API Design for UniHaven Accommodation Management System

Zhu Yecheng 3036061373

API Design for UniHaven Accommodation Management System

Assumptions:

- **Authentication:** We'll assume a standard token-based authentication (JWT) for all requests requiring a user context.
- **Database:** I'm assuming a relational database (e.g., PostgreSQL, MySQL) for storing the data.
- **UniHaven:** The UniHaven database is the central data store.
- **CEDARS Integration:** User authentication is handled through CEDARS authentication API.

1. Data Models (JSON Schemas):

These define the structure of the data being passed back and forth.

- **Accommodation:**

```
{
  "id": "string (UUID)",
  "type": "string (e.g., 'Single_Room', 'Shared_Room', 'Apartment')",
  "period_of_availability": "string (e.g., 'Semester', 'Yearly')",
  "number_of_rooms_available": "integer",
  "Shared_Bathroom": "boolean",
  "price": "number (float)",
  "location": "string (e.g., address, neighborhood)",
  "latitude": "number (float)",
  "longitude": "number (float)",
  "amenities": "array of strings (e.g., 'WiFi', 'Air Conditioning', 'Laundry')",
  "photos": "array of strings (URLs to images)",
  "landlord_id": "string (UUID)",
  "availability_calendar": "array of dates",
  "description": "string",
  "created_at": "timestamp",
  "updated_at": "timestamp"
}
```

- **User (HKU Student):** (Simplified, assumes CEDARS provides more details)

```
{
  "id": "string (UUID)",
  "student_id": "string", // HKU Student/Staff ID
  "name": "string",
  "email": "string",
  "phone": "string"
}
```

- **Landlord:**

```
{
  "id": "string (UUID)",
  "name": "string",
  "email": "string",
  "phone": "string"
}
```

- **Application:**

```
{
  "id": "string (UUID)",
  "accommodation_id": "string (UUID)",
  "user_id": "string (UUID)",
  "application_date": "timestamp",
  "status": "string (e.g., 'pending', 'approved', 'rejected', 'canceled')",
  "rental_contract_start_date": "date",
  "rental_contract_end_date": "date",
  "notes": "string"
}
```

- **Rating/Review:**

```
{
  "id": "string (UUID)",
  "accommodation_id": "string (UUID)",
  "user_id": "string (UUID)",
  "overall_rating": "integer (0-5)",
  "value_for_money": "integer (0-5)",
  "location_convenience": "integer (0-5)",
  "property_condition": "integer (0-5)",
  "landlord_communication": "integer (0-5)",
  "review_text": "string",
  "photos": "array of strings (URLs)",
  "created_at": "timestamp",
  "updated_at": "timestamp"
}
```

2. API Endpoints:

Here's a breakdown of the endpoints, grouped by functionality, relating to the use cases.

A. Accommodation Management (UC01, UC02, UC05):

- `GET /accommodations`: Retrieve a list of accommodations. Supports filtering and pagination.
  - Query Parameters: `type`, `location`, `price_min`, `price_max`, `amenities`, `page`, `page_size`
  - Returns: Array of `Accommodation` objects.
  - ie. `GET /accommodations?type=Single_Room&location=Swire_Hall&price_min=1000&page=1&page_size=10`
- `GET /accommodations/{id}`: Retrieve a specific accommodation by ID.
  - Returns: `Accommodation` object.
  - ie. `GET /accommodations/a1b2c3d4-e5f6-7890-1234-567890abcdef`
- `POST /accommodations`: Create a new accommodation (Admin/Landlord). Requires authentication.
  - Request Body: `Accommodation` object (without the `id`).
  - Returns: `Accommodation` object (with the created `id`).
  - ie. `POST /accommodations`

  ```
  {
    "type": "Single_Room",
    "period_of_availability": "Yearly",
    "number_of_rooms_available": 5,
    "Shared_Bathroom": true,
    "price": 1500.00,
    "location": "Swire Hall",
    "latitude": 22.3000000,
    "longitude": 114.2000000,
    "amenities": ["WiFi", "Air Conditioning"],
    "photos": ["url1", "url2"],
    "landlord_id": "f1g2h3i4-j5k6-7890-1234-567890abcdef"],
    "availability_calendar": ["2024-01-01", "2024-12-31"],
    "description": "A single room in Swire Hall."
  }
  ```

  A successful response would return the created accommodation object with its `id`.
- `POST /accommodations/{id}/update`: Update an existing accommodation (Admin/Landlord). Requires authentication.
  - Request Body: `Accommodation` object (with the `id`).
  - Returns: `Accommodation` object.

- `POST /accommodations/{id}/delete`: Delete an accommodation (Admin/Landlord). Requires authentication.

B. Application Management (UC03, UC04):

- `GET /applications`: Retrieve a list of applications. Supports filtering (e.g., by user, accommodation, status). Requires authentication.
  - Query Parameters: `user_id`, `accommodation_id`, `status`, `page`, `page_size`
  - Returns: Array of `Application` objects.
  - ie. `GET /applications?user_id={UUID}&status=pending&page=1&page_size=10`
- `GET /applications/{id}`: Retrieve a specific application by ID. Requires authentication.
  - Returns: `Application` object.
  - ie. `GET /applications/a1b2c3d4-e5f6-7890-1234-567890abcdef`
- `POST /applications`: Create a new application. Requires authentication.
  - Request Body: `Application` object (without the `id`).
  - Returns: `Application` object (with the created `id`).
  - ie. `POST /applications`

```
{
  "accommodation_id": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "user_id": "f1g2h3i4-j5k6-7890-1234-567890abcdef",
  "application_date": "2024-01-26T10:00:00Z",
  "rental_contract_start_date": "2024-09-01",
  "rental_contract_end_date": "2025-06-30"
}
```

  A successful response would return the created application object with its `id`.
- `POST /applications/{id}/update`: Update an existing application (e.g., change status). Requires authentication.
  - Request Body: `Application` object (with the `id`).
  - Returns: `Application` object.

C. User & Landlord Management:

- `GET /users/{id}`: Retrieve a user by ID. Requires authentication (may be public or restricted).
  - Returns: `User` object.
  - ie. `GET /users/f1g2h3i4-j5k6-7890-1234-567890abcdef`
- `GET /landlords/{id}`: Retrieve a landlord by ID.
  - Returns: `Landlord` object.
  - ie. `GET /landlords/f1g2h3i4-j5k6-7890-1234-567890abcdef`

D. Rating & Review (UC07):

- `GET /accommodations/{id}/ratings`: Retrieve all ratings for a specific accommodation. Supports pagination.
  - Query Parameters: `page`, `page_size`
  - Returns: Array of `Rating/Review` objects.
  - ie. `GET /accommodations/a1b2c3d4-e5f6-7890-1234-567890abcdef/ratings?page=1&page_size=10`
- `POST /ratings`: Create a new rating/review. Requires authentication.
  - Request Body: `Rating/Review` object (without the `id`).
  - Returns: `Rating/Review` object (with the created `id`).
  - ie. `POST /ratings`

```
{
  "accommodation_id": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "user_id": "f1g2h3i4-j5k6-7890-1234-567890abcdef",
  "overall_rating": 4,
  "value_for_money": 4,
  "location_convenience": 5,
  "property_condition": 4,
  "landlord_communication": 5,
  "review_text": "Great place to stay!",
  "photos": ["url1", "url2"]
}
```

  A successful response would return the created rating/review object with its `id`.
- `GET /users/{user_id}/ratings`: Retrieve all ratings given by a specific user.
  - Returns: Array of `Rating/Review` objects.
  - ie. `GET /users/f1g2h3i4-j5k6-7890-1234-567890abcdef/ratings`

E. Reservation Cancellation (UC05, UC06):

- `POST /applications/{id}/cancel`: Cancel a reservation (Application). Requires authentication.
  - Returns: `Application` object with updated `status`.
  - ie. `POST /applications/a1b2c3d4-e5f6-7890-1234-567890abcdef/cancel`

3. Key Considerations:

- **Pagination:** Use consistent pagination for list endpoints (e.g., `page` and `page_size` query parameters).
- **Filtering:** Allow filtering on key fields for most list endpoints.
- **Error Handling:** Return meaningful error messages (e.g., HTTP status codes, JSON error objects).
- **Validation:** Validate incoming data to ensure data integrity.
- **Security:** Use HTTPS for all communication. Protect against common web vulnerabilities (e.g., XSS, CSRF).
- **Caching:** Implement caching (e.g., Redis) for frequently accessed data.
- **API Versioning:** Use API versioning (e.g., `/v1/accommodations`) to allow for future changes without breaking existing clients.
- **Asynchronous Tasks:** Use a message queue (e.g., RabbitMQ, Kafka) for tasks that don't need to be done immediately (e.g., sending notifications).