Thank you for applying for the Full Stack Intern position at ServiceHive. We were impressed with your application and would like to invite you to the next stage of our process: a technical challenge.
We've designed this challenge to be a practical and engaging way for you to showcase your skills across the entire stack. It's designed to be more in-depth than a simple to-do app and will touch on data modeling, complex API logic, and frontend state management.
Your challenge is to build **"SlotSwapper."**

The Concept

SlotSwapper is a peer-to-peer time-slot scheduling application.
The core idea: Users have calendars with busy slots. They can mark a busy slot as "swappable." Other users can then see these swappable slots and request to swap one of their *own* swappable slots for it.
Example:

- **User A** has a "Team Meeting" on Tuesday from 10:00-11:00 AM. They mark it as "swappable."
- **User B** has a "Focus Block" on Wednesday from 2:00-3:00 PM. They also mark it as "swappable."
- User A sees User B's slot and requests a swap, offering their Tuesday slot.
- User B receives a notification and can **Accept** or **Reject** the swap.
- If accepted, the application updates the calendars for both users—User A now has the Wednesday slot, and User B has the Tuesday slot.

Core Technical Requirements

Here are the features we'd like you to build.

1. User Authentication

- Users must be able to **Sign Up** (e.g., Name, Email, Password) and **Log In**.
- Implement **JWT (JSON Web Tokens)** for managing authenticated sessions. The token should be sent (e.g., as a Bearer token) with all protected API requests.

2. Backend: Calendar & Data Model

- Design a database schema to support the application. You'll likely need tables/collections for Users, Events (or Slots), and SwapRequests.

- An **Event** record should have at least:
  - title
  - startTime (timestamp)
  - endTime (timestamp)
  - status (e.g., an enum: BUSY, SWAPPABLE, SWAP_PENDING)
  - A link to its owner (e.g., userId).
- Create **CRUD API endpoints** for a user to manage their *own* events.

3. Backend: The Swap Logic (The Core Challenge)

This is the most critical part of the backend.

- **GET /api/swappable-slots**
  - This endpoint must return all slots from **other users** that are marked as SWAPPABLE.
  - It should *not* include the logged-in user's own slots.
- **POST /api/swap-request**
  - This endpoint should accept the ID of the user's slot (mySlotId) and the ID of the desired slot (theirSlotId).
  - **Server-side logic:** You must verify that both slots exist and are currently SWAPPABLE.
  - Create a new SwapRequest record (e.g., with a PENDING status), linking the two slots and users.
  - Update the status of **both** original slots to SWAP_PENDING to prevent them from being offered in other swaps.
- **POST /api/swap-response**
  - This endpoint allows a user to respond to an *incoming* swap request (e.g., /api/swap-response/:requestId).
  - The request body should indicate acceptance (true/false).
  - **If Rejected:** The SwapRequest status is set to REJECTED, and the two slots involved must be set back to SWAPPABLE.
  - **If Accepted:** This is the key transaction. The SwapRequest is marked ACCEPTED. The owner (or userId) of the two slots must be *exchanged*. Both slots' status should be set back to BUSY.

4. Frontend: UI/UX

- **Authentication:** Create sign-up and log-in pages/forms.
- **Calendar/Dashboard View:**
  - A page where a user can see their own events (a simple list view or a calendar grid is fine).
  - They must be able to **create** new events.

    - They must be able to **update** an existing event's status (e.g., click a "Make Swappable" button on a BUSY event).
- **Marketplace View:**
    - A page that fetches and displays the list of available slots from GET /api/swappable-slots.
    - Add a "Request Swap" button to each slot. Clicking this should (e.g., in a modal) show the user a list of *their own* SWAPPABLE slots to choose from as their offer.
- **Notifications/Requests View:**
    - A page that shows two lists:
        1. **Incoming Requests:** Swaps other users have offered them. Each must have **"Accept"** and **"Reject"** buttons that call the POST /api/swap-response endpoint.
        2. **Outgoing Requests:** Swaps they have offered to others (showing "Pending...").
- **State Management:**
    - The application must update its state dynamically. For example, after a user accepts a swap, their calendar view should reflect this change without requiring a manual page refresh.
    - Authenticated routes should be protected.

Technology Stack

- **Frontend:** You may use any modern framework (React, Vue, Svelte, or Angular). TypeScript is a plus but not required.
- **Backend:** You may use any framework (e.g., Node.js/Express, Python/Django/FastAPI, Go, Ruby on Rails, etc.).
- **Database:** You may use any database (e.g., PostgreSQL, MySQL, MongoDB, or even SQLite for simplicity).

Bonus Features (To Stand Out)

These are **not required**, but if you have extra time and want to showcase more of your skills, consider adding one or two:

- **Unit/Integration Tests:** Write tests for your backend API, especially for the complex swap-response logic.
- **Real-time Notifications:** Use WebSockets to notify a user instantly when they receive a swap request or when their request is accepted.
- **Deployment:** Deploy the live application (e.g., frontend on Vercel/Netlify, backend on Render/Heroku).
- **Containerization:** Include a Dockerfile and docker-compose.yml for easy local setup.

deliverables

Please submit the following:

1. A **GitHub repository link** (please ensure it's public or that you've given us access).
2. A comprehensive **README.md** file in your repository that **must** include:
   - A brief overview of your project and any design choices you made.
   - **Clear, step-by-step instructions** on how to set up and run the project locally (this is crucial for us to review your code).
   - A list of your API endpoints and how to use them (e.g., a simple table or a link to a Postman collection).
   - Any assumptions you made or challenges you faced.
3. **(Optional but highly recommended)** A link to the live, deployed application.