Feb W1 Report

Hi Sara. Schema and auth backend is complete and Course content code from our Nov MVP is what I was editing so I can integrate it into this, but I wanna make sure it doesn't conflict anything. BUT, I'll be helping Subodh set up the UI first, so he can work alongsides. Also, lmk if you like this format and if you want me to attach code files. I've done things slightly differently.

What I Built

I developed a role-based user system that enforces a clear hierarchy:

- **Admin:** Automatically approved and can approve anyone.
- **SchoolGroup:** Approves Schools and any Teachers/Students linked to those Schools.
- **School:** Approves Teachers or Students tied to that School.
- **Teacher:** Approves Students in the same School.
- **Student:** Only attends classes, no approval rights.

This structure ensures that everyone in Spark OS has a clear chain of command, preventing invalid signups and keeping data organized.

Why It Matters

Security & Validation

No one can create a Teacher or Student account without a legitimate School. Schools require an approved SchoolGroup, and only an Admin can set up a SchoolGroup.

Simple Approvals

When a Student signs up, the system checks their School. If the School is valid and approved, then a School or Teacher can quickly approve the Student, ensuring no orphaned accounts.

Clear Logins

Users log in with their email or username and password. The system handles roles and validations behind the scenes while keeping the login process straightforward.

How It Works (Pseudo Code)

```
REGISTER_USER(email, username, password, role, references):
  1. Validate input fields.
  2. For 'school' role, confirm a valid, approved schoolGroup.
  3. For 'teacher' or 'student' role, confirm a valid, approved school.
  4. Securely hash the password.
  5. Set status: 'approved' if admin; otherwise 'pending'.
  6. Save the user record.

LOGIN_USER(login, password):
  1. Find user by email or username.
  2. Compare given password with stored hash.
  3. Generate and return a JWT token indicating the user's role.

APPROVE_USER(approver, userIdToApprove):
  1. Ensure user is in a 'pending' state.
  2. Validate that the approver has permission (admin, or within the same school/
schoolGroup).
  3. Mark the user as 'approved'.
```

Why I Chose This Design

- **Keeps control:** Each role has a defined scope of approval.
- **Scales easily:** The hierarchy is robust for adding new roles or features.
- **Reduces errors:** Prevents "floating" accounts that don't belong anywhere.

Final Thoughts

Now that the hierarchical approvals are up and running, we can onboard new users at each level without the risk of unauthorized signups or tangled data.

If anything is unclear or you want more details, just let me know.

Also, I'm curious if this report gives enough insight into the work I put in—it was a solid effort!

Key Components (Pseudo Code Overview)

User Model

```
// Define user fields: username, email, password, role, name, status, and hierarchy
references.
// Additional fields for students: date of birth, XP, streak, and last active date.
```

Authentication Controller

```
// For registration:
//   - Validate input.
//   - Check for existing users.
//   - Validate hierarchy references.
//   - Hash password and set status.
//   - Create and save user.
// For login:
//   - Locate user by email/username.
//   - Verify password and return JWT token.
// For approvals:
//   - Check pending status.
//   - Validate approver's permissions based on role and hierarchy.
//   - Approve user if valid.
```

Middleware & Server Setup

```
// Middleware checks for a valid JWT token and attaches user info to the request.
// Server connects to the database, sets up JSON parsing, and defines routes for
authentication.
```