

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	4
2	Methodology & Planning	4
2.1	Choice of Development Methodology	4
2.2	Following of Development Methodology	4
2.3	Review of Development Methodology	4
2.4	Project Design Project Planning	5
2.4.1	Project Design Approach	5
2.4.2	Design Principles	5
2.4.3	Project Planning Methodology	5
2.4.4	Development Timeline	6
2.4.5	Risk Management	6
2.5	Stakeholder Interaction	6
2.5.1	User-Centric Design Process	6
2.5.2	Communication Strategy	7
2.5.3	Stakeholder Roles and Responsibilities	7

2.5.4	Stakeholder Feedback Summary	7
3	Architecture	8
3.1	API Documentation	8
3.2	Database Documentation	11
3.3	Developer Guides Tech Stack	20
3.3.1	Technology Stack Overview	20
3.3.2	Frontend Development Guide	21
3.3.3	Backend Development Guide	22
3.3.4	Database Development Guide	22
3.3.5	Development Workflow	23
3.3.6	Deployment Guide	24
4	External APIs and Third-Party Integrations	25
4.1	External API Integration	25
4.1.1	Clubs Connect API	25
5	Testing	29
5.1	Automated Testing Procedure	29
6	Testing	29
6.1	User Feedback Procedure	33
6.2	User Feedback Analysis and Review	34
7	Deployment & Integration	37
7.1	Deployment Platform Choice	37
7.2	Deployment Platform Review	37
7.3	Integration Review	37
8	General Discussion	37

8.1 Challenges & Future Development & Project Evaluation	37
9 Conclusion	37
10 Supporting Documents	38

1 Introduction

The **Study Nest** project is a platform designed to help students find study partners, join study groups, schedule collaborative sessions, and track progress and match with other students. The database plays a central role in ensuring that user information, groups, sessions, notifications, study notes, and study-related activities are stored and retrieved efficiently.

The database is responsible for:

- Managing user accounts, authentication, and roles (students, group admins).
- Storing study material about study groups and their members.
- Keeping track of scheduled study sessions.
- Supporting real-time features such as notifications.
- Enabling future scalability for analytics.

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

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

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

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

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 [+ 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










	study_preferences	No description	0	16 kB	×	6 columns 	
	study_sessions	No description	0	8192 bytes	×	7 columns 	
	topics	No description	6	32 kB	×	7 columns 	

Figure 3: Database tables















Tables with fields

	Primary key		Identity		Unique		Nullable		Non-Nullable
---	-------------	---	----------	---	--------	---	----------	--	--------------

Figure 4: study logs table

This is the description of the tables(primary,foreign unique) keys

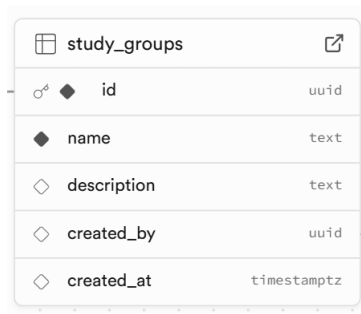
Study groups Notifications

	students	
	 user_id	uuid
	 email	text
	 university	text
	 course	text
	 year	int4
	 created_at	timestampz
	 updated_at	timestampz
	 skills	text
	 studyPreference	text
	 profileImage	text
	 full_name	text

auth.users.id

Figure 5: Student table

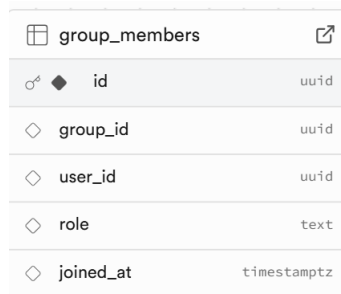
This table(student) stores the information of a student with the fields they are required when they login the first time. It uses the auth table to uniquely identity the user. This acts as a profile page



study_groups		
id	uuid	
name	text	
description	text	
created_by	uuid	
created_at	timestampz	

Figure 6: study groups table

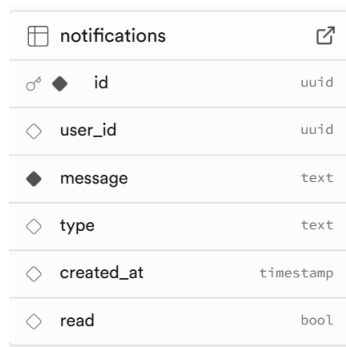
The table provides all the study groups users create with the details of the groups and the user who created the group for unique identification.



group_members		
id	uuid	
group_id	uuid	
user_id	uuid	
role	text	
joined_at	timestampz	

Figure 7: group members table

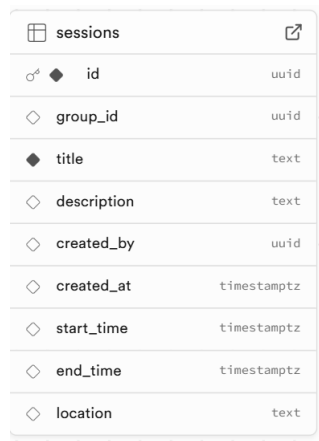
After creation of the study group, the user who created a group(admin) gets stored in the group members table together with the users who are going to join the same group. This uses group id to categories which members are in which group



notifications	
id	uuid
user_id	uuid
message	text
type	text
created_at	timestamp
read	bool

Figure 8: notification table

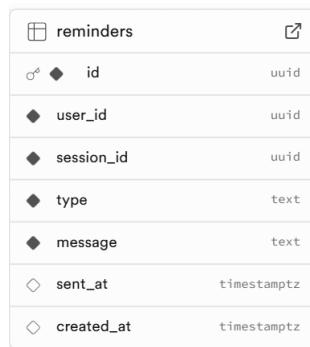
Each time a member join a group, the group admin gets an email notification about this event and it gets stored in this table. Also when a session is created within a group, all the group members within the group gets the notification and the sessions are stored in the session table below.



sessions	
id	uuid
group_id	uuid
title	text
description	text
created_by	uuid
created_at	timestampz
start_time	timestampz
end_time	timestampz
location	text

Figure 9: sessions table

After a session has been created, all members are sent regular reminder notifications in interval of 1h or 24 hours before the session which are stored in the reminder table.



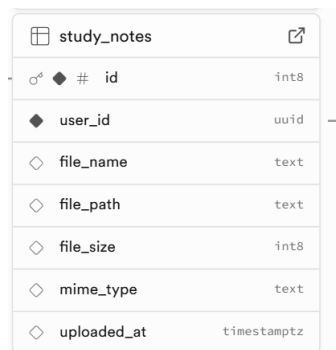
The diagram shows a table named 'reminders' with a grid icon and a link icon. It contains seven rows of fields. The first row has a primary key icon (a circle with a dot) and a diamond icon next to the field 'id', which has the data type 'uuid'. The next three rows have diamond icons next to the fields 'user_id', 'session_id', and 'type', all with the data type 'uuid'. The last two rows have diamond icons next to the fields 'message' and 'sent_at', both with the data type 'text'. The final row has a diamond icon next to the field 'created_at', which has the data type 'timestampz'.

reminders	
id	uuid
user_id	uuid
session_id	uuid
type	text
message	text
sent_at	timestampz
created_at	timestampz

Figure 10: reminder table

This table stores all the created notifications and keep track of all the send/unsent notifications

User Progress

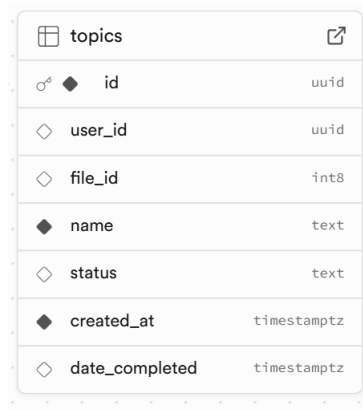


The diagram shows a table named 'study_notes' with a grid icon and a link icon. It contains seven rows of fields. The first row has a primary key icon (a circle with a dot) and a diamond icon next to the field 'id', which has the data type 'int8'. The next row has a diamond icon next to the field 'user_id', which has the data type 'uuid'. The next three rows have diamond icons next to the fields 'file_name', 'file_path', and 'file_size', all with the data type 'text'. The final row has a diamond icon next to the field 'mime_type', which has the data type 'text'. The last row has a diamond icon next to the field 'uploaded_at', which has the data type 'timestampz'.

study_notes	
# id	int8
user_id	uuid
file_name	text
file_path	text
file_size	int8
mime_type	text
uploaded_at	timestampz

Figure 11: study notes table

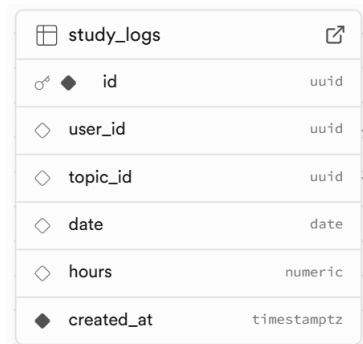
This provides a reference to the files which a user uploads in the provided section in order to add/share study notes



topics	
id	uuid
user_id	uuid
file_id	int8
name	text
status	text
created_at	timestampz
date_completed	timestampz

Figure 12: study topics table

After notes have been uploaded a topic can be created in which all user can interact and study all together.

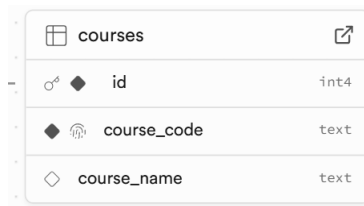


study_logs	
id	uuid
user_id	uuid
topic_id	uuid
date	date
hours	numeric
created_at	timestampz

Figure 13: study logs table

To allow effective user tracking the study logs allow a user to log hours to a certain topic.

Partner matching



The image shows a table schema for a table named 'courses'. The table has three columns: 'id' of type 'int4', 'course_code' of type 'text', and 'course_name' of type 'text'. The 'id' column is marked as the primary key with a diamond icon. The 'course_code' column is marked with a diamond and a globe icon, indicating it might be a foreign key or have a specific constraint. The 'course_name' column is marked with a diamond icon. The table name 'courses' is at the top, and there is a small icon in the top right corner of the table header.

courses		
id	int4	
course_code	text	
course_name	text	

Figure 14: study logs table

Users have courses which a certain universities offers, This tables allow for partner matching based of similar courses.

ERD Diagrams

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

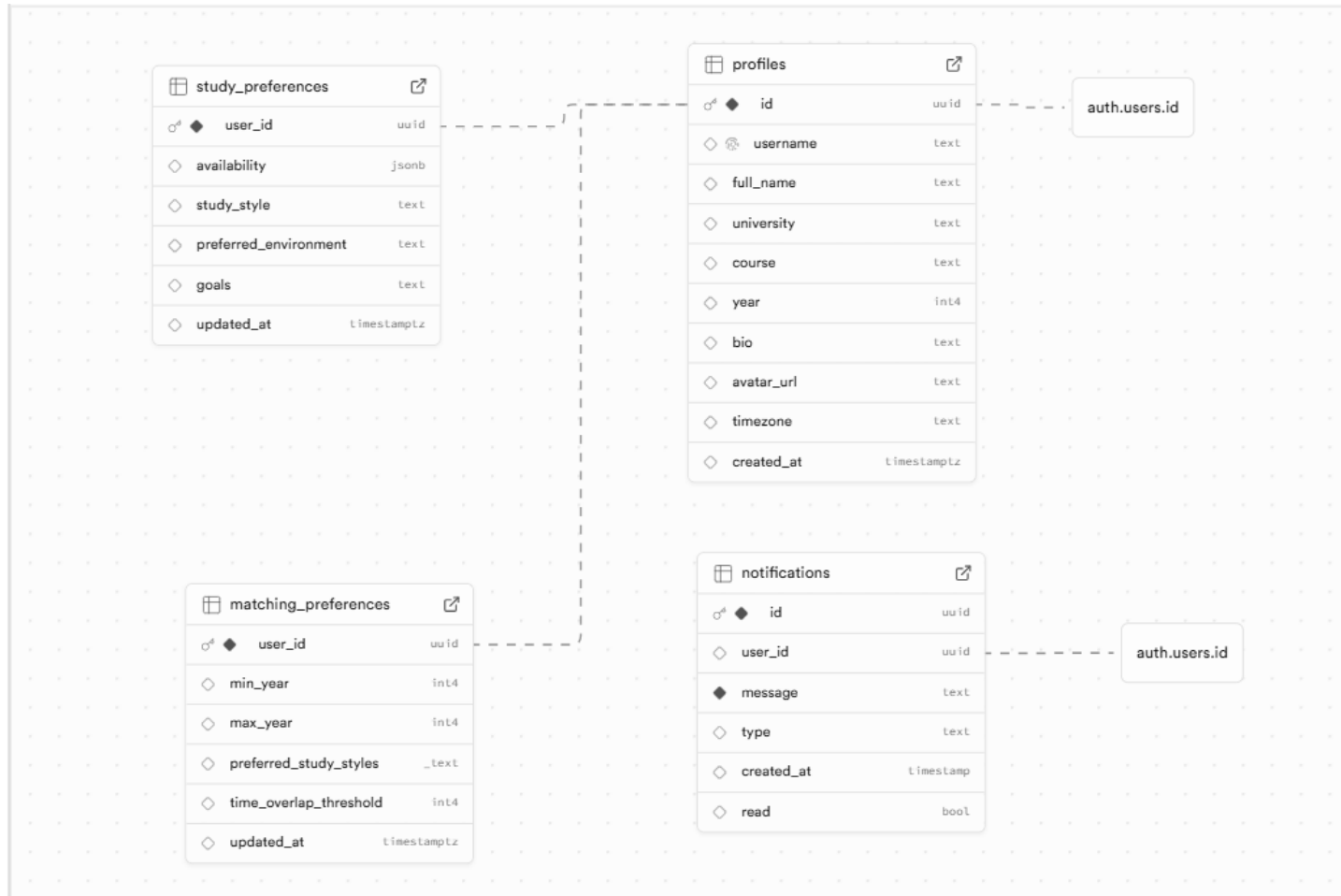


Figure 15: Initial Part of the ERD

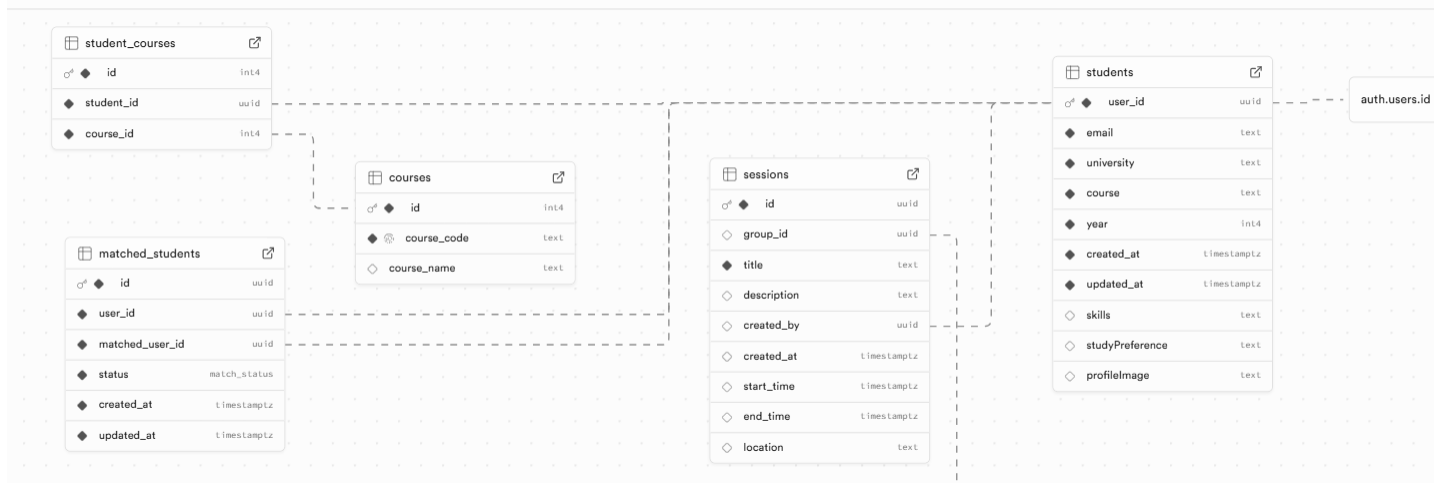


Figure 16: Middle Part of the ERD

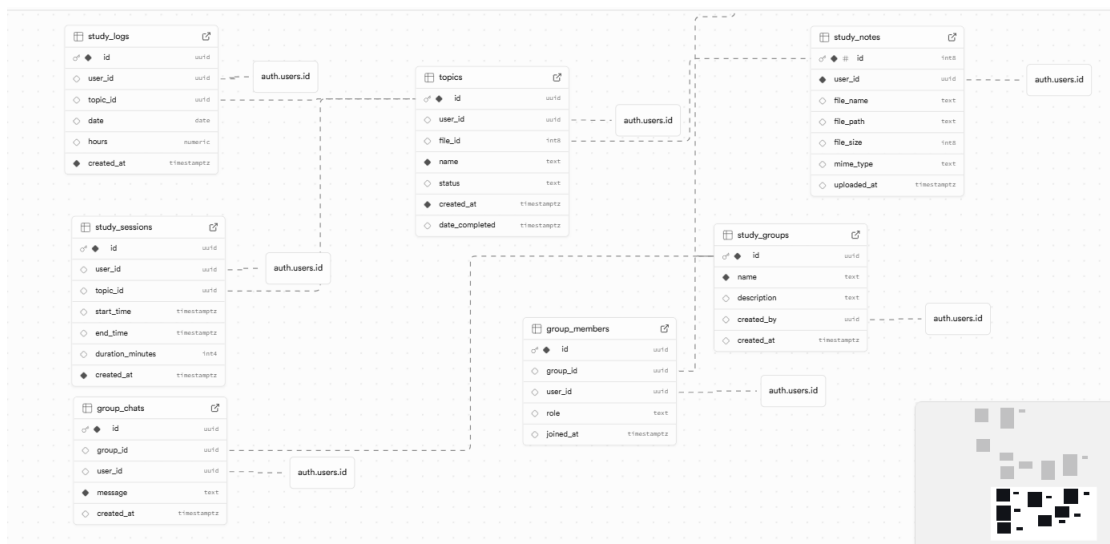


Figure 17: Last Part of the ERD

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

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

4 # Install dependencies
5 npm install

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

10 # Start development server
11 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

```
1 src/
2     app/
3         components/      # Reusable UI components
4         services/        # Data and business logic services
5         pages/           # Feature pages and routes
6         models/          # TypeScript interfaces
7         guards/          # Route protection
8     assets/              # Static resources
9     environments/        # Configuration files
```

3.3.3 Backend Development Guide

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

4 # Install dependencies
5 npm install

7 # Setup environment variables
8 cp .env.example .env

10 # Run database migrations
11 npx supabase db push

13 # Start development server
14 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

```
1 src/
2     auth/                # Authentication logic
3     users/               # User management
4     groups/              # Study groups feature
5     progress/            # Progress tracking
6     notifications/       # Notification system
7     common/              # Shared utilities and decorators
8     main.ts              # Application entry point
```

3.3.4 Database Development Guide

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

4 # Apply migrations
5 npx supabase db push

7 # Reset database (development)
8 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
- **hotfix/*:** Critical production fixes

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

```
1 # Build production version
2 npm run build

4 # Deploy to Cloudflare Pages
5 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:

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

5 # Backend (.env)
6 DATABASE_URL=postgresql://...
7 JWT_SECRET=your_jwt_secret
8 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

```
1 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

5 Testing

5.1 Automated Testing Procedure

6 Testing

This document 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:

Run frontend tests with coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	71.51	46.73	66.81	71.73	
app	100	100	100	100	
app.ts	100	100	100	100	
app/features/about-us	100	100	100	100	
about-us.html	100	100	100	100	
about-us.ts	100	100	100	100	
app/features/auth	54.67	29.78	45.83	56.25	
auth.guard.ts	50	0	0	45.45	8-17
auth.service.ts	67.03	35.89	78.57	71.95	25-28,64-75,92,96-98,109-120,136-137
callback.component.ts	25	0	0	22.85	37-95
app/features/group-session	84.48	59.09	72.22	83.92	
group-session.html	100	100	100	100	
group-session.ts	84.21	59.09	72.22	83.63	48,67,82,92,100,122,128,136-145
app/features/home	75	100	33.33	71.42	
home.html	100	100	100	100	
home.ts	71.42	100	33.33	66.66	15-18
app/features/log-hours-dialog	100	50	100	100	
log-hours-dialog.html	100	100	100	100	
log-hours-dialog.ts	100	50	100	100	47
app/features/matches	92.77	100	78.57	92.4	
matches.html	100	100	100	100	
matches.ts	92.68	100	78.57	92.3	49,66,83,97,111,131

Figure 18: Frontend Code Coverage

NestJS Backend Tests
failed 3 hours ago in 35s

Search logs Explain error

Run backend tests with coverage 7s

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	64.14	48.5	75.4	63.27	
src	100	75	100	100	
app.controller.ts	100	75	100	100	6
app.service.ts	100	100	100	100	
src/chats	19.14	14.1	0	18.18	
chats.controller.ts	33.33	32	0	31.42	9-10,15,20-25,35-50,59-63,73-75,83-85
chats.service.ts	9.09	5.66	0	7.14	6-155
src/files	25	17.64	50	21.42	
files.controller.ts	100	75	100	100	20
files.service.ts	9.58	10	0	7.04	16-187
src/groups	72.58	57.89	50	70.68	
groups.controller.ts	63.15	66.66	0	58.82	14,18,23,28,41,46,54
groups.service.ts	76.74	55.17	100	75.6	19,31,49,61,74,94-98,109,126
src/groups/dto/create-group.dto	100	100	100	100	
create-group.dto.ts	100	100	100	100	
src/mailler	47.72	21.42	80	45.23	
mailler.service.ts	47.72	21.42	80	45.23	68-69,75-123
src/notifications	75.51	57.69	90.9	73.33	
notifications.controller.ts	100	75	100	100	14
notifications.service.ts	65.71	54.54	83.33	63.63	19,35,47,58-75,88,96-97,106
src/progress	92.59	81.57	100	92.1	
progress.controller.ts	100	75	100	100	25-43
progress.service.ts	80.47	84.63	100	88.88	55-56,74-75,100,101

Figure 19: Backend Code Coverage

6.1 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

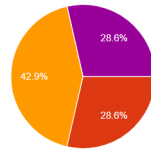
The aim of this feedback procedure was to assess user satisfaction, identify technical issues, and prioritize improvements for the next iteration of the application. This ensured that the StudyNest app aligns with the real needs of students and enhances collaborative learning efficiency.

6.2 User Feedback Analysis and Review

How often do you use StudyNest?

[Copy chart](#)

7 responses

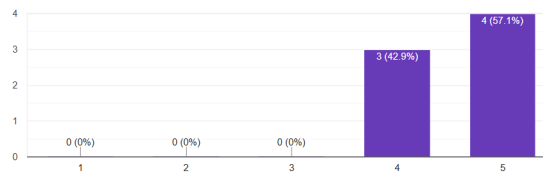


● Daily
● Several times a week
● Once a week
● Rarely
● I just started using it-How often do you use StudyNest?

How easy is it to navigate the app?

[Copy chart](#)

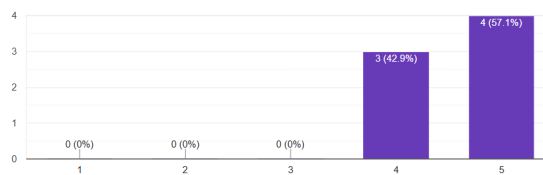
7 responses



How intuitive is creating and joining study groups?

[Copy chart](#)

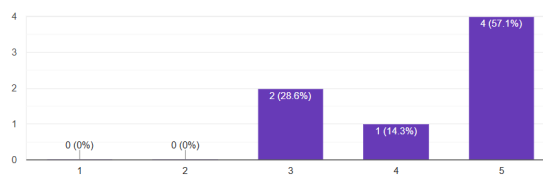
7 responses

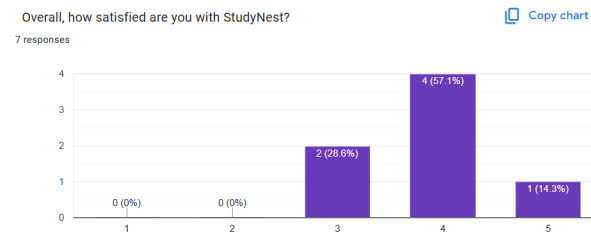
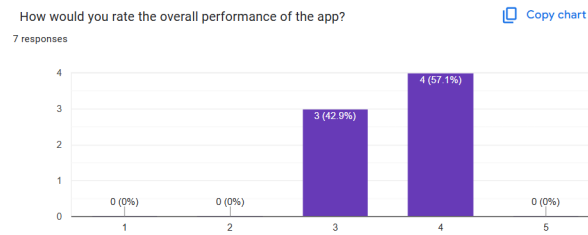


How helpful is the chat and notes sharing feature?

[Copy chart](#)

7 responses





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

What features would you like to see added?

7 responses

Automatic partner matching and chat system with lectures.

Automatic partner matching

A map location thing to show where a session is

Map Tracker: to maybe see the location of your group

Sending images of notes over chat

Chat system with lecturers

AI tools

Figure 20: 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.

7 Deployment & Integration

7.1 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.

7.2 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.

7.3 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.

8 General Discussion

8.1 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. - W.N

9 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.

10 Supporting Documents