gamification & User Reputation Engine

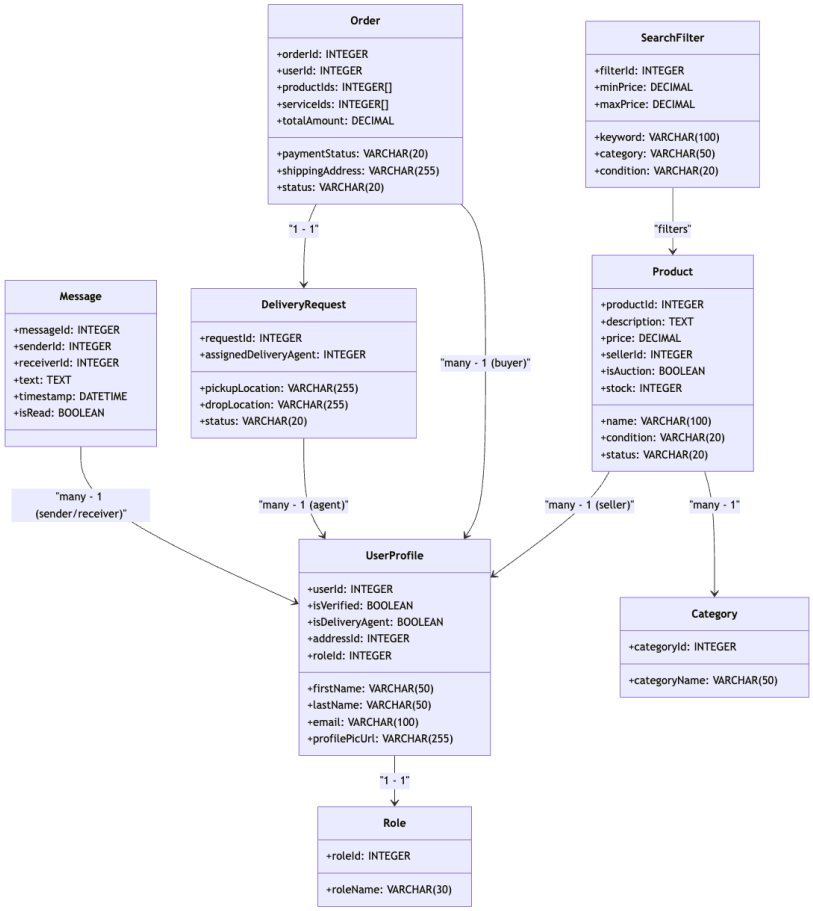
To improve long-term engagement, a gamified point and badge system tracks user activity:

- **+10 points** for completing a sale
- **+5 points** for leaving a review
- **+20 points** for completing a delivery
- **Level-Up**: Every 100 points
- **Badges**: Earned for milestones (e.g., “Trusted Buyer”, “Top Seller”)

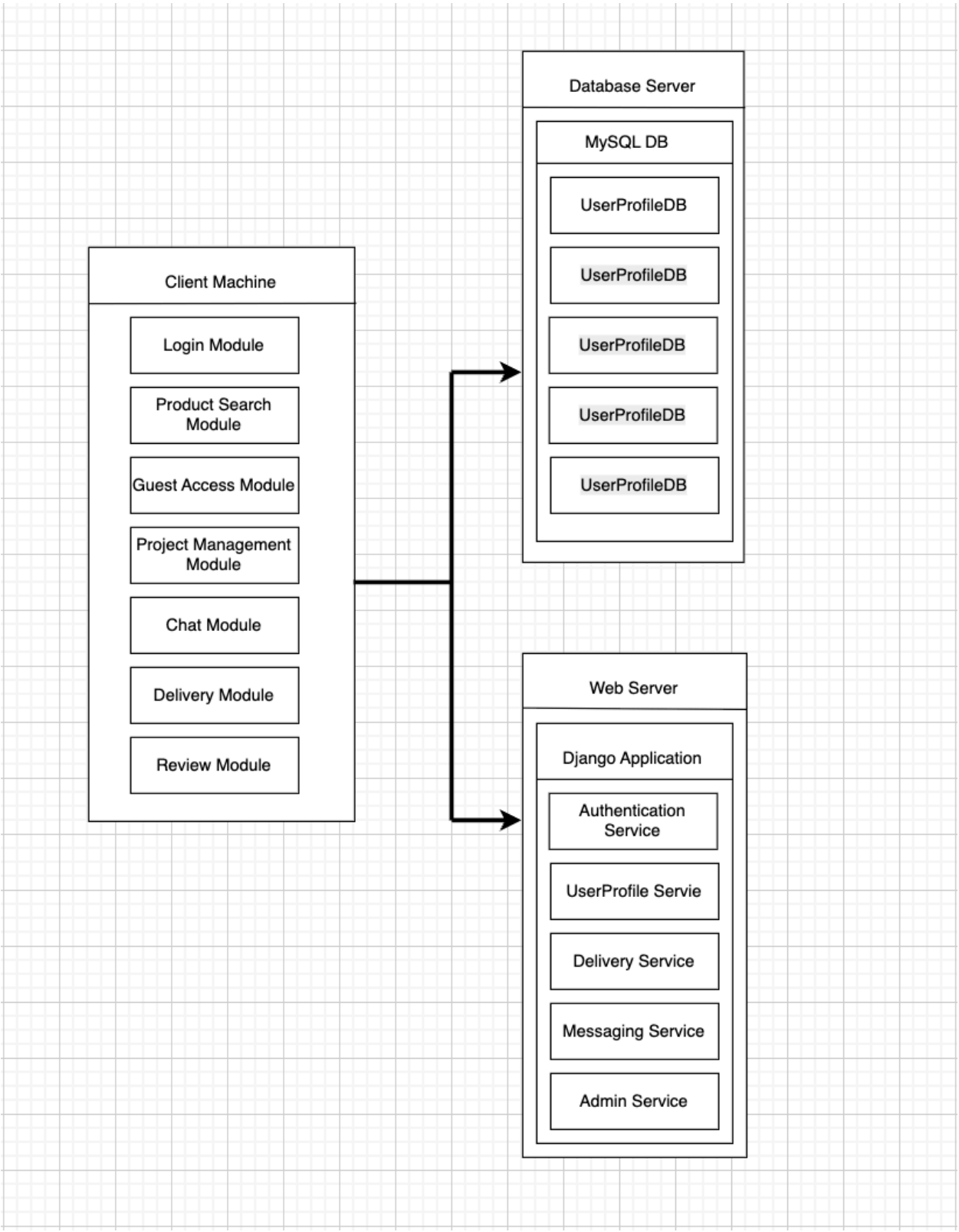
This system builds trust and keeps users returning to the app.

5. HIGH-LEVEL UML DIAGRAMS

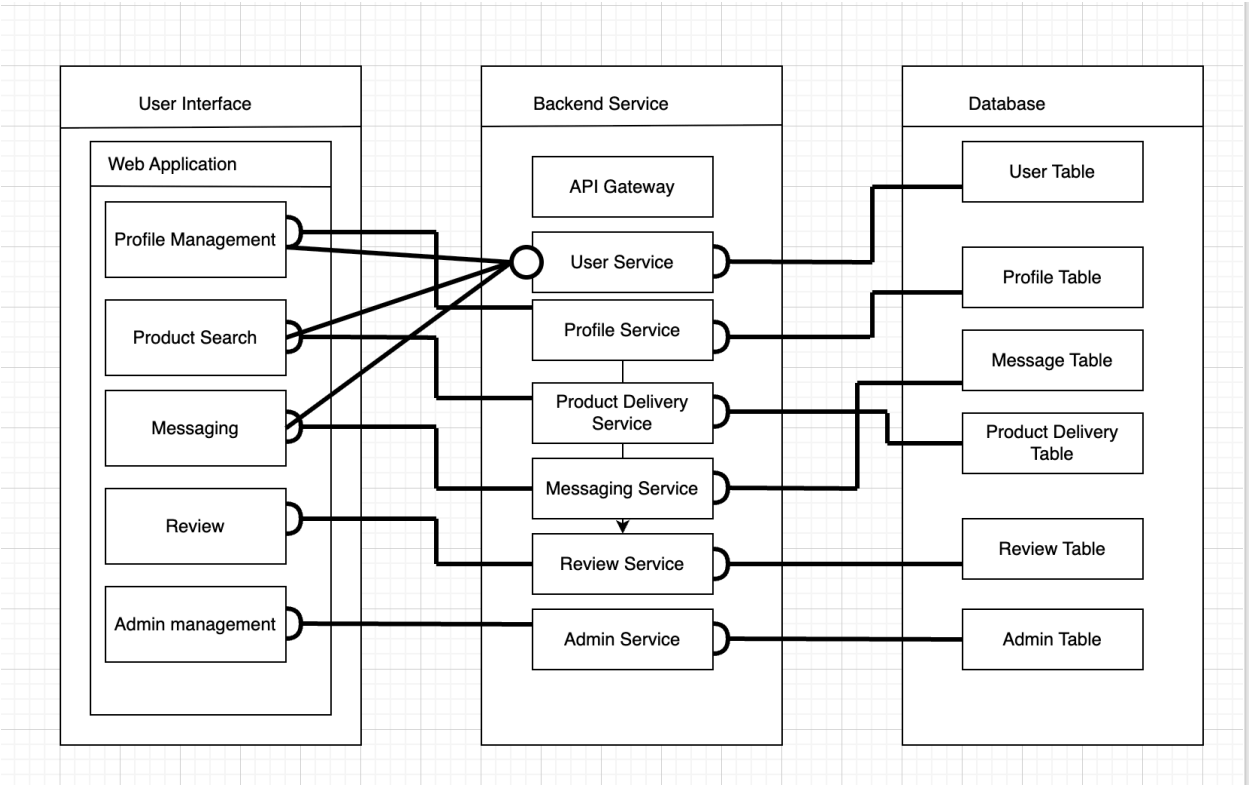
Class Diagram



Deployment Diagram



Component Diagram



6. KEY RISKS

6.1. Skills Risks

Risk:
The two major technologies our team is using — React and Django — are new for two of our members. Besides that, we need to learn to use some lightweight software for some basic tasks such as Miro for UI, Azure for deployment, which are easy to learn but would need some time to set up and understand.

Mitigation:
The team has already started to learn the technologies and are trying to learn most of it before setting up the project, and we also track the progress of each member.

6.2. Schedule Risks

Risk:
Multiple complex features (auction, delivery matching, multilingual chat, gamification) may stretch the timeline.

Mitigation:
We will prioritize core features for the P1 (listings, messaging, basic delivery, wanted items page), and list the complex ones (gamification, bidding) to P2 and P3.

6.3. Technical Risks

Risk:
Some planned features (e.g., real-time bidding, in-app translation, delivery tracking) may face compatibility or performance issues with the selected tech stack — especially integrating these with React 19, Python 3.13, and MySQL 8.0. There may also be bugs or missing support in the latest versions of these tools.

Mitigation:
Early prototyping and feature-specific testing will be done to verify feasibility. The team will monitor official release notes and known issues for React 19 and Python 3.13. If major incompatibilities arise, fallback libraries or version downgrades will be considered to ensure stable implementation.

6.4. Teamwork Risks

Risk:
Coordination across different roles (frontend, backend, CI/CD) may be challenging, especially when syncing features or fixing bugs across branches. Additionally, since most team members are in their first or second semester, they may face a heavier load of written exams this semester, which could affect availability later in the project.

Mitigation:
The team has decided to begin development early to reduce pressure toward the end of the term. Regular stand-ups and biweekly integration sessions are scheduled to ensure alignment. Tasks will be clearly assigned and tracked using GitHub issues, Trello, and Discord for continuous collaboration and progress updates.

6.5. Legal/Content Risks

Risk 1:
Unfree images are used to illustrate certain properties and show their interior. While highly unlikely, a copyright holder could stumble upon the website to check for an apartment and report the infringement.

Mitigation 1:
Check the licensing of all images before using them.

Risk 2:
The lack of clear communication about the switch from Azure's free credits to paid usage for a student account could lead to unexpected charges, which might create a legal risk.

Mitigation 2:
Stopping the server and contacting Azure support to resolve the billing issue can help prevent further unexpected charges.

7. PROJECT MANAGEMENT

Starting Milestone 2, our team has made significant progress in organizing and streamlining our development workflow. We are now working more independently in our respective frontend and backend roles, allowing us to focus on our specific areas while maintaining alignment through regular online meetings. Pair programming sessions have become more frequent, particularly to ensure a solid project setup and smooth integration between components. We have also started actively using our Trello board to manage tasks, track progress, and coordinate responsibilities more efficiently. To further enhance collaboration, we rely on API documentation to facilitate clear communication between frontend and backend teams. Additionally, we've set up dedicated Discord channels to compartmentalize our discussions, making it easier to stay focused and organized as the project evolves.