The Hive - Final Project Report

Name: Metin Cem Doğan
Course: SWE573 Software Development Practice
Date: December 2025
Project Name: The Hive - Community Time Banking Platform
Deployment URL: http://ec2-52-59-134-106.eu-central-1.compute.amazonaws.com:3000/
Git Repository URL: https://github.com/metincd/SWE573-FALL2025
Git Tag Version URL: https://github.com/metincd/SWE573-FALL2025/releases/tag/v0.9 **Functionality of the System Video**: https://drive.google.com/file/d/1-7QTCoadfYialKfkYjolmDY4QOMmyoyD/view?usp=sharing

Honor Code Declaration

I, **Metin Cem Doğan**, being a student in the Software Engineering MS program at Boğaziçi University, registered for SWE573 during Fall 2025, hereby declare that:

1. All submitted material (project repository, final project report, and supplementary documents) has been exclusively prepared by me.
2. The material was prepared individually without assistance, except for explicitly disclosed permitted peer assistance.
3. All third-party software used in this project adheres to their respective licenses (see Third-Party Software section).

Signature: Metin Cem Doğan **Date**: December 2025

Table of Contents

1. Overview

1.1 Project Description

The Hive is a community-driven time banking platform designed to facilitate service exchange and strengthen local community bonds. The platform enables users to offer and request services using a virtual currency called Time Bank Hours (TBH), where one hour of service equals one TBH. By replacing monetary transactions with time-based exchanges, The Hive promotes reciprocity, mutual aid, and community solidarity.

1.2 Project Objectives

The primary objectives of The Hive project were:

1. **Develop a functional time banking platform** that enables users to exchange services using time credits
2. **Create an intuitive user interface** for service discovery, management, and community interaction
3. **Implement a robust backend system** with proper authentication, authorization, and data management
4. **Build community features** including forums, messaging, and review systems
5. **Ensure platform security and moderation** through reporting and administrative tools
6. **Deploy the application** to a production environment accessible to users

1.3 Technology Stack

Backend

- **Framework**: Django 4.2.25
- **API**: Django REST Framework 3.16.1
- **Authentication**: JWT via djangorestframework-simplejwt 5.5.1
- **Database**: PostgreSQL 15
- **Geocoding**: OpenStreetMap Nominatim integration
- **File Storage**: Local media storage for avatars and service images
- **Web Server**: Gunicorn 21.2.0 (production)

Frontend

- **Framework**: React 19.1.1 with TypeScript
- **Build Tool**: Vite 7.1.14
- **Styling**: Tailwind CSS 4.1.16
- **Maps**: Leaflet 1.9.4 with react-leaflet 5.0.0
- **State Management**: TanStack Query (React Query) 5.90.6
- **Routing**: React Router DOM 7.9.5
- **HTTP Client**: Axios 1.13.1

Infrastructure

- **Containerization**: Docker and Docker Compose
- **Database**: PostgreSQL with health checks
- **Deployment**: AWS EC2 (eu-central-1 region)
- **CORS**: Configured for cross-origin requests

1.4 Key Features

- **Service Exchange**: Create service offers and needs with interactive map visualization
- **Time Banking System**: Virtual currency (TBH) with automatic credit transfers
- **User Management**: Registration, authentication, profile management
- **Communication**: Private messaging and public forums
- **Review System**: Comprehensive review and rating system
- **Moderation**: Content reporting and administrative tools

---

2. Software Requirements Specification

The complete Software Requirements Specification (SRS) document is available as a separate document: `SOFTWARE_REQUIREMENTS_SPECIFICATION.md`

The SRS follows IEEE 830-1998 standards and includes:

- **Functional Requirements**: 19 categories (FR-1 through FR-19) covering all system features
- **Non-Functional Requirements**: 7 categories (NFR-1 through NFR-7) covering performance, security, reliability, usability, scalability, accessibility, and maintainability
- **External Interface Requirements**: User, hardware, software, and communication interfaces
- **System Constraints**: Design and implementation constraints

2.1 Requirements Summary

The SRS document contains:

- **Functional Requirements**: 200+ individual requirements
- **Non-Functional Requirements**: 30+ individual requirements

- All requirements are marked with implementation status (Implemented, Partially Implemented, Not Implemented)

---

3. Design Documents

3.1 System Architecture

The Hive follows a three-tier architecture:

```
┌─────────────────────────────────────────────────────────┐
│                   Frontend Layer                          │
│  React 19.1.1 + TypeScript + Vite + Tailwind CSS        │
│  - Component-based architecture                           │
│  - TanStack Query for server state                       │
│  - React Router for navigation                            │
└────────────────────────────────────────────────────────┘
                   │
                   │ HTTP/REST API
┌──────────────────▼─────────────────────────────────────┐
│                   Backend Layer                           │
│  Django 4.2.25 + Django REST Framework 3.16.1           │
│  - RESTful API design                                     │
│  - JWT authentication                                     │
│  - Business logic and validation                          │
└────────────────────────────────────────────────────────┘
                   │
                   │ SQL Queries
┌──────────────────▼─────────────────────────────────────┐
│                 Database Layer                            │
│  PostgreSQL 15                                            │
│  - 20+ models                                             │
│  - Proper indexes and constraints                         │
└─────────────────────────────────────────────────────────┘
```

3.2 Database Design

The database schema includes 20+ models organized into the following categories:

User Management

- **User**: Core user authentication model (email-based)
- **Profile**: Extended user profile with display name, bio, avatar, location

Service Management

- **Service**: Service offers and needs with location, tags, capacity
- **ServiceRequest**: Service request workflow management
- **ServiceSession**: Scheduled service sessions
- **Completion**: Service completion tracking

Time Banking

- **TimeAccount**: User time account with balance tracking
- **TimeTransaction**: Complete transaction history with audit trail

Communication

- **Conversation**: Private messaging channels
- **Message**: Individual messages within conversations
- **Thread**: Public forum discussion threads
- **Post**: Individual posts in forum threads

Community Features

- **Review**: User reviews with ratings (1-5 stars)
- **ReviewHelpfulVote**: Helpful votes on reviews
- **UserRating**: Aggregated rating summaries
- **ThankYouNote**: Thank you notes between users

Moderation

- **Report**: Content reporting system (generic foreign key)
- **ModerationAction**: Administrative actions taken
- **Notification**: System notifications

System

- **Tag**: Service and thread categorization with Wikidata integration

3.3 API Design

The REST API follows RESTful conventions:

- **Resource-based URLs**: `/api/services/`, `/api/service-requests/`, etc.
- **HTTP Methods**: GET (retrieve), POST (create), PUT/PATCH (update), DELETE (remove)
- **JSON Format**: All requests and responses use JSON
- **Authentication**: JWT tokens via Authorization header
- **Pagination**: Page-based pagination for list endpoints
- **Filtering**: Query parameters for filtering and search
- **Custom Actions**: Custom endpoints for complex operations (e.g., `/api/service-requests/{id}/complete/`)

3.4 Frontend Architecture

The frontend follows a component-based architecture:

```
hive_frontend/
├── src/
│   ├── pages/              # Main application pages
│   │   ├── Home.tsx        # Home page with map
│   │   ├── Services.tsx    # Service list view
│   │   ├── Profile.tsx     # User profile
│   │   ├── Chat.tsx        # Messaging interface
│   │   ├── Forum.tsx       # Forum discussions
│   │   └── ...
│   ├── components/         # Reusable components
│   │   ├── Layout.tsx      # Main layout wrapper
│   │   ├── ReportButton.tsx
│   │   └── ui/             # UI components (Card, TextInput, etc.)
│   ├── contexts/           # React contexts
│   │   └── AuthContext.tsx # Authentication context
│   ├── api.ts             # API client configuration
│   └── main.tsx           # Application entry point
```

3.5 Deployment Architecture

```
┌─────────────────────────────────────────────────────────┐
│                    AWS EC2 Instance                      │
│                                                          │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐ │
│  │   Frontend   │  │   Backend    │  │  Database    │ │
│  │   (Nginx)    │  │  (Gunicorn)  │  │ (PostgreSQL) │ │
│  │   Port 3000  │  │   Port 8000  │  │   Port 5432  │ │
│  └──────────────┘  └──────────────┘  └──────────────┘ │
│                                                          │
│  All services containerized with Docker Compose         │
└─────────────────────────────────────────────────────────┘
```

---

4. Status of Your Project

This section describes the status of each requirement category from the SRS document. "Completed" means the requirement is documented, tested, and deployed.

4.1 Functional Requirements Status

FR-1: User Account Management

- **Status**: Completed

- **Implementation**: All 13 requirements implemented
- **Testing**: Unit tests (UT-1), System tests (ST-1)
- **Deployment**: Deployed and functional

FR-2: Service Map and Post Management

- **Status**: Completed
- **Implementation**: All 14 requirements implemented
- **Testing**: Unit tests (UT-4), System tests (ST-2)
- **Deployment**: Deployed and functional

FR-3: Search and Discovery

- **Status**: Completed
- **Implementation**: All 14 requirements implemented
- **Testing**: System tests (ST-2, ST-7)
- **Deployment**: Deployed and functional

FR-4: Service Exchange ("Handshake") Process

- **Status**: Completed
- **Implementation**: All 19 requirements implemented
- **Testing**: Unit tests (UT-5), System tests (ST-3), Use case tests (UC-1)
- **Deployment**: Deployed and functional

FR-5: Time Bank System

- **Status**: Completed
- **Implementation**: All 17 requirements implemented
- **Testing**: Unit tests (UT-6, UT-7), System tests (ST-4), Use case tests (UC-4)
- **Deployment**: Deployed and functional

FR-6: Community Forums (Common Area)

- **Status**: Completed
- **Implementation**: All 14 requirements implemented
- **Testing**: System tests (ST-8)
- **Deployment**: Deployed and functional

FR-7: Administration and Moderation

- **Status**: Completed
- **Implementation**: All 17 requirements implemented
- **Testing**: System tests (ST-10), Use case tests (UC-5)
- **Deployment**: Deployed and functional

FR-8: Forum and Discussion System

- **Status**: Completed
- **Implementation**: All 14 requirements implemented
- **Testing**: System tests (ST-8)
- **Deployment**: Deployed and functional

FR-9: Review and Rating System

- **Status**: Completed
- **Implementation**: All 16 requirements implemented
- **Testing**: Unit tests (UT-11), System tests (ST-9)
- **Deployment**: Deployed and functional

FR-10: Thank You Notes

- **Status**: Completed
- **Implementation**: All 10 requirements implemented
- **Testing**: Included in system tests
- **Deployment**: Deployed and functional

FR-11: Reporting and Moderation System

- **Status**: Completed
- **Implementation**: All 19 requirements implemented
- **Testing**: System tests (ST-10), Use case tests (UC-5)
- **Deployment**: Deployed and functional

FR-12: Notification System

- **Status**: Partially Completed
- **Implementation**: 12 out of 15 requirements implemented

- **Testing**: Basic functionality tested
- **Deployment**: Deployed with partial functionality
- **Notes**: Notification generation for some events not fully implemented

FR-13: User Profile Management

- **Status**: Completed
- **Implementation**: All 8 requirements implemented
- **Testing**: System tests (ST-1)
- **Deployment**: Deployed and functional

FR-14: Admin Panel

- **Status**: Completed
- **Implementation**: All 8 requirements implemented
- **Testing**: Manual testing performed
- **Deployment**: Deployed and functional

FR-15: Service Sessions

- **Status**: Completed
- **Implementation**: All 7 requirements implemented
- **Testing**: Included in system tests
- **Deployment**: Deployed and functional

FR-16: Completion Tracking

- **Status**: Completed
- **Implementation**: All 6 requirements implemented
- **Testing**: Included in use case tests
- **Deployment**: Deployed and functional

FR-17: Map Visualization

- **Status**: Completed
- **Implementation**: All 8 requirements implemented
- **Testing**: Manual testing performed
- **Deployment**: Deployed and functional

FR-18: Service Images

- **Status**: Partially Completed
- **Implementation**: 4 out of 5 requirements implemented
- **Testing**: Basic functionality tested
- **Deployment**: Deployed with partial functionality
- **Notes**: Image format validation partially implemented

FR-19: Health Check and Monitoring

- **Status**: Partially Completed
- **Implementation**: 2 out of 4 requirements fully implemented
- **Testing**: Basic health check tested
- **Deployment**: Deployed with partial functionality
- **Notes**: Database connectivity check partially implemented

4.2 Non-Functional Requirements Status

NFR-1: Usability

- **Status**: Completed
- **Implementation**: 8 out of 9 requirements implemented
- **Notes**: Query result caching not implemented (NFR-1.8)

NFR-2: Performance

- **Status**: Completed
- **Implementation**: All 9 requirements implemented

NFR-3: Scalability

- **Status**: Partially Completed
- **Implementation**: 6 out of 7 requirements implemented
- **Notes**: Database transactions for multi-step operations partially implemented (NFR-3.6)

NFR-4: Security and Privacy

- **Status**: Partially Completed

- **Implementation**: 9 out of 11 requirements implemented
- **Notes**: HTTPS not fully configured in production (NFR-4.2), some accessibility features partially implemented

NFR-5: Reliability

- **Status**: Partially Completed
- **Implementation**: 5 out of 7 requirements implemented
- **Notes**: 99.5% uptime target partially met (NFR-5.1), automatic daily backups not implemented (NFR-5.2), API documentation not provided (NFR-5.7)

NFR-6: Accessibility

- **Status**: Partially Completed
- **Implementation**: 7 out of 8 requirements implemented
- **Notes**: Screen reader support partially implemented (NFR-6.1)

NFR-7: Maintainability

- **Status**: Partially Completed
- **Implementation**: 2 out of 3 requirements implemented
- **Notes**: Open source license not specified (NFR-7.2), code documentation partially complete (NFR-7.3)

4.3 Overall Project Status

- **Total Functional Requirements**: 200+
- **Completed**: ~185 (92.5%)
- **Partially Completed**: ~15 (7.5%)
- **Not Completed**: ~5 (2.5%)
- **Total Non-Functional Requirements**: 30+
- **Completed**: ~20 (66.7%)
- **Partially Completed**: ~10 (33.3%)

Overall Completion Rate: ~85%

---

5. Status of Deployment

5.1 Deployment Information

- **Deployment URL**: http://ec2-52-59-134-106.eu-central-1.compute.amazonaws.com:3000/
- **Backend API URL**: http://ec2-52-59-134-106.eu-central-1.compute.amazonaws.com:8000/api
- **Admin Panel URL**: http://ec2-52-59-134-106.eu-central-1.compute.amazonaws.com:8000/admin
- **Dockerized**: Yes
- **Deployment Platform**: AWS EC2 (eu-central-1 region)
- **Deployment Date**: December 2025

5.2 Docker Configuration

The project is fully containerized using Docker and Docker Compose:

- **Backend Container**: Django application with Gunicorn
- **Frontend Container**: React application served via Nginx
- **Database Container**: PostgreSQL 15
- **Docker Compose**: Orchestration of all services
- **Production Configuration**: Separate docker-compose.prod.yml for production settings

5.3 Deployment Status

- **Frontend**: Deployed and accessible
- **Backend API**: Deployed and accessible
- **Database**: Deployed and connected
- **Health Check**: Available at `/api/health/`
- **SSL/HTTPS**: Not configured (HTTP only)
- **Monitoring**: Basic monitoring in place

5.4 Test Credentials

Note: For testing the deployed system, you can register a new account or use the following test accounts (if available):

- **Test User 1**:
  - Email: `ayberk@gmail.com`
  - Password: `ayberk123`

    - Initial Balance: 3 TBH

- **Test User 2 (Admin)**:
    - Email: `deniz@gmail.com`
    - Password: `deniz123`
    - Initial Balance: 3 TBH

Note: These are example test accounts from the test suite. For production testing, please register new accounts through the registration page.

---

6. Full Installation Instructions

6.1 System Requirements

Minimum Requirements

- **Operating System**: Linux, macOS, or Windows (with WSL2 for Windows)
- **Python**: 3.10 or higher
- **Node.js**: 18 or higher
- **npm**: Comes with Node.js
- **PostgreSQL**: 15 or higher
- **Docker**: 20.10 or higher (optional, for containerized deployment)
- **Docker Compose**: 2.0 or higher (optional, for containerized deployment)
- **Git**: For cloning the repository

Recommended Requirements

- **RAM**: 4GB or more
- **Disk Space**: 2GB free space
- **Internet Connection**: Required for package installation and geocoding API

6.2 Option 1: Docker Compose Installation (Recommended)

Step 1: Clone the Repository

```
git clone https://github.com/metincd/SWE573-FALL2025.git
cd SWE573-FALL2025
```

Step 2: Create Environment File

```
cp .env.example .env
```

Edit `.env` file with your configuration:

```
# Database Configuration
POSTGRES_DB=hive_db
POSTGRES_USER=hive_user
POSTGRES_PASSWORD=hive_password

# Django Configuration
DEBUG=True
SECRET_KEY=your-secret-key-here
ALLOWED_HOSTS=localhost,127.0.0.1

# CORS Configuration
CORS_ALLOWED_ORIGINS=http://localhost:3000,http://127.0.0.1:3000

# Frontend API URL
VITE_API_BASE_URL=http://localhost:8000/api
```

Step 3: Build and Start Services

```
docker-compose up --build
```

For production deployment:

```
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build
```

Step 4: Run Database Migrations

```
docker-compose exec backend python manage.py migrate
```

Step 5: Create Superuser (Optional)

```
docker-compose exec backend python manage.py createsuperuser
```

Step 6: Access the Application

- **Frontend**: http://localhost:3000
- **Backend API**: http://localhost:8000/api
- **Admin Panel**: http://localhost:8000/admin

6.3 Option 2: Manual Installation

Backend Setup

1. **Navigate to project directory**:

   ```
   cd SWE573-FALL2025
   ```

2. **Create virtual environment**:

   ```
   python -m venv venv
   source venv/bin/activate  # On Windows: venv\Scripts\activate
   ```

3. **Install Python dependencies**:

   ```
   pip install -r requirements.txt
   ```

4. **Set up environment variables**:

   ```
   export DEBUG=True
   export SECRET_KEY=your-secret-key-here
   export DB_NAME=hive_db
   export DB_USER=hive_user
   export DB_PASSWORD=hive_password
   export DB_HOST=localhost
   export DB_PORT=5432
   ```

5. **Create PostgreSQL database**:

   ```
   createdb hive_db
   ```

6. **Run migrations**:

   ```
   python manage.py migrate
   ```

7. **Create superuser** (optional):

   ```
   python manage.py createsuperuser
   ```

8. **Start development server**:

   ```
   python manage.py runserver
   ```

Frontend Setup

1. **Navigate to frontend directory**:

   ```
   cd hive_frontend
   ```

2. **Install dependencies**:

```
npm install
```

3. **Create `.env` file**:

```
echo "VITE_API_BASE_URL=http://localhost:8000/api" > .env
```

4. **Start development server**:

```
npm run dev
```

The frontend will be available at http://localhost:5173

6.4 Production Deployment

AWS EC2 Deployment Steps

1. **Launch EC2 Instance**:
   - Choose Ubuntu 22.04 LTS
   - Select instance type (t2.micro or larger)
   - Configure security groups to allow ports 3000, 8000, and 22
2. **Connect to Instance**:

```
ssh -i your-key.pem ubuntu@your-ec2-ip
```

3. **Install Docker and Docker Compose**:

```
sudo apt update
sudo apt install docker.io docker-compose -y
sudo usermod -aG docker ubuntu
```

4. **Clone Repository**:

```
git clone https://github.com/metincd/SWE573-FALL2025.git
cd SWE573-FALL2025
```

5. **Configure Environment Variables**:

```
export DEBUG=False
export SECRET_KEY=your-production-secret-key
export ALLOWED_HOSTS=ec2-52-59-134-106.eu-central-1.compute.amazon.com,localhost
export CORS_ALLOWED_ORIGINS=http://ec2-52-59-134-106.eu-central-1.compute.amazonaws.com:3000
export VITE_API_BASE_URL=http://ec2-52-59-134-106.eu-central-1.compute.amazonaws.com:8000/api
```

6. **Start Services**:

```
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build
```

7. **Run Migrations**:

```
docker-compose exec backend python manage.py migrate
```

8. **Collect Static Files**:

```
docker-compose exec backend python manage.py collectstatic --noinput
```

6.5 Troubleshooting

Common Issues

1. **Database Connection Error**:
   - Ensure PostgreSQL is running
   - Check database credentials in `.env`
   - Verify database exists: `psql -U hive_user -d hive_db`
2. **Port Already in Use**:
   - Change ports in `docker-compose.yml`
   - Or stop conflicting services
3. **Frontend Can't Connect to Backend**:

3. **Frontend Can't Connect to Backend**:

   - Check `VITE_API_BASE_URL` in frontend `.env`
   - Verify backend is running
   - Check CORS configuration

4. **Migration Errors**:

   - Ensure database is created
   - Check database user permissions
   - Try: `python manage.py migrate --run-syncdb`

---

7. User Manual

7.1 Getting Started

7.1.1 Registration

1. Navigate to the registration page
2. Enter your email address
3. Create a password (minimum requirements apply)
4. Confirm your password
5. Optionally provide your location (latitude/longitude)
6. Click "Register"
7. You will automatically receive 3 TBH (Time Bank Hours) as a welcome bonus

7.1.2 Login

1. Navigate to the login page
2. Enter your email and password
3. Click "Login"
4. You will be redirected to the home page

7.1.3 Profile Setup

1. Click on your profile icon in the navigation bar
2. Click "Edit Profile"
3. Add your display name (public name shown to others)
4. Write a bio describing yourself and your interests
5. Upload an avatar image (optional, max 5MB)
6. Update your location if needed
7. Click "Save"

7.2 Creating Services

7.2.1 Create a Service Offer

1. Click "Create Service" in the navigation bar
2. Select "Offer" as service type
3. Enter a title (e.g., "Math Tutoring")
4. Write a detailed description
5. Specify estimated hours required
6. Add your location (address or coordinates)
7. Select or create tags (e.g., "tutoring", "education")
8. Optionally upload a service image
9. Set capacity (default: 1, increase for group services)
10. Click "Create Service"

7.2.2 Create a Service Need

1. Follow the same steps as creating an offer
2. Select "Need" as service type
3. Describe what service you are seeking

7.3 Discovering Services

7.3.1 Map View

1. Navigate to the home page
2. You will see an interactive map with service markers
3. **Green markers**: Service offers
4. **Blue markers**: Service needs
5. Click on a marker to see service details
6. Use zoom and pan to explore different areas

7.3.2 List View

1. Click "Services" in the navigation bar
2. View all services in a list format
3. Use filters to narrow down results:
   - Filter by type (Offer/Need)
   - Filter by status (Active/Inactive/Completed)
   - Filter by tags
   - Search by title or description
4. Click on a service to view details

7.4 Requesting Services

7.4.1 Send a Service Request

1. Browse services on the map or list view
2. Click on a service to view details
3. Click "Request Service"
4. Optionally add a message to the service owner
5. Click "Send Request"
6. The system will check if you have sufficient time credits
7. A private conversation will be automatically created

7.4.2 Manage Service Requests

1. Click "Messages" in the navigation bar
2. View your service requests in the conversations list
3. Request statuses:
   - **Pending**: Waiting for owner response
   - **Accepted**: Owner accepted your request
   - **Rejected**: Owner rejected your request
   - **In Progress**: Service has started
   - **Completed**: Service completed

7.5 Completing Services

7.5.1 Start a Service

1. Once a request is accepted, both parties must approve to start
2. Click "Approve Start" when ready
3. Wait for the other party to approve
4. Service status changes to "In Progress"

7.5.2 Complete a Service

1. After performing the service, navigate to the service request
2. Click "Complete Service"
3. Wait for the other party to also mark as complete
4. Once both parties confirm, time credits are automatically transferred
5. Service status changes to "Completed"

7.6 Messaging

7.6.1 Send Messages

1. Click "Messages" in the navigation bar
2. Select a conversation
3. Type your message in the input field
4. Press Enter or click "Send"
5. Messages are displayed in chronological order

7.6.2 Manage Conversations

1. View all conversations in the messages list
2. Conversations are sorted by most recent activity
3. Unread message count is shown for each conversation
4. Click on a conversation to view messages
5. Archive conversations you no longer need

7.7 Forum Participation

7.7.1 Create a Thread

1. Click "Forum" in the navigation bar
2. Click "Create Thread"
3. Enter a title
4. Write your post content
5. Add tags if relevant
6. Click "Create Thread"

7.7.2 Reply to Threads

1. Browse forum threads
2. Click on a thread to view posts
3. Scroll to the bottom
4. Type your reply
5. Click "Post Reply"

7.8 Reviews and Ratings

7.8.1 Write a Review

1. After completing a service, navigate to the service details
2. Click "Write Review"
3. Select review type (Service Provider, Service Receiver, or Service Quality)
4. Rate from 1 to 5 stars
5. Write a title and detailed review
6. Optionally mark as anonymous
7. Click "Submit Review"

7.8.2 View Reviews

1. Navigate to a user's profile
2. View their rating summary and reviews
3. Filter reviews by type
4. Mark reviews as helpful

7.9 Reporting Content

1. Navigate to the content you want to report (service, post, message, etc.)
2. Click "Report" button
3. Select a reason (spam, inappropriate content, harassment, etc.)
4. Provide a detailed description
5. Optionally add evidence URL
6. Click "Submit Report"
7. Administrators will review your report

7.10 Admin Panel

7.10.1 Access Admin Panel

1. Login as a staff user
2. Navigate to Admin Panel
3. View system statistics:
    - Service statistics
    - User statistics
    - Report statistics
    - Forum statistics

7.10.2 Moderate Content

1. View pending reports
2. Review report details
3. Take appropriate action:
    - Delete content
    - Suspend user
    - Ban user
    - Dismiss report
4. System will notify affected users

---

8. Use Case: End-to-End Demo

8.1 Scenario: Complete Service Exchange

This use case demonstrates a complete service exchange from request to completion and time transfer.

Actors

- **Aysegul**: Service provider (offers "Math Tutoring")
- **Metin**: Service requester (needs math tutoring)

Preconditions

- Aysegul has registered and created a service offer

- Metin has registered and has sufficient time credits (3 TBH initial balance)
- Both users are logged in

Steps

Step 1: Aysegul Creates a Service Offer

1. Aysegul logs in and navigates to "Create Service"
2. Selects "Offer" as service type
3. Enters:
    - Title: "Math Tutoring for High School Students"
    - Description: "I offer one-on-one math tutoring sessions. I can help with algebra, geometry, and calculus."
    - Estimated Hours: 2
    - Location: "Istanbul, Turkey" (coordinates: 41.0082, 28.9784)
    - Tags: "tutoring", "education", "math"
    - Capacity: 1
4. Clicks "Create Service"
5. **Result**: Service is created and appears on the map with a green marker

Step 2: Metin Discovers the Service

1. Metin logs in and navigates to the home page
2. Sees Aysegul's service on the map (green marker)
3. Clicks on the marker to view service details
4. Reads the description and sees it requires 2 TBH
5. Checks his balance: 3 TBH (sufficient)

Step 3: Metin Requests the Service

1. Metin clicks "Request Service"
2. Adds message: "Hi Aysegul, I need help with calculus. Are you available this weekend?"
3. Clicks "Send Request"
4. **Result**:
    - Service request created with status "Pending"
    - Private conversation automatically created
    - Metin's message is added as first message in conversation

Step 4: Aysegul Accepts the Request

1. Aysegul receives notification (if implemented) or checks her messages
2. Navigates to "Messages" and sees new conversation
3. Opens conversation and reads Metin's request message
4. Navigates to service requests dashboard
5. Sees Metin's request with status "Pending"
6. Clicks "Accept Request"
7. Adds response note: "Yes, I'm available this weekend. Let's schedule a time."
8. **Result**:
    - Request status changes to "Accepted"
    - Conversation is active for coordination

Step 5: Both Parties Approve to Start

1. Aysegul and Metin coordinate details via private messaging
2. They agree on Saturday at 2 PM
3. Aysegul clicks "Approve Start" on the service request
4. Metin clicks "Approve Start" on the service request
5. **Result**:
    - Service request status changes to "In Progress"
    - Service is now active

Step 6: Service is Performed

1. On Saturday, Aysegul provides 2 hours of math tutoring to Metin
2. Both parties are satisfied with the service

Step 7: Both Parties Complete the Service

1. Aysegul navigates to the service request
2. Clicks "Complete Service"
3. **Result**: System shows "Waiting for Metin to confirm"
4. Metin navigates to the service request
5. Clicks "Complete Service"
6. **Result**:
    - Both parties have confirmed
    - Time transfer is automatically processed:
        - Metin's account: 3 TBH - 2 TBH = 1 TBH (debit transaction)
        - Aysegul's account: 3 TBH + 2 TBH = 5 TBH (credit transaction)
    - Service request status changes to "Completed"
    - Service status changes to "Completed"

Step 8: Metin Writes a Review

1. Metin navigates to Aysegul's profile
2. Clicks "Write Review"
3. Selects "Service Provider" review type
4. Rates 5 stars
5. Writes title: "Excellent tutor!"
6. Writes content: "Aysegul was very patient and explained calculus concepts clearly. Highly recommended!"
7. Clicks "Submit Review"

8. **Result**: Review is published and appears on Aysegul's profile

Step 9: Aysegul Sends Thank You Note

1. Aysegul navigates to Metin's profile
2. Clicks "Send Thank You Note"
3. Writes: "Thank you for being such an engaged student! Good luck with your studies."
4. Clicks "Send"
5. **Result**: Thank you note is sent to Metin

Postconditions

- Service exchange is completed
- Time credits have been transferred (Metin: 1 TBH, Aysegul: 5 TBH)
- Review has been written and published
- Thank you note has been sent
- Both users have positive experience with the platform

Verification

- Service request shows status "Completed"
- Metin's time account balance: 1 TBH
- Aysegul's time account balance: 5 TBH
- Transaction history shows debit for Metin and credit for Aysegul
- Review appears on Aysegul's profile
- Thank you note appears in Metin's received notes

---

9. Test Plan and Results

Test Documentation Reference: https://github.com/metincd/SWE573-FALL2025/wiki/Testing-Documentation

9.1 Test Strategy

The testing strategy follows a three-tier approach as documented in the Testing Documentation wiki page:

1. **Unit Tests (UT-X.Y.Z)**: Test individual model methods and properties
2. **System Tests (ST-X.Y.Z)**: Test API endpoints and integration
3. **Use Case Tests (UC-X.Y)**: Test complete user workflows

9.2 Test Execution

Tests are executed using Django's test framework:

```
python manage.py test the_hive.tests --verbosity=2
```

Or using the test script:

```
./run_tests.sh
```

The test script generates:

- `test_results.txt` - Detailed test execution output
- `coverage_report.txt` - Code coverage report
- `htmlcov/` - HTML coverage report

9.3 Test Execution Summary

Execution Date: December 15, 2025
Test Environment: Django Test Framework, PostgreSQL Test Database
Total Tests: 53
Test Framework: Django TestCase, REST Framework APIClient

Test Category	Total	Passed	Failed	Errors	Status
Unit Tests (UT-*)	28	28	0	0	Pass
System Tests (ST-*)	20	20	0	0	Pass
Use Case Tests (UC-*)	5	5	0	0	Pass
Total	**53**	**53**	**0**	**0**	**100% Pass**

9.4 Unit Test Results (UT-X.Y.Z)

UT-1: User Model Tests - All Pass

- **UT-1.1.1**: User creation with email and password PASSED
- **UT-1.1.2**: User full_name property with first and last name PASSED
- **UT-1.1.3**: User full_name fallback to email when names empty PASSED
- **UT-1.1.4**: User string representation PASSED

UT-2: Profile Model Tests - All Pass

- **UT-2.1.1**: Profile creation with all fields PASSED
- **UT-2.1.2**: Profile one-to-one relationship with User PASSED
- **UT-2.1.3**: Profile location coordinates storage PASSED

UT-3: Tag Model Tests - All Pass

- **UT-3.1.1**: Tag creation with name and slug PASSED
- **UT-3.1.2**: Automatic slug generation from name PASSED
- **UT-3.1.3**: Wikidata URL generation from wikidata_id PASSED

UT-4: Service Model Tests - All Pass

- **UT-4.1.1**: Service creation with all fields PASSED
- **UT-4.1.2**: Default status is 'active' PASSED
- **UT-4.1.3**: Default capacity is 1 PASSED

UT-5: ServiceRequest Model Tests - All Pass

- **UT-5.1.1**: Service request creation PASSED
- **UT-5.1.2**: Unique constraint - one request per user per service PASSED

UT-6: TimeAccount Model Tests - All Pass

- **UT-6.1.1**: Time account creation PASSED
- **UT-6.1.2**: Participation ratio calculation PASSED
- **UT-6.1.3**: Participation ratio when total_spent is zero PASSED
- **UT-6.1.4**: Is positive balance property PASSED

UT-7: TimeTransaction Model Tests - All Pass

- **UT-7.1.1**: Transaction creation PASSED
- **UT-7.1.2**: Signed amount for credit transactions PASSED
- **UT-7.1.3**: Signed amount for debit transactions PASSED

UT-8: Conversation Model Tests - All Pass

- **UT-8.1.1**: Conversation creation PASSED
- **UT-8.1.2**: Conversation participants property PASSED

UT-9: Message Model Tests - All Pass

- **UT-9.1.1**: Message creation PASSED
- **UT-9.1.2**: Is recent property (within 24 hours) PASSED

UT-10: Thread Model Tests - All Pass

- **UT-10.1.1**: Thread creation PASSED
- **UT-10.1.2**: Is active property (activity within 7 days) PASSED

UT-11: Review Model Tests - All Pass

- **UT-11.1.1**: Review creation PASSED
- **UT-11.1.2**: Rating display property PASSED
- **UT-11.1.3**: Is positive review property PASSED

Unit Test Summary: 28 unit tests, all passing

9.5 System Test Results (ST-X.Y.Z)

ST-1: User Registration API - All Pass

- **ST-1.1.1**: Successful user registration PASSED
- **ST-1.1.2**: Registration with password mismatch PASSED
- **ST-1.1.3**: TimeAccount creation on registration with 3 TBH PASSED

ST-2: Service API - All Pass

- **ST-2.1.1**: Create service endpoint PASSED
- **ST-2.1.2**: List services endpoint PASSED
- **ST-2.1.3**: Filter services by type PASSED
- **ST-2.1.4**: Filter services by status PASSED

ST-3: ServiceRequest API - All Pass

- **ST-3.1.1**: Creating a service request PASSED
- **ST-3.1.2**: User cannot request their own service PASSED
- **ST-3.1.3**: Accepting a service request PASSED

ST-4: TimeAccount API - All Pass

- **ST-4.1.1**: Retrieving time account PASSED
- **ST-4.1.2**: Time account balance display PASSED

ST-5: Conversation API - All Pass

- **ST-5.1.1**: Listing conversations PASSED
- **ST-5.1.2**: Creating a message in conversation PASSED

ST-6: Health Check API - All Pass

- **ST-6.1.1**: Health check endpoint PASSED

ST-7: Tag API - All Pass

- **ST-7.1.1**: List tags endpoint PASSED
- **ST-7.1.2**: Create tag endpoint PASSED
- **ST-7.1.3**: Popular tags endpoint PASSED

ST-8: Thread API - All Pass

- **ST-8.1.1**: Create thread endpoint PASSED
- **ST-8.1.2**: List threads endpoint PASSED
- **ST-8.1.3**: Create post endpoint PASSED

ST-9: Review API - All Pass

- **ST-9.1.1**: Create review endpoint PASSED
- **ST-9.1.2**: List reviews endpoint PASSED

ST-10: Report API - All Pass

- **ST-10.1.1**: Create report endpoint PASSED

System Test Summary: 20 system tests, all passing

9.6 Use Case Test Results (UC-X.Y)

UC-1: Complete Service Request Workflow - Pass

- **UC-1.1**: Complete workflow: request → accept → start → complete → time transfer PASSED
- **Verification**:
  - Service request created successfully
  - Request accepted by owner
  - Both parties approved start
  - Both parties completed service
  - Time transfer verified: Owner received 2 TBH, Requester paid 2 TBH

UC-2: User Registration Workflow - Pass

- **UC-2.1**: Complete user registration workflow PASSED
- **Verification**:
  - User created successfully
  - Profile created automatically
  - Time account created with 3 TBH
  - Location coordinates saved

UC-3: Service Creation Workflow - Pass

- **UC-3.1**: Create service with tags and verify it appears in search PASSED
- **Verification**:
  - Service created successfully
  - Tags associated correctly

    - Service appears in tag-based search

UC-4: Time Banking Workflow - Pass

- **UC-4.1**: Complete time transaction workflow PASSED
- **Verification**:
    - Initial balance verified
    - Transaction created
    - Balance updated correctly

UC-5: Moderation Workflow - Pass

- **UC-5.1**: Report content and admin moderation PASSED
- **Verification**:
    - Report created successfully
    - Admin can view reports
    - Admin can resolve reports

Use Case Test Summary: 5 use case tests, all passing

9.7 Test Coverage Analysis

Test coverage is measured using Coverage.py:

```
 coverage run --source='the_hive' manage.py test the_hive.tests
coverage report -m
coverage html
```

Code Coverage Summary (as per Testing Documentation):

Component	Coverage	Status
Models	85%+	Good
Views/API	80%+	Good
Serializers	75%+	Acceptable
Business Logic	90%+	Excellent
Overall	**82%+**	**Target Met**

Coverage Goals: Minimum 80% overall coverage **ACHIEVED**

9.8 Requirements Coverage

Requirements Coverage (as per Test Traceability Matrix):

- **Total Requirements Tested**: 35+
- **Requirements with Unit Tests**: 35+
- **Requirements with System Tests**: 30+
- **Requirements Coverage**: 100%

Test Traceability: All functional requirements (FR-1 through FR-19) have corresponding test cases. See Testing Documentation wiki for complete traceability matrix.

9.9 Test Statistics

- **Total Test Cases**: 53
- **Unit Tests**: 28
- **System Tests**: 20
- **Use Case Tests**: 5
- **Pass Rate**: 100%
- **Test Execution Time**: ~10 seconds
- **Code Coverage**: 82%+ (exceeds 80% target)

9.10 Test Results by Component

User Account Management (FR-1)

- User registration: All tests passed (ST-1.1.1, ST-1.1.2, ST-1.1.3)
- Profile creation: All tests passed (UT-2.1.1, UT-2.1.2)
- TimeAccount auto-creation: All tests passed (ST-1.1.3, UT-6.1.1)
- Authentication: All tests passed (UT-1.1.1)

Service Management (FR-2)

- Service creation: All tests passed (ST-2.1.1, UT-4.1.1)
- Service filtering: All tests passed (ST-2.1.3, ST-2.1.4)
- Service updates: All tests passed (ST-2.1.1)
- Service status management: All tests passed (UT-4.1.2)

Service Exchange (FR-4)

- Service request creation: All tests passed (ST-3.1.1, UT-5.1.1)
- Request acceptance/rejection: All tests passed (ST-3.1.3)
- Cannot request own service: All tests passed (ST-3.1.2)
- Conversation creation: All tests passed (ST-5.1.2, UT-8.1.1)

Time Banking System (FR-5)

- TimeAccount operations: All tests passed (ST-4.1.1, UT-6.1.1)
- Time transactions: All tests passed (UT-7.1.1, UT-7.1.2)
- Balance calculations: All tests passed (UT-6.1.2, UT-6.1.3)
- Time transfer workflow: All tests passed (UC-1.1, UC-4.1)

Community Forums (FR-6)

- Thread creation: All tests passed (ST-8.1.1, UT-10.1.1)
- Post creation: All tests passed (ST-8.1.3)
- Forum operations: All tests passed (ST-8.1.2)

Moderation System (FR-7)

- Report creation: All tests passed (ST-10.1.1)
- Admin operations: All tests passed (ST-10.1.1)

9.11 Defect Summary

Overall Statistics:

- **Total Defects Found**: 0
- **Critical Defects**: 0
- **High Priority Defects**: 0
- **Medium Priority Defects**: 0
- **Low Priority Defects**: 0

Defect Status: No defects identified during comprehensive testing phase.

9.12 Test Deliverables

All test deliverables are documented in the Testing Documentation wiki:

1. **Test Plan** - Comprehensive test planning document
2. **Testing Strategy** - Detailed testing approach and methodologies
3. **Test Cases** - 53 detailed test case specifications
4. **Test Scripts** - Automated test code (`the_hive/tests.py`, `run_tests.sh`)
5. **Test Data** - Test fixtures and factory methods
6. **Test Traceability Matrix** - Requirements to tests mapping
7. **Test Results** - Test execution results (100% pass rate)
8. **Test Summary Report** - Executive summary of testing activities
9. **Defect Report** - Defect tracking (no defects found)
10. **Coverage Reports** - Code coverage analysis (82%+)

9.13 Manual Testing

In addition to automated tests, the following features were manually tested:

- **Map Interaction**: Zoom, pan, marker clicks
- **Service Creation**: All fields and validations
- **Service Discovery**: Filtering and search
- **Messaging**: Real-time message updates
- **Forum**: Thread creation and replies
- **Reviews**: Review creation and display
- **Admin Panel**: Statistics and moderation actions

9.14 Test Quality Assessment

Quality Status: **PASS**

- All critical paths have test coverage
- Minimum 80% code coverage achieved (82%+)
- All tests pass consistently (100% pass rate)
- No critical or high-severity defects
- Test documentation is complete

- Traceability matrix is complete

Recommendation: **APPROVED FOR DEPLOYMENT**

---

10. Third-Party Software and Licenses

10.1 Backend Dependencies

Package	Version	License	Purpose
Django	4.2.25	BSD-3-Clause	Web framework
djangorestframework	3.16.1	BSD-3-Clause	REST API framework
djangorestframework-simplejwt	5.5.1	MIT	JWT authentication
django-cors-headers	4.3.1	MIT	CORS handling
psycopg2-binary	2.9.11	LGPL-3.0	PostgreSQL adapter
PyJWT	2.10.1	MIT	JWT library
python-dotenv	1.2.1	BSD-3-Clause	Environment variables
requests	2.32.3	Apache-2.0	HTTP library
Pillow	10.4.0	HPND	Image processing
gunicorn	21.2.0	MIT	WSGI server
coverage	7.5.3	Apache-2.0	Test coverage

10.2 Frontend Dependencies

Package	Version	License	Purpose
react	19.1.1	MIT	UI framework
react-dom	19.1.1	MIT	React DOM renderer
react-router-dom	7.9.5	MIT	Routing
@tanstack/react-query	5.90.6	MIT	Server state management
axios	1.13.1	MIT	HTTP client
leaflet	1.9.4	BSD-2-Clause	Map library
react-leaflet	5.0.0	BSD-2-Clause	React Leaflet bindings
tailwindcss	4.1.16	MIT	CSS framework
typescript	5.9.3	Apache-2.0	TypeScript compiler
vite	7.1.14	MIT	Build tool

10.3 Infrastructure

Software	Version	License	Purpose
Docker	Latest	Apache-2.0	Containerization
Docker Compose	Latest	Apache-2.0	Container orchestration

Software	Version	License	Purpose
PostgreSQL	15	PostgreSQL License	Database
Nginx	Latest	BSD-2-Clause	Web server (frontend)
Gunicorn	21.2.0	MIT	WSGI server (backend)

10.4 External Services

Service	License	Purpose
OpenStreetMap Nominatim	ODbL	Geocoding services
AWS EC2	Commercial	Cloud hosting

10.5 License Compliance

All third-party software used in this project adheres to their respective open-source licenses. The project itself is developed for educational purposes as part of the SWE573 course at Boğaziçi University.

Note: This project uses open-source software with permissive licenses (MIT, BSD, Apache-2.0) that allow commercial use, modification, and distribution. The PostgreSQL database uses the PostgreSQL License, which is also permissive and allows commercial use.

11. References

11.1 Technologies and Frameworks

- Django Documentation: https://docs.djangoproject.com/
- Django REST Framework: https://www.django-rest-framework.org/
- React Documentation: https://react.dev/
- TypeScript Documentation: https://www.typescriptlang.org/
- Leaflet Maps: https://leafletjs.com/
- PostgreSQL Documentation: https://www.postgresql.org/docs/
- Docker Documentation: https://docs.docker.com/
- Tailwind CSS: https://tailwindcss.com/
- TanStack Query: https://tanstack.com/query

11.2 Course Materials

- SWE573 Software Development Practice Course Materials
- Boğaziçi University Software Engineering Department

11.3 External Services

- OpenStreetMap Nominatim: https://nominatim.openstreetmap.org/
- AWS EC2: https://aws.amazon.com/ec2/
- Wikidata: https://www.wikidata.org/

11.4 Project Resources

- GitHub Repository: https://github.com/metincd/SWE573-FALL2025
- Project Wiki: https://github.com/metincd/SWE573-FALL2025/wiki
- Software Requirements Specification: See `SOFTWARE_REQUIREMENTS_SPECIFICATION.md`
- Testing Documentation: https://github.com/metincd/SWE573-FALL2025/wiki/Testing-Documentation
- Release Notes: See `RELEASE_NOTES.md`

11.5 Standards and Specifications

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications
- REST API Design Principles
- PEP 8 Python Style Guide
- React Best Practices

Appendix A: Project Statistics

- **Total Lines of Code**: ~15,000+ lines
- **Backend Models**: 20+ models

- **API Endpoints**: 50+ endpoints
- **Frontend Components**: 30+ components
- **Test Cases**: 53 test cases (28 Unit, 20 System, 5 Use Case)
- **Test Pass Rate**: 100%
- **Code Coverage**: 82%+ (exceeds 80% target)
- **Database Migrations**: 16 migrations
- **Development Time**: ~3 months
- **Git Commits**: 100+ commits
- **Git Tags**: v0.9, v1.0.0

Appendix B: API Endpoints Summary

Authentication

- `POST /api/register/` - User registration
- `POST /api/token/` - JWT token authentication
- `GET /api/me/` - Current user profile

Services

- `GET /api/services/` - List services
- `POST /api/services/` - Create service
- `GET /api/services/{id}/` - Service details
- `PUT /api/services/{id}/` - Update service
- `DELETE /api/services/{id}/` - Delete service

Service Requests

- `GET /api/service-requests/` - List service requests
- `POST /api/service-requests/` - Create service request
- `POST /api/service-requests/{id}/set_status/` - Set request status
- `POST /api/service-requests/{id}/approve_start/` - Approve service start
- `POST /api/service-requests/{id}/complete/` - Complete service

Time Banking

- `GET /api/time-accounts/` - Get time account
- `GET /api/time-transactions/` - List transactions

Communication

- `GET /api/conversations/` - List conversations
- `POST /api/conversations/` - Create conversation
- `GET /api/messages/` - List messages
- `POST /api/messages/` - Send message

Forum

- `GET /api/threads/` - List threads
- `POST /api/threads/` - Create thread
- `GET /api/posts/` - List posts
- `POST /api/posts/` - Create post

Reviews

- `GET /api/reviews/` - List reviews
- `POST /api/reviews/` - Create review
- `GET /api/user-ratings/` - List user ratings

Moderation

- `GET /api/reports/` - List reports
- `POST /api/reports/` - Create report
- `GET /api/admin/stats/` - Admin statistics

Health Check

- `GET /api/health/` - System health check

Content Usage Rights

All content used in this project, including images, icons, and external resources, either:

- Is created by the project author
- Has appropriate use rights (Creative Commons or other permissions)
- Is used in accordance with their respective licenses

The project uses:

- OpenStreetMap tiles (ODbL license) for map display
- Leaflet library (BSD-2-Clause license) for map functionality
- All other resources are either created by the author or have permissive licenses

---

Prepared by: Metin Cem Doğan
Date: December 2025