COMS3011A – Group Report

Big-O-No
Mareme Mogoru 2675094
Wakhiwe Ndzimandze 2694083
Sesihle Goniwe 2680440
Khuselo Sofohlo 2729931
Doyen Simango 1602758

October 19, 2025

Contents

1 Introduction

In today's dynamic academic landscape, students face unprecedented challenges in finding effective study collaboration and maintaining consistent learning momentum. The traditional barriers of geographical isolation, scheduling conflicts, and difficulty in locating compatible study partners often hinder the collaborative learning experience that is crucial for academic success. **StudyNest** emerges as a comprehensive solution to these challenges, providing a sophisticated digital ecosystem designed to transform how students connect, collaborate, and track their academic progress. StudyNest represents a full-stack web application built on cutting-edge technologies including **Angular** for the frontend, **NestJS** for the backend, and **PostgreSQL** on Supabase for crucial data management. The platform serves as a centralized hub where students can seamlessly discover study partners based on shared courses and academic interests.

2 Methodology & Planning

2.1 Choice of Development Methodology

Our team adopted a **Kanban-based Agile methodology** for the StudyNest WebApp. Kanban was selected for its visual task management and flexibility in continuous delivery. We integrated Scrum elements with weekly sprints to maintain structure and accountability while leveraging Kanban's workflow visualization principles.

2.2 Following of Development Methodology

We maintained strict adherence to our methodology through:

1. **Visual Workflow Management**: Active Kanban board with To-Do, In-Progress, and Done columns for real-time task visibility through Notion.
2. **Structured Sprints**: Five weekly sprints (Sept 5–29, 2025) with documented agendas and deliverables.
3. **Issue Tracking**: Systematic bug management through Linear.
4. **Team Communication**: Regular coordination via WhatsApp for rapid response.
5. **Clear Ownership**: Task distribution based on team member expertise.

2.3 Review of Development Methodology

The Kanban approach proved effective, with the visual board facilitating transparency and early bottleneck identification. Weekly sprints provided valuable checkpoints, particularly during mid-sprint reviews addressing authentication bugs.

Strengths: Enabled continuous feature delivery, maintained accountability, and supported iterative improvements through retrospectives.

Areas for Improvement: Future weeks of Kanban should implement explicit work-in-progress limits and clearer definition of done criteria.

The methodology successfully balanced structure with flexibility, contributing to effective project delivery.

2.4 Project Design Project Planning

2.4.1 Project Design Approach

The StudyNest platform was designed using a modular, component-based architecture that separates concerns across three distinct layers:

- **Frontend Layer:** Angular-based single-page application providing responsive user interfaces
- **Backend Layer:** NestJS RESTful API handling business logic and data processing
- **Database Layer:** PostgreSQL database managed through Supabase for data persistence

2.4.2 Design Principles

The system architecture follows these core design principles:

- **Separation of Concerns:** Clear division between presentation, business logic, and data layers
- **API-First Design:** RESTful endpoints designed before implementation to ensure consistency
- **Component Reusability:** Frontend components built for maximum reuse across different features
- **Scalability:** Stateless backend design allowing horizontal scaling
- **Security by Design:** Authentication and authorization integrated from initial design phase

2.4.3 Project Planning Methodology

Our planning process utilized a hybrid Agile-Kanban approach with the following elements:

- **Sprint Planning:** Weekly sprints with clearly defined deliverables and acceptance criteria
- **Task Breakdown:** User stories decomposed into technical tasks with estimated effort points
- **Progress Tracking:** Visual Kanban board showing task status (To Do, In Progress, Review, Done)
- **Regular Syncs:** Daily stand-ups and weekly retrospectives to adapt plans based on progress

2.4.4 Development Timeline

The project followed a structured timeline across five development sprints:

1. **Sprint 1 (Week 1):** Foundation setup, authentication system, basic user profiles
2. **Sprint 2 (Week 2):** Study group creation and management features
3. **Sprint 3 (Week 3):** Partner matching algorithm and user discovery
4. **Sprint 4 (Week 4):** Progress tracking and study session scheduling
5. **Sprint 5 (Week 5):** Notifications system, polish, and user testing

2.4.5 Risk Management

Potential risks were identified and mitigated through:

- **Technical Risks:** Early prototyping of complex features like real-time notifications
- **Scope Risks:** MoSCoW prioritization (Must-have, Should-have, Could-have, Won't-have)
- **Team Risks:** Cross-training on critical components to prevent knowledge silos
- **Integration Risks:** Early API contract definitions between frontend and backend teams

2.5 Stakeholder Interaction

Key stakeholders for the StudyNest platform include:

- **Primary Users:** University students seeking study collaboration tools
- **Secondary Users:** Group admins managing study sessions and materials
- **Academic Institutions:** Potential institutional adoption for student support Wits, Rosebank
- **Development Team:** Internal stakeholder, Yintla, and group members

2.5.1 User-Centric Design Process

Stakeholder needs were incorporated through:

- **User Personas:** Developed detailed profiles of target student users
- **Feature Prioritization:** Regular feedback sessions to rank feature importance
- **Usability Testing:** Continuous testing with actual students throughout development
- **Feedback Integration:** Structured process for incorporating user suggestions

2.5.2 Communication Strategy

Stakeholder engagement was maintained through:

- **Regular Demos:** Bi-weekly showcases of new features and progress
- **Feedback Channels:** Multiple avenues for users to report issues and suggest improvements
- **Transparent Roadmap:** Clear communication of upcoming features and timelines
- **Responsive Support:** Quick turnaround on user questions and technical issues

2.5.3 Stakeholder Roles and Responsibilities

Each stakeholder group contributed uniquely to the project's success:

- **Students:** Provided user stories, participated in usability testing, and validated design choices.
- **Group Admins:** Advised on moderation and access control requirements.
- **Academic Institutions:** Offered insights on potential institutional integration and compliance requirements.
- **Development Team:** Implemented and maintained the system, ensuring alignment with stakeholder expectations.

2.5.4 Stakeholder Feedback Summary

Key feedback from stakeholders shaped several major design decisions:

- Students requested a **chat feature** for real-time study collaboration.
- Admins emphasized the need for **role-based permissions**.

These insights directly influenced the final design and implementation roadmap.

3 Architecture

3.1 API Documentation

The Campus Study Buddy API powers the platform's main features: partner matching, group sessions, progress tracking, and notifications.

- **Base URL:** `https://studynester.onrender.com/`
- **Content Type:** `application/json`

Partner Matching API

- **Base URL:** `https://studynester.onrender.com/StudentCourses/`

Endpoints

Method & Endpoint	Description
GET /partners	Retrieve list of potential partners
POST /partners/preferences	Set or update partner matching preferences
GET /partners/{id}	View student profile
PATCH /partners/{id}	Update user profile
DELETE /partners/{id}	Remove a profile

Groups API

- **Base URL:** `https://studynester.onrender.com/groups`

Endpoints

Method & Endpoint	Description
POST /groups	Create a study group

GET /groups List groups user belongs to

GET /groups/{id} View group details

POST /groups/{id}/members Join a group

POST /groups/{id}/schedule Schedule a study session

DELETE /groups/{id} Delete a group

Progress API

Endpoints

- **Base URL:** `https://studynester.onrender.com/progress`

Method & Endpoint Description

POST /progress Log a new topic/section

GET /progress/{userId} Retrieve progress for a user

PATCH /progress/{id} Update progress entry

DELETE /progress/{id} Remove a progress entry

Notifications API

- **Base URL:** `https://studynester.onrender.com/notifications`

Endpoints

Method & Endpoint Description

GET /notifications/{userId} Fetch user notifications

PATCH /notifications/{id} Mark as read or update

DELETE /notifications/{id} Delete a notification

This API supports all core features of the Campus Study Buddy platform: partner discovery, group collaboration, study progress tracking, and personalized reminders.

3.2 Database Documentation

Database Choice and Justification

For this project, we selected **PostgreSQL**, hosted on the **Supabase** platform, as our database solution.

Database Type

PostgreSQL is a relational database management system (RDBMS). It supports complex queries, transactions, and relationships between entities, which are crucial for managing user interaction with stored information.

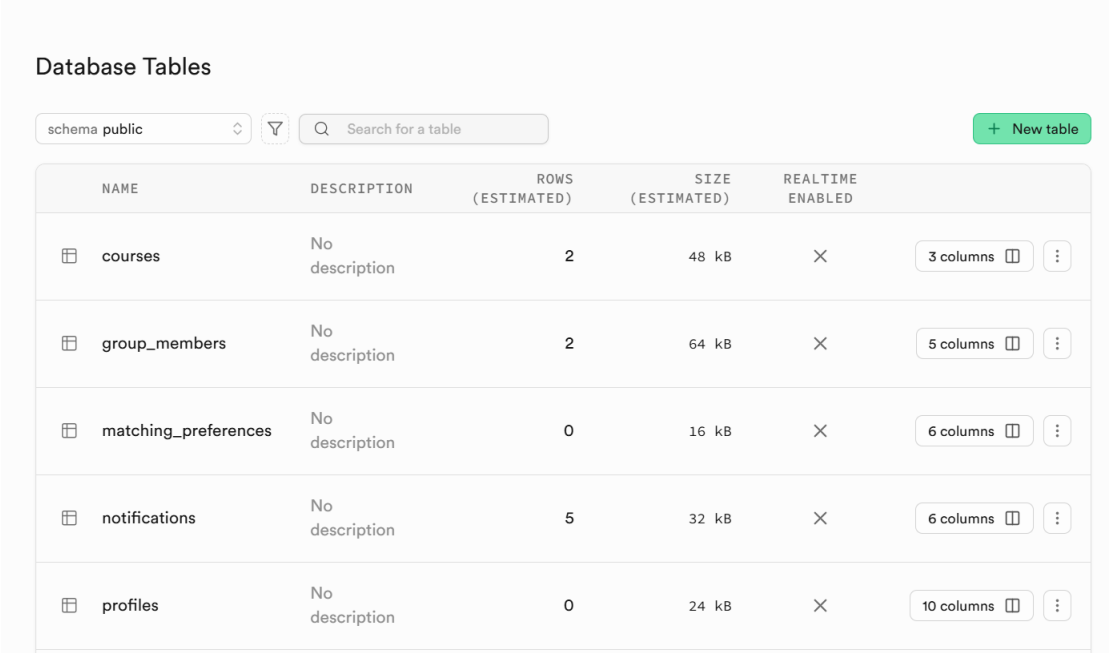
Justification for PostgreSQL with Supabase

- **Relational Structure:** The platform requires multiple interrelated entities (e.g., users, groups, sessions, notifications etc). A relational database ensures consistency and enforces constraints such as foreign keys and unique identifiers.
- **Integration with Backend:** Supabase provides a PostgreSQL database with built-in APIs and authentication, which integrates seamlessly with our **NestJS** backend and Angular frontend.
- **Scalability:** PostgreSQL is highly scalable in terms of performance and schema evolution. Supabase further simplifies scaling by handling cloud hosting and database provisioning.
- **Developer Productivity:** Supabase offers ready-to-use features such as Row-Level Security (RLS), user authentication, and migration tools. This reduces overhead and allows the team to focus on application logic rather than database setup.
- **Team Familiarity:** Team members have prior experience with SQL and relational models, which minimizes the learning curve and improves development speed.

In summary, PostgreSQL on Supabase was chosen because it provides a robust, scalable, and developer-friendly relational database solution that supports the functional and non-functional requirements of Campus Study Buddy.

Database Schema

Our database for this sprint consists on several tables which we used to implement out pregress.

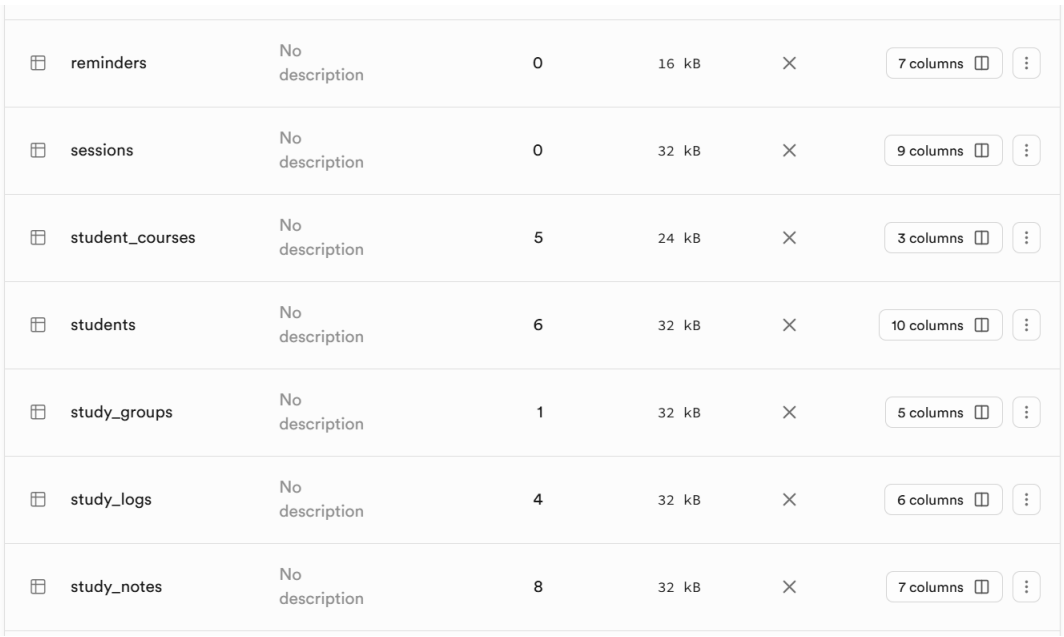


Database Tables

schema public | Search for a table | + New table

NAME	DESCRIPTION	ROWS (ESTIMATED)	SIZE (ESTIMATED)	REALTIME ENABLED	
courses	No description	2	48 kB	✕	3 columns
group_members	No description	2	64 kB	✕	5 columns
matching_preferences	No description	0	16 kB	✕	6 columns
notifications	No description	5	32 kB	✕	6 columns
profiles	No description	0	24 kB	✕	10 columns

Figure 1: Database tables



reminders	No description	0	16 kB	✕	7 columns
sessions	No description	0	32 kB	✕	9 columns
student_courses	No description	5	24 kB	✕	3 columns
students	No description	6	32 kB	✕	10 columns
study_groups	No description	1	32 kB	✕	5 columns
study_logs	No description	4	32 kB	✕	6 columns
study_notes	No description	8	32 kB	✕	7 columns

Figure 2: Database tables

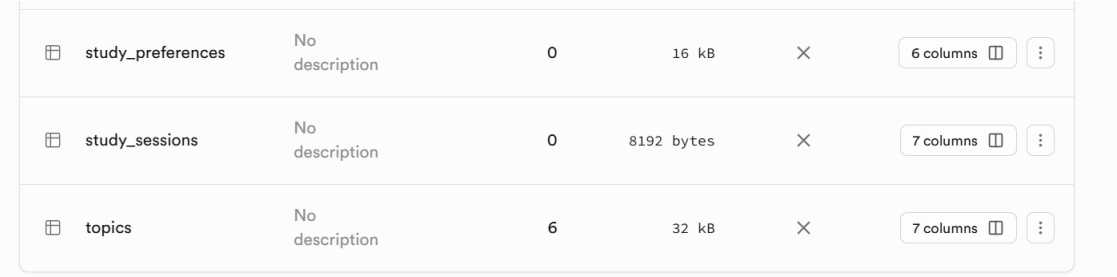


Figure 3: Database tables

ERD Diagrams

This diagrams illustrate all the relationships between all the tables above and their perspective relationships.

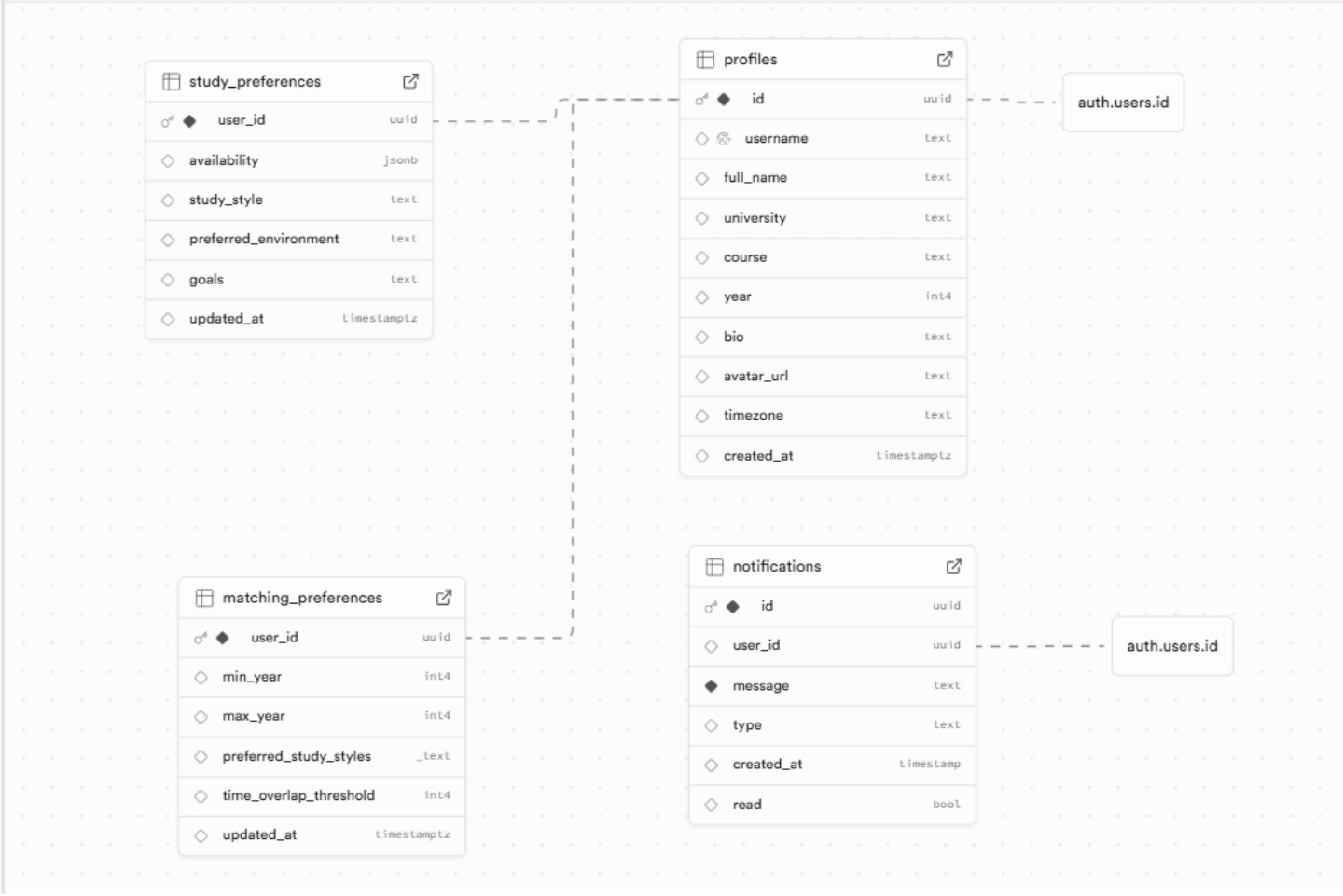


Figure 4: Initial Part of the ERD

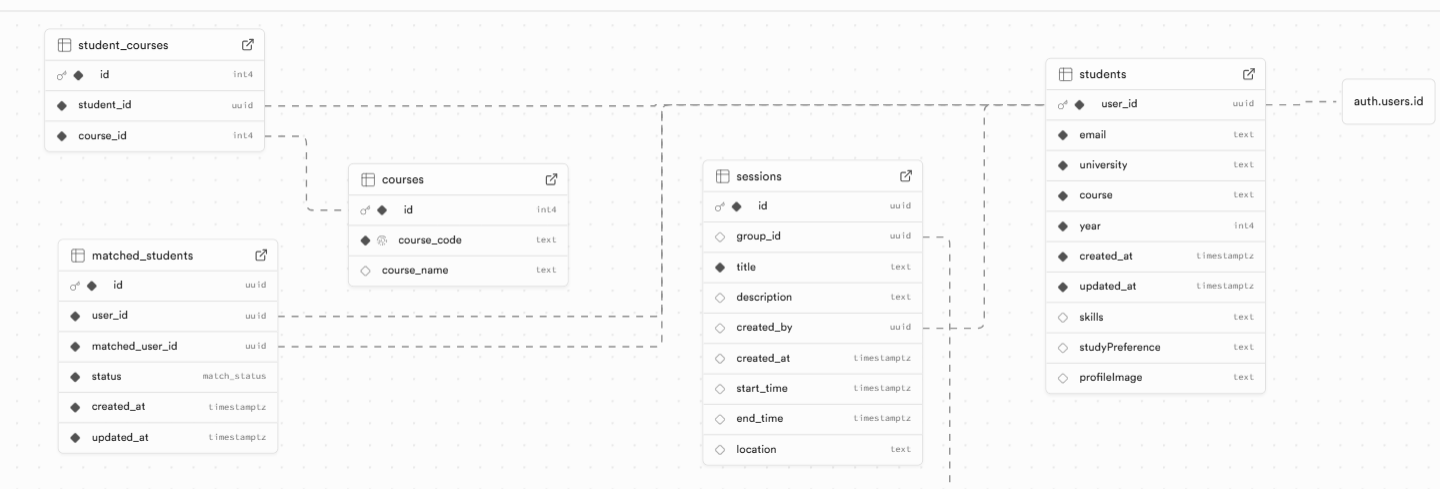


Figure 5: Middle Part of the ERD

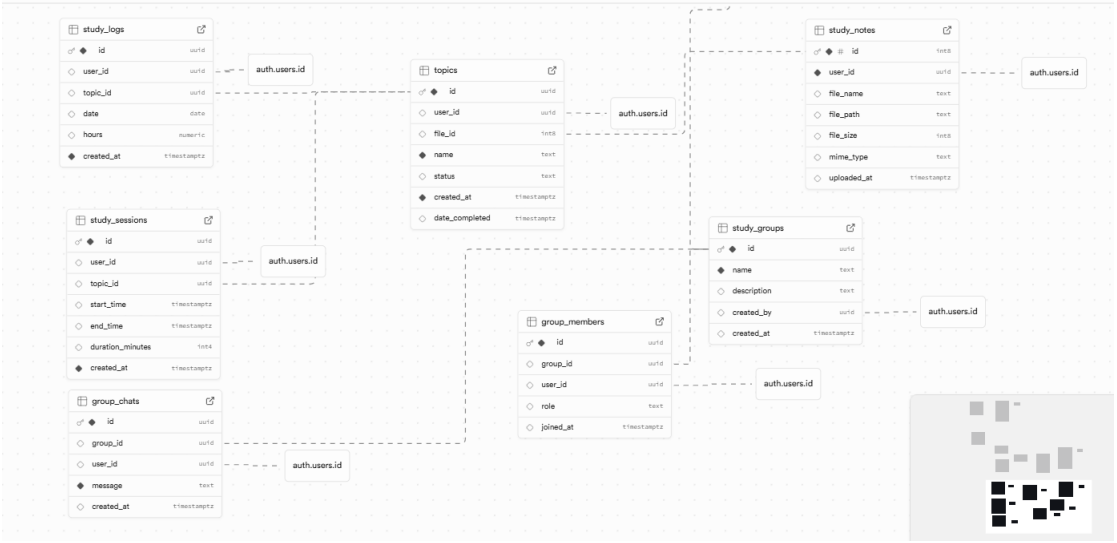


Figure 6: Last Part of the ERD

3.3 Developer Guides Tech Stack

3.3.1 Technology Stack Overview

StudyNest is built using a modern, full-stack JavaScript/TypeScript technology stack:

- **Frontend:** Angular 16+ with TypeScript
- **Backend:** NestJS with Node.js and TypeScript
- **Database:** PostgreSQL hosted on Supabase
- **Authentication:** Supabase Auth with Row-Level Security
- **Deployment:** Cloudflare Pages (Frontend) and Render (Backend)

3.3.2 Frontend Development Guide

```
# Clone the repository
git clone https://github.com/studynest/frontend.git

# Install dependencies
npm install

# Configure environment
cp src/environments/environment.example.ts src/environments/environment.ts

# Start development server
ng serve
```

Key Dependencies

- **@angular/core:** Core Angular framework
- **@angular/material:** UI component library
- **@supabase/supabase-js:** Supabase client for authentication and data
- **rxjs:** Reactive programming library
- **chart.js:** Data visualization for progress tracking

```
src/
          app/
                components/     # Reusable UI components
                services/       # Data and business logic services
                pages/          # Feature pages and routes
                models/         # TypeScript interfaces
                guards/         # Route protection
          assets/             # Static resources
          environments/       # Configuration files
```

3.3.3 Backend Development Guide

```
# Clone the repository
git clone https://github.com/studynest/backend.git

# Install dependencies
npm install

# Setup environment variables
cp .env.example .env

# Run database migrations
npx supabase db push

# Start development server
npm run start:dev
```

Key Dependencies

- **@nestjs/core:** NestJS framework core
- **@nestjs/jwt:** JWT authentication utilities
- **@supabase/supabase-js:** Database and auth client
- **class-validator:** Input validation decorators
- **@nestjs/mapped-types:** Utility types for DTOs

```
src/
          auth/                # Authentication logic
          users/               # User management
          groups/              # Study groups feature
          progress/            # Progress tracking
          notifications/       # Notification system
          common/              # Shared utilities and decorators
          main.ts             # Application entry point
```

3.3.4 Database Development Guide

```
# Create new migration
npx supabase migration new migration_name

# Apply migrations
npx supabase db push

# Reset database (development)
npx supabase db reset
```

Row-Level Security (RLS) Policies All database tables implement RLS policies for security:

- **Users:** Can only access their own profile data
- **Groups:** Members can read group data, admins have full access
- **Study Sessions:** Visible to group members only
- **Progress:** Private to each user unless explicitly shared

3.3.5 Development Workflow

Branch Strategy

- **main:** Production-ready code
- **develop:** Integration branch for features
- **feature/*:** Individual feature development

Code Quality Standards

- **TypeScript:** Strict type checking enabled
- **ESLint:** Consistent code style enforcement
- **Prettier:** Automated code formatting
- **Husky:** Pre-commit hooks for quality checks

Testing Strategy

- **Unit Tests:** Jest for backend, Jasmine for frontend
- **Integration Tests:** API endpoint testing
- **E2E Tests:** Cypress for critical user flows
- **Test Coverage:** Minimum 80% coverage requirement

3.3.6 Deployment Guide

```
# Build production version
npm run build

# Deploy to Cloudflare Pages
npx wrangler pages publish dist/studynest
```

Backend Deployment (Render)

- Automatic deployment from GitHub main branch
- Environment variables configured in Render dashboard
- Health checks at /health endpoint

Environment Configuration Required environment variables:

```
# Frontend (.env)
SUPABASE_URL=your_supabase_url
SUPABASE_ANON_KEY=your_anon_key

# Backend (.env)
```

```
DATABASE_URL=postgresql://...
JWT_SECRET=your_jwt_secret
SUPABASE_SERVICE_KEY=your_service_key
```

This comprehensive tech stack and developer guide ensures consistent development practices, smooth onboarding of new team members, and maintainable code quality throughout the project lifecycle.

4 External APIs and Third-Party Integrations

The Campus Study Buddy platform integrates with external APIs and third-party libraries to extend functionality and enhance user experience. This section documents both external API integrations and supporting libraries used throughout the application.

4.1 External API Integration

4.1.1 Clubs Connect API

The Campus Study Buddy platform integrates with the **Clubs Connect API** developed by another student group to provide users with relevant upcoming events from campus clubs and societies.

API Details

- **Provider:** Clubs Connect Team
- **Base URL:** `https://clubs-connect-api.onrender.com`
- **Authentication:** Public API (no authentication required)
- **Purpose:** Fetch upcoming campus events relevant to study topics

Integration Architecture The integration follows a proxy pattern where our backend service acts as an intermediary between the frontend and the external API. This approach provides several advantages:

- Centralized error handling and logging
- Ability to filter and transform data before sending to frontend
- Protection of external API endpoints from direct client access
- Simplified frontend implementation

Data Flow

1. User navigates to the events section in the frontend
2. Frontend service makes HTTP GET request to `/events/upcoming` on our backend
3. Backend service proxies the request to Clubs Connect API at `https://clubs-connect-api.onrender.com/api/events`
4. External API returns all upcoming events
5. Backend returns the data to frontend
6. Frontend service filters events based on study-relevant keywords
7. Filtered events are displayed to the user

Third-Party Libraries and Services

The platform utilizes several third-party libraries to provide specialized functionality that would be time-consuming and resource-intensive to develop in-house.

PDF Viewer - NgxExtendedPdfViewerModule

Overview NgxExtendedPdfViewerModule is a full-featured PDF viewer for Angular applications that enables in-browser PDF viewing without external plugins.

Integration Purpose

- Display lecture notes and study materials in PDF format
- Enable annotation and highlighting features for collaborative studying
- Provide seamless reading experience within the application

```
npm install ngx-extended-pdf-viewer
```

Key Features

- Browser locale detection for internationalization
- Responsive height settings (80% viewport height)
- Zoom controls for enhanced readability
- No server-side processing required
- Mobile-responsive design
- Accessibility features including text-to-speech support

Charts - Chart.js

Overview Chart.js is a flexible JavaScript charting library that provides eight different chart types with extensive customization options and responsive design capabilities.

Integration Purpose

- Visualize study progress and time tracking analytics

```
npm install chart.js
```

Chart Types Utilized

- **Line charts:** Study progress tracking over time
- **Bar charts:** Comparing study hours across subjects
- **Pie charts:** Visualizing study method distribution

Email Notifications - Mailjet

Overview Mailjet is a cloud-based email service provider that offers reliable transactional and marketing email capabilities through a robust API, making it ideal for scalable notification systems.

Integration Purpose

- Send study session reminders and notifications

```
npm install node-mailjet
```

Email Templates

- Session reminders with detailed information

Security Features

- TLS encryption for all email communications

4.2 Automated Testing Procedure

Testing

This section provides an overview of the testing frameworks and code coverage results for the project. It outlines the tools and methodologies used for both frontend and backend testing, explains why they were selected, and highlights the benefits of the different types of tests performed.

Testing plays a crucial role in software development. It not only verifies that the application meets its functional requirements but also ensures long-term maintainability, prevents regressions, and provides confidence when introducing new features or refactoring existing code.

Frontend Testing

Framework Used: Jasmine

The frontend is tested using the **Jasmine** testing framework. Jasmine is a behavior-driven development (BDD) framework for JavaScript that emphasizes readability and simplicity. It was chosen because:

- It integrates seamlessly with most frontend environments without requiring additional dependencies.
- Its descriptive syntax makes test cases easy to understand, even for new developers joining the project.
- It provides built-in support for spies, mocks, and asynchronous testing, which are particularly useful for testing frontend logic and interactions.

Types of Tests

For the frontend, the following types of tests were implemented:

- **Unit Tests:** Focused on individual components, functions, and services to ensure they behave correctly in isolation.
- **Integration Tests:** Verified the interaction between multiple components, such as ensuring that user inputs correctly propagate through forms and services.

Benefits

These tests are beneficial because they:

- Help detect errors early in the development cycle before they reach production.
- Increase confidence when making UI or logic changes, as regressions are quickly identified.
- Support maintainability by acting as executable documentation of the expected behavior of the application.

Backend Testing

Framework Used: Jest

The backend is tested using the **Jest** framework. Jest was chosen because:

- It is widely adopted in the JavaScript and TypeScript ecosystem, with strong community support.
- It provides an extensive set of features out of the box, such as mocking, assertions, and coverage reporting.
- Its snapshot testing capability is particularly useful for verifying structured data responses and ensuring consistent API outputs.

Types of Tests

For the backend, the following types of tests were implemented:

- **Unit Tests:** Checked the correctness of individual functions, utilities, and smaller modules of the backend.
- **Integration Tests:** Verified that multiple modules, such as controllers, services, and databases, work correctly when combined.
- **API Tests:** Ensured that REST endpoints respond with the correct data and status codes under various conditions.
- **Snapshot Tests:** Used to compare structured outputs against stored "snapshots" to detect unintended changes.

Benefits

The benefits of backend testing include:

- Preventing regressions in critical business logic and API endpoints.
- Increasing developer confidence when making changes to the codebase.
- Ensuring data consistency, correctness of responses, and resilience against invalid input or edge cases.

Coverage Screenshots

The following figures show the coverage results for the frontend and backend:

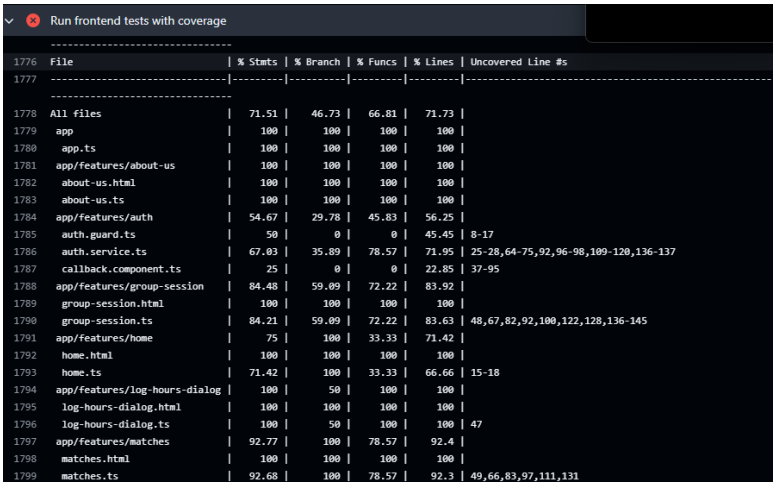


Figure 7: Frontend Code Coverage

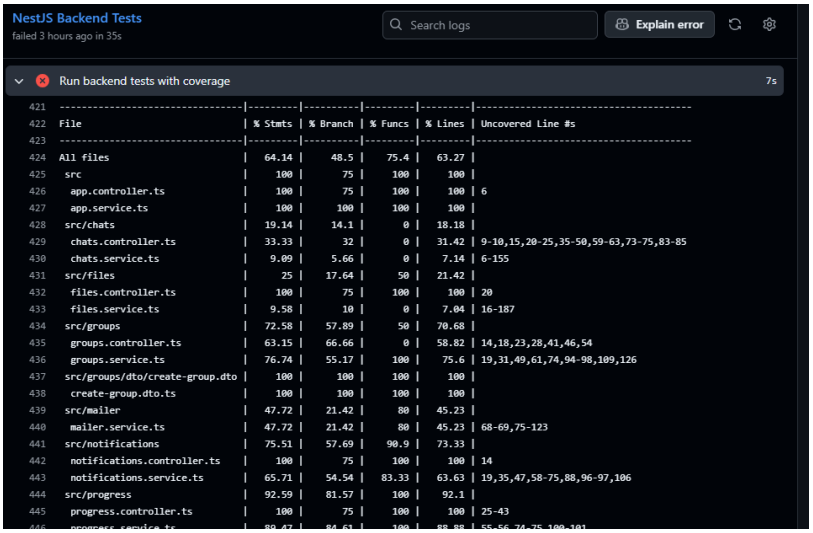


Figure 8: Backend Code Coverage

4.3 User Feedback Procedure

To evaluate the usability and overall user experience of the StudyNest application, a structured user feedback survey was conducted among a group of undergraduate students from various universities, including the University of the Witwatersrand and Rosebank College. The objective was to gather both quantitative and qualitative data regarding the app's design, ease of use, performance, and overall satisfaction.

Survey Design

The survey was designed using Google Forms and distributed to a sample of 5 users who had interacted with the StudyNest platform during its testing phase. It included a combination of Likert-scale and open-ended questions to capture both measurable and descriptive feedback. The key focus areas were:

- Frequency of app usage.
- Ease of navigation and intuitiveness of creating/joining study groups.
- Usefulness of features such as chat and note sharing.
- Technical performance and reliability.
- Desired additional features or improvements.

Data Collection

Responses were collected electronically and stored in a CSV format for analysis. Each participant provided details about their institution, year of study, and their experience with the application.

The collected data was then categorized into quantitative (ratings) and qualitative (comments and suggestions) components for analysis.

Purpose of Evaluation

The aim of this feedback procedure was to assess user satisfaction, identify technical issues, and prioritize improvements for the next iteration of the application. This ensured that the StudyNest app aligns with the real needs of students and enhances collaborative learning efficiency.

4.4 User Feedback Analysis and Review

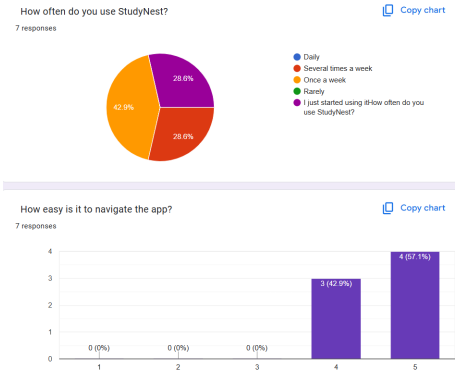
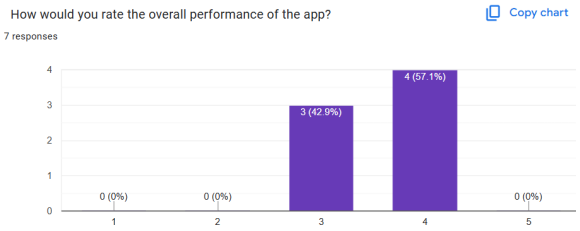


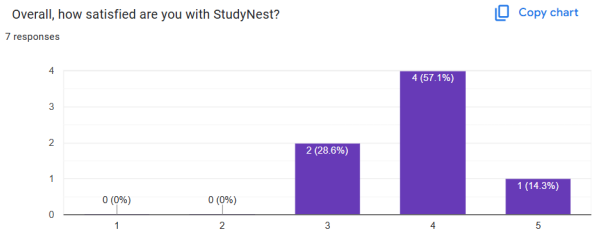
Figure 9: Frequency and Ease of Use of StudyNest

(a) Groups and Sharing Notes



(b) Performance

Figure 10: Comparison of Group and Performance Feedback



The responses gathered from the survey provided valuable insights into user experience and areas requiring refinement. Both numerical ratings and written feedback were analyzed to identify key themes and patterns.

Quantitative Analysis

Overall, the responses indicated a positive reception:

- **Ease of Navigation:** Average rating of 4.6/5 – users found the interface simple and accessible.
- **Creating and Joining Study Groups:** Average rating of 4.6/5 – participants felt that group creation and participation were intuitive.
- **Chat and Note Sharing:** Average rating of 4.0/5 – while useful, some users suggested performance improvements.
- **Overall App Performance:** Average rating of 3.4/5 – users reported occasional bugs and slow loading.
- **Overall Satisfaction:** Average rating of 3.8/5 – most participants were satisfied with the app but saw room for enhancement.

Qualitative Feedback

Common themes and comments included:

- **Positive Remarks:** Users praised the app's clean user interface and ease of use.
- **Technical Issues:** Reports of slow page refreshes, unsaved sessions, and delayed group chat loading.
- **Feature Requests:** Suggestions for automatic study partner matching, lecturer chat integration, a map-based session locator, and AI-assisted study tools.

Findings and Improvements

The feedback demonstrated that while the core functionality of StudyNest is effective and user-friendly, certain optimizations could improve the experience. Based on the responses:

- Backend optimization and caching strategies will be implemented to improve response time and chat synchronization.

- A new feature for **automatic study partner matching** was prioritized in future releases.
- User profile persistence and session management is enhanced to prevent loss of progress when navigating back.

Conclusion

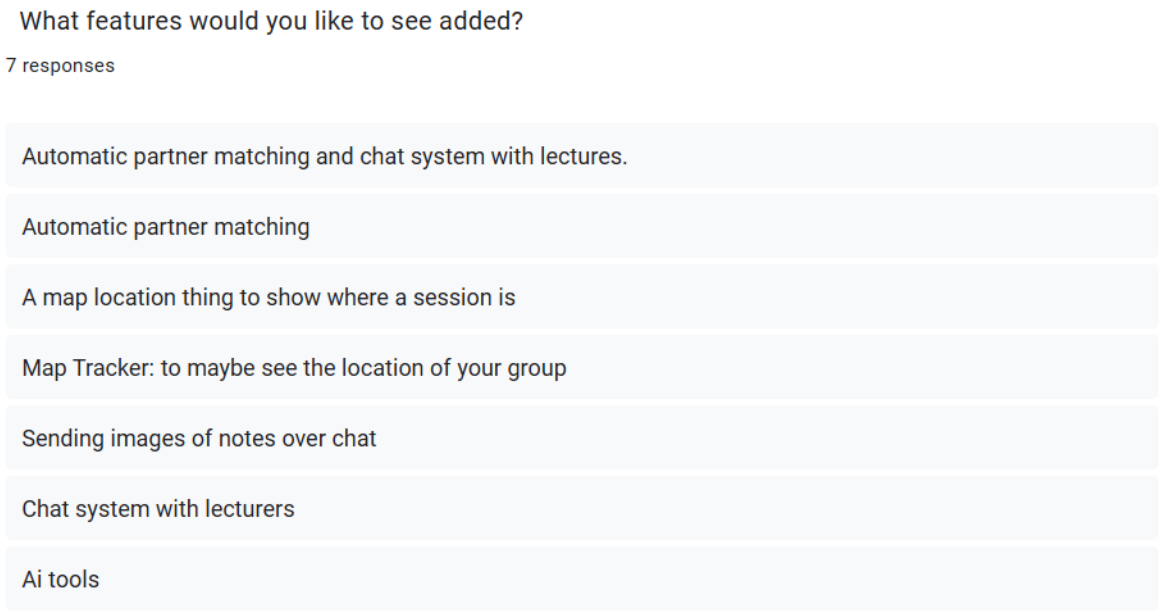


Figure 11: Features implemented and features to be implemented

The user evaluation confirmed that the StudyNest platform successfully meets its primary goal of fostering collaborative study among students. However, the feedback also highlighted valuable areas for refinement, which will guide the next development sprint. The continuous collection of user feedback will remain integral to ensuring that the app evolves in alignment with student needs.

5 Deployment & Integration

5.1 Deployment Platform Choice

Our deployment architecture utilizes three platforms: **Cloudflare** for the Angular frontend, **Render** for the NestJS backend, and **Supabase** for the PostgreSQL database. Cloudflare was selected for its global CDN capabilities and fast content delivery, while Render provides seamless Node.js application hosting with automatic deployments. Supabase offers a managed PostgreSQL instance, eliminating manual database provisioning. Database migrations are handled through the Supabase CLI, allowing consistent schema changes across environments.

5.2 Deployment Platform Review

The multi-platform approach proved effective, though each service exhibited distinct characteristics. Cloudflare's CDN provided excellent frontend performance with minimal latency. Render's backend deployment featured automatic wake-up from idle state, introducing initial response delays that required consideration in our architecture. The application connects to Supabase through secure connection strings stored in environment variables, preventing credential exposure. Access control is managed through Row-Level Security (RLS) policies and API keys, ensuring only authorized users can access sensitive data.

5.3 Integration Review

Integrating the three platforms was a valuable learning experience. Configuring CORS policies between Cloudflare and Render required careful attention to cross-origin requests. Understanding CDN caching behaviors and backend wake-up times helped optimize user experience. The separation of concerns across platforms enhanced security and scalability, with each service performing its specialized role effectively. Environment-based configuration management ensured smooth transitions between development and production environments.

6 General Discussion

6.1 Challenges & Future Development & Project Evaluation

Working with the Kanban methodology provided valuable flexibility throughout the development process and enhanced the conducive and fun working environment. Learning TypeScript proved particularly rewarding, as its strong typing system significantly enhanced code quality and caught potential errors during development. This project feels secure and super robust. For our testing, we used **Jest** on the backend and **Karma** on the frontend. Jest worked really well with NestJS because it was easy to set up, ran fast, and made mocking backend services simple. It helped us make sure the API and database logic behaved as expected without needing real Supabase calls.

On the frontend, Karma (with Jasmine) was a good fit for Angular. It let us test components, UI behaviour, and data binding early, which helped catch bugs before they got too deep.

We did face a few challenges though. Testing async backend functions that talked to Supabase was tricky at first — we had to mock the calls properly using Jest. Karma also gave us trouble in our CI setup until we switched to a headless Chrome setup. Once those were fixed, our tests ran smoothly across both environments.

Overall, the mix of Jest and Karma gave us good confidence that both the frontend and backend were stable and ready for deployment.

7 Conclusion

In conclusion, the Study Nest project successfully delivers a comprehensive platform built on modern web technologies and sound architectural principles. The database design provides a critical and scalable foundation, ensuring data integrity through well-defined relationships and constraints while enabling efficient queries that support collaborative features. The project demonstrates effective integration of frontend, backend, and database layers, with security considerations embedded throughout the architecture.

8 Supporting Documents

All supporting documentation for the project, including architecture, API endpoints, and database schema, can be found in the main documentation file. It is hosted in the project's GitHub repository at the following URL:

https://github.com/Sesihle-Goniwe/studynest/blob/main/supportdocs.pdf