Star Infinity - Software Project Proposal

Muhammad Bilal Khan

Syed Ather Saeed

Instructor*: Rakibul Islam*

Dated: *2025-03-13*

Author Note

(Junior) Developers:

1. Muhammad Bilal Khan (Backend, Fullstack) Developer
2. Syed Ather Saeed (Database) Developer
3. Nishant Durant (Frontend) Developer

(Senior) Developer / Evaluator:

1. Rakibul Islam

Abstract

Star Infinity is a cutting-edge online teaching platform focused on computer science, coding languages, and DevOps, designed to take learners from foundational concepts to mastery. Users sign up via Zoom, streamlining the enrollment process by leveraging existing meeting credentials, while the platform does not include a separate user dashboard, an integrated admin dashboard enables course management: administrators can upload course content and images, and the system automatically creates or updates course information accordingly. Built with a modern tech stack including ReactJS for the frontend, a GraphQL server for seamless communication, NodeJS for backend logic, and PostgreSQL for data persistence, Star Infinity also employs ArcJet for enhanced security, auth.js for administrative verification, and Prisma as its ORM. Although the specifics of Zoom integration are still under development, the project is structured to provide an engaging and scalable educational experience to meet evolving industry demands.

Technical Requirements / Diagrams

This section outlines the specific technical needs of the project, including functional and non-functional requirements that must be stated. It will also provide sources of reference and truth to bridge business requirements and technical implementation.

Functional Requirement No.	Functional Requirement Description
FR-001	System can process and manage user logins for admin only.
FR-002	System can display a list of available instructor-led courses covering topics that students can register for.
FR-003	System should display a search result for courses based on a search query.
FR-004	System can process user information when registering for a live instructor-led meeting.

FR-005	System can prevent security attacks from entities such as bots with bot detection capabilities.
FR-006	System can prevent security attacks from malicious users through rate limiting.
FR-007	System can consume API from backend processes and display accurate and timely information.
FR-008	System can prevent constant “ghost registering” through the use of IP blocking.
FR-009	System must store personalized information and preferences for the user in localStorage.
FR-010	System must transfer the user to the zoom registration page after a user clicks “register” to register to an instructor-led course.

Non-Functional Requirement No.	Non-Functional Requirements Description
NFR-001	System should be able to handle around 5,000 concurrent users performing various tasks.
NFR-002	System should respond to user requests and relay users to zoom registration within 5 seconds.
NFR-003	System should provide clear and concise error messages in both the frontend and backend domains.
NFR-004	System should be able to scale to handle more than 5,000 concurrent users when necessary.
NFR-005	System should be available with 99.9% uptime.
NFR-006	System should be able to run on multiple browsers (Chromium, Firefox, Safari).
NFR-007	System should be easy to maintain by other developers and easily updatable.
NFR-008	System should be able to provide an interface for users that is easy-to-use and intuitive.
NFR-009	System must send an email notification when a user successfully enrolls to an

	instructor-led course.
NFR-010	Codebase documentation should be updated regularly and follow industry-standard practices (e.g., JSDoc for JavaScript)
NFR-011	System should be deployed on cloud platforms such as AWS, DigitalOcean, Azure , or GCP without requiring significant code changes.
NFR-012	Containerization (e.g., using Docker) should be used to ensure that the deployment can work across different environments.
NFR-013	System should be compatible for both desktop and mobile devices, with the exception of the Admin dashboard (only for desktop).
NFR-014	System should recover from failures within 5 minutes without any significant data loss.
NFR-015	System's backend should allow for auto-scaling during peak usage times (e.g., during course enrollment periods which are handled externally by zoom).
NFR-016	System must prevent SQL injection, cross-site scripting (XSS), and other common vulnerabilities as per OWASP.

Application Overview (Presentation, Business, Database)

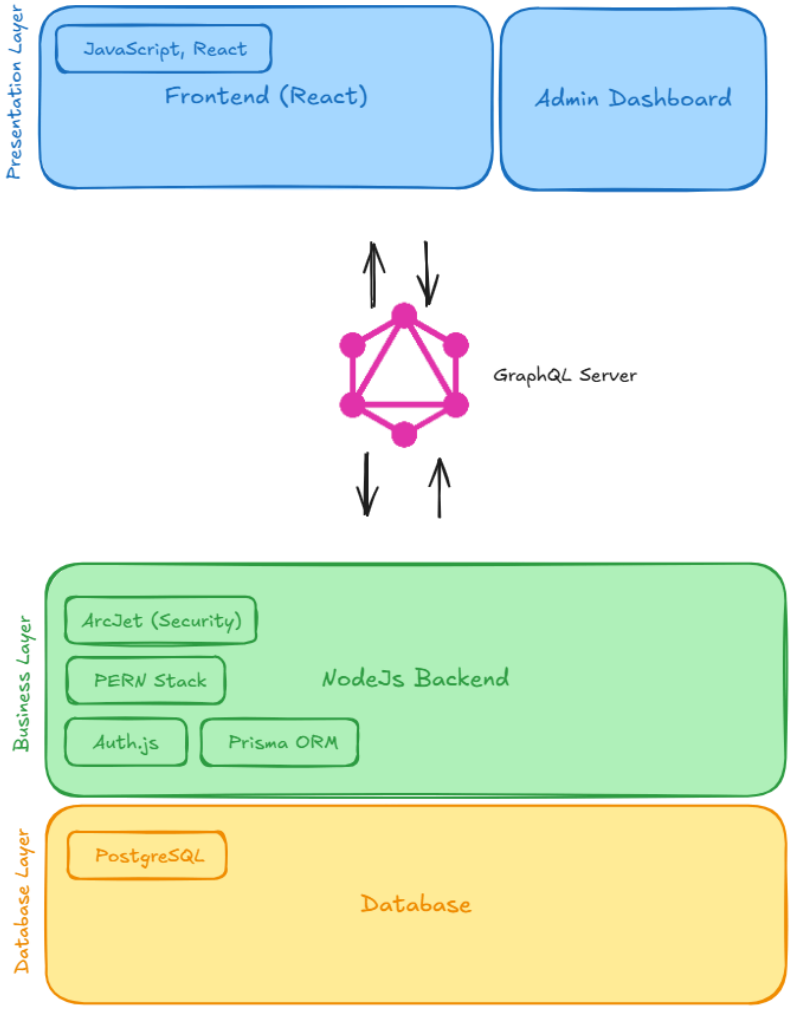


Figure 1: Application Overview Covering Presentation, Business, and Database Layers.

The figure above provides a comprehensive overview of the applications architecture, which is divided into its 3 respective layers being:

1. Presentation Layer;
2. Business Layer;
3. Database Layer.

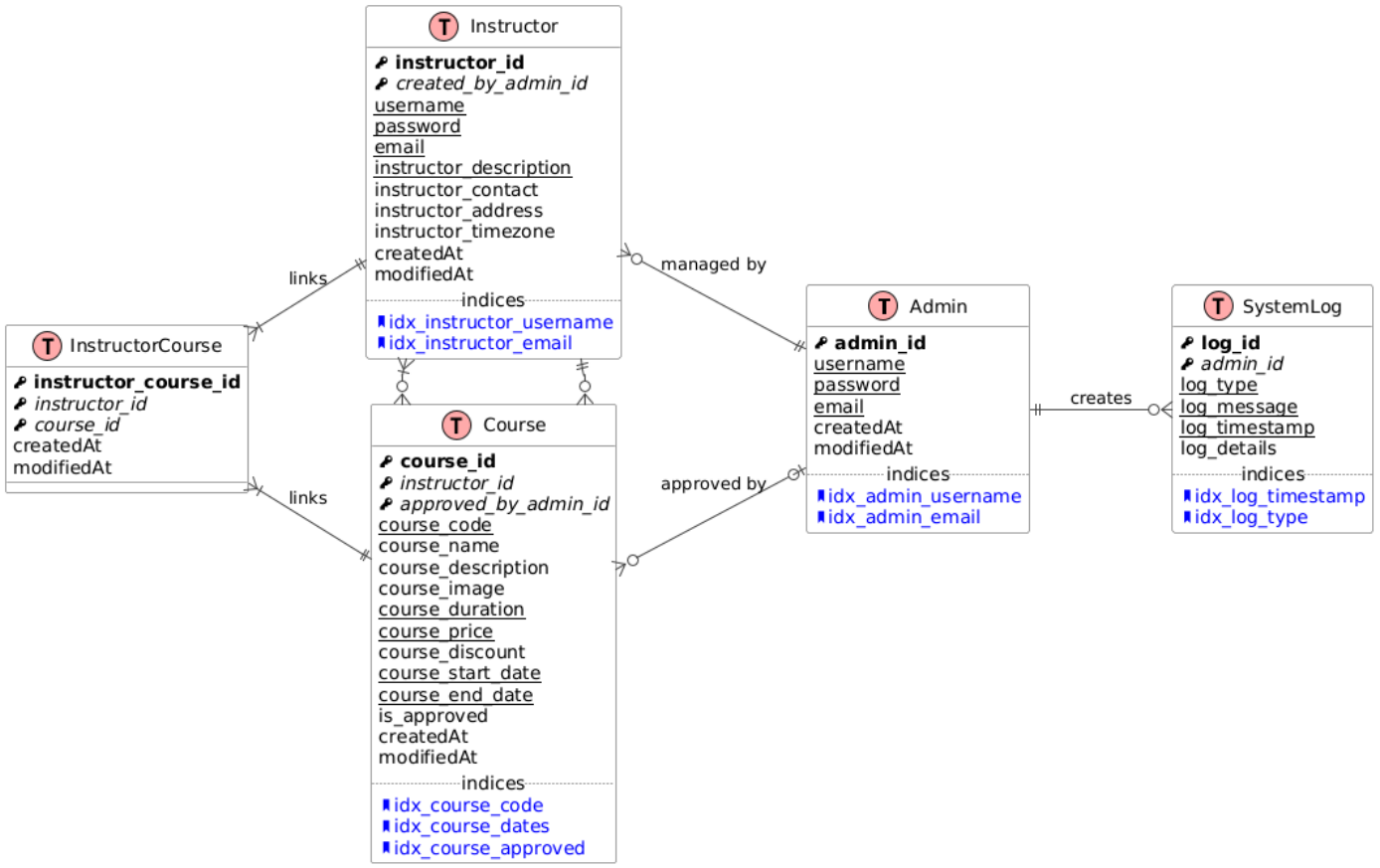
For the presentation layer, we will use React (NextJs) for the frontend application with Typescript/JavaScript coupled with a UI library (Shadcn UI) that will address all our needs.

Alongside this, we will include an Admin Dashboard that will be specifically designed for more administrative tasks, providing a dedicated interface for managing the frontend applications data and settings.

For the business layer, we will process business logic and communicate between the presentation layer and database layer. Our technologies will include the following:

1. PENQ Stack (PostgreSQL, Express, NodeJs) with GraphQL.
2. Auth.js for authentication and authorization of the application.
3. ArcJet as our security framework integration into the application to protect against various types of attacks to ensure data integrity.
4. Prisma ORM as our ORM tool that will simplify database operations by providing a high-level abstraction over the underlying database layer.

For the database layer, we will store and manage the application's data. Using PostgreSQL are our primary database of choice alongside its RDBMS or opting for the use of more streamlined RDBMS such as DBeaver.



At the center of everything, we have a GraphQL server, which will act as an intermediary between the Presentation Layer and Business Layer. GraphQL will serve as our query language for APIs and provide a more efficient and flexible way to fetch and manipulate data compared to more traditional RESTful APIs which are prone to failure. Our GraphQL server will receive requests from the frontend and admin dashboard, process, and forward back to appropriate services within the business layer.

Simple Application Overview

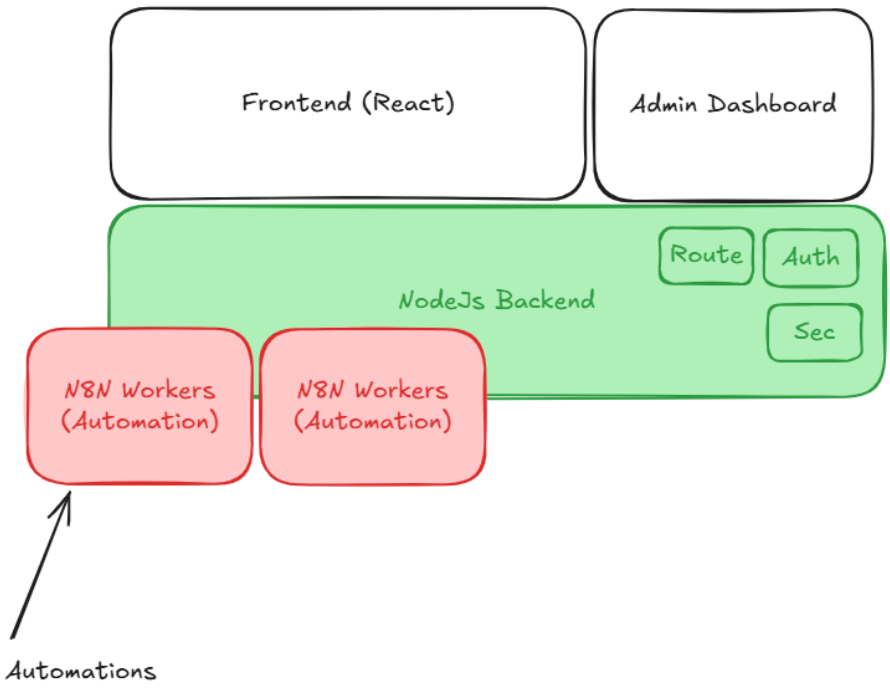
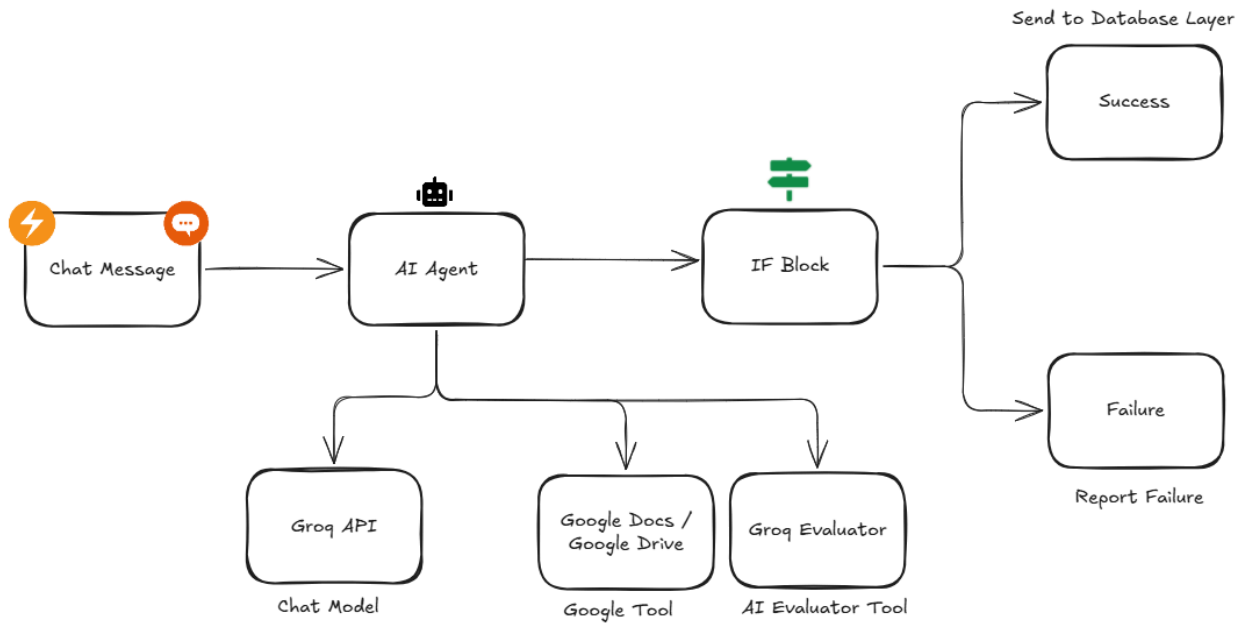


Figure 2: Simple Application Overview w/N8N Workers (Automation)

The figure above presents a more simplified overview and architecture of how we will implement n8n workers to perform automations on our business layer. Automations that may fit the use case of this application is using Agentic frameworks that will automatically deploy a live session/course to the backend by fetching relative data from a google doc/form that contains all information required, or sending a chat message with all the content the Agentic AI needs to create a JSON structure. Provided below is a rough outline of how the automation should be within n8n:



Project Plan

This section will outline the roadmap for executing the project, detailing its scope, goals, tasks, deliverables, and milestones within a tabular format. We will provide a comprehensive overview of how the project should be managed from initiation to completion, including timelines, resource allocation, and any risks we must resolve.

Goals, Tasks, and Deliverables

Goal	Tasks	Deliverables
Zoom-Based User Access	- Develop Zoom integration for sign-up and content viewing. - Configure session settings and restrictions.	- Functional Zoom-based enrollment. - Verified user access workflows.
Structured Course Management	- Design admin dashboard for managing courses. - Create modules and lessons management features. - Integrate content upload and update functionality.	- Admin dashboard with course structure (modules and lessons). - Documentation for course management processes.
Scalability & Performance	- Architect system to support up to 5000 concurrent users. - Perform load testing and performance optimization.	- Load-tested system capable of handling peak loads.
Security & Compliance	- Implement ArcJet and auth.js for security. - Research and determine compliance standards for software projects in India.	- Secure system implementation. - Compliance recommendations report.
Modern Technology Stack Implementation	- Develop frontend using ReactJS. - Create backend services with NodeJS and GraphQL. - Set up a PostgreSQL database using Prisma ORM.	- Fully functional platform built on the modern tech stack.
Team Coordination	- Define team roles and responsibilities (within a four-man team). - Establish regular status meetings and documentation workflows.	- Team structure document. - Scheduled project meetings and communication plan.

1. Project Overview

Star Infinity is an innovative online teaching platform focusing on computer science, coding languages, and DevOps. It is designed to guide learners from foundational concepts to mastery. Users are limited to viewing content on Zoom, with enrollment streamlined via Zoom credentials. Course management is handled exclusively through an integrated admin dashboard.

2. Objectives

I. **Seamless Zoom-Based Access:** Ensure enrolled users can securely access and view course content via Zoom.
II. **Structured Course Delivery:** Organize course content into modules and lessons for clear, systematic learning.
III. **Scalability:** Build a robust platform capable of handling up to 5000 users during peak loads.
IV. **Security & Compliance:** Integrate ArcJet and auth.js for robust security while researching appropriate compliance standards for India.
V. **Modern Technology:** Use ReactJS, GraphQL, NodeJS, PostgreSQL, and Prisma to create a high-performance, scalable system.

3. Scope & Deliverables

In Scope:

I. **User Enrollment & Access:**
    i. Zoom-based sign-up and secure viewing access.
II. **Admin Dashboard:**
    i. Manage course content structured into modules and lessons.
    ii. Upload and update functionalities for course materials and images.
    iii. Basic payment handling via the admin dashboard (future expansion possible).
III. **Backend Services:**
    i. NodeJS backend with GraphQL server and PostgreSQL database integration using Prisma.
IV. **Security & Verification:**
    i. Integration of ArcJet and auth.js.
V. **Scalability:**
    i. Design and testing for 5000 peak concurrent users.

Out of Scope:

I. A separate user dashboard.
II. Payment integrations outside of current admin dashboard functionalities.

III. Detailed role assignment within the four-man team (to be decided).

4. Technical Architecture & Stack

I. **Frontend:** ReactJS.
II. **API Layer:** GraphQL server.
III. **Backend:** NodeJS.
IV. **Database:** PostgreSQL.
V. **ORM:** Prisma.
VI. **Security:** ArcJet, auth.js.
VII. **Integration:** Zoom (for enrollment and content viewing).

5. Project Phases & Timeline

Phase 1: Requirements & Planning

I. Finalize functional and non-functional requirements.
II. Define user flows for Zoom-based content viewing and admin course management.
III. Clarify Zoom integration specifics and begin compliance research for India.
IV. Develop initial team role assignments and project schedules.

Phase 2: Design & Prototyping

I. Create wireframes and design mockups for the admin dashboard.
II. Architect backend services and database schema focused on course structure.
III. Define API endpoints for course management and Zoom integration.

Phase 3: Development

I. **Frontend:** Build ReactJS components for the admin dashboard.
II. **Backend:** Develop NodeJS services and GraphQL APIs.
III. **Database:** Set up PostgreSQL schemas and integrate with Prisma.
IV. **Security:** Implement ArcJet and auth.js integrations.
V. **Integration:** Develop and test Zoom-based enrollment and content viewing workflows.

Phase 4: Quality Assurance & Testing

I. Conduct unit, integration, and system tests.
II. Perform load and performance tests simulating 5000 concurrent users.
III. Validate security and compliance requirements through pilot testing.

Phase 5: Deployment & Maintenance

I. Deploy the platform in stages (beta and full production).

II. Monitor system performance and gather feedback.
III. Establish a maintenance schedule for iterative updates and improvements.

6. Resource Allocation & Team Structure

I. **Team Size:** 4-man team (role assignments pending final discussion)
II. **Key Areas:** Project management, frontend development, backend development, database and security management, integration, and testing.
III. **Communication:** Regular weekly meetings and comprehensive documentation throughout the project.

7. Risk Management

I. **Technical Risks:** Challenges with Zoom integration and scalability under peak loads.
II. **Compliance Risks:** Uncertainty in compliance standards for software projects in India.
III. **Operational Risks:** Limited team size may affect resource allocation and timelines.
IV. **Mitigation Strategies:**
    i. Incremental development and iterative testing.
    ii. Regular stakeholder reviews and clear communication.
    iii. Early-stage compliance research and potential consultation with experts.

8. Communication & Reporting

I. **Status Meetings:** Regular (weekly) project meetings.
II. **Documentation:** Detailed documentation for code, APIs, and processes.
III. **Feedback Loops:** Continuous feedback from pilot testing and administrative reviews.

Project Budget

This section will provide a detailed financial overview of the project, outline cost estimates, funding sources, and expenditure plans. We will go over a breakdown of the potential costs associated with software, hardware, and third-party services, along with a contingency plan to address any unexpected expenses that may arise.

[DISCUSS WITH RAKIB]

[TODO]

References

This section contains all of the sources and documents references throughout the proposal.

https://nextjs.org/docs

https://docs.arcjet.com/

https://graphql.org/learn/

https://authjs.dev/

https://www.prisma.io/

https://nodejs.org/docs/latest/api/