tables

Below is a proposed starting point for your table schema along with some thoughts on the feasibility of building DocConnect within three days.

Proposed Table Schema

1. Users Table

This table stores common information for all users (both professors and students).

- **id** (Primary Key, auto-increment)
- **full_name** (VARCHAR)
- **email** (VARCHAR, unique)
- **password_hash** (VARCHAR)
- **role** (ENUM: 'Professor', 'Student')
- **created_at** (TIMESTAMP)
- **updated_at** (TIMESTAMP)

2. Professors Table

Since professors have additional fields, this table will store professor-specific information. It links to the Users table via a foreign key.

- **id** (Primary Key, auto-increment)
- **user_id** (Foreign Key referencing Users.id)
- **department** (VARCHAR)
- **office_location** (VARCHAR)
- **working_hours** (VARCHAR or JSON if you need structured hours)
- **email_verified** (BOOLEAN) – for extra verification
- **created_at** (TIMESTAMP)

- **updated_at** (TIMESTAMP)

3. Status History Table

This table will log every change in a professor’s status, which can be used for accountability and real-time updates.

- **id** (Primary Key, auto-increment)
- **professor_id** (Foreign Key referencing Professors.id)
- **status** (ENUM: 'Available', 'Busy', 'Away', 'In Meeting')
- **timestamp** (TIMESTAMP)

4. Announcements Table

This table handles professor announcements that are visible to students.

- **id** (Primary Key, auto-increment)
- **professor_id** (Foreign Key referencing Professors.id)
- **content** (TEXT)
- **posted_at** (TIMESTAMP)

5. Notifications Table

This table keeps track of student notifications for when a professor becomes available.

- **id** (Primary Key, auto-increment)
- **student_id** (Foreign Key referencing Users.id)
- **professor_id** (Foreign Key referencing Professors.id)
- **notification_set_at** (TIMESTAMP)
- **notified** (BOOLEAN) – to indicate if notification has been sent

6. Favorites Table

This table records the relationships for students who have starred (favorited) a professor.

- **id** (Primary Key, auto-increment)
- **student_id** (Foreign Key referencing Users.id)
- **professor_id** (Foreign Key referencing Professors.id)
- **created_at** (TIMESTAMP)

Thoughts on Feasibility in 3 Days

- **Scope of Functionality:**

  The core functionality—user registration, professor status updates, real-time notifications, and basic dashboards—can be developed as a minimum viable product (MVP) in three days, especially if you're using a modern web framework that supports rapid development (such as Django, Ruby on Rails, or Node.js with Express).

- **Real-Time Features:**

  Implementing real-time updates (e.g., live status changes and immediate notifications) might require using websockets or a similar technology. Depending on your experience with these, this could extend your timeline slightly.

- **Testing and Refinement:**

  While a basic version might be achievable in three days, thorough testing, debugging, and UI refinements (like the smooth animations and responsive design details) could need extra time.

- **Team and Tools:**

  If you're working solo with a clear plan and have used similar projects before, three days might be possible for a prototype. However, if you're new to some of the required technologies, you might need additional time.

Conclusion

Your schema covers the essential data relationships needed for DocConnect. It provides a good starting point that aligns with the outlined user flows. Building a basic, functioning prototype in three days is ambitious but achievable, especially if you focus on core functionality and are comfortable with rapid development. For production-level quality with all interactive and real-time features, consider additional time for polish and thorough testing.

Does this schema align with your vision, or would you like to dive deeper into any particular aspect?

Thanks for the tech stack clarification—**Spring Boot + Angular + MySQL + Firebase Auth** is a solid and modern choice. Let's break it down based on your stack and what you're aiming to do in 3 days:

🔥 Should You Use Firebase All the Way?

Quick Answer:

Use **Firebase only for authentication**, and **keep your business logic and data in Spring Boot + MySQL**. That gives you the best of both worlds.

🧠 Why This Hybrid Approach Works Best:

✅ Use Firebase for Authentication Only

- Super fast setup (email/password auth out of the box)
- Handles secure login/registration flows, password resets, etc.
- Comes with SDKs for Angular (AngularFire), so you can protect routes and read user info easily
- Reduces a lot of boilerplate and lets you focus on app logic

🚫 Don't Use Firebase as Full Backend (in your case)

- You already have Spring Boot + MySQL for business logic (which is more flexible and structured)
- Firebase Firestore or Realtime DB would be overkill/chaotic for managing all the custom logic (e.g., status history, role-based dashboards, relationships between users)
- Complex joins, real-time filters, or status logs are **much cleaner in MySQL**

🛠️ Suggested Architecture Overview

🔐 Firebase Authentication (Frontend)

- Handle login/register using Angular + Firebase SDK
- After login, retrieve the user's Firebase ID token and send it in headers to your backend

🔄 Spring Boot Backend (API + MySQL)

- Use Firebase Admin SDK to verify the ID token sent from frontend
- Maintain your own `Users`, `Professors`, `Status`, etc., in MySQL
- Backend determines role-based access (student/professor)
- REST APIs for:
  - Registering user metadata (role, department, etc.)
  - Updating professor status
  - Managing announcements
  - Managing favorites and notifications

🧾 MySQL

- Use the schema we started building above (customized to match your final entities)
- Handles data relationships, status history, filters, etc.

⚡ Can You Build It in 3 Days?

If you:

- Stick to MVP features (core flows only)
- Use Firebase for Auth (saves lots of time)
- Reuse Angular Material for UI (to save time on styling)
- Keep animations/real-time stuff minimal or mocked

Then YES, you can build a functional prototype in 3 days.

✅ Tips for Speed

- Use Angular CLI to scaffold everything fast
- Use Spring Boot + JPA for quick entity-repository-service layers
- Use Firebase Auth emulator locally if needed for testing
- Don’t spend time on full user management—Firebase already handles it
- Use JWT or Firebase ID token verification filters in Spring Security (can help set up if you need)

Would you like help with:

- Setting up Firebase Auth integration with Angular?
- A starter Spring Boot Firebase ID token verifier?
- Scaffolded structure for your project in either Angular or Spring Boot?

Let me know how hands-on you want to go—I can generate starter code too if it helps you go faster!