

Project HiMate: Master Design Document

Version: 1.0.0 **Mission:** "Scientific Roommate Allocation based on Chronobiology and Psychometrics."

1. Global Architecture & Standards

Everyone must strictly follow these rules to ensure the 5 slices fit together.

1.1 Tech Stack

- **Frontend:** React 18, Axios, Tailwind CSS (Styling), React Router Dom.
- **Backend:** Python 3.10+, Django 4.2, Django REST Framework (DRF), SimpleJWT.
- **Database:** MySQL (Use `mysqlclient` connector).
- **Version Control:** Git (Feature Branch Workflow).

1.2 The "Shared Foundation" (Dev 1 Responsibility)

Before splitting up, **Dev 1** establishes these standards.

A. Visual Theme (Tailwind config):

- **Primary Color (Trust):** Blue-600 (#2563EB) - Buttons, Links.
- **Secondary Color (Calm/Sleep):** Teal-500 (#14B8A6) - Success states, Profile tags.
- **Background:** Slate-50 (#F8FAFC) - App Background.
- **Font:** 'Inter', sans-serif (Google Fonts).

B. Directory Structure (Strict Adherence):

Plaintext

```
/himate-project
    └── /backend (Django)
        |   └── manage.py
        |   └── /core (Settings, Shared Utilities)
        |       └── /users (Dev 1 - Auth & Profile)
        |           └── /housing (Dev 2 - Hostels & Rooms)
        |               └── /allocation (Dev 3 - The Algo)
        |                   └── /requests (Dev 4 - Swaps & Leave)
        |                       └── /operations (Dev 5 - Dashboard & Issues)
```

```
|  
└─ /frontend (React + Vite)  
    ├─ /src  
    |    ├─ /assets  
    |    ├─ /components (Shared: Button, Input, Modal, Card)  
    |    ├─ /context (AuthContext)  
    |    ├─ /features  
    |    |    ├─ /auth (Dev 1)  
    |    |    ├─ /housing (Dev 2)  
    |    |    ├─ /allocation (Dev 3)  
    |    |    ├─ /requests (Dev 4)  
    |    |    └─ /dashboard (Dev 5)  
    └─ App.jsx
```

1.3 Database Master Schema (MySQL)

All developers must agree on these field names.

1. Users (users_customuser) - Dev 1

- `id` (PK), `username`, `email`, `password`.
- `role`: Enum ['STUDENT', 'WARDEN'].
- `gender`: Enum ['MALE', 'FEMALE'].
- `is_profile_complete`: Boolean (Default False).

2. Profiles (users_studentprofile) - Dev 1

- `user_id` (FK -> Users).
- `wake_up_time`: Time (e.g., 08:00:00).
- `requires_darkness`: Boolean.
- `cleanliness`: Int (1-5).
- `guest_tolerance`: Int (1-5).
- `dominance`: Int (1-5).

3. Hostels (housing_hostel) - Dev 2

- `id` (PK), `name` (e.g., "Block A").
- `gender_type`: Enum ['MALE', 'FEMALE'].
- `caretaker_name`: String.

4. Rooms (`housing_room`) - Dev 2

- `id` (PK), `hostel_id` (FK).
- `room_number`: String (e.g., "101").
- `capacity`: Int (Default 4).
- `current_occupancy`: Int (Default 0).
- `status`: Enum ['AVAILABLE', 'FULL', 'MAINTENANCE'].

5. Allocations (`allocation_allocation`) - Dev 3

- `id` (PK).
- `student_id` (FK -> User).
- `room_id` (FK -> Room).
- `semester`: String (e.g., "Fall 2025").

6. Requests (`requests_swap`, `requests_outpass`) - Dev 4

- **Swap:** `student_a` (FK), `student_b` (FK), `status` ['PENDING', 'APPROVED', 'REJECTED'].
- **Outpass:** `student_id` (FK), `leave_date`, `return_date`, `status`.

7. Operations (`operations_ticket`) - Dev 5

- `student_id` (FK), `category` (WiFi/Plumbing), `description`, `status` ['OPEN', 'RESOLVED'].

2. Developer Specifications (The "To-Do" Lists)

Developer 1: The Identity Lead (Auth & Profile)

Focus: Security & Scientific Data Collection.

Backend Tasks:

1. Setup Django Project & `core` app.
2. Install `djangorestframework`, `djoser` (for JWT).
 1. Create `allocation` app.
 2. Create `Allocation` model.
3. **The Engine (`services.py`):** Implement the `calculate_compatibility(student_a, student_b)` function (as detailed in previous plan).

- *Constraint:* Must import `StudentProfile` from `users` app.
- 1. **The Loop:** Create logic to fetch *all* unassigned profiles, sort them, and commit to Allocation table.
- 2. **API Endpoint:** `POST /api/allocation/run/` (Admin only trigger).
- 3. **API Endpoint:** `GET /api/allocation/my-room/` (Student view).
- 3. **API Endpoint:** `POST /api/auth/register/` (Standard Djoser).

Frontend Tasks:

1. Setup React + Tailwind.
2. Create `Login.jsx` & `Register.jsx`.
1. Create `features/allocation/AllocationControl.jsx` (Warden View).
 - Button: "Run Smart Allocation".
 - Display: Loading Spinner -> Success Message "Allocated 120 Students".
1. Create `MyRoom.jsx` (Student View).
 - Display: "Room 101" and "Roommate Names".
- 3.

Developer 2: The Property Manager (Inventory)

Focus: CRUD Operations & Data Integrity.

Backend Tasks:

1. Create housing app.
2. Models: Hostel and Room.
3. **Seeding Script:** Create `management/commands/seed_hostels.py`.
 - Action: Auto-generate 2 Hostels (Boys/Girls) with 50 rooms each.
1. **API Endpoint:** `GET /api/housing/hostels/` (List all).
2. **API Endpoint:** `GET /api/housing/hostels/{id}/rooms/` (List rooms for a building).

Frontend Tasks:

1. Create `features/housing/RoomGrid.jsx`.
2. **UI:** Display rooms as small cards.
 - Green Card = Available (0/4).
 - Yellow Card = Filling (2/4).
 - Red Card = Full (4/4).
 - Grey Card = Maintenance.

Developer 3: The Algorithm Engineer (The Core)

Focus: Python Logic & Optimization.

Backend Tasks:

4. Create `User` model (Custom User Model) extending `AbstractUser`.
5. Create `StudentProfile` model with the 5 specific scientific fields.
6. **API Endpoint:** `PATCH /api/profile/update/` (Saves survey answers).
 - o *Logic:* When profile is saved, set `user.is_profile_complete = True`.

Frontend Tasks:

4. **Core Feature:** `SurveyWizard.jsx`.
 - o Slide 1: "When do you wake up?" (Time Picker).
 - o Slide 2: "Can you sleep with lights on?" (Yes/No Toggle).
 - o Slide 3: "Cleanliness Level" (Range Slider 1-5).
 - o Slide 4: "Guest Tolerance" (Range Slider 1-5).
 - o Slide 5: "Social Battery/Dominance" (Range Slider 1-5).

Developer 4: The Workflow Manager (Requests)

Focus: State Management & Transactions.

Backend Tasks:

1. Create `requests` app.
2. Models: `SwapRequest` and `OutPass`.
3. **Swap Logic:** Use `transaction.atomic()`.
 - o *If Approved:* Update Allocation table for Student A AND Student B.
 - o *If Failed:* Rollback both.
1. **API Endpoint:** `POST /api/requests/swap/` (Create request).
2. **API Endpoint:** `POST /api/requests/outpass/` (Create pass).

Frontend Tasks:

1. Create `features/requests/SwapForm.jsx`.
 - o Dropdown: Select Student to swap with (Search by ID).
1. Create `features/requests/OutpassQR.jsx`.
 - o Show a static QR code image if status == 'APPROVED'.

Developer 5: The Operations Lead (Dashboard)

Focus: Data Viz & Maintenance.

Backend Tasks:

1. Create operations app.
 2. Model: MaintenanceTicket.
 3. **Dashboard Stats (`views.py`):**
 - o Count User where `role='STUDENT'`.
 - o Count Allocation rows.
 - o Count Room where `status='AVAILABLE'`.
1. **API Endpoint:** GET `/api/dashboard/stats/`.
 2. **API Endpoint:** POST `/api/operations/ticket/`.

Frontend Tasks:

1. Create `features/dashboard/WardenDashboard.jsx`.
 - o **Charts:** Use recharts or chart.js.
 - o Pie Chart: Occupied vs Empty Rooms.
 - o Bar Chart: Maintenance Issues by Category (Wifi, Water, Electric).
1. Create `TicketList.jsx`: Table of active complaints with "Mark Resolved" button.

3. Integration & Execution Order

To avoid "Merge Hell", follow this timeline:

Day 1:

- **Dev 1:** Pushes `User` model and `Auth` system.
- **All Devs:** Pull the repo so everyone has the `User` model.

Day 2:

- **Dev 2:** Builds Hostels/Rooms and pushes Seeding Script.
- **Dev 3:** Writes "Mock Students" script to test algorithm locally.
- **Dev 1:** Finishes Profile API.

Day 3:

- **Dev 3:** Connects Algorithm to real Database (Dev 1's Profiles + Dev 2's Rooms).
- **Dev 4 & 5:** Build features on top of existing data.

Day 4:

- **Frontend Integration:** Connect React pages to Django APIs.
- **Testing:** Run the "Happy Path" (Register -> Allocator -> Swap).

4. Final Deliverable Checklist

1. **Repo:** A single GitHub repo with `/backend` and `/frontend`.

2. **Report:** Includes "The Science of Compatibility" section.
3. **Demo:**
 - o Show Admin Dashboard (Empty).
 - o Register 4 Students (2 Early Birds, 2 Night Owls).
 - o Click "Run Allocation".
 - o Show Database: Early birds paired together, Owls paired together.
 - o File a "Broken Fan" ticket.