AI Assisted Online Learning Platform

Team A3

Anirudh Vijayaraghavan, 002995983

Sunil Srinivaas, 002711958

Devki Patel, 002912471

Kaushik Elangovan, 002793494

Arvind Ram Mahesh, 002819495

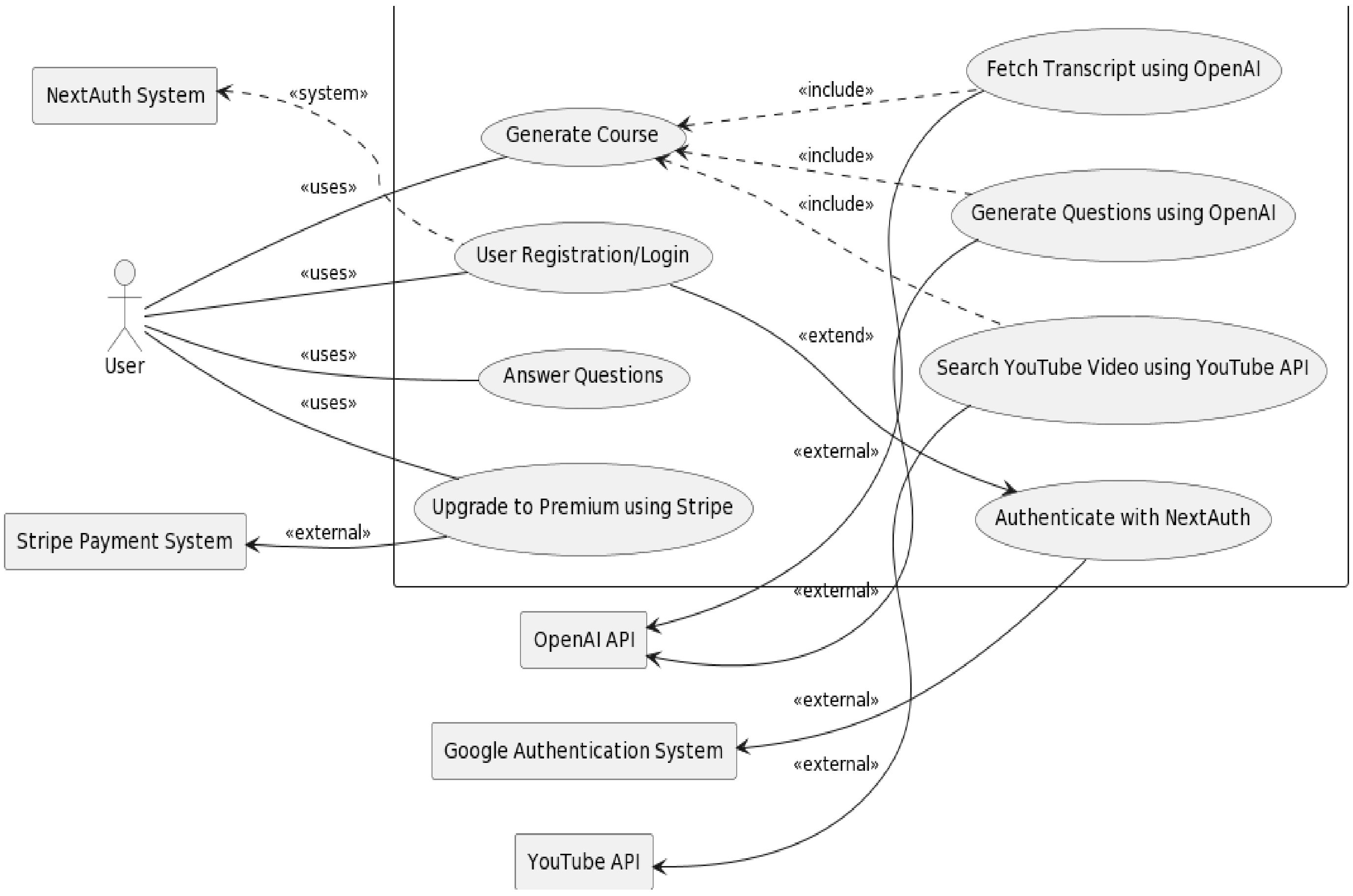
Github repository :

https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/tree/main

Analysis Description

We have tried our best to ensure each diagram listed below provides insight to view the application's requirements and functionalities, as per our knowledge obtained in the lectures. When combined, they should provide a good understanding/overview of how the application is developed, the workflows it should facilitate, the data/object structures it should implement, and the different ways users can interact with it.

<u>Use case Diagram:</u>

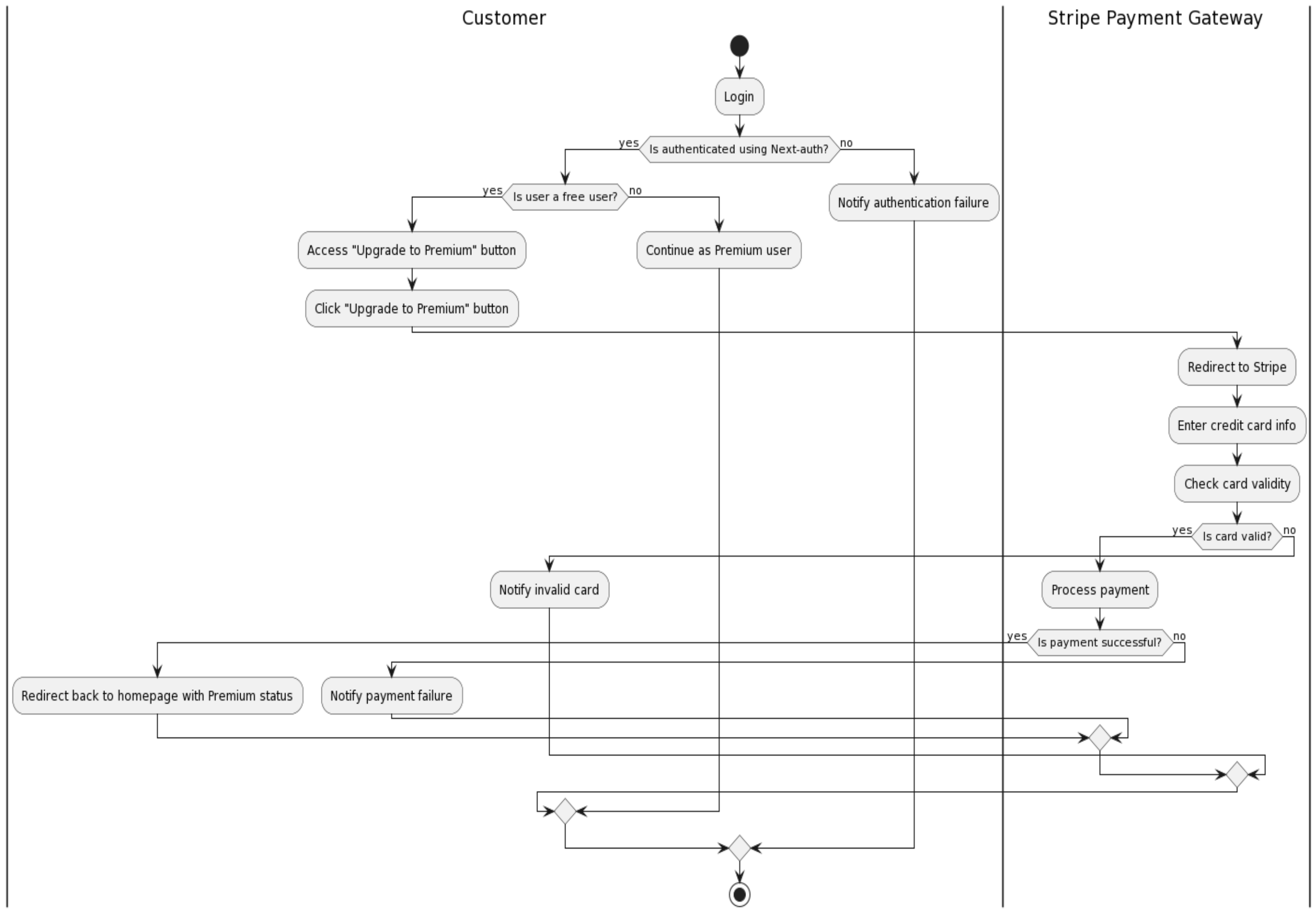


Use Case Diagram:

Purpose: Illustrates user-system interactions.

How the diagram corresponds: Shows essential activities – verification, course development, billing, and content access. Demonstrates user and system actions. Allows stake holders understand user objectives and system deliverables.

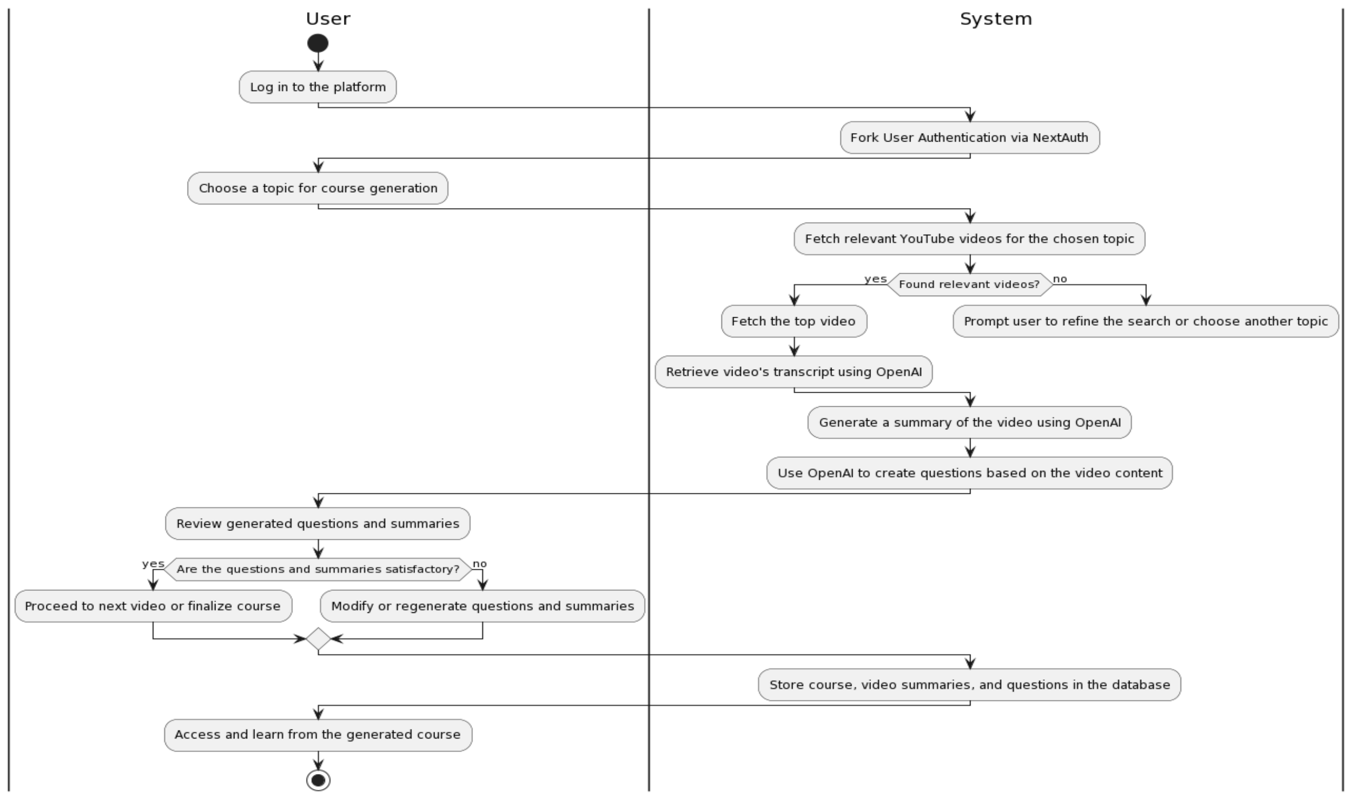
Activity Diagrams:



^

|

Part 1


User
System
Log in to the platform
Fork User Authentication via NextAuth
Choose a topic for course generation
Fetch relevant YouTube videos for the chosen topic
yes
Found relevant videos?
no
Fetch the top video
Prompt user to refine the search or choose another topic
Retrieve video's transcript using OpenAI
Generate a summary of the video using OpenAI
Use OpenAI to create questions based on the video content
Review generated questions and summaries
yes
Are the questions and summaries satisfactory?
no
Proceed to next video or finalize course
Modify or regenerate questions and summaries
Store course, video summaries, and questions in the database
Access and learn from the generated course

^

|

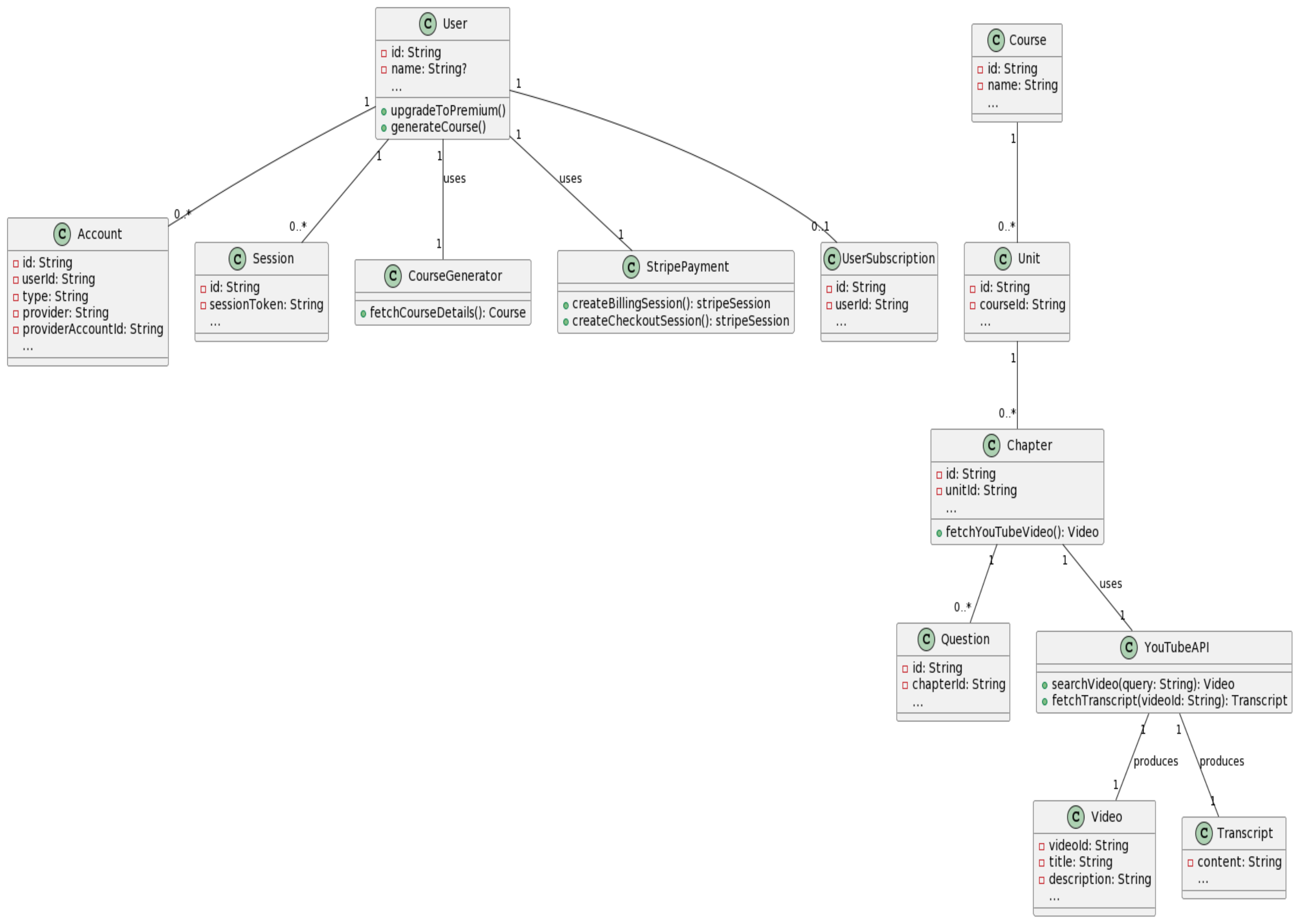
Part 2

Activity Diagram:

Purpose: Displays system workflow and activities.

How the diagram corresponds: Searching the video, transcription, summary, questions, and payment process is described by step. Shows decision point, alternative path as lack of transcript or payment problems. Useful for identifying workflow bottlenecks.

Class Diagram:

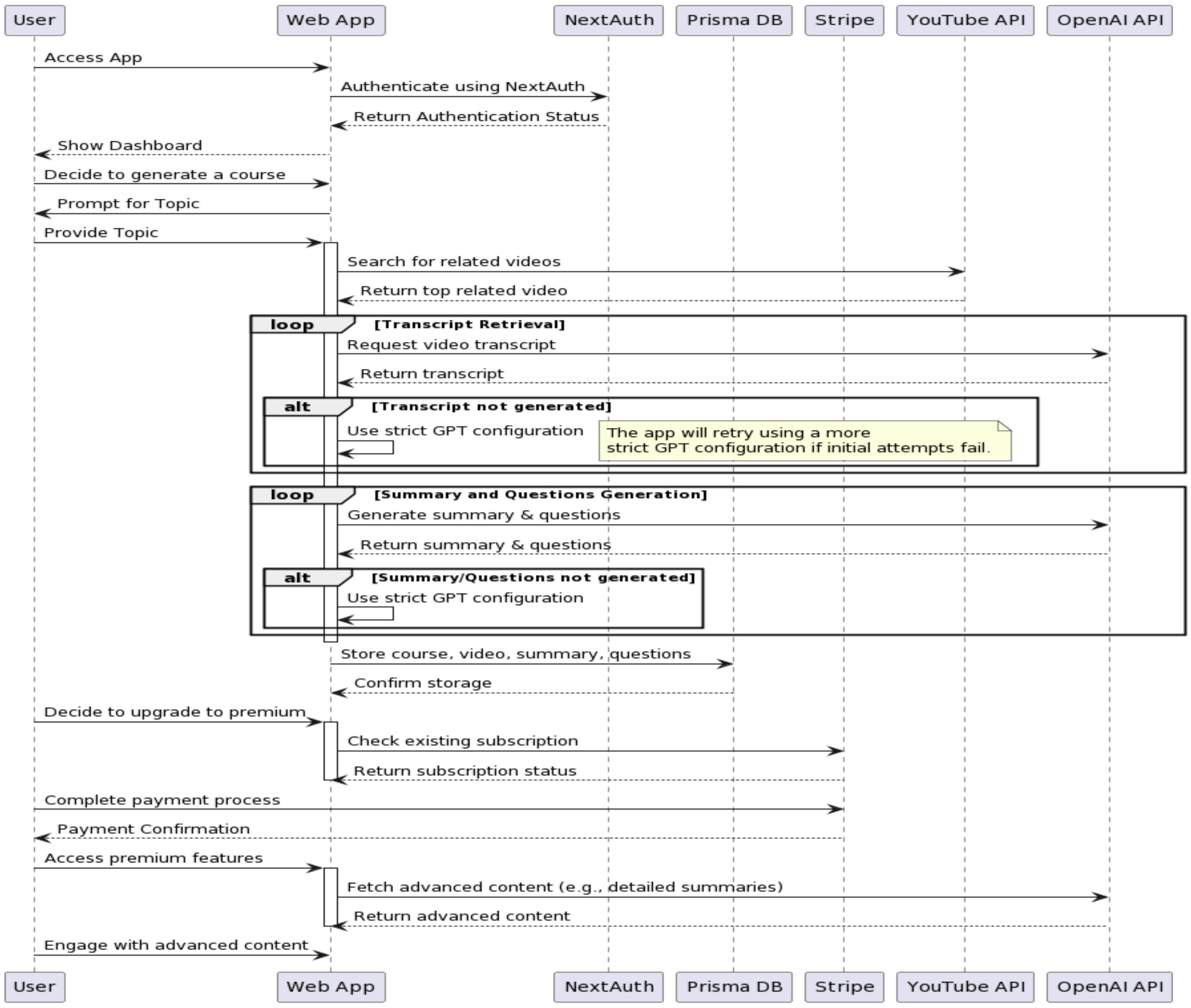


Class Diagram:

Purpose: This represents the system statical structure – classes, attributes, methods, and relationships.

How the diagram corresponds: Provision of user, session, course, unit, and chapter entities. Shows relationships (associations, aggregations, compositions). Ensures implementation of entities and guides developers through the process of coding.

Sequence Diagram:

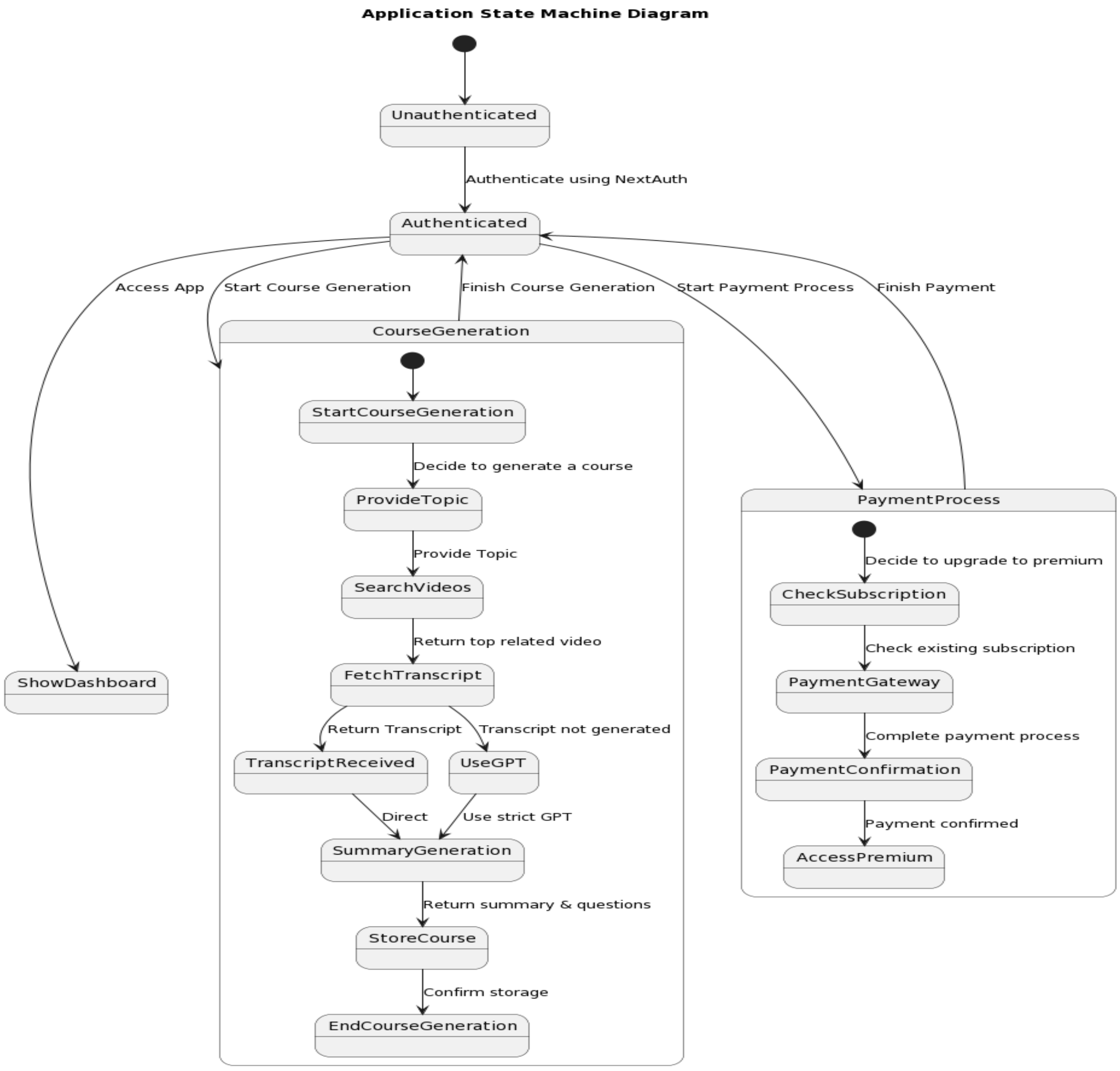

User
Web App
NextAuth
Prisma DB
Stripe
YouTube API
OpenAI API
Access App
Authenticate using NextAuth
Return Authentication Status
Show Dashboard
Decide to generate a course
Prompt for Topic
Provide Topic
Search for related videos
Return top related video
loop [Transcript Retrieval]
Request video transcript
Return transcript
alt [Transcript not generated]
Use strict GPT configuration
The app will retry using a more strict GPT configuration if initial attempts fail.
loop [Summary and Questions Generation]
Generate summary & questions
Return summary & questions
alt [Summary/Questions not generated]
Use strict GPT configuration
Store course, video, summary, questions
Confirm storage
Decide to upgrade to premium
Check existing subscription
Return subscription status
Complete payment process
Payment Confirmation
Access premium features
Fetch advanced content (e.g., detailed summaries)
Return advanced content
Engage with advanced content

Sequence Diagram:

Purpose: Shows object interactions over time.

How the diagram corresponds: Action flow – authentication, video search, content creation, and payment visualized. Describes interaction order that helps understand developer. Highlights component connections: WebAPP, next AUTH, YouTube API, Stripe, etc.

State Machine Diagram:



State Machine Diagram:

Purpose: Illustrates object states and transitions.

How the diagram corresponds: For core component displays “for course generation”, “authentication” and “payment”. Displays how a system reacts on certain conditions like missing payment by students or creating courses. Clarifies object lifecycles.

Architecture Description

1. Frontend (User Interface):

- Navbar/Landing Page: The primary user interaction point. The sidebar gives navigation to different areas, such as a gallery, creating the course, settings, or user's account management.

- User Authentication: User sign-in associated components, user account. This means that only authorized users can design courses or have access to specific areas of an application.

- Theme Toggle: Allows for switching between various visual themes and providing a good user experience.

2. Backend (Server Side):

- User Authentication and Management: Handles user authentication sessions using the getAuthSession() function. It dictates through which functions (such as making courses) a user's access is permitted based on his or her authentication.

- Database: The database is accessed using Prisma ORM to retrieve information. It serves to facilitate the recovery and storage of data related to users, courses, and subscriptions.

- Course Generation API: It gets data from users, analyzes, communicates with an AI-based model to propose items for the learning process, and builds courses.

- Stripe Integration: This is a section that deals with the logic behind any subscriptions associated with premium features. It deals with the user's subscription status and hosts a user billing portal for active subscribed users including a gateway to a third-party stripe system that facilitates the handling of new user subscriptions.

3. External API Services:

- Stripe: Combined for managing transactions and subscriptions. It smoothly converts your users to a premium variant that has many additional qualities.

- NextAuth: Utilized for verification reasons whereby all users have authenticated and granted permission to certain sections.

- AI Models: API models from openAPI and in combination with the YouTube API and Unsplash API are used for recommending course content and relevant image generation.

4. Middleware & Utilities:

- Theme Management: Controls visual themes for the user to personalize their UI experience.

- Error Handling: Although the specific elements are not yet fully developed, for now, there is error handling, especially on Stripe-related operations, which should ensure a smooth operation with unforeseen problems.

5. Data Storage:

- Database: Our application uses PlanetScale as a database to save information regarding students, courses, and subscriptions. Prisma acts as a connecting point between the backend logic and the database.

<u>First Level: Across Components (Application Landscape Patterns):</u>

1) Architecture Pattern Chosen: N-tier

Why:

1. Presentation Tier (Frontend): This includes navigation/landing page, theme toggle, and user authentication.

2. Logic Tier (Application Server): It contains the Course Generation API, User Authentication and Management, Stripe Integration, and perhaps AI models.

3. Data Tier (Database): Accessed through Prisma, which is a data storage system.

The N-tier architecture is beneficial for a few reasons:

- Separation of Concerns It also provides an opportunity to separate concerns.
- It is scalable; we can scale the logic tier horizontally without impacting the presentation or data tier(s). For instance, each of these tiers could be managed separately.

2) How it addresses the needs of the app.:

- Scalability: The Logic Tier and various others can be scaled up as the number of users or amount of content increases. This is an advantage as it means the systems do not have to be totally revamped when changing the size of the system.

- Flexibility: When we choose to change the front-end framework or use a different database, the N-tier architecture will allow to implement changes in one layer without major impacts on other layers.

- Security: By separating user-facing components from backend logic and database, robust security practices can be implemented at every stage. For example, the Data Tier may be encrypted and the Logic Tier deals with authorization and safely handles business logic.

Other Patterns and Their Applicability:

1. Monolith:
   Given that our case has various separate features in our application plus there being the need for scalability, using a monolith architecture could be problematic in our context. However, it could limit our future scaling and adjustment capacities.

2. Service-oriented:
   Although parts of our app, such as Course Generation API or Stripe Integration are essentially services, our whole program does not center on SOA.

3. Microservices:
   With time, as we build more functionalities in our backend, there could be a benefit in decomposing it into separate microservices. Every microservice would focus on one particular task like user management, course generation or payment processing. However, this method is more scalable and sustainable.

4. Serverless:
   We thought of implementing serverless architecture in some parts of the application such as the Course Generation API. They take advantage of automatic scaling and efficient costs for this selection. Our app's functioning peculiarities must be scrutinized before fully transforming it into serverless.

5. Peer-to-peer:
   The problem is that our application's structures do not correspond to this peer-to-peer pattern, which makes its connection to our practice quite doubtful.

Second Level: Within Components (Application Structure Patterns):

1) Application Structure Chosen: Layered

Why:

1. Presentation Layer: It deals with the user interface including the navbar, landing page, and other User Interface related UI components.
2. Application Layer: Controls business logic and handles requests from the view layer; it also includes generating courses, authenticating users, and interfacing with Stripe.
3. Persistence Layer: Holds privileges of data access, containing query to Prisma's databases as well as cache mechanisms.
4. Services Layer: However, it is not a mandatory component under conventional layered architectures; however, it accommodates third party integration and some external services like Stripe for future purposes.

2) How it addresses the needs of the app.:

- Separation of Concerns: Ensures orderly development and control by clearly defining responsibilities for each layer.
- Reusability: This allows for easy reuse or replacement of business logic in the application layer without impacting other layers.
- Scalability: Provides separation and allows layers, especially Application Layer, to scale independently to support demand.

Other Patterns and Their Applicability:

1. Microkernel: The main functionality of this pattern is in its center and the plug-and-play characteristics are supported by it. This would be useful for us if we intend to offer additional options, such as modules and features in the future.

2. CQRS (Command Query Responsibility Segregation): If the number of writes is substantially different than the reads think about using CQRS. An example would be separating course production and viewing when the latter takes place more frequently than the former to improve performance.

3. Event Sourcing: Store changes as events rather than only the current state. Event sourcing, for example, can help track the history of the course's life cycle and restore the state in case it is necessary.

Hence, the layered architecture meets our requirements for an intelligent educational app owing to its conceptualization and division of concerns.

Third Level: UI Components (User Interface Patterns):

1) Selected Pattern: Model-View-Controller (MVC)

Why:

1. Model: Represents the most essential aspects of our business logic and physical data model. Such cases involve specifying the course arrangements, store users' information, related metadata, among others.

2. View: It consists of all visual elements which our users come across and manipulate. This encompasses course display, user settings, and the like.

3. Controller: It is a middleman connecting between the Model and the View. This component oversees taking user input, processing requests, possibly refreshing the model, and sending the outcome back to the view. The controller is responsible for vital actions such as developing a course, signing users in, and Stripe payments.

2) How It Addresses Our App's Needs:

The MVC architectural pattern allows us to easily deal with and scale our app's parts. This modularity will increase our development cycles immensely and reduce some of these nagging pains associated with debugging. It should be noted that this is especially significant to our app which needs to develop along with our reactions to changes in user demand.

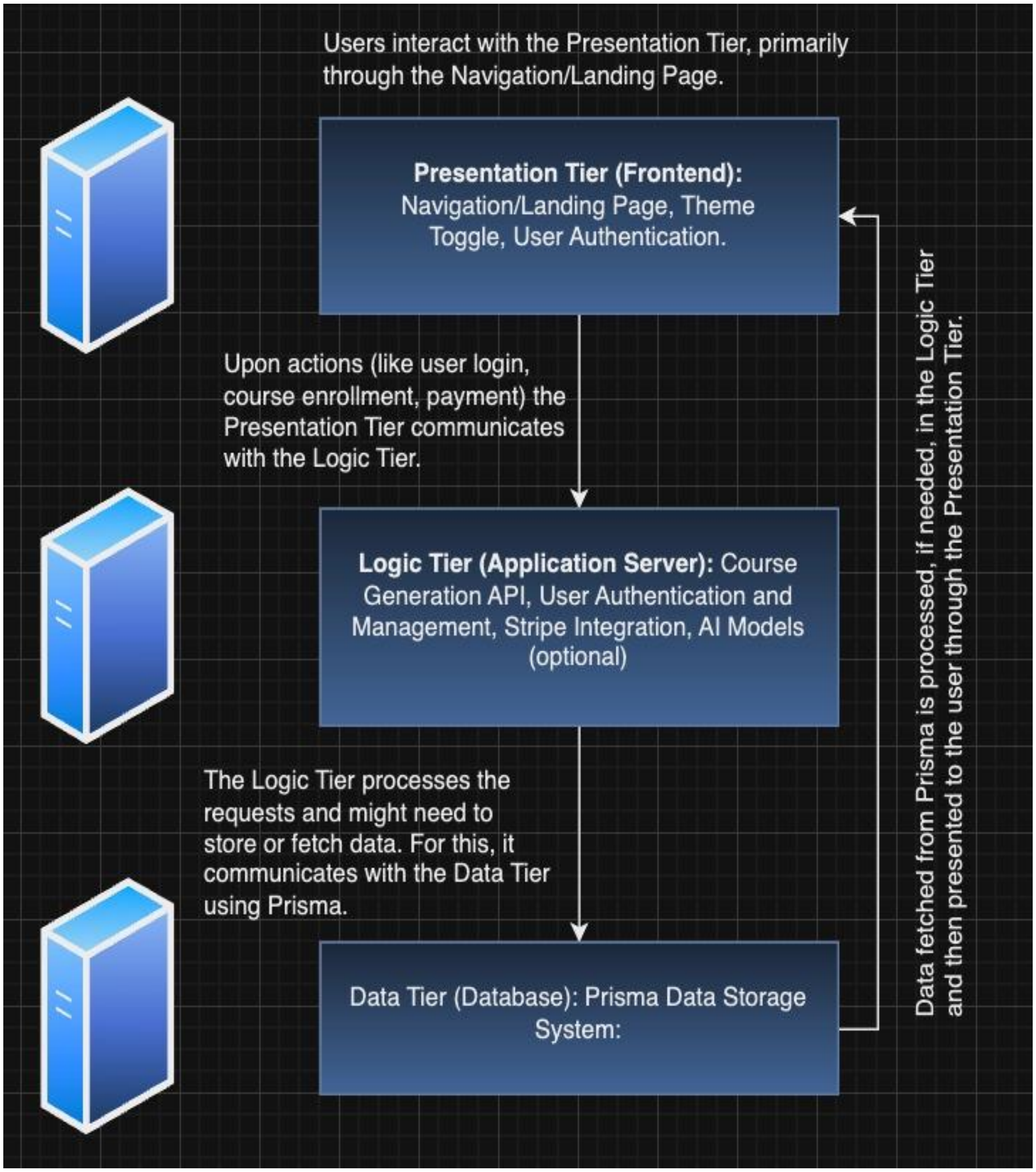
Considerations for Other Patterns:

1. Model-View-Presenter (MVP):
   The view is supposed to be 'dumb', with most of the logic in the presenter. Usually, it is a pattern employed for such purposes when thorough unit testing for UI components is demanded. It can work well for us, but we may already have something more complicated than our application on offer.

2. Model-View-ViewModel (MVVM):
   Developers today adopt MVVM as a way of separating logic in an application that has sophisticated UI. Presentational logic is stored in ViewModel as a middle layer.

N-Tier Application Landscape Pattern relevant to our app.:

Once again, as per what we have learnt in class, we feel this is the architecture that fits our use cases and concept, based on our understanding.


Users interact with the Presentation Tier, primarily through the Navigation/Landing Page.
Presentation Tier (Frontend): Navigation/Landing Page, Theme Toggle, User Authentication.
Upon actions (like user login, course enrollment, payment) the Presentation Tier communicates with the Logic Tier.
Logic Tier (Application Server): Course Generation API, User Authentication and Management, Stripe Integration, AI Models (optional)
The Logic Tier processes the requests and might need to store or fetch data. For this, it communicates with the Data Tier using Prisma.
Data Tier (Database): Prisma Data Storage System:
Data fetched from Prisma is processed, if needed, in the Logic Tier and then presented to the user through the Presentation Tier.

Design Description

We are trying to structure our project as a modern educational web application with a well-defined architecture, consisting of key components and API routes. Its design revolves around three primary functionalities: course creation, interaction with YouTube content, and a pro user upgrade feature.

<u>Structure:</u>

1. Course Creation: Users create courses by specifying a title and adding units. Chapters are structured and confirmed by users.
2. YouTube Interaction: The API route interacts with YouTube, fetching video transcripts and generating quiz questions. Questions are saved in the database for later use within courses.
3. Course Viewing: Users navigate through course chapters, viewing video summaries and interactive quiz cards.
4. Pro User Upgrade: Users have the option to upgrade to a pro account. The Stripe integration handles payment processing, enabling users to access premium features.

<u>Behavior:</u>

1. Course Creation:
    a. Users create courses by specifying a title and adding units.
    b. YouTube API is used to search for relevant videos, retrieve their transcripts, and generate questions.
    c. Chapters are structured and confirmed by users.
2. YouTube Interaction:
    a. The API route interacts with YouTube, fetching video transcripts and generating quiz questions.
    b. Questions are saved in the database for later use within courses.
3. Course Viewing:
    a. Users navigate through course chapters, viewing video summaries and interactive quiz cards.
4. Pro User Upgrade:
    a. Users have the option to upgrade to a pro account.
    b. The Stripe integration handles payment processing, enabling users to access premium features.

Motivations behind our design choices:

In the design of our project, we considered several key factors to align with the requirements identified during the analysis phase:

1. User-Friendly Interface: Our foremost objective was to create a user-friendly platform. We invested considerable effort in ensuring a seamless user experience, particularly during course creation. The intuitive interface allows users to add or remove course sections effortlessly.

2. Integration with YouTube: Given the primary requirement of generating course material from YouTube videos, we strategically chose to integrate with the YouTube platform. This decision facilitates the seamless extraction and utilization of video content, meeting a central project need.

3. Graceful Error Handling: Recognizing the importance of user satisfaction, we prioritized graceful error handling. In the event of an issue, the platform provides clear, user-friendly error messages, offering both an explanation of the problem and guidance on resolution.

4. Pro Subscription Model: To ensure the project's sustainability and continued operation, we introduced a Pro subscription option. This approach not only supports the project financially but also extends additional features, such as unlimited course creation, to subscribers, enhancing the overall value proposition.

5. Scalability and Modularity: A crucial aspect of our design philosophy involved building for the future. Leveraging RESTful APIs and adopting a modular component structure, we've created a platform that can seamlessly expand and integrate with other services or features in the future. This forward-looking approach aligns with our goal of continuously enhancing the platform.

Implementation Status

Our app is still in the development phase. We have set up the fundamental building blocks. Until now, we have been establishing/tinkering around with the database, authenticating the users using Google auth, and integrating the next auth to ensure a smooth session. Next, we will develop the necessary UI elements such as navbars and improve the form component critical to our course creation system. Further information as to the backlogs present, are available in this section below: Current Implementation Status (% of completion).

Epics and User Stories:

User Stories:

1. https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/issues/6
2. https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/issues/4

Epics:

1. https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/issues/5
2. https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/issues/7
3. https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/issues/8

Scrum challenges and positives:

Positives:

1. Breaking tasks down into small and achievable sprints has contributed towards making sure all team members are working towards meeting set targets by using Scrum. Weekly standup has promoted communication to a point where each member knows what the other person is saying or doing, thus, they are all on the same page.
2. The scrum process also gave us an experience so far, on how to adapt our project development to a higher standard, like how it is done in a very professional environment. E.g : Proper comments, Best practices for UX, Bouncing around and accepting/challenging others ideas.

Challenges:

1. We have experienced difficulties, especially in prioritizing backlogs and also fully understanding, actually how the SCRUM process needs to be carried out (Planning, managing, milestone creation, Epics, and all terms, as well as the relevant updates on GitHub).
2. Since we are getting started, we will soon try to pick up and follow some more essential things, if we have missed something for now in the scrum process.
3. In addition, estimating the duration for tasks precisely and holding up with sprint commitments on several occasions has not proven easy but also remains an educative process to be improved upon continually, which we will do from here on.

Current Implementation Status (% of completion):

So far, we have set up some only a couple of crucial elements.

Specifically:

1. Authentication Components: 100% complete.
   - https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/tree/main/src/app/api/auth/%5B...nextauth%5D
2. Database Setup: 100% complete.
   - https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/tree/main/prisma
   - https://github.com/AnirudhVijayaraghavan/CSYE7230-Software-Engineering-Project_A3/tree/main/src/lib
3. Initial UI Components(Landing page, UI forms): 80% complete.

Backlogs remaining (listed at a high level only):

1. Backend API setup: Essential systems such as Unsplash, Youtube API, Subscription management and Enhancement, Strict GPT configuration, and OpenAI API integration.
2. UI Components: We still need to produce chapter cards, multiple choice questions, and some other features that will make navigation and use of this site easier.
3. Deployment, Infrastructure setup, QA etc.

Libraries and Frameworks employed for now:

1. Next.js: Provides server side rendering capabilities.
2. TypeScript: Offers static typing, making our codebase more robust and less error-prone.

3. Tailwind CSS: Allows for rapid UI development with a utility-first approach.
4. Prisma: An excellent ORM, making database interactions more intuitive.
5. Planetscale Database: Ensuring scalability for our growing user base.
6. Stripe: For handling payments efficiently and securely.
7. Shad CN: Adopted for its impressive UI components.
8. Strict GPT JSON Extension: To simplify the transcript processing of our video content.
9. Open AI API and Youtube API: To furnish user content, quiz cards, and dynamic interaction with the user for the answer choices.

Tradeoffs considered:

We accounted for several things when choosing these tools. These frameworks and libraries increase our productivity and application robustness. However, they are associated with a learning curve for newcomers, as well as, having the dreaded scenario of not working together/integrating properly, due to the number of API's involved. We believe that even though there are some concerning shortcomings such as the learning curve, and potential integration issues, our tech. stack is sufficient for the job, and ensures both reliability and performance. The documentation, community forums, as well as examples of use cases/literature available for these technologies, are stellar.

Figma file link to high-level initial prototype:

https://www.figma.com/file/NV9dIPNqsJZqhLPf88quF0/AnirudhVijayaraghavan_Spring2023?type=design&node-id=1930-1762&mode=design

Presentation link:

https://docs.google.com/presentation/d/1qN6pLTzir6Ivfvv3E333xbBoJQbG6Xgr/edit#slide=id.p1