

INTEGRATED PROJECT REPORT

On

Dev Insights

Submitted in partial fulfilment of the requirement for the
Course Integrated Project (CS 203) of

COMPUTER SCIENCE AND ENGINEERING
B.E. Batch-2022 (VI Semester)



Submitted by :

Under the Guidance of :
Dr. Lekha Rani

Name: Sanskriti
Roll. No. : 2210992252
Name: Sanya
Roll. No. : 2210992255
Name: Ishaan Singla
Roll. No. : 2210992582
Name: Nutan
Roll. No. : 2210992005

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHITKARA UNIVERSITY
PUNJAB

(Annexure–C)

CERTIFICATE

This is to be certified that the project entitled “Dev Insights” has been submitted for the Bachelor of Computer Science Engineering at Chitkara University, Punjab during the academic semester January 2025-

May-2025 is a bonafide piece of project work carried out by “Sanskriti (2210992252), Sanya (2210992255), Ishaan Singla (2210992582), Nutan (2210992005)” towards the partial fulfillment for the award of the course Integrated Project (CS 203) under the guidance of “Dr. Lekha Rani” and supervision.

Sign. of Project Guide

Dev Insights

Dr. Lekha Rani



(Annexure–D)

CANDIDATE’SDECLARATI

ON

We, “Sanskriti (2210992252), Sanya (2210992255), Ishaan Singla (2210992582), Nutan (2210992005)”, B.E.-2022 of the Chitkara University, Punjab hereby declare that the Integrated Project Report entitled “Dev Insights” is an original work and data provided in the study is authentic to the best of our knowledge. This report has not been submitted to any other Institute for the award of any other course.

Sign.of Student1

Sanskriti

ID No. 2210992252

Sign.of Student2

Sanya

ID No. 2210992255

Sign.of Student3

Ishaan Singla

ID No. 2210992582

Sign.of Student4

Nutan

ID No. 2210992005

Place: Rajpura

Date: 3-03-2025

(Annexure-E)

ACKNOWLEDGEMENT

It is our pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behavior and acts during the course of study.

We express our sincere gratitude to all for providing mean opportunity to undergo Integrated Project as the part of the curriculum.

We are thankful to “Project Guide Name” for his support, cooperation, and motivation provided to us during the training for constant inspiration, presence and blessings.

We also extend our sincere appreciation to **Dr. Lekha Rani** Who provided his valuable suggestions and precious time in accomplishing our Integrated project report.

Lastly, we would like to thank the almighty and our parents for their moral support and friends with whom we share dour day-to day experience and received lots of suggestions that improve our quality of work.

Sanskriti
ID No. 2210992252

Sanya
ID No. 2210992255

Ishaan Singla
ID No. 2210992582

Nutan
ID No. 2210992005

Table of Content

Sr No.	Title	Page No.
1	Abstract/Keywords	1
2	Introduction to the project	2
3	Software and Hardware Requirement Specification	4
4	Database Analyzing, design and implementation	7
5	Program's Structure Analyzing and GUI Constructing	9
6	Code-Implementation and Database Connections	15
7	System Testing	25
8	Limitations	27
9	Conclusion	28
10	Future Scope	28
11	Bibliography/References	29

1. Abstract/Keywords

DevInsights is an AI-powered web application designed to help developers efficiently manage and understand code changes in large repositories. By leveraging AI-generated commit summaries, natural language-based code search, and meeting summaries, DevInsights streamlines the development process, saving time and improving productivity. The application also stores and summarizes meeting recordings, ensuring that all team discussions are easily accessible and actionable.

In today's fast-paced software development ecosystem, engineers work with increasingly large and complex codebases that require constant updates, collaboration, and tracking. Traditional development workflows often fall short when it comes to understanding historical changes, reviewing commits, and documenting meetings efficiently. DevInsights is an AI-powered web application developed to address these challenges by introducing automation, intelligence, and enhanced user interaction into the development process.

DevInsights provides developers with the ability to automatically generate summaries for code commits, conduct natural language searches across repositories, and summarize technical meetings. By integrating state-of-the-art technologies such as Google Gemini AI for code-related natural language processing and Assembly AI for speech-to-text transcription, the platform ensures a seamless bridge between code and communication.

The core of DevInsights lies in its ability to simplify understanding — whether it's about recent code changes, finding specific logic within a repository, or reviewing past discussions from meetings. Users can interact with the system through a clean and intuitive frontend built using Next.js and ShadCN UI, while the backend leverages Bun, Clerk, and LangChain to ensure scalable, secure, and efficient performance. The application is deployed via Vercel and uses Neon PostgreSQL and Supabase for data storage and management.

With AI taking over tedious and repetitive tasks like summarization, indexing, and search, DevInsights improves team collaboration, boosts productivity, and accelerates onboarding for new developers. The system also supports multi-user access to shared projects, enabling collaborative development and centralized project tracking.

This report delves into the architectural structure, database design, implementation logic, GUI workflow, and real-world use cases of DevInsights, and further proposes enhancements that could make it even more powerful and developer-centric.

Keywords:

AI-Powered Code Insights, Commit Summarization, Natural Language Code Search, Repository Management, Next.js, Google Gemini AI, GitHub API, Clerk Authentication, Neon PostgreSQL, Bun, PostgreSQL.

2. Introduction to the Project

In today's fast-paced development environment, keeping track of code changes and maintaining clear documentation can be challenging. DevInsights is a full-stack Next.js application designed to simplify project management by leveraging AI to generate insightful summaries of GitHub repositories.

With DevInsights, users can upload their GitHub repositories and receive AI-generated commit summaries, making it easier to understand recent changes and track progress. Additionally, the platform allows users to ask questions about their project's code and files, streamlining code comprehension and collaboration.

Beyond code analysis, DevInsights also offers AI-powered meeting transcription and summarization. Users can upload audio recordings of team meetings, and the system will generate structured summaries, ensuring that key discussions and decisions are well-documented.

2.1 Background

With the rapid growth of software development, managing repositories efficiently has become crucial. Developers often rely on commit messages and manual code reviews, which can be time-consuming and error-prone. Existing tools offer limited AI-driven insights. DevInsights leverages Next.js, AI models like Google Gemini, and GitHub API to streamline repository management and enhance productivity.

The modern software development lifecycle has evolved to become highly collaborative, fast-paced, and data-driven. With the widespread adoption of GitHub and similar platforms, teams now maintain massive codebases where hundreds or even thousands of commits occur across multiple branches. In such environments, keeping track of changes, understanding historical development progress, and ensuring efficient onboarding of new developers has become increasingly difficult.

Traditional code review processes, such as manual browsing of commit histories and reading verbose commit messages, are often inefficient and time-consuming.

2.2 Problem Statement

The major issues in the current market include:

- **Lack of intelligent insights:** Existing tools do not provide AI-powered commit summarization.
- **Complex repository navigation:** Developers struggle to locate specific code changes.
- **Inefficient onboarding:** New team members face difficulties understanding project history.
- **Manual Commit Review Is Time-Consuming:** Developers often spend hours reading through diffs or logs to understand project progress.
- **Loss of Information from Verbal Discussion:** Important technical decisions or issues discussed during meetings often go undocumented.
- **Difficult Onboarding Process for New Developers:** Without a high-level overview of a project's structure and progress, newcomers require significant time to get productive.

2.3 Scope of the Project

DevInsights focuses on enhancing developer workflows with the help of AI and seamless integrations:

- **Supported Actions:** Users can upload repositories, view commit summaries, search code using questions, and upload audio for transcription.
- **Target Users:** Software development teams, open-source contributors, and freelance developers.
- **Technology Focus:** Combines the best of frontend (React/Next.js), backend (Bun, LangChain), and AI APIs (Gemini, Assembly).
- **Deployment:** Cloud-based, scalable, and accessible via browser without needing local installations.

2.4 Objectives

The main objectives of DevInsights include:

- Automating commit summarization using Google Gemini AI.
- Enabling semantic code search through natural language queries.
- Generating meeting summaries from uploaded audio using Assembly AI.
- Offering a clean, responsive, and secure frontend built with Next.js and Clerk authentication.
- Supporting multi-user projects and team-based collaboration.

3. Software and Hardware Requirement Specification

3.1 Methods

DevInsights was developed using the **Agile Software Development Life Cycle (SDLC)** model. Agile promotes flexibility, continuous feedback, and rapid iterative improvements—qualities essential for building an application that heavily depends on real-time integration with APIs, evolving AI capabilities, and fast deployment environments.

Each feature in DevInsights was developed in short sprints that followed the sequence:

1. Requirement gathering and user story creation
2. Component-level planning and UI wireframing
3. Development and integration with backend/API
4. Continuous testing and deployment via Vercel
5. User feedback and iteration

Agile allowed the team to ship features quickly and adapt as third-party APIs (such as Gemini AI or GitHub API) changed or introduced updates.

3.2 Programming/Working Environment

The Dev Insights is developed using:

- a) **Bun** -- Fast JavaScript runtime & package manager
- b) **Next.js** -- React framework for Server Side rendering & static sites
- c) **TypeScript** -- Typed superset of JavaScript for safer code
- d) **Prisma** – Database Management
- e) **Gemini API (v2.0-flash)** -- Google’s AI model for NLP tasks (*upgraded from v1.5-flash*)
- f) **Assembly API**– For generating Meeting Summary
- g) **GitHub API** -- Interface for repo & data access
- h) **LangChain** – Framework for AI-driven applications
- i) **TailwindCSS** – Utility-first CSS framework for styling
- j) **ShadCN** – Prebuilt UI components for React
- k) **SupaBase**—For Storing Files of the meetings
- l) **Stripe API** – For handling payments and subscription management

3.3 Software Requirements to Run the Application

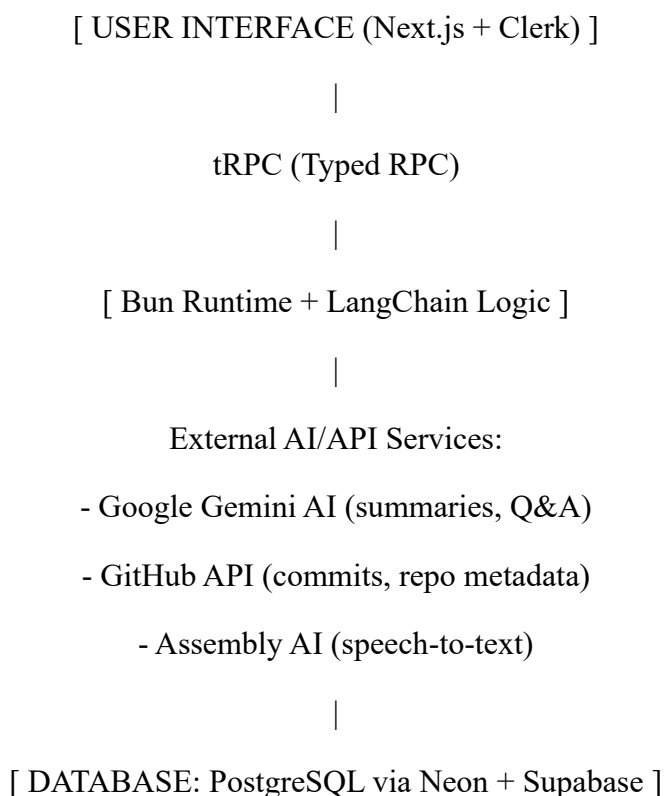
- | a) Component | Minimum Specification |
|------------------------|--|
| b) Operating System | Windows 10 / Linux (Ubuntu) / macOS Catalina or higher |
| c) Code Editor | Visual Studio Code or any IDE with JS support |
| d) Runtime Environment | Node.js (v18+), Bun (v1.0+) |
| e) Browser | Google Chrome / Firefox / Microsoft Edge |
| f) Version Control | Git |
| g) API Access | GitHub Developer Token, Gemini API Key, Clerk Keys |
| h) Database | PostgreSQL instance via Neon Console or Supabase |

3.4 Hardware Requirements to Run the Application

Specification	Minimum	Recommended
Processor	Intel Core i5	Intel Core i7 or Apple M1
RAM	4 GB	8 GB or more
Storage	20 GB free disk space	50 GB SSD
Screen Resolution	1024 x 768	1920 x 1080 (Full HD)
Internet Connection	Required	High-speed (5+ Mbps)

3.5 Architecture Overview

DevInsights uses a modular and scalable architecture comprising multiple independent layers. This separation allows the frontend, backend, and AI logic to evolve independently.



- **Frontend:** Built with React components and ShadCN UI for quick design.

- **Backend:** Bun runtime with fast server responses, interacting with Clerk for auth and AI APIs for processing.
- **Database:** Cloud-hosted, real-time enabled with Supabase support.
- **Deployment:** Vercel automates builds from GitHub commits, ensuring zero-downtime updates.

4. Database Analyzing, Design, and Implementation

4.1 Database Overview

The backbone of DevInsights lies in a well-structured relational database. We chose PostgreSQL, hosted on Neon Console, as the core data storage system. PostgreSQL offers rich SQL support, ACID compliance, and scalability features essential for a real-time, multi-user web application. Integration with Supabase allows real-time updates, file storage for meeting recordings, and simplified client-server communication.

The database is designed to efficiently store:

- User credentials and authentication details
- GitHub repository metadata
- Commit data and AI-generated summaries
- Meeting recordings and summaries
- Semantic code embeddings and search mappings
- AI-generated answers to user questions

4.2 Data Design

The DevInsights database is designed to efficiently manage user data, project details, AI-generated insights, and collaborative features. Below is an overview of the key database entities:

- **User :** Stores user-related data, including authentication details and profile information.
- **Project :** Stores details of projects uploaded by users, including repository metadata and associated files.

- **UserToProject** : Establishes a many-to-many relationship between users and projects, allowing multiple users to collaborate on a shared project.
- **SourceCodeEmbedding** : Stores AI-generated embeddings of the uploaded source code.
- **Meeting** : Stores details of meetings uploaded by users, including audio files and metadata.
- **Issue** : Records key discussions from meetings along with timestamps, enabling users to track important topics.
- **Question** : Stores user-submitted questions about the project. Contains AI-generated responses based on the source code embeddings.
- **Commit** : Stores commits fetched from GitHub repositories. Includes AI-generated summaries for each commit to help users quickly understand code changes.

4.3 Database Design Principles

Our design goals included:

- Scalability: Capable of handling growth in users, projects, and commits.
- Security: Storing only non-sensitive metadata while relying on Clerk for authentication.
- Redundancy Elimination: Normalized design using 3NF principles.
- Query Performance: Indexed primary keys and foreign key relationships for fast retrieval.

4.4 Normalization

All database tables are normalized to Third Normal Form (3NF):

- 1NF: Atomic values; each field contains indivisible data.
- 2NF: No partial dependencies; all non-key attributes depend on the entire key.
- 3NF: No transitive dependencies; attributes depend only on primary keys.

This ensures:

- Reduced data redundancy
- Easier updates and maintenance
- Better scalability and referential integrity

4.5 Data Flow Summary

- Commits are fetched via the GitHub API and stored in the Commits table with AI-generated summaries.
- Uploaded meeting recordings are transcribed and summarized, then stored in the Meetings and Issues tables.
- Developers can ask questions about a codebase, which are recorded in the Questions table with AI responses.

5. Program's Structure Analyzing and GUI Constructing

5.1 High-Level Design:

- a) Flowchart & Use Case Diagram: Shows user interactions with the system.
- b) Architecture Diagram: Explains system components (frontend, backend, database).



Account


Manage your account info.

Profile


Security

Profile details

Profile



 Shunsuke [Update profile](#)

Email addresses


 Primary

+ Add email address

Connected accounts

 Google 

+ Connect account

Secured by  clerk

DevInsights

Application

Dashboard

Q&A

Meetings

Your Projects

Dev

food app

Loan prediction23

Loan prediction24

+ Create Project

This project is linked to <https://github.com/Ishaan282/Dev-Insights>

Ask a question

ask me anything about the project!

Ask AI

Create a new meeting

Analyse your meeting with our Tool.
Powered by AI.

Upload Meeting

Ishaan282 committed

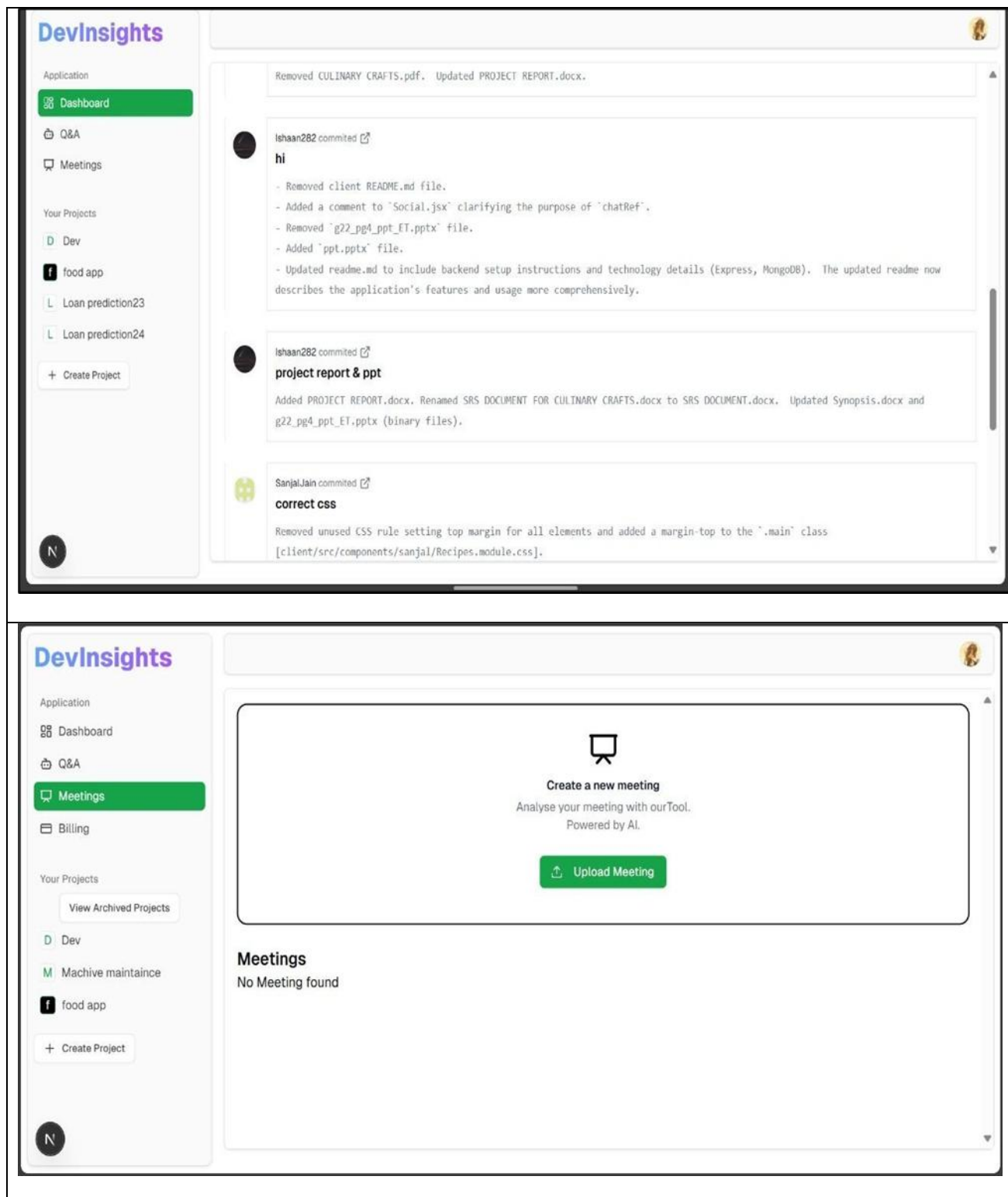
applied supabase

Added Supabase integration for file uploads [src/lib/firebase.ts], [src/lib/testsupabase.ts], added '@supabase/supabase-js' package [package.json], 'bun.lockb' was also modified. The Firebase file upload function was replaced with a Supabase-based implementation.

Ishaan282 committed

added archiving project button

Page | 11





DevInsights

Application

- Dashboard
- Q&A
- Meetings
- Billing**

Your Projects

View Archived Projects

- Dev
- Machive maintainece
- food app

+ Create Project

Billing

You currently have 620 credits.

Each credit allows you to index 1 file in a repository
E.g. If your project has 100 files, you will need 100 credits to index it.

Buy 370 credits for Rs370.00

← Dev-insights sandbox

370 code credits

₹370.00

Pay with link

Or

Email

Card information

1234 1234 1234 1234

MM / YY CVC

Cardholder name

Full name on card

Country or region

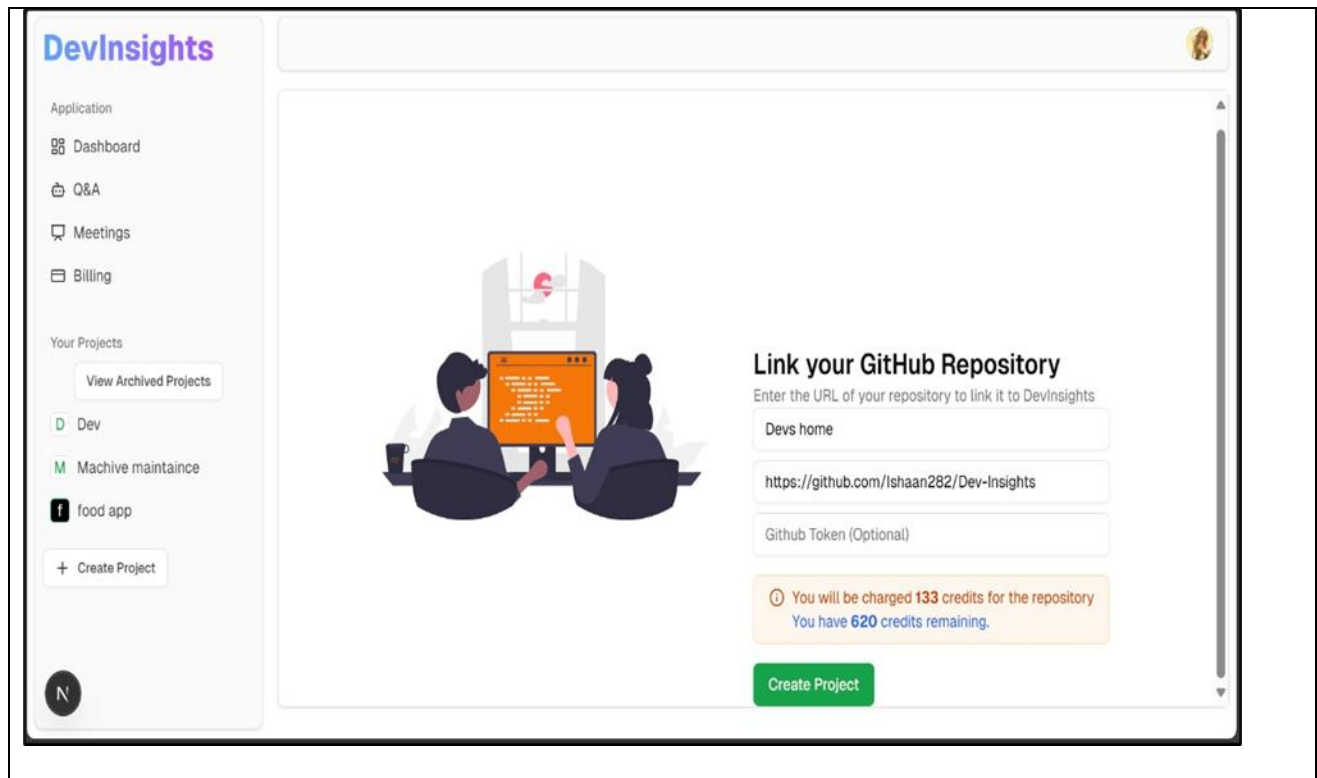
India

☐ Securely save my information for 1-click checkout
Pay faster on Dev-insights sandbox and everywhere Link is accepted.

Pay

Powered by stripe

Terms Privacy



6. Code Implementation and Database Connections

6.1 Frontend Code Implementation

The frontend of **DevInsights** is developed using **Next.js**, a robust React framework that supports both **server-side rendering (SSR)** and **static site generation (SSG)**. This hybrid approach ensures fast load times, SEO optimization, and a seamless user experience. The UI is styled using **Tailwind CSS**, a utility-first CSS framework, and enhanced with **ShadCN UI**, which provides accessible and customizable component libraries.

Core Technologies Used

- Next.js: For routing, SSR, and API integration.
- Tailwind CSS: For responsive and utility-based styling.
- ShadCN UI: For prebuilt, accessible UI components.
- tRPC: For type-safe API communication between frontend and backend.

Key frontend components and features

1. Dashboard

- a. The dashboard is the central hub of the application. It displays:
 - i. A list of all commits fetched from the linked GitHub repository.
 - ii. AI-generated summaries of each commit.
 - iii. Metadata such as the author's name, commit message, and timestamp.
 - iv. A clean, card-based layout for easy readability.
- b. This allows developers to quickly understand what changes were made, by whom, and when—without diving into raw diffs.

2. Q&A interface

- a. This section allows users to interact with the AI using **natural language queries**.
Users can:
 - i. Ask questions like “What does the authentication module do?”
 - ii. Receive answers in Markdown format, making them easy to read and copy.
 - iii. View references to specific files or functions in the codebase.

- b. This feature is powered by **LangChain** and **Google Gemini AI**, which interpret the question and search the embedded source code for relevant answers.
- 3. Repository Upload Form
 - a. Users can link their GitHub repositories by:
 - i. Entering the repository URL.
 - ii. Assigning a custom name to the project.
 - iii. Triggering a backend process that fetches commit history and metadata.
 - b. This form is validated in real-time and provides feedback on successful or failed uploads.
- 4. Meeting Upload Interface
 - a. Users can upload **audio recordings** of team meetings. The system:
 - i. Sends the audio to Assembly AI for transcription.
 - ii. Stores the summary and timestamps in the database.
 - iii. Displays the summary in a structured format with clickable time markers.
 - b. This ensures that important discussions and decisions are documented and accessible.
- 5. Credit system
 - a. DevInsights uses a **credit-based system**:
 - i. Each credit allows the user to upload and process one file.
 - ii. Users can purchase credits via Stripe integration.
 - iii. The current credit balance is shown on the dashboard.
 - b. This system ensures fair usage and supports monetization.
- 6. **Collaboration tools**
 - a. Users can **collaborate** with others by:
 - i. Generating an invite link to share with teammates.
 - ii. Allowing invited users to access the same project, view summaries, and ask questions.
 - iii. Managing collaborators through a simple UI.
 - b. This feature supports team-based workflows and enhances productivity.
- 7. **Component architecture**
 - a. Each UI feature is built as a **modular React component**, following best practices such as:

- i. Separation of concerns: Logic, styling, and rendering are separated.
 - ii. Reusability: Components like buttons, cards, and modals are reused across pages.
 - iii. Accessibility: All components follow ARIA guidelines for screen readers.
8. API integration with tRPC
 - a. All frontend interactions with the backend are handled using **tRPC**, which provides:
 - i. End-to-end type safety between client and server.
 - ii. Automatic inference of input/output types.
 - iii. Simplified API calls without needing REST or GraphQL boilerplate.
 - b. This ensures a smooth developer experience and reduces runtime errors.

6.2 Backend Code implementation

The backend of **DevInsights** is designed to be **modular, scalable, and efficient**, handling everything from AI integrations to database operations. It is built using **Next.js API routes**, powered by the **Bun runtime** for high performance, and structured with **tRPC** for type-safe communication between the frontend and backend.

Core technologies used

- Next.js API Routes: For defining server-side logic.
- Bun: A fast JavaScript runtime used for executing backend code and improving performance.
- tRPC: Enables type-safe API calls without REST or GraphQL.
- Prisma ORM: For interacting with the PostgreSQL database.
- LangChain: For orchestrating AI workflows.
- Google Gemini AI: For generating commit summaries and answering code-related questions.
- Assembly AI: For transcribing and summarizing meeting recordings.
- GitHub API: For fetching repository data and commit history.

Backend Responsibilities

1. Commit Summary Generation

- a. When a user links a GitHub repository:

- i. The backend fetches the commit history using the GitHub API.
 - ii. Each commit message and diff is sent to Google Gemini AI.
 - iii. The AI returns a natural language summary of the commit.
 - iv. These summaries are stored in the Commit table in the database.
- b. This process is automated and runs in the background, ensuring that users always have up-to-date insights.

2. Natural Language Code Search

- a. Users can ask → *where is the login logic implemented*
- b. The backend handles this by:
 - i. Converting the question into an embedding using Gemini.
 - ii. Comparing it with precomputed source code embeddings stored in the SourceCodeEmbedding table.
 - iii. Returning the most relevant code snippets and a generated answer.
- c. This is powered by LangChain, which manages the prompt engineering and retrieval-augmented generation (RAG) pipeline.

3. Meeting Transcription and Summarization

- a. When a user uploads a meeting recording:
 - i. The file is stored in Supabase.
 - ii. The backend sends the audio to Assembly AI.
 - iii. The returned transcript and summary are stored in the Meeting and Issue tables.
 - iv. The summary includes timestamps, headlines, and key discussion points.
- b. This ensures that team discussions are documented and searchable.

4. User and Project Management

- a. The backend handles:
 - i. **User registration and authentication via Clerk.**
 - ii. Project creation and linking to GitHub repositories.
 - iii. Collaboration features, allowing users to invite others to their projects.
 - iv. Credit management, where each file upload deducts one credit from the user's balance.
- b. All user-related data is stored in the User, Project, and UserToProject tables.

5. Stripe Integration for Payments

- a. To support monetization:
 - i. The backend integrates with Stripe API.
 - ii. Users can purchase credits through a secure checkout flow.
 - iii. Transactions are recorded in the StripeTransaction table.
 - iv. Webhooks are used to update credit balances in real-time.

6. Code Structure and Modularity

- a. The backend is organized into the following layers:
 - i. Routes: Defined using Next.js API routes (e.g., /api/commit, /api/meeting).
 - ii. Controllers: Handle business logic (e.g., summarizing commits, processing audio).
 - iii. Services: Interact with external APIs (e.g., GitHub, Gemini, Assembly AI).
 - iv. Database Layer: Uses Prisma to perform CRUD operations on PostgreSQL.
 - v. Middleware: Handles authentication, error handling, and logging.
- b. This modular architecture ensures that the codebase is **maintainable**, **testable**, and **scalable**

7. Security and Authentication

- a. Security is a top priority in DevInsights. Key measures include:
 - i. Clerk Authentication: Ensures secure login and access control.
 - ii. API Rate Limiting: Prevents abuse of endpoints.
 - iii. Environment Variables: All sensitive keys (e.g., API tokens) are stored securely in .env files.
 - iv. Input Validation: All user inputs are validated to prevent injection attacks.

8. Performance Optimization

- a. To ensure fast response times:
 - i. Bun is used for its high-speed runtime and efficient bundling.
 - ii. Database queries are optimized using Prisma's query builder.
 - iii. Caching is implemented for frequently accessed data (e.g., commit summaries).
 - iv. Asynchronous processing is used for long-running tasks like transcription and summarization.

6.3 Database connection

The backend of DevInsights uses PostgreSQL as its primary relational database, hosted on Neon Console for scalability and performance. The database schema is defined using Prisma ORM, which provides a type-safe and intuitive way to interact with the database in a TypeScript environment.

This section outlines the structure, relationships, and purpose of each model in the Prisma schema, which supports the core functionalities of the platform such as user management, project collaboration, AI-powered insights, and meeting summaries.

1. Prisma Schema overview

- a. The Prisma schema is defined in the schema.prisma file and includes:
 - i. Data source: PostgreSQL with vector extension support.
 - ii. Generator: Prisma client for JavaScript.
 - iii. Models: Representing tables like User, Project, Commit, Meeting, etc.
 - iv. Enums: For status tracking (e.g., MeetingStatus).

```
4 generator client {
5   provider = "prisma-client-js"
6   previewFeatures = ["postgresqlExtensions"]
7 }
8
9 datasource db {
10  provider = "postgresql"
11  url       = env("DATABASE_URL")
12  extensions = [vector] // importing the vector extension
13 }
```

b.

2. User and Authentication Models

- a. User
 - i. Stores user profile and authentication data. Each user has:
 - ii. A unique email address
 - iii. Optional profile fields (name, image)
 - iv. A default credit balance (used for file uploads)
 - v. Relationships to projects, questions, and Stripe transactions

b. UserToProject

- i. A **many-to-many** relationship model that links users to projects. This allows multiple users to collaborate on the same project.

```
1  model User {  
2    id          String  @id @default(cuid())  
3    createdAt   DateTime @default(now())  
4    updatedAt   DateTime @updatedAt  
5    imageUrl    String?  
6    firstName   String?  
7    lastName    String?  
8    emailAddress String  @unique  
9    credits     Int      @default(450)  
10  
11    userToProjects UserToProject[]  
12    questionsAsked Question[]  
13    stripeTransactions StripeTransaction[]  
14  }
```

c.

3. Project and collaboration model

a. Project

- i. Represents a GitHub repository linked by a user. It stores:
1. Project metadata (name, GitHub URL)
 2. Timestamps for creation and updates
 3. Relationships to commits, embeddings, meetings, and questions



```
model Project {
  id          String @id @default(cuid())
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  name        String
  githubUrl   String
  deletedAt   DateTime?

  userToProjects UserToProject[]
  commits         Commit[]
  sourceCodeEmbeddings SourceCodeEmbedding[]
  savedQuestions Question[]
  meetings        Meeting[]
}
```

b.

```
model UserToProject {
  id          String @id @default(cuid())
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  userId      String
  projectId   String

  user        User @relation(fields: [userId], references: [id])
  project     Project @relation(fields: [projectId], references: [id])

  @@unique([userId, projectId])
}
```

c.

4. AI Embeddings and Commit Summaries

a. SourceCodeEmbedding

- i. Stores AI-generated **vector embeddings** of source code files. Each record includes:
 1. The original source code
 2. A summary
 3. The file name
 4. A vector embedding (768 dimensions, compatible with Gemini)

b. Commit

- i. Stores commit data fetched from GitHub, including:
 1. Commit message, hash, author, and timestamp
 2. AI-generated summary of the commit

```
model SourceCodeEmbedding {
  id                String @id @default(cuid())
  summaryEmbedding  Unsupported("vector(768)")?
  sourceCode        String
  fileName          String
  summary           String
  projectId         String
  project           Project @relation(fields: [projectId], references: [id])
}
```

```
model Commit {
  id                String @id @default(cuid())
  createdAt         DateTime @default(now())
  updatedAt         DateTime @updatedAt
  projectId         String
  project           Project @relation(fields: [projectId], references: [id])
  commitMessage     String
  commitHash        String
  commitAuthorName  String
  commitAuthorAvatar String
  commitDate        DateTime
  summary           String
}
```

5. Meeting summaries and issues

- a. Meeting
 - i. Stores uploaded meeting recordings and metadata. Each meeting is linked to a project and has a status (PROCESSING or COMPLETED).
- b. Issue
 - i. Represents key discussion points extracted from a meeting. Each issue includes:
 - 1. Start and end timestamps
 - 2. A headline and summary
 - 3. A reference to the parent meeting



```
model Meeting {
  id          String @id @default(cuid())
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  name        String
  meetingUrl  String
  projectId   String
  project     Project @relation(fields: [projectId], references: [id])
  status      MeetingStatus @default(PROCESSING)
  issues      Issue[]
}
```

c.

```
enum MeetingStatus {
  PROCESSING
  COMPLETED
}
```

d.

```
model Issue {
  id          String @id @default(cuid())
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  start       String
  end         String
  gist        String
  headline    String
  summary     String
  meetingId   String
  meeting     Meeting @relation(fields: [meetingId], references: [id])
}
```

e.

6. Questions and AI responses

a. Question

- i. Stores user-submitted questions and AI-generated answers. Each question is linked to:
 1. A user
 2. A project
 3. Optional file references (in JSON format)

```
1  model Question {
2      id          String @id @default(cuid())
3      createdAt   DateTime @default(now())
4      updatedAt   DateTime @updatedAt
5      question    String
6      answer      String
7      fileReferences Json?
8      projectId   String
9      project     Project @relation(fields: [projectId], references: [id])
10     userId      String
11     user        User @relation(fields: [userId], references: [id])
12 }
```

b.

7. Payment and transactions

a. StripeTransaction

i. Tracks credit purchases made by users. Each transaction includes:

1. The number of credits purchased
2. A reference to the user
3. Timestamps for auditing

```
1  model StripeTransaction {
2      id          String @id @default(cuid())
3      createdAt   DateTime @default(now())
4      updatedAt   DateTime @updatedAt
5      userId      String
6      user        User @relation(fields: [userId], references: [id])
7      credits     Int
8  }
```

ii.

7. System Testing

Testing is a critical phase in the software development lifecycle, ensuring that the application functions as intended, is free of major bugs, and provides a smooth user experience. DevInsights underwent multiple levels of testing to validate its functionality, performance, and usability.

7.1 Unit testing

Unit testing was performed on individual components and functions to ensure they work in isolation. This included:

- Frontend components (e.g., buttons, forms, modals)
- Backend services (e.g., commit summarization, meeting transcription)
- Utility functions (e.g., date formatting, API response handling)

7.2 Integration testing

Integration testing ensured that different modules of the application worked together seamlessly. For example:

- Uploading a GitHub repository triggers commit fetching and AI summarization.
- Uploading a meeting file results in transcription and summary generation.

7.3 User Acceptance Testing (UAT)

A group of developers and testers used the application in real-world scenarios to validate:

- Ease of use
- Accuracy of AI-generated summaries
- Responsiveness of the UI
- Clarity of search results

8. Limitations

While DevInsights offers a powerful set of features, it also has certain limitations that are important to acknowledge:

8.1 AI Accuracy

- AI-generated summaries may not always capture the full context of code changes or discussions.
- Misinterpretations can occur, especially with complex logic or ambiguous commit messages.

8.2 Limited Context in AI Responses

- AI relies on source code embeddings, which may not fully represent the entire project structure or external dependencies.
- This can lead to incomplete or less relevant answers to user queries.

8.3 Scalability Challenges

- As the number of users and projects increases, the system may face challenges in:
 - Efficiently storing and retrieving embeddings
 - Managing API rate limits
 - Maintaining low-latency responses

8.4 Dependency on External Services

- The platform relies on third-party APIs such as:
 - GitHub API (for commit data)
 - Google Gemini AI (for summarization and Q&A)
 - Assembly AI (for meeting transcription)
- Any downtime or rate limiting from these services can affect functionality.

9. Conclusion

DevInsights is a modern, AI-powered platform that transforms how developers interact with their codebases. By integrating advanced AI models with intuitive UI and seamless GitHub integration, it offers:

- Commit Summarization: Quickly understand code changes without reading diffs.
- Natural Language Code Search: Ask questions in plain English and get relevant answers.
- Meeting Summaries: Upload recordings and receive structured summaries with timestamps.
- Collaboration Tools: Work with teammates on shared projects.
- Credit System: Manage usage through a scalable, monetized model.

Despite some limitations, DevInsights significantly improves developer productivity, onboarding, and collaboration. It serves as a valuable tool for teams managing large and evolving codebases.

10. Future Scope

The future development of DevInsights includes several enhancements aimed at improving accuracy, usability, and scalability:

Enhanced AI Capabilities

- Fine-tune AI models for better summarization and Q&A accuracy.
- Incorporate feedback loops to improve AI responses over time.

Offline Functionality

- Enable local analysis of repositories without internet access.
- Use local embeddings and models for privacy-sensitive environments.

Advanced Collaboration Features

- Add team dashboards with shared insights and activity logs.
- Implement threaded discussions on commits and summaries.
- Integrate with communication tools like Slack or Microsoft Teams.

Analytics and Insights

- Visualize commit trends, contributor activity, and code churn.

- Provide AI-driven suggestions for refactoring and optimization.

11. Bibliography/References

1. Next.js Documentation – <https://nextjs.org/docs>
2. Google Gemini AI - <https://ai.google.dev/gemini-api/docs>
3. GitHub API Documentation - <https://docs.github.com/en/rest>
4. Vercel Documentation - <https://vercel.com/docs>
5. Neon PostgreSQL - <https://neon.tech/docs/reference/api-reference>
6. Clerk Authentication - <https://clerk.com/docs/quickstarts/nextjs>
7. ShadCN UI - <https://ui.shadcn.com/docs>
8. AssemblyAI - <https://www.assemblyai.com/docs/>
9. Stripe- <https://docs.stripe.com/api/>
10. AssemblyAI - <https://www.assemblyai.com/docs/>
11. LangChain - <https://docs.langchain.com>
12. Neon PostgreSQL - <https://neon.tech/docs>
13. Stripe API - <https://stripe.com/docs>
14. Cypress - <https://docs.cypress.io>
15. Markdown Guide - <https://www.markdownguide.org>
16. Jest - <https://jestjs.io/docs>
17. LightHouse - <https://developer.chrome.com/docs/lighthouse/>
18. Prisma ORM - <https://www.prisma.io/docs>
19. Tailwind CSS - <https://tailwindcss.com/docs>
20. Supabase - <https://supabase.com/docs>