Scott McAnally

SWENG 837

03/02/2025

AI Powered Chatbot for University Advisors

**Problem Statement**

1. **Clearly Define the Problem the System Aims to Solve**

The problem that the AI Chabot is solving is the inefficiency, and timeliness of answering technical questions about the university courses and majors for potential future students. Currently advisors are taking down the questions and emailing professors to help answer the inquiries. In addition many students ask similar questions and some of the questions can be time consuming to answer. A couple examples of prospectus student questions are:

* What are the prerequisites for taking STAT 500?
* Is python covered as part of the Data Analytics Masters program?
* Am I able to take DAAN 541 If I have never used pandas?

The constant back and forth leads to delayed responses, overwhelmed academic advisors, and inconsistent answers.

The AI chatbot will automate the process of answering technical questions and it will provide accurate, timely and consistent responses to students while also freeing up advisors to focus on other initiatives.

1. **Specify the Functionalities the System Needs to Provide**

The core functionalities of the Chatbot are:

1. Answer Technical Questions about courses and majors that are offered
2. Real Time Data Integration on programs from the university’s systems
3. Natural Language Understanding of questions from students of differing backgrounds
4. Seamless Integration into the schools website
5. **Identify the Target Users and Their Needs**

| **User** | **Needs** |
| --- | --- |
| Students | * Quick and accurate answers to technical questions about courses and majors * Guidance on requirements and course selection * Ongoing access to course and program data without relying on an advisor |
| University Advisors | * Free up workload of answering student inquiries * Additional time to aid students with academic guidance, and career planning * Accurate and dependable information provided to students |
| University Administrators | * Improved student experience * Better alignment of academic advisors * Data on common questions which can be used to improve university processes |

1. **Outline any Business Goals the System Should Support**

The system must support the following Business Goals:

1. Improve the Student Experience
2. Boost Operational Efficiency
3. Refine Data Driven Decision Making
4. Maintain Consistency
5. Support Scalability
6. Promote Accessibility

**Requirements**

1. **Performance Requirements**

The performance requirements are required to ensure the system holds up under differing workloads.

1. Scalability:
   1. The system must hold up to 5,000 concurrent users without any impact to performance
   2. The system must be able to scale horizontally to accommodate the potential for growth in future users
2. Response Time:
   1. The chatbot should respond to user questions within 2 seconds of the request
   2. For queries that are more complex, the chatbot should respond within 5 seconds
3. Throughput:
   1. The system is able to process at least 100 queries per second during peak usage times (i.e. registration time)
4. Availability:
   1. The system must have a 99.9% availability, to ensure students and advisors can leverage it during any time of the day.
5. **Security Requirements**

The purpose for the security requirements are to ensure the system enforces the CIA triad, confidentiality, integrity and availability for its users.

1. Authentication:
   1. Users must be authentication with the university’s single sign on (SSO) process to ensure secure login
2. Authorization:
   1. Access to information should be relevant to the user's role. Students should only be able to view course content, and advisors can view broader program information.
3. Data Encryption:
   1. All data must be encrypted in transit using the highest version of TLS
   2. Sensitive data that’s stored in databases should be encrypted at rest using AES-256 encryption.
4. Data Privacy:
   1. The system must comply with the latest data privacy rules and regulations to safeguard student information.
5. Audit Logging:
   1. Robust logging of user interactions must be logged for audit and troubleshooting activities.
   2. Logs must contain timestamps, unique IDs, and query details without exposing sensitive information.
6. **Maintainability Requirements**

The maintainability requirements are to ensure that the system is easy to update, debug and maintain over time.

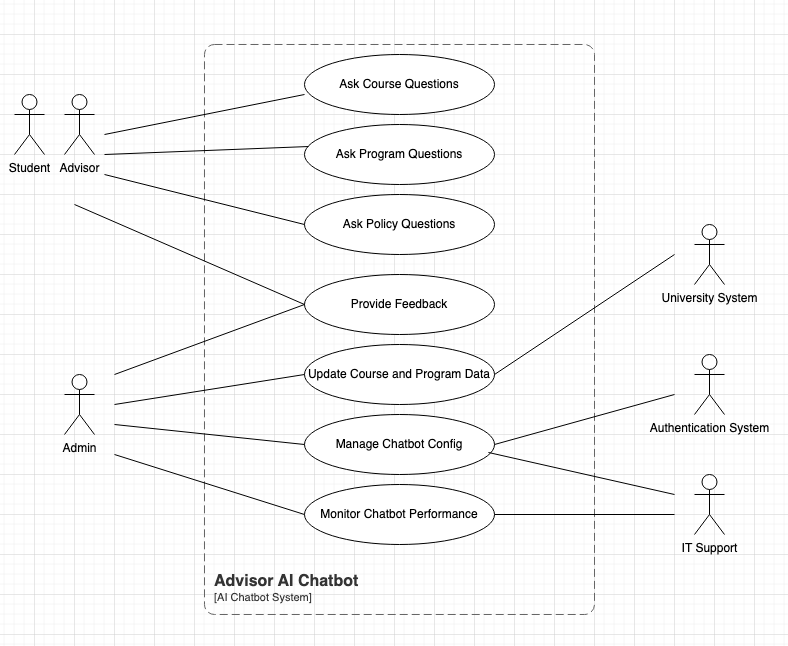
1. Code Modularity:
   1. The system should be structured with modular components to separate functionalities such as NLP, database connections, and API endpoints.
   2. The system must adhere to Object Oriented Design patterns to enable extension and reusability.
2. Documentation:
   1. Extension documentation shall be created:
      1. Architecture Diagrams (i.e UML class, sequence diagrams etc)
      2. API Swagger documentation to assist external consumers
      3. User documentation to assist advisors and administrators of the tool
3. Testing Strategies:
   1. Unit tests, integration tests, and load tests shall be incorporated to ensure proper functionality of components
   2. A CI/CD pipeline should be implemented to run automated tests and to deploy the application code.
4. Version Control:
   1. Git should be leveraged to enforce version control, branching to manage feature development, bug fixes and various releases.
5. **Other Non-Functional Requirements**

Additional non-functional requirements are defined to make sure the system operates smoothly and meets end user expectations.

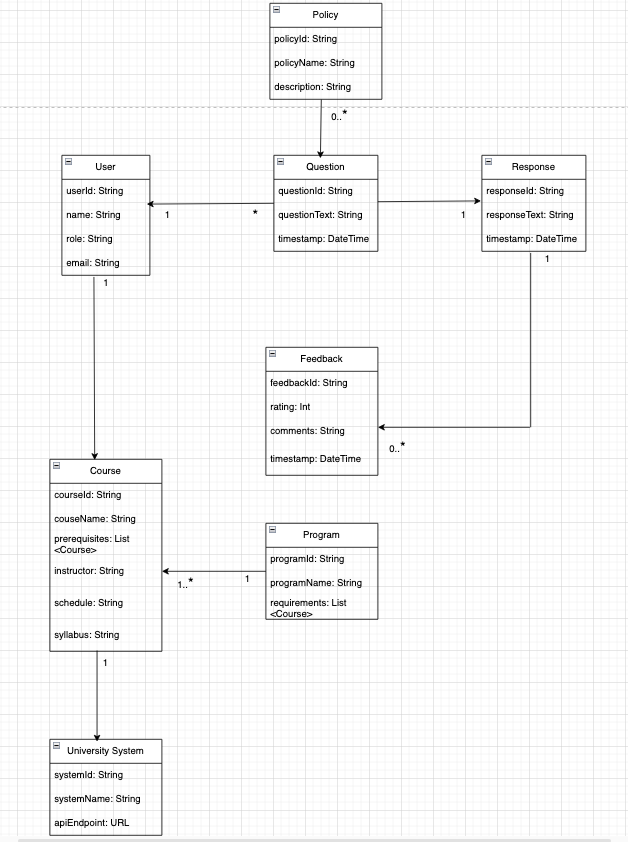
1. Usability:
   1. The chatbot interface must be intuitive to use and provide error messages and instructions for use
   2. Accessibility standards should be incorporated for users with disabilities
2. Interoperability:
   1. The chatbot must be able to integrate with the university’s existing wordpress website.
   2. RESTful APIs must provide integration to external services.
3. Cost Efficiency:
   1. Resource shall be optimize to limit the cost of operations such as serverless architecture

**System Design and Domain Modeling**

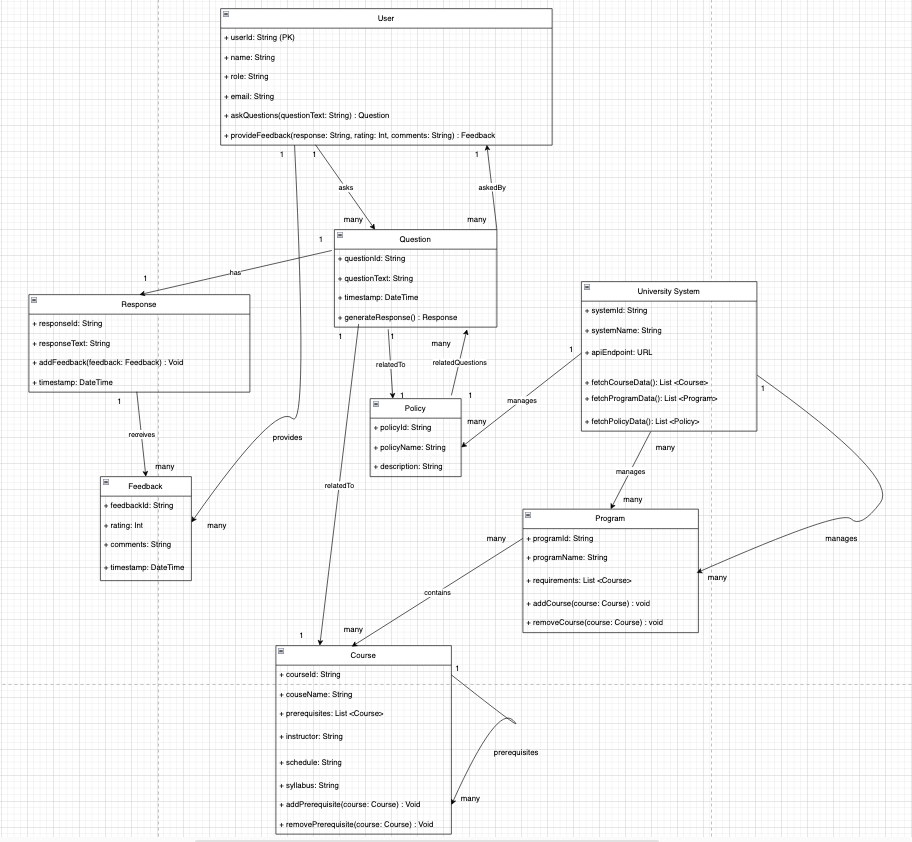
1. UML Case Diagram



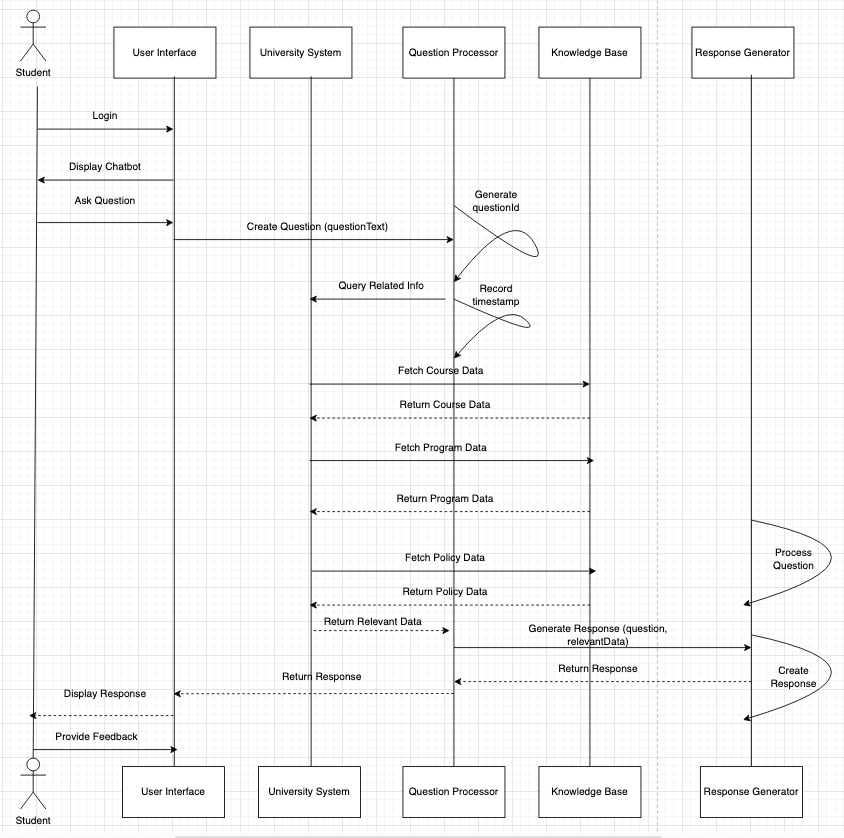
1. UML Domain Model



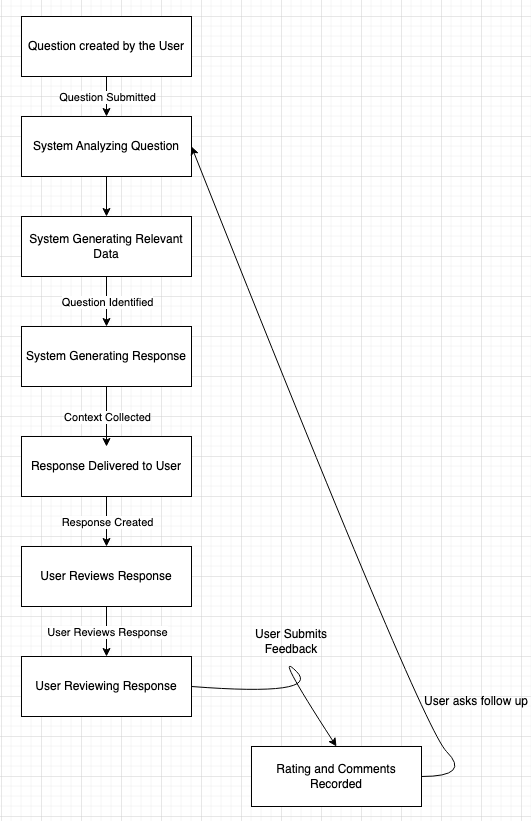
1. UML Class Diagram



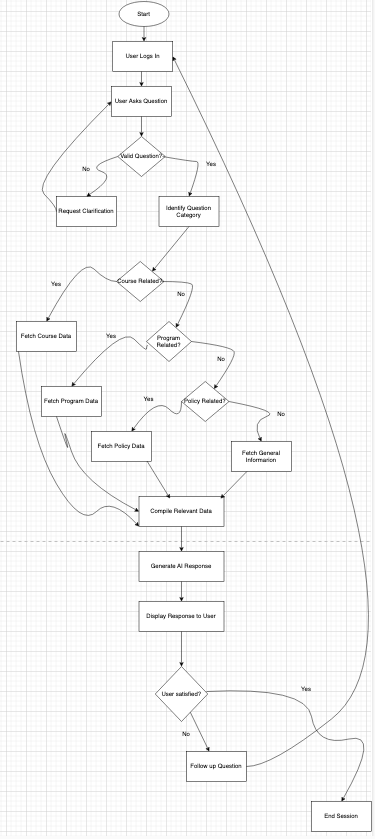
1. UML Sequence Diagram



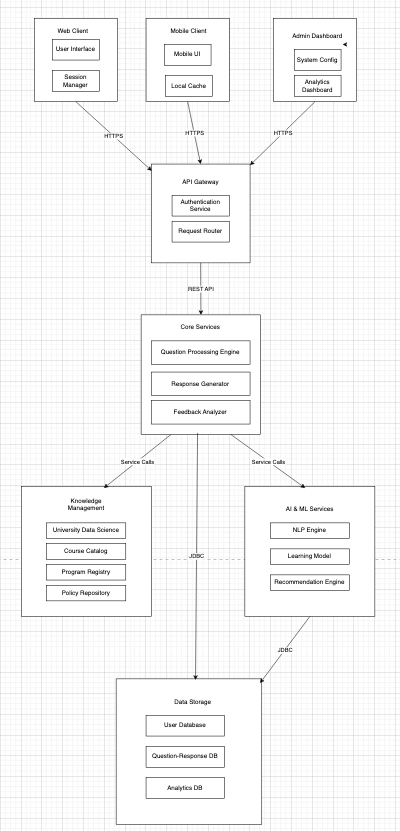
1. UML State Diagram



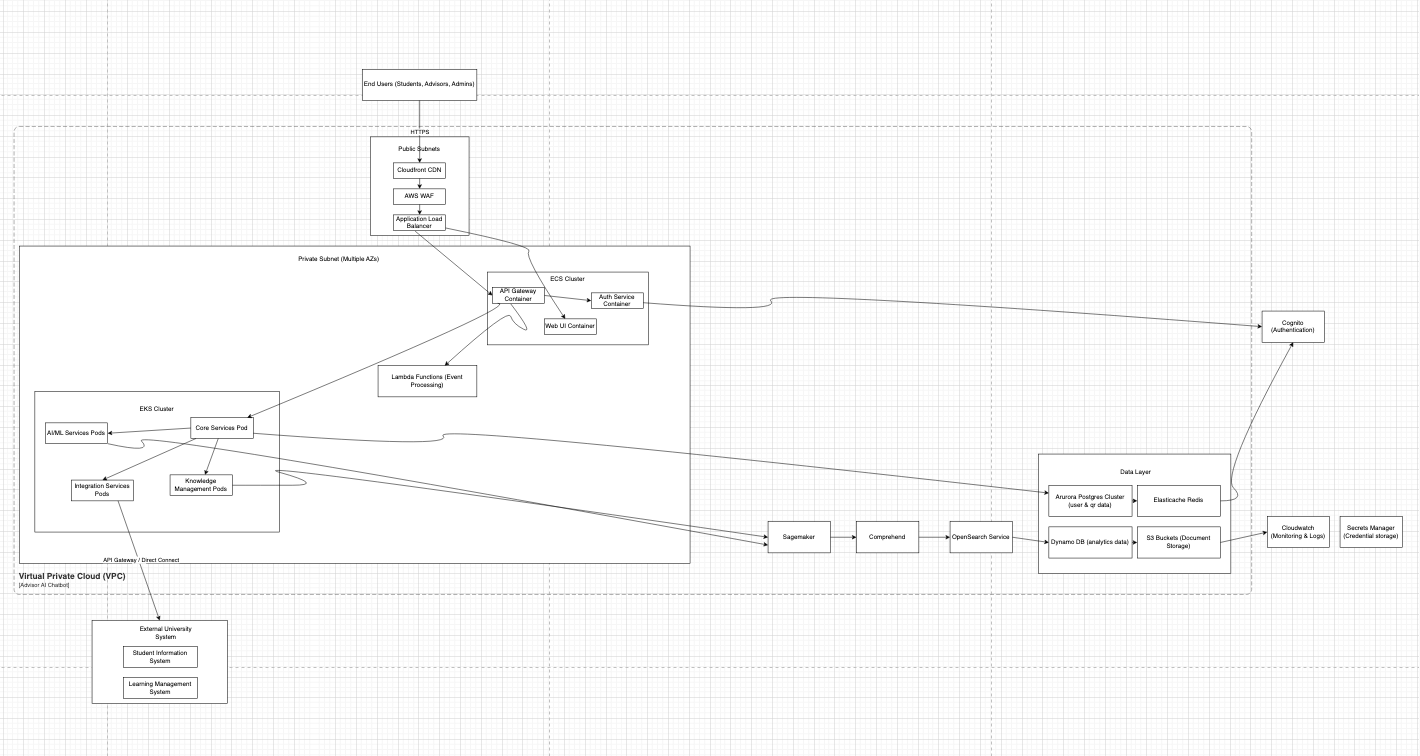
1. UML Activity Diagram



1. UML Component Diagram



1. Cloud Deployment Diagram



1. Skeleton Class and Table Definitions

| -- Users table CREATE TABLE users (  user\_id VARCHAR(36) PRIMARY KEY,  name VARCHAR(100) NOT NULL,  email VARCHAR(100) NOT NULL UNIQUE,  role VARCHAR(20) NOT NULL CHECK (role IN ('STUDENT', 'ADVISOR', 'ADMIN')),  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP );  -- Questions table CREATE TABLE questions (  question\_id VARCHAR(36) PRIMARY KEY,  question\_text TEXT NOT NULL,  timestamp TIMESTAMP NOT NULL,  asked\_by\_id VARCHAR(36) NOT NULL,  related\_to\_type VARCHAR(20) CHECK (related\_to\_type IN ('COURSE', 'PROGRAM', 'POLICY')),  related\_to\_id VARCHAR(36),  FOREIGN KEY (asked\_by\_id) REFERENCES users(user\_id) );  -- Responses table CREATE TABLE responses (  response\_id VARCHAR(36) PRIMARY KEY,  response\_text TEXT NOT NULL,  timestamp TIMESTAMP NOT NULL,  question\_id VARCHAR(36) NOT NULL,  FOREIGN KEY (question\_id) REFERENCES questions(question\_id) );  -- Feedback table CREATE TABLE feedback (  feedback\_id VARCHAR(36) PRIMARY KEY,  rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5),  comments TEXT,  timestamp TIMESTAMP NOT NULL,  provided\_by\_id VARCHAR(36) NOT NULL,  given\_on\_id VARCHAR(36) NOT NULL,  FOREIGN KEY (provided\_by\_id) REFERENCES users(user\_id),  FOREIGN KEY (given\_on\_id) REFERENCES responses(response\_id) );  -- Courses table CREATE TABLE courses (  course\_id VARCHAR(36) PRIMARY KEY,  course\_name VARCHAR(100) NOT NULL,  syllabus TEXT,  instructor VARCHAR(100),  schedule TEXT );  -- Course prerequisites (self-referencing relationship) CREATE TABLE course\_prerequisites (  course\_id VARCHAR(36) NOT NULL,  prerequisite\_id VARCHAR(36) NOT NULL,  PRIMARY KEY (course\_id, prerequisite\_id),  FOREIGN KEY (course\_id) REFERENCES courses(course\_id),  FOREIGN KEY (prerequisite\_id) REFERENCES courses(course\_id) );  -- Programs table CREATE TABLE programs (  program\_id VARCHAR(36) PRIMARY KEY,  program\_name VARCHAR(100) NOT NULL );  -- Program\_courses (many-to-many relationship) CREATE TABLE program\_courses (  program\_id VARCHAR(36) NOT NULL,  course\_id VARCHAR(36) NOT NULL,  is\_required BOOLEAN DEFAULT false,  PRIMARY KEY (program\_id, course\_id),  FOREIGN KEY (program\_id) REFERENCES programs(program\_id),  FOREIGN KEY (course\_id) REFERENCES courses(course\_id) );  -- Policies table CREATE TABLE policies (  policy\_id VARCHAR(36) PRIMARY KEY,  policy\_name VARCHAR(100) NOT NULL,  description TEXT );  -- University systems table CREATE TABLE university\_systems (  system\_id VARCHAR(36) PRIMARY KEY,  system\_name VARCHAR(100) NOT NULL,  api\_endpoint VARCHAR(255) NOT NULL );  -- Session tracking CREATE TABLE user\_sessions (  session\_id VARCHAR(100) PRIMARY KEY,  user\_id VARCHAR(36) NOT NULL,  login\_time TIMESTAMP NOT NULL,  last\_activity TIMESTAMP NOT NULL,  ip\_address VARCHAR(45),  user\_agent TEXT,  FOREIGN KEY (user\_id) REFERENCES users(user\_id) );  -- Analytics tracking CREATE TABLE analytics\_events (  event\_id VARCHAR(36) PRIMARY KEY,  event\_type VARCHAR(50) NOT NULL,  event\_data JSON,  user\_id VARCHAR(36),  timestamp TIMESTAMP NOT NULL,  FOREIGN KEY (user\_id) REFERENCES users(user\_id) );  -- Indexes for performance CREATE INDEX idx\_questions\_asked\_by ON questions(asked\_by\_id); CREATE INDEX idx\_questions\_timestamp ON questions(timestamp); CREATE INDEX idx\_responses\_question ON responses(question\_id); CREATE INDEX idx\_feedback\_response ON feedback(given\_on\_id); CREATE INDEX idx\_program\_courses ON program\_courses(program\_id, course\_id); CREATE INDEX idx\_analytics\_timestamp ON analytics\_events(timestamp); CREATE INDEX idx\_analytics\_user ON analytics\_events(user\_id); |
| --- |

1. Design Patterns

GRASP

1. Information Expert - An example of the information expert design pattern can be seen in the Question Class as it contains the question from the user and contains enough supporting details for the model to create a response. Ultimately this pattern puts the responsibility of the Question Class and makes the design more cohesive.
2. Creator - The AI advisor chatbot leverages the creator pattern in that the User class is in charge of creating the Question and Feedback objects. For the application, it is the User that is the initiator of information and thus it flows to the Question and Feedback objects as everything is based on the users input.
3. Controller - Within the AWS architecture pattern the APi Gateway service is used to control or direct all incoming requests to different services. This helps to provide a singular entry point and it separates the backend processing from the front end components.
4. Low Coupling - The overall design leverages micro services with individual interfaces in between the different components. The pattern helps to provide flexibility and maintainability for future development.

SOLID

1. Single Responsibility Principle - When designing the system, each class only services on purpose. An example is the Policy class and how it only contains policy data and actions which helps to improve maintainability.
2. Open / Closed Principle - The relatedTo attribute within the Question class is able to reference different entities such as ‘Course’, ‘Program’, or ‘Policy’. This approach enables for more simple extensions without heavily modifying the code.
3. Liskov Substitution Principle - The User class serves as a class and has subclasses for Student, Advisor and Admin. This enables each role to extend on the User functionality and promotes reusability and changeability.
4. Interface Segregation Principle - There is a distinct separation of the University System interface for pulling specific school data. This pattern allows clients to only depend on certain methods within the interface and not the entire interface.

GOF

1. Strategy - The system allows for differing NPL algorithms which can be beneficial for processing different types of questions from Students, Advisors or Admins. This gives the system the autonomy to determine which NLP to use based on the question context provided by the user.
2. Facade - An example of the Facade pattern can be seen in the Knowledge Management component and how it simplifies access to different data sources. This aids in creating a unified interface so that other subsystems can be more easily used.
3. Adapter - The application has been designed with an integration layer to simplify external connections to the University data sources. This helps bridge the gap between external systems and makes them compatible with internal interfaces for the chatbot.

Microservices

1. Database per Service - The system was designed with different databases for data, questions/responses and analytics. This design helps enable independent scaling and makes service evolution more simple.
2. Event Sourcing - The AI Advisor chatbot has an events table with the analytics database which is used to track usage and behavior. This service helps provide auditing and debugging within the system and it does not impact the applications main functionalities.
3. Circuit Breaker - All of the service API calls to external university systems are protected with API gateway. This design helps to prevent cascading failures if an external system is unavailable for example.