Technical Background

The capstone project was developed using a modern and efficient tech stack that aligns with industry standards, ensuring scalability, maintainability, and optimal performance. The frontend of the application was implemented using Next.js, selected for its ability to support server-side rendering (SSR) and static site generation (SSG). These features improved page load times and search engine optimization (SEO). With its React-based component structure, Next.js enabled the creation of a dynamic and interactive user interface while seamlessly integrating with the backend through REST APIs, resulting in a responsive and user-friendly experience.

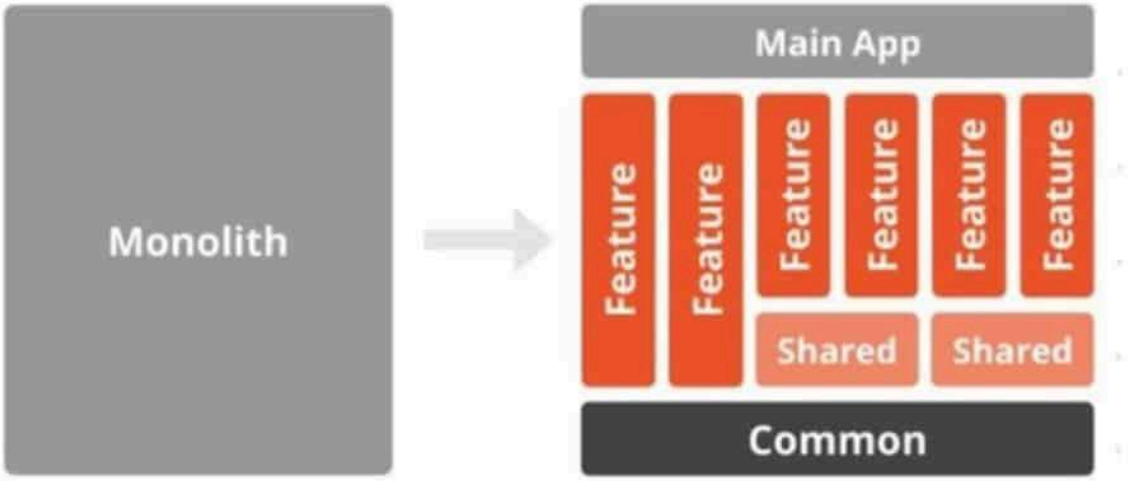

Figure 1. Modular Architecture Design

The backend was built using NestJS, a progressive Node.js framework that leverages TypeScript. NestJS's modular architecture provided a clean and scalable codebase, simplifying the management of the project's increasing complexity. Its service-oriented design allowed functionalities to be encapsulated into discrete modules, enhancing maintainability and promoting code reuse. Features such as dependency injection, middleware, and decorators further streamlined the

development of REST APIs, ensuring efficient request handling and robust business logic implementation.

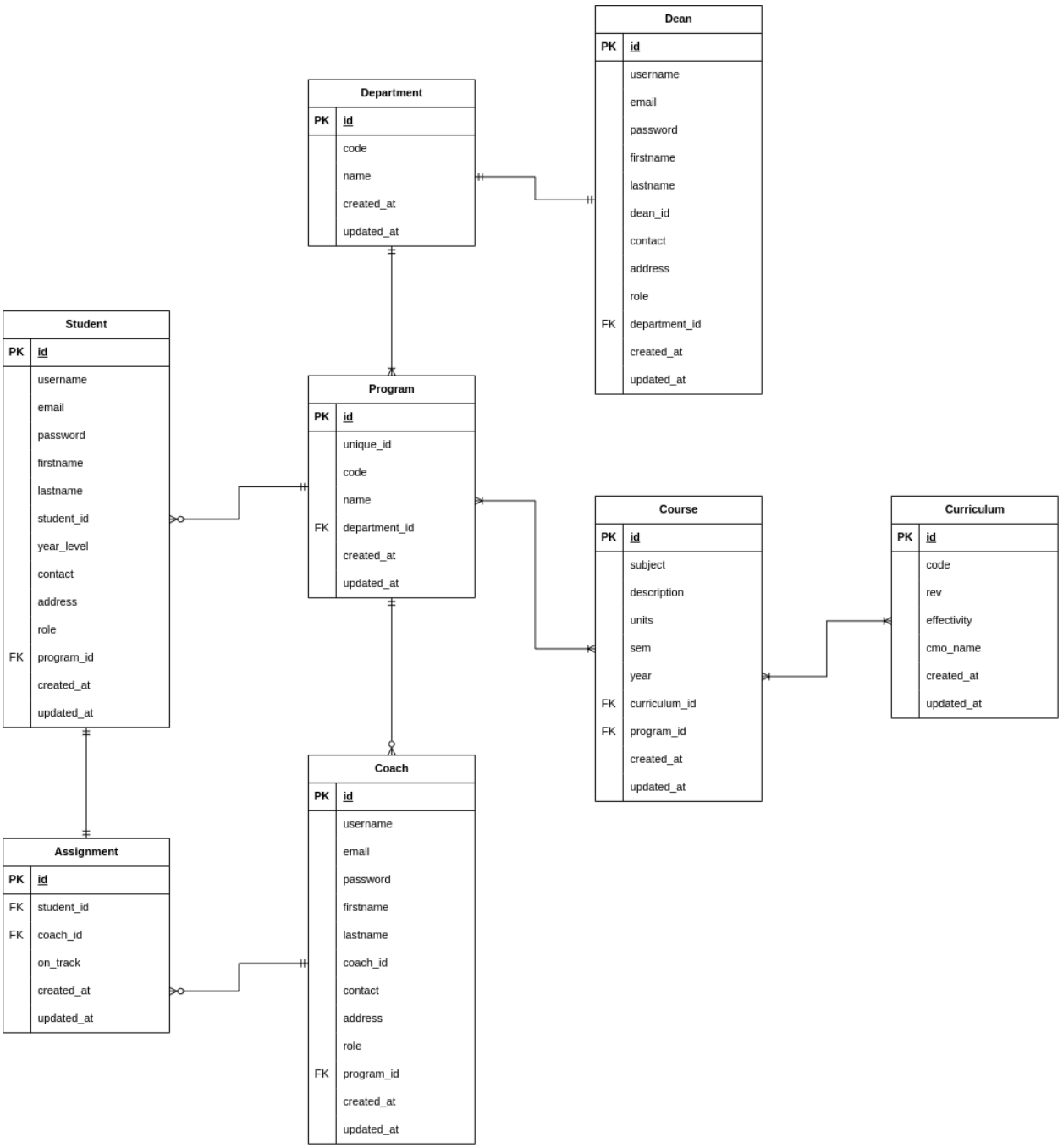

Figure 2. Entity Relationship Diagram

PostgreSQL was chosen as the relational database system for the project due to its powerful features, including support for complex queries, ACID compliance,

and extensibility. Its seamless integration with modern ORMs and compatibility with NestJS facilitated reliable data storage and management. PostgreSQL's capabilities ensured data integrity and consistency across the application.

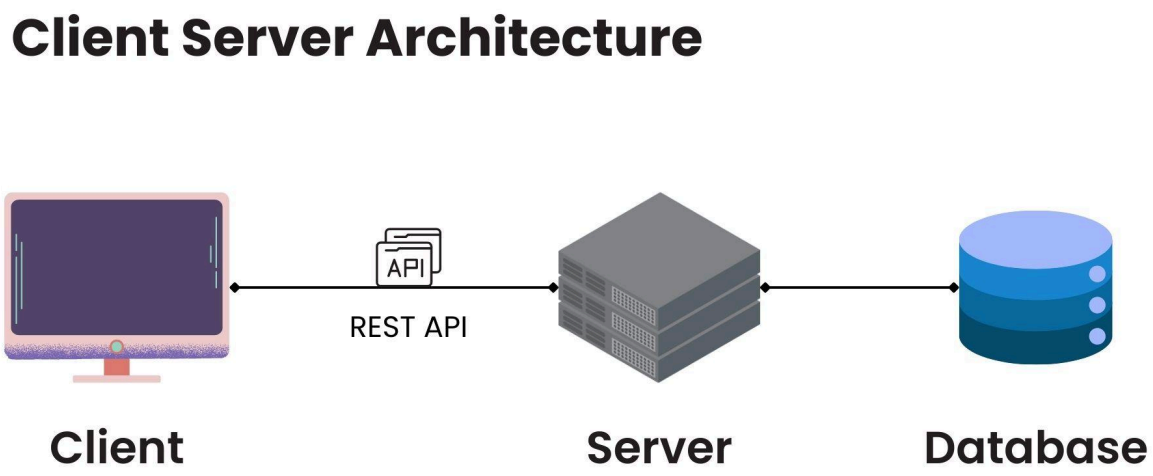

Figure 3. Client Server Architecture

The project adhered to a client-server architecture, which established a clear separation between the frontend and backend. The Next.js-based frontend served as the client, responsible for managing user interactions and delivering a responsive UI. Simultaneously, the NestJS backend functioned as the server, handling business logic, database operations, and exposing RESTful APIs for efficient communication. This architecture enabled independent development and optimization of the frontend and backend, fostering flexibility and scalability.

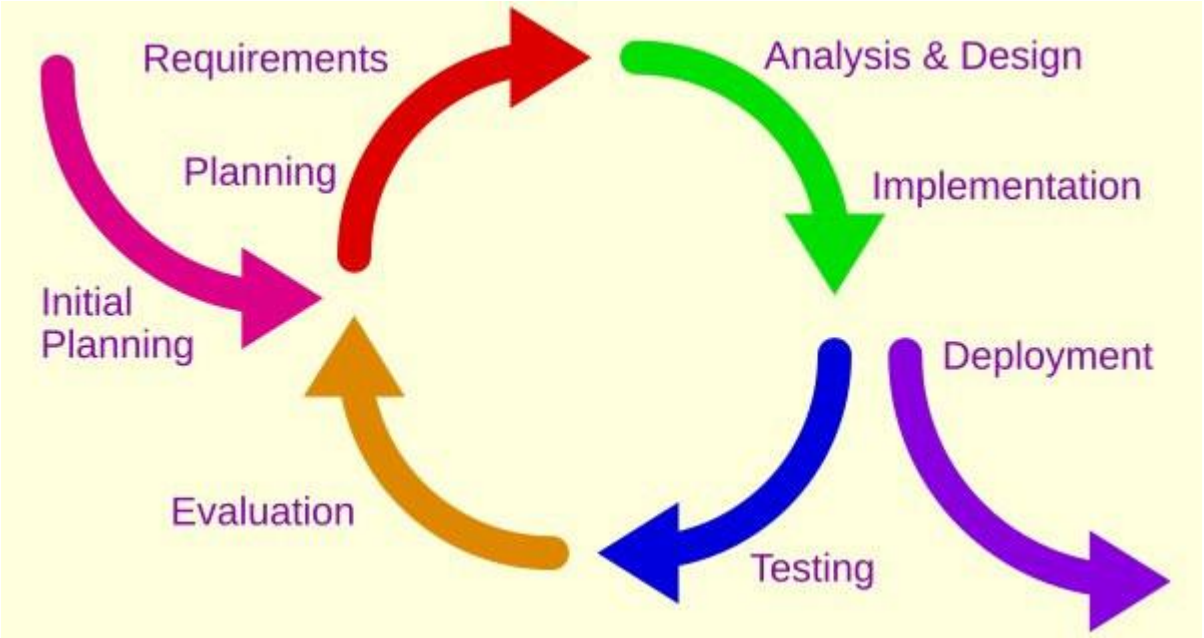

Figure 4. Incremental Software Methodology

An incremental software methodology guided the development process, emphasizing the creation of the application in manageable, iterative increments. Each iteration produced a functional version of the system, allowing for early feedback and continuous improvement. This approach minimized risks, ensured consistent progress, and provided adaptability to evolving requirements.

The chosen tech stack and methodology offered numerous advantages. The combination of SSR, REST APIs, and a modular architecture ensured high performance and maintainability. PostgreSQL provided reliable data management, while the client-server model supported future scalability. The incremental methodology facilitated a steady and adaptive development process, resulting in a scalable and user-focused application built on a solid technical foundation.