bldr Course Scheduler

Initial Architecture Document

Team Number: 29
Team Members: Ansuman Sharma, Achinth Ulagapperoli, Jahnvi Maddila, Vamsi Doddapaneni, Taha Khalid
Project Name: *bldr Course Scheduler*
Project Synopsis: KU course-schedule builder that helps students plan schedules for a given semester from all courses offered by KU across all campuses.

1. Architecture Description

The **bldr** system is a modular web application that assists university students in constructing, validating, and visualizing semester schedules. The architecture emphasizes maintainability, modularity, and real-time responsiveness across three primary layers: the **frontend (React/Next.js)**, the **backend API (Next API)**, and the **data layer (Supabase PostgreSQL)**.

1.1 Overall System View

At the highest level, *bldr* operates on a client-server model. Users interact with a React/Next.js frontend hosted via Vercel or a similar deployment platform. The frontend communicates with a Node/Express backend through RESTful API endpoints exposed under `/api/v1/`. The backend interfaces with Supabase, which provides both an authenticated PostgreSQL database and built-in session handling.

When a student logs in, Supabase Auth issues a session token that is verified by backend middleware. The backend fetches relational course data, user schedules, and constraints directly from Supabase. The frontend displays a unified dashboard that includes the active term calendar, course search, and schedule visualization.

1.2 Backend Components

The backend consists of modular controllers mapped to core functionalities:

- **Auth Controller:** Validates Supabase sessions and manages user profiles.

- **Schedule Controller:** Handles CRUD operations for schedule items (add, remove, replace).
- **Catalog Controller:** Queries the Supabase `courses` table using department or course filters.
- **Constraint Module:** Applies business rules such as credit limits, overlapping time checks, and prerequisite enforcement.
- **Deployment Module:** Defines Dockerfiles for both backend and frontend for reproducible builds.

Each controller uses a shared `db.js` connection pool configured via Supabase's REST interface or PostgreSQL client. Requests are validated through middleware that checks payload structure and authentication before committing transactions.

1.3 Frontend Architecture

The frontend follows a component-driven Next.js structure with clear routing:

- `/signup` – Supabase Auth UI flow
- `/login` – Supabase Auth UI flow
- `/dashboard` – Main schedule building/class search workspace

Core React components:

- **ScheduleCalendar.tsx:** Renders enrolled courses using an in-house calendar view of selected courses.
- **CourseSearch.tsx:** Custom made component embedded in the `/dashboard` page that handles search and add-to-schedule functionality.
- **SidebarNavigation.tsx:** Provides persistent navigation between different schedules and semesters.

Frontend communication with backend APIs is performed through fetch API, while state management and data fetching are handled using React Query or SWR. A global context provider maintains session and user preference persistence.

1.4 Data Layer Design

The project uses five core tables that separate *catalog data* (all available classes) from *user-owned schedules* and a *join table* that links schedules to classes. Ownership is tracked through a lightweight user table and a user↔schedule association table. Primary keys and foreign keys are chosen to keep writes simple while enabling efficient reads for common UI flows (load active schedule, list classes in a schedule, add/remove a class).

Access Patterns

- **Load active schedule for a user:** resolve `onlineid` → `userschedule(isactive=true)` → `scheduleid` → join `scheduleclasses` + `allclasses`.
- **Add a class:** insert into `scheduleclasses(scheduleid, classid)` after validation (time overlap, credit cap, seat availability if enforced).
- **Remove a class:** delete from `scheduleclasses` by (`scheduleid`, `classid`).
- **Catalog queries:** filter `allclasses` by `dept`, `code`, `component`, `days`, or time ranges.

Indexes and Constraints (recommended)

- `allclasses`: `UNIQUE(classid)`, indexes on `(dept, code)`, `(semester?, year?)` if added.
- `scheduleclasses`: `UNIQUE(scheduleid, classid)`, btree on `scheduleid` for fast expansion of a schedule.
- `userschedule`: partial index `(onlineid)` where `isactive=true` to accelerate dashboard default load.

Ownership and Security Each row in `userschedule` binds a user to a schedule; API endpoints must scope all `allschedules/scheduleclasses` operations by `onlineid` through `userschedule`. This enforces *row ownership* without exposing cross-user data. If using Supabase RLS, policies can mirror the same ownership rule using `auth.uid()` mapped to `userdata.onlineid`.

1.5 Data Flow and Interactions

1. User opens *bldr* and authenticates via Supabase.
2. Frontend loads schedule and course data through API calls.

3. User adds a course → frontend sends POST request → backend validates time conflicts → writes to `schedules`.
4. Backend returns updated data → frontend updates calendar view dynamically.

This asynchronous feedback loop ensures responsiveness while maintaining data integrity.

1.6 Non-Functional Considerations

- **Security:** Supabase RLS and JWT validation.
- **Scalability:** Docker-based modular deployments.
- **Reliability:** Transaction rollbacks for failed writes.
- **Performance:** Indexed catalog queries and efficient caching.

1.7 Technology Stack

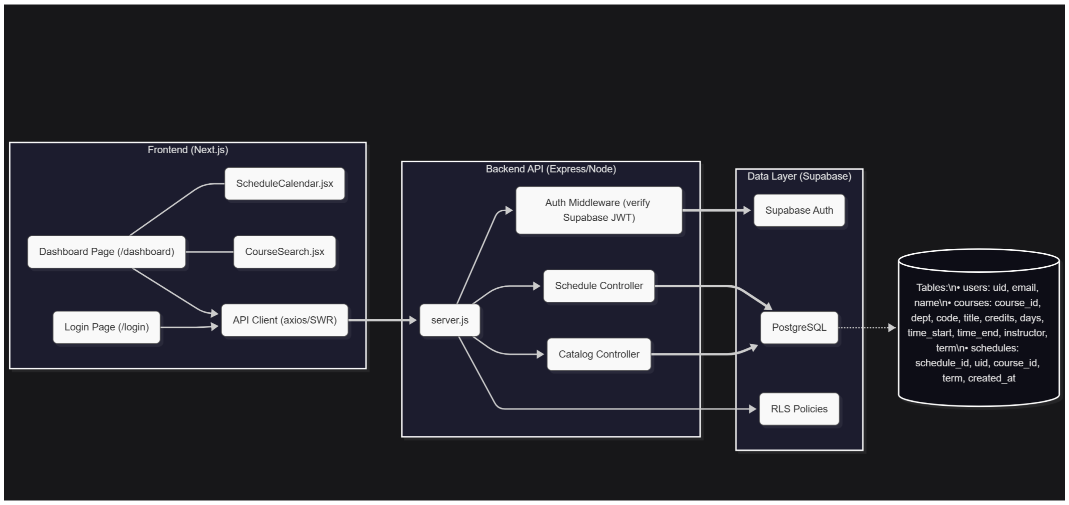
Layer	**Tools / Technologies**
Frontend	Next.js (React 18), Tailwind CSS
Backend	Node.js (Express), Axios
Database	Supabase (PostgreSQL + Auth)
Deployment	Docker, Vercel
Testing	Manual Sprint Verification

1.8 Future Extensions

The modular design allows future integration of AI-driven class recommendations and constraint-based optimization without altering core modules.

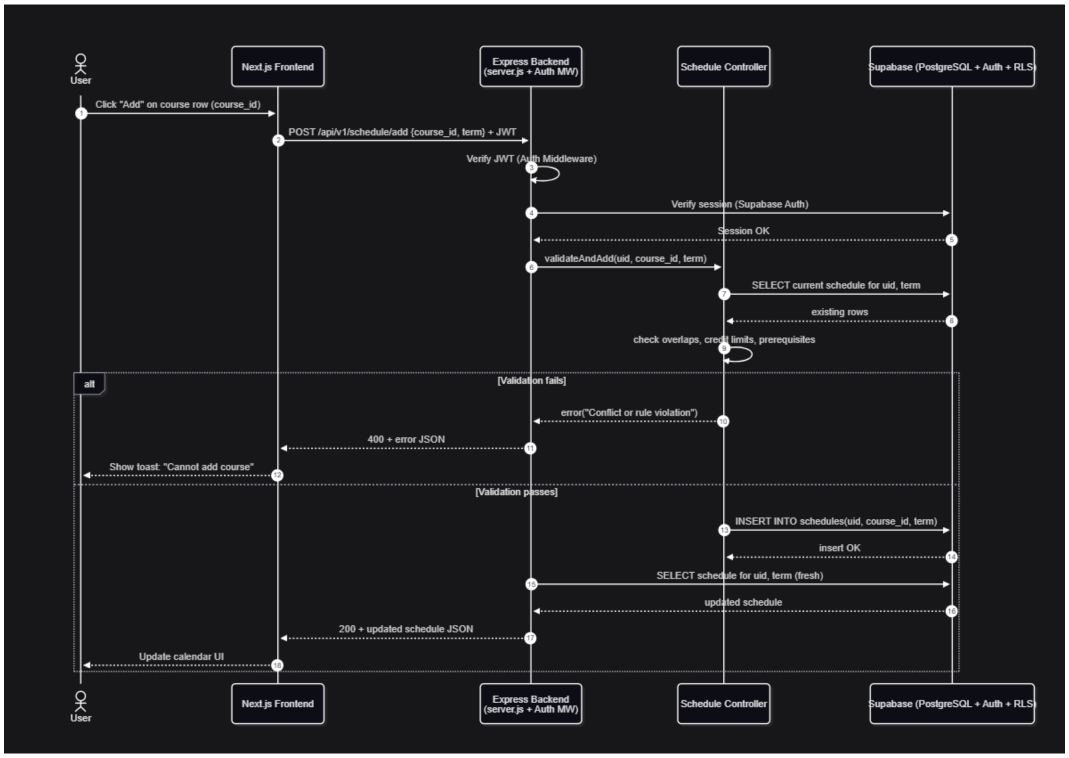
2. UML Diagrams

2.1 Component Diagram


Frontend (Next.js)
ScheduleCalendar.jsx
Dashboard Page (/dashboard)
CourseSearch.jsx
Login Page (/login)
API Client (axios/SWR)
Backend API (Express/Node)
Auth Middleware (verify Supabase JWT)
Schedule Controller
server.js
Catalog Controller
Data Layer (Supabase)
Supabase Auth
PostgreSQL
RLS Policies
Tables:\n• users: uid, email, name\n• courses: course_id, dept, code, title, credits, days, time_start, time_end, instructor, term\n• schedules: schedule_id, uid, course_id, term, created_at

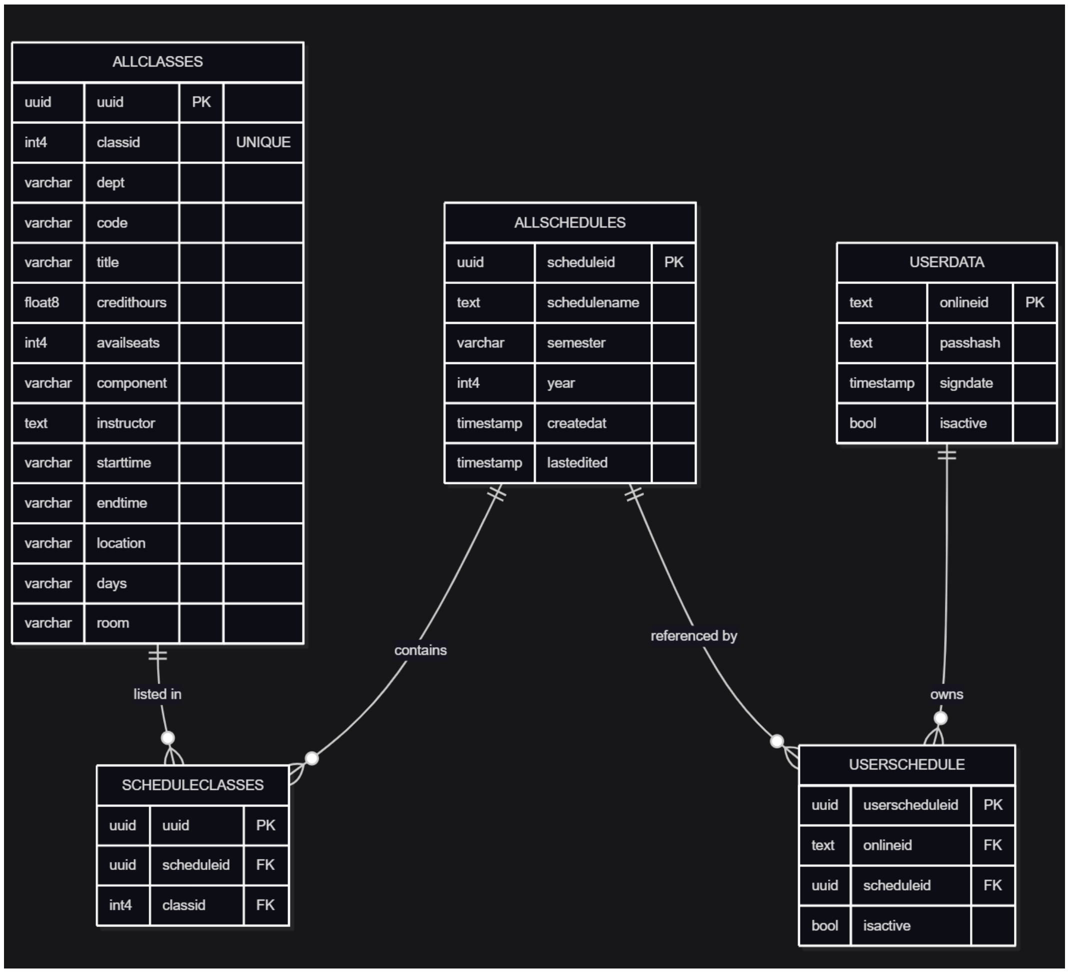
Depicts the relationship between frontend, backend, Supabase, and Auth subsystems.

2.2 Data Flow Diagram (Level 1)


User
Next.js Frontend
Express Backend (server.js + Auth MW)
Schedule Controller
Supabase (PostgreSQL + Auth + RLS)
1 Click "Add" on course row (course_id)
2 POST /api/v1/schedule/add {course_id, term} + JWT
3 Verify JWT (Auth Middleware)
4 Verify session (Supabase Auth)
5 Session OK
6 validateAndAdd(uid, course_id, term)
7 SELECT current schedule for uid, term
8 existing rows
9 check overlaps, credit limits, prerequisites
alt
[Validation fails]
10 error("Conflict or rule violation")
11 400 + error JSON
12 Show toast: "Cannot add course"
[Validation passes]
13 INSERT INTO schedules(uid, course_id, term)
14 insert OK
15 SELECT schedule for uid, term (fresh)
16 updated schedule
17 200 + updated schedule JSON
18 Update calendar UI

Shows the flow of data between user, API, and database layers.

2.3 Entity Relationship Diagram



ALLCLASSES			
uuid	uuid	PK	
int4	classid		UNIQUE
varchar	dept		
varchar	code		
varchar	title		
float8	credithours		
int4	availseats		
varchar	component		
text	instructor		
varchar	starttime		
varchar	endtime		
varchar	location		
varchar	days		
varchar	room		

ALLSCHEDULES		
uuid	scheduleid	PK
text	schedulename	
varchar	semester	
int4	year	
timestamp	createdat	
timestamp	lastedited	

USERDATA		
text	onlineid	PK
text	passhash	
timestamp	signdate	
bool	isactive	

SCHEDULECLASSES		
uuid	uuid	PK
uuid	scheduleid	FK
int4	classid	FK

USERSCHEDULE		
uuid	userscheduleid	PK
text	onlineid	FK
uuid	scheduleid	FK
bool	isactive	

Illustrates `users`, `courses`, `schedules`, *and* `constraints` *relations.*