Towson Academic Pathway (TAP)

Fall 2024

TAP Dev

Nyle Clements, Romerico David, Tim DeLloyd, Mitchell Griff, Zachary Hall

11/05/2024

Assignments Board

Assignee	Email	Task	Duration (hours)	Dependency	Due Date
Nyle Clements	nclemen1@students.towson.edu	Task 6: Formatting sprint 2 report paper	3	Google Drive MS Visio	11/5/24
Romerico David	rdavid6@students.towson.edu	Task 5: Specifying database table and database management system Task 6: Formatting sprint 2 report paper	7	Google Drive	11/5/24
Timothy DeLloyd	tdello3@students.towson.edu	Task 3: Revising the problem statement Task 5: Identify objects and create class diagrams	10.5	Google Drive Gliffy	11/5/24
Mitchell Griff	mgriff30@students.towson.edu	Task 1: Make a scheduling chart and assign tasks. Task 4: List all use cases, create use case diagrams, Task 4: Describe the requirements for each use case.	4	Google Drive MS Visio	11/5/24
Zachary Hall	zhall6@students.towson.edu	N/A	N/A	N/A	11/5/24

Problem Statement

Towson Academic Pathway (TAP) is a web application designed to assist Towson University students in planning and organizing their academic path. TAP generates personalized degree completion plans as well as potential schedules for upcoming semesters and allows users to adjust those plans and schedules dynamically based on their preferences, progress, and Towson's academic policies.

TAP is for incoming and current students of all majors at Towson University, regardless of whether they are full or part-time. Auditors and non-students who are merely considering a select few classes may not find it as useful.

For a student, the process of planning out what courses one will take over the next few years can be very difficult and overwhelming. TAP eases out that process using artificial intelligence. Additionally, plans may need to change if a student changes their major or is not able to take a course when planned because it was not offered or because the course did not fit their schedule. TAP can also help in these instances as well, making the process of replanning less stressful for the student.

TAP also helps students schedule classes for an upcoming semester. Using information from a students' career path, such as the courses they have taken and what they need to take, and allowing for various preferences, class meeting time or class professor, TAP can provide the student with scheduling options for the upcoming semester from Towson's list of offered courses. The logistical nightmare of planning a schedule for the next semester is often a time-consuming and stressful obligation for students; TAP would help alleviate the stress and free up time to let students focus on other obligations in their lives.

For Towson students, the only current alternative to TAP is to work manually, for example, with an Excel spreadsheet. This is time consuming and can distract a student from other obligations they may have in their lives.

Students have many responsibilities and obligations during a semester; they have classes, may have jobs, and may even be parents. The logistical nightmare of determining what classes one should take in the upcoming semester is always stressful, time-consuming, and distracting. TAP helps busy Towson students by taking the scheduling process off their backs, making their lives just a bit easier.

The top-level objective of TAP is to provide the target customer, a Towson student, with a (usually four-year) degree plan and a list of potential schedules for the upcoming semester for them to choose from.

TAP considers what courses the student has taken, what courses the student needs to take, and what preferences the student has regarding course sections such as meeting time and whether a section is online to output a degree plan.

The scheduler is limited to semesters where the course offerings have already been finalized.

TAP does not have any known competitors, which makes TAP new and innovative.

AI-based scheduling software has already been built for other purposes, such as Data317, which has software to help school districts create class schedules across a whole school; it only makes sense, then, that our scheduling system would be buildable.

By simply inputting preferences and requirements, students can use TAP to generate degree plans and several potential schedules, making the process of planning their academic plan simpler, efficient, and less stressful.

Our design for TAP uses “service” objects (adapters) to handle calls between components and the database and between components and the OpenAI API, in accordance with the Dependency Inversion Principle. The objects stored in the databases, Course, CourseSection, and StudentData are examples of data abstraction. Finally, by separating CourseSections from Courses, we decrease coupling by only having CourseSection objects depend on Course objects and not vice versa, which follows rationally when one considers that every CourseSection must be of some Course.

Requirements

User Requirements:

Use Case Number:1 **Use Case Name:** Log in/log out
Actors: User, AuthenticationComponent, Database
Goal in context: To grant the user access to their account if they are logging in or to lock their account if they are logging out.
Alternate Path: If the user is attempting to log in the user fails to provide valid credentials and therefore fails to log in. If the user is not logged in, logging out will not be an accessible option.
Pre-condition: The user has registered a valid account in the database. If the user is logging out, they must already be logged in.

Use Case Number: 2 **Use Case Name:** Submit personal information
Actors: User, DegreePlannerComponent, Database
Goal in context: The DegreePlannerComponent prompts the user for their personal information, their chosen degree path, existing academic progress and quantifiable goals (graduation date, etc.). The user complies and enters the requested information which is then saved.
Alternate Path: The user refuses to submit the requested information. The user's account information remains empty until the user decides to submit the requested information. Until then, the user can continue to use the application, but functionality will be limited since critical information about the user is missing.
Pre-condition: The user has successfully logged into a valid account.

Use Case Number: 3 **Use Case Name:** Modify account information
Actors: User, StudentDataComponent, Database
Goal in context: The StudentDataComponent allows the user to modify their account information.
Alternate Path: There is no information to modify as a result of a refusal of the FormComponent information request. The user will be prompted to enter their information.
Pre-condition: The user has successfully logged into a valid account.

Use Case Number: 4 **Use Case Name:** Catalog access
Actors: User, CourseCatalogComponent, Database
Goal in context: The user uses the filters and search criteria to successfully locate a desired course.
Alternate Path: The user enters invalid criteria or there is no course that matches their conditions. In such a case, no course is displayed.
Pre-condition: The user has successfully logged into a valid account.

Use Case Number: 5 **Use Case Name:** Course selection
Actors: User, ActiveSemesterPlannerComponent, CourseCatalogComponent, Database
Goal in context: The user selects courses from the CourseCatalogComponent and adds them to their enrollment list for the coming semester.

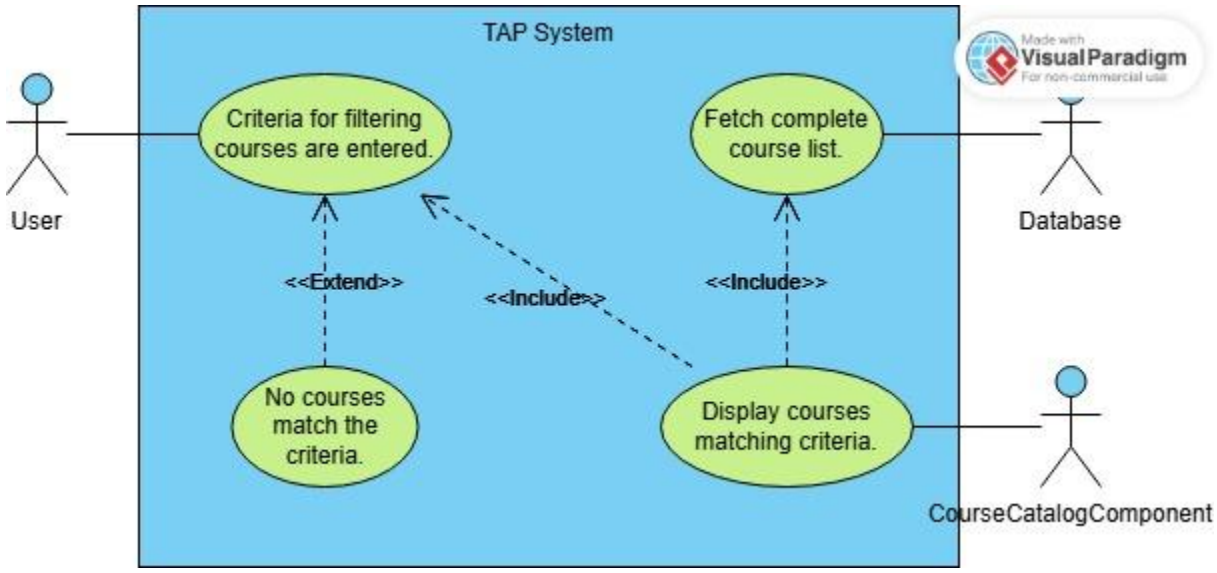
Alternate Path: If the user does not have the prerequisite courses for their selection, the selection is rejected. If the selected course is full, the user is given the option to join a waitlist.
Pre-condition: The user has successfully logged into a valid account and there is an available course presented in the CourseCatalogComponent.

Use Case Number: 6 **Use Case Name:** Degree planning
Actors: User, DegreePlannerComponent, CourseCatalogComponent, Database
Goal in context: The user adds courses to their plan, indicating what courses they plan to take over the lifetime of their degree and in what order.
Alternate Path: If the user selects a course such that it would be taken before its prerequisites, the selection is rejected. If the plan the user creates would not fulfill their degrees requirements, their plan is rejected.
Pre-condition: The user has successfully logged into a valid account, they have chosen a degree path, and there are valid courses presented in the CourseCatalogComponent.

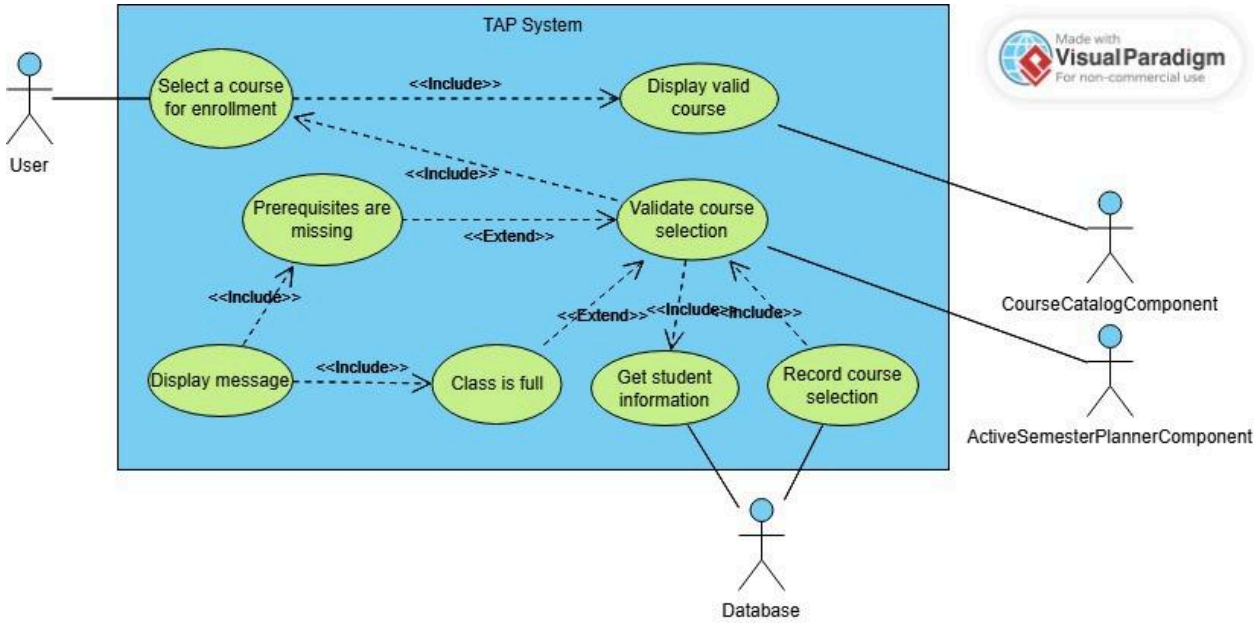
Use Case Number: 7 **Use Case Name:** Search FAQs
Actors: User, FAQComponent
Goal in context: The user finds an FAQ that matches their problem.
Alternate Path: There is no FAQ that matches the user's problem, they are given the option to submit their question in an email.
Pre-condition: The user has successfully logged into a valid account and there is a valid email address for the question to be sent to.

Use Case Diagrams:

Use Case 4:



Use Case 5:



System Requirements:

Requirement number: 1 **Use Case number:** 1
Introduction: This requirement describes how a user interacts with the AuthenticationComponent to log in or log out.
Inputs: User's student email and password.
Requirements Description:

1) The login page presents a prompt to the user for their student email and password.
2) The input from the user is validated to ensure the email is formatted correctly and the entered password meets our password conventions.
    a) Extension: If the inputs are not formatted correctly, they are rejected before checking the database.
3) The user inputs are compared to the data that corresponds to that student email in our database.
4) If the credentials match, a successful login message is displayed to the user, an access token is provided, and the user is directed to the TAP homepage.
    a) Extension: If the credentials do not match, an error message is displayed to the user and they remain on the login page.

Outputs: Message indicating whether login was successful.

Requirement number: 2 **Use Case number:** 2
Introduction: This requirement describes the user's interaction with the FormComponent when prompted to enter their student information and the process of saving that information.
Inputs: User's pertinent information: personal identifying information, current degree path, existing academic progress, quantifiable goals(graduation date, GPA, etc.)

Requirements Description:

1) The FormComponent presents a prompt to the user asking if they want to submit their pertinent information.
2) After an affirmative response, the user is presented with a series of forms to be populated by the user with their information.
    a) Extension: If the user refuses, they are allowed to continue using the application. However, functionality will be severely limited due to a lack of critical information.
3) User's inputs are validated for reasonableness and security. Ex: GPA goal is on a 4.0 scale, date is in a MM/DD/YYYY format, and alphanumeric responses are screened for prohibited characters, such as any that might indicate an injection attack.
    a) Extension: If any invalid or suspicious inputs are detected, the input(s) is rejected and a message is displayed indicating the problematic input(s).
4) User's responses are stored in the database and attached to their student profile to be recalled later.

Outputs: Message indicating successful submission of user's information.

Requirement number: 3 **Use Case number:** 3

Introduction: This requirement describes how a user can use the SettingsComponent to modify their account information and preferences.

Inputs: User chooses which items to modify and what modifications to make.

Requirements Description:

1) User navigates to the settings page.
2) User selects which information field or preference to modify.
    a) Extension: If the user has not already entered their pertinent information through the DegreePlannerComponent, they will be prompted to do so.
3) The SettingsComponent displays a field for the user to enter their modification.
4) User enters their modified information.
5) User's modifications are validated for reasonableness and security. Ex: GPA goal is on a 4.0 scale, date is in a MM/DD/YYYY format, and alphanumeric responses are screened for prohibited characters, such as any that might indicate an injection attack.
    a) Extension: If any invalid or suspicious inputs are detected, the input(s) is rejected and a message is displayed indicating the problematic input(s).
6) The SettingsComponent saves the user's modifications to the database.

Outputs: Message indicating that the selected information was successfully updated.

Requirement number: 4 **Use Case number:** 4

Introduction: The user uses the CourseCatalogComponent to locate a desired course within the catalog.

Inputs: Criteria from the user to narrow the list of courses displayed.

Requirements Description:

1) User navigates to the catalog page.
2) The CourseCatalogComponent displays a blank page and fields for relevant criteria.

3) User enters desired criteria into any number of fields.
    a) Extension: If the user enters no criteria, the page will continue to be blank.
4) The CourseCatalogComponent filters the courses by the entered criteria and displays those that meet them.
    a) Extension: If the selected criteria do not match any course in the catalog, a message will display indicating that fact.

Outputs: Courses meeting the criteria entered by the user.

Requirement number: 5 **Use Case number:** 5
Introduction: This requirement describes how a user may select a course through the CourseCatalogComponent and enroll in it through the ActiveSemesterPlannerComponent.
Inputs: A valid course selection from the CourseCatalogComponent.
Requirements Description:

1) The user selects a course from the list available through the CourseCatalogComponent.
    a) Extension: If there is not a valid course displayed through the CourseCatalogComponent, no course can be selected.
2) The user selects to enroll in the course.
    a) Extension: If the user has not completed the selected course's prerequisites, the selection will be rejected and a message will display indicating the rejection and its cause.
3) The course is added to the student's enrollment plan.
    a) Extension: If the class is full, the user may instead choose to join the waitlist.
4) The new enrollment plan is saved to the student's profile in the database.

Outputs: Message indicating successful enrollment.

Requirement number: 6 **Use Case number:** 6
Introduction: This requirement describes how a user may add a course from the CourseCatalogComponent to their Degree Completion Plan using the DegreePlannerComponent.
Inputs: A valid course selection from the CourseCatalogComponent.
Requirements Description:

1) The user selects a course from the list available through the CourseCatalogComponent.
    a) Extension: If there is not a valid course displayed through the CourseCatalogComponent, no course can be selected.
2) The user selects to add it to their Degree Completion Plan.
3) The DegreePlannerComponent prompts the user to choose a semester to add the course to, depending on when the course is offered.
    a) Extension: If the course is not offered during a semester the user will be enrolled, the selection is rejected and a message will display indicating the rejection and its cause.
4) The course is added to the semester indicated by the user.

   a) Extension: If the course indicated falls before the user is scheduled to take the course's prerequisites, the user will be prompted to add the relevant courses to their Degree Completion Plan.
   b) If the user declines to add those classes, a warning will be indicated on their Degree Completion Plan indicating that the chosen courses are invalid in their current order.

Outputs: Message indicating successful addition to the Degree Completion Plan.

Requirement number: 7 **Use Case number:** 7

Introduction: This requirement describes how the user can use the FAQComponent to find an FAQ response that addresses their issue.

Inputs: Navigation inputs from user.

Requirements Description:

1) User navigates to the FAQ page.
2) The FAQ Component displays the list of available FAQ responses
3) The user selects a FAQ response that addresses their issue.
   a) Extension: If there are no responses that address the user's issue, the user will be given the option to submit their issue in an email form. The information in the form will be stored in the database for later retrieval.

Outputs: FAQ response matching the user's issue.

System Modeling

Class Analysis

- **Course**
    - **Description**: Contains comprehensive course information, including both active courses (for upcoming semesters like Spring 2025) and general course data across all terms.
    - **Fields**:
        - _id? (String): Unique Identifier
        - institution (String): Institution code
        - acadCareer (String): Academic career
        - subject (String): Subject code
        - catalogNumber (String): Catalog number
        - courseTitle (String): Title
        - termsOffered (String): The specific term(s) for which the course is offered
        - description (String): Course description
        - units (String): Credit hours
        - gradingBasis (String): Grading basis
        - components (Array of Objects):
            - component (String): Type
            - required (Boolean): Whether it's required
        - isActive (Boolean): A flag to indicate if the course is currently active. true means the course is available for an upcoming term; false means it is inactive.
        - campus (String): Campus name
- · **CourseComponent**
    - Description
        - Used by SemesterComponent and CourseCatalogComponent to display a Course object
    - Attributes
        - Course: a Course object
        - String: a generic message about whether special approval is required for a Course
    - Methods
        - Display(): creates the display of the CourseComponent object
- **CourseSectionData** (Active Courses for Specific Terms)
    - Description
        - Contains information about courses that are currently active and available for specific upcoming semesters. This collection can dynamically update each term to reflect only those courses being offered.
    - Attributes
        - _id? (String): Unique Identifier

        - courseBridgeId (String): Id to course bridge
        - termActive (String): The term the course is active
        - classNumber (Number): Unique class number for the specific offering
        - classSection (String): Section ID
        - component (String): Class type
        - session (String): Session description
        - startDate (Date): Start date of the course for the term.
        - endDate (Date): End date of the course for the term.
        - meetings (Array of Objects):
            - days (String): Meeting days
            - startTime (String): Start time
            - endTime (String): End time
            - buildingCode (String): Building code
            - room (String): Room number
            - facilityDescr (String): Facility description
        - instructors (Array of Objects):
            - name (String): Instructor's name
            - email (String): Instructor's email
        - enrollmentInfo (Object): Enrollment details.
            - status (String): Enrollment status
            - classCapacity (Number): Class capacity
            - enrollmentTotal (Number): Current enrollment
            - enrollmentAvailable (Number): Seats available
            - waitListCapacity (Number): Waitlist capacity
            - waitListTotal (Number): Waitlist total
        - location (String): Location
- **CourseSectionComponent**
    - Description
        - Used by SemesterComponent and CourseSectionCatalogComponent to display a Course object
    - Attributes
        - CourseSection: a CourseSection object
        - Course: a Course object (the one associated with the CourseSection)
        - String: a generic message about whether special approval is required for a Course
        - String: a message displayed only if the Course has multiple meeting times
    - Methods
        - Display(): creates the display of the CourseSection object along with the Course object
- **CourseService**
    - Description
        - Handles queries to the Course database
    - Methods
        - Implements operations for CRUD

- **CourseSectionService**
  - Description
    - Handles queries to the CourseSection database
  - Methods
    - Implements operations for CRUD
- **CourseBridge**
  - Description
    - Acts as a middleman between CourseData and CourseSectionData. It contains information about each course and includes an array of specific course sections/class offerings.
  - Attributes
    - _id? (String): Unique identifier
    - courseId (String): Id for course data linking each active course section to its general course details
    - activeClasses (Array of ObjectIds): Array of ids of CourseSectionData documents, where each document represents a specific class offering for this section in the specified term.
- **CourseCatalogComponent**
  - Description
    - A page listing all courses available at Towson University; makes calls to the CourseService
  - o Attributes
    - int 2D array: for each subject, an array of course IDs offered by that department
  - o Methods
    - Populate(): populates the attribute array with the Course IDs via a call to the CourseService
    - Display(): iterates over the attribute array; at the top level, it creates a heading for each subject; at the lower level, for each ID, it uses the ID to obtain the Course object from the CourseService and passes it to a CourseComponent to display it, and also handles calls to the CourseService for prerequisite and corequisite displays
- **CourseSectionCatalogComponent**
  - Description
    - An object serving list of courses available at Towson University; makes calls to the CourseSectionService and the CourseService
  - Attributes
    - int 2D array: for each department, an array of course section IDs offered by that department
  - Methods
    - Populate(): populates the attribute array with the Course IDs via a call to the CourseService
    - Display(): iterates over the attribute array; at the top level, it creates a heading for each subject; at the middle level, it creates a header for each

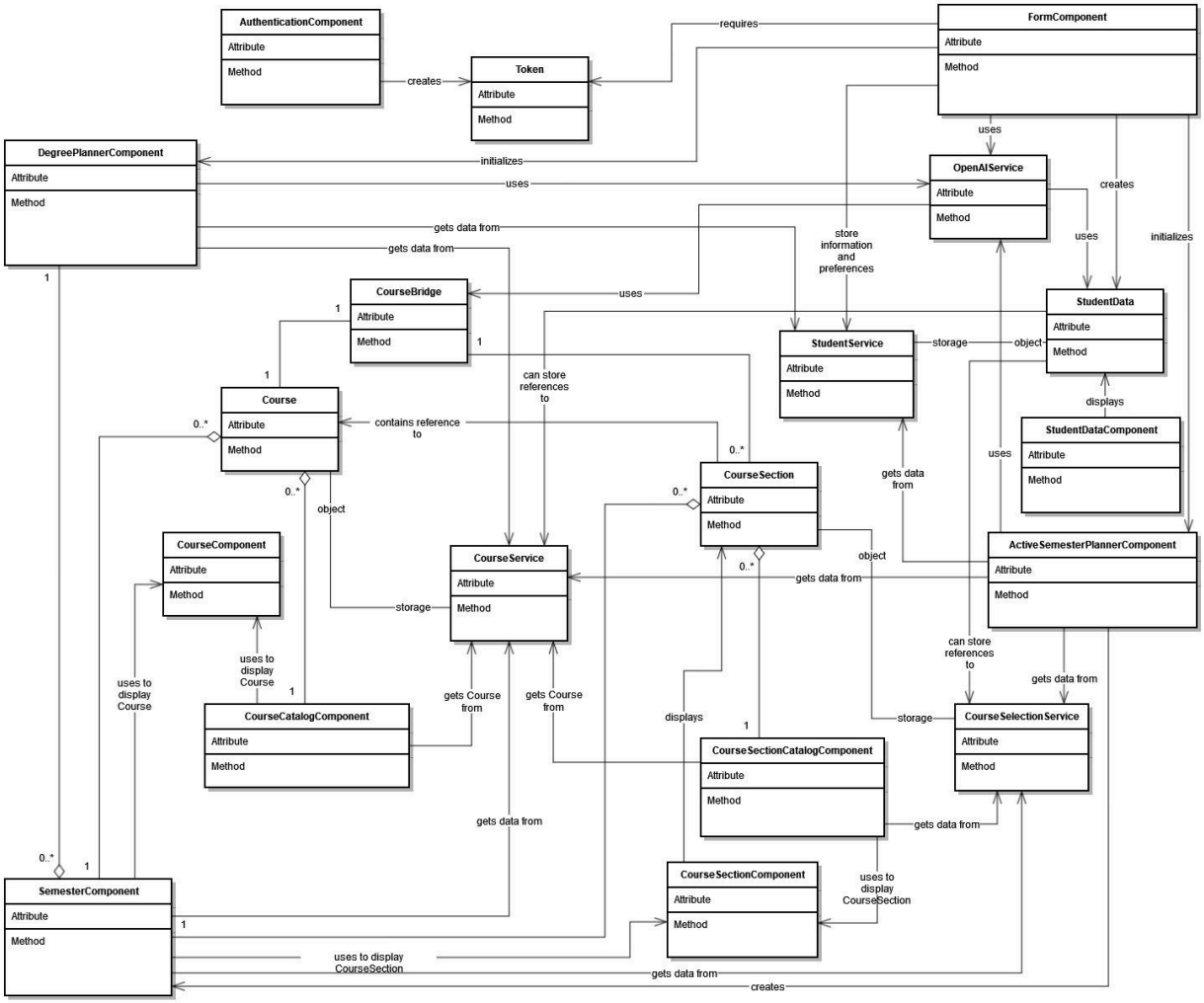
course; at the lower level, it uses the ID to obtain the CourseSection and Course objects from the CourseSection and CourseServices to build a CourseSectionComponent, and also makes calls to the CourseService for the prerequisite and corequisite displays

- **SemesterComponent**
    - Description
        - Displays either Course or CourseSection data for a semester
    - Attributes
        - int: semester ID (a 1, followed by the last 2 digits of the year, followed by a 1 if the January minimester, a 2 if the spring semester, a 3 if the summer semester, and a 4 if the fall semester) (e.g. “1244”, the current semester)
        - object array: a list of Course or CourseSection objects
        - String: name of the semester (e.g. “Fall 2024”)
    - Methods
        - Initializer
        - CreatePlan(): generates a header with the name of the Semester, figures out whether Course or CourseSection objects were given, then iteratively generates either CourseComponent objects or CourseSectionComponent objects
- **DegreePlannerComponent**
    - Description
        - The page displaying a student’s degree plan
    - Attributes
        - StudentInformation: a StudentInformation object
    - Methods
        - Private GetCourseIDs(): int 2D array; gets a 2D array of course IDs via a call to the OpenAIService with the StudentInformation object
        - Display(): void; generates a display via calls to the CourseService and creating SemesterComponent objects
- **ActiveSemesterPlannerComponent**
    - Description
        - The page displaying a student’s class plan for the next semester
    - Attributes
        - StudentInformation: a StudentInformation object
    - o Methods
        - Private GetCourseSectionIDs(): int 2D array; gets a 2D array of course IDs via a call to the OpenAIService with the StudentInformation object; checks to make sure the StudentInformation object contains an ActiveSettings object that is not null
        - Display(): void; generates a display via calls to the CourseSectionService, CourseService, and creating SemesterComponent objects
- **StudentService**
    - Description

        - The database containing all of the StudentInformation data
    - Attributes
        - The StudentInformation objects
    - Methods
        - Implements all methods of CRUD
- **OpenAIService**
    - Description
        - An adapter for obtaining information from ChatGPT; it has access to the CourseService and CourseSectionService
    - Attributes
        - Finagle
    - Methods
        - CreateDegreePlan(): Course 2D array
        - CreateActiveSemesterPlan(): CourseSection 2D array
- **StudentData**
    - **Description**:
        - Stores student records, academic plans, preferences, and scheduling information
    - **Attributes**:
        - _id? (String): Unique identifier
        - studentId (String): Unique Towson University student identifier
        - firstName (String): First name
        - lastName (String): Last name
        - academicInfo (Object): User's current academic standing/information
        - preferences (Object): User preferences
        - degreePlan (2D Array of Course Objects): top level is semester, bottom level is Course objects for that semester
        - activeSemesterPlan (Array of CourseSections): CourseSection objects for that semester
- **FormComponent**
    - Description
        - The page where we get personal data and preferences from the students
    - Methods
        - GenerateStudent(): Student; generates a Student object from the input settings
- **AuthenticationComponent**
    - Description
        - The login page; also doubles as an adapter to the Authentication API; generates a Token object that contains the user's studentData ID
    - Methods
        - Authenticate(): verifies the authenticity of the user
- **Token**
    - Description

- An object whose presence is required to store student data and ensure student privacy
- o Attributes
  - int: internal student ID

Class Diagram



Database Specification and Analysis

MongoDB is a NoSQL database where data is organized into collections of documents. Each collection represents a specific category of data to TAP's application.

Collections:

StudentData

- **Description**: Stores student records, academic plans, preferences, and scheduling information
- **Fields**:
    - _id (ObjectId): Primary Key
    - studentId (String): Unique Towson University student identifier
    - firstName (String): First name
    - lastName (String): Last name
    - academicInfo (Object): User's current academic standing/information
    - preferences (Object): User preferences (e.g., preferred course types, academic goals)
    - degreePlan (Array of Objects): General academic outline
        - semester (String): Term, e.g., "Fall 2025"
        - plannedCourses (Array of Strings): Course names or subject areas
        - creditHours (Number): Total credit hours for that semester
        - notes (String, optional): Notes on the semester plan
    - activeSemesterPlan (Object): Detailed schedule for the upcoming semester
        - courseCode (String): Course code, e.g., "COSC101"
        - title (String): Full course title
        - units (Number): Credit hours
        - schedule (Array of Objects): Class meeting details
            - day (String): Days, e.g., "MoWe"
            - startTime (String): Class start time
            - endTime (String): Class end time
            - location (String): Classroom or online location
            - instructor (String): Instructor's name.
- **Relationships**:
    - studentId can be referenced in other collections

CourseData

- **Description**: Contains comprehensive course information, including both active courses (for upcoming semesters like Spring 2025) and general course data across all terms.
- **Fields**:
    - _id (ObjectId): Primary Key

    - institution (String): Institution code, e.g., "TOWSN"
    - acadCareer (String): Academic career, e.g., "UGRD"
    - subject (String): Subject code, e.g., "AADS"
    - catalogNumber (String): Catalog number, e.g., "305"
    - courseTitle (String): Title, e.g., "HISTORY OF DISABILITY"
    - termsOffered (String): The specific term(s) for which the course is offered, e.g., "Fall", “Spring”, “Summer”, “Winter”
    - description (String): Course description
    - units (String): Credit hours, e.g., "3 units"
    - gradingBasis (String): Grading basis, e.g., "UNDERGRADUATE GRADING"
    - components (Array of Objects):
        - component (String): Type, e.g., "Lecture"
        - required (Boolean): Whether it’s required
    - isActive (Boolean): A flag to indicate if the course is currently active. true means the course is available for an upcoming term; false means it is inactive.
    - campus (String): Campus name, e.g., "Main Academic Campus"
    - campusCode (String): Foreign Key reference to CampusData.campus
- **Relationships**:
    - campusCode: Links to CampusData.campus

CourseBridge (Term-Specific Active Sections of a Course)

- **Description**: Acts as a middleman between CourseSectionData and ActiveCourseData. It contains information about each active course section and includes an array of specific class offerings.
- **Fields**:
    - _id (ObjectId): Primary Key
    - courseId (ObjectId): Reference to CourseData._id, linking each active course section to its general course details.
    - courseSections (Array of ObjectIds): Array of references to CourseSectionData documents, where each document represents a specific class offering for this section in the specified term.

CourseSectionData (Active Courses for Specific Terms)

- **Description**: Contains information about courses that are currently active and available for specific upcoming semesters. This collection can dynamically update each term to reflect only those courses being offered.
- **Fields:**
    - _id (ObjectId): Primary Key
    - courseBridgeId (ObjectId): Reference to CourseBridge._id, linking each active course back to its general course details.
    - termActive (String): The specific term for which the course is active, e.g., "Spring 2025"

    - classNumber (Number): Unique class number for the specific offering, e.g., 4776.
    - classSection (String): Section ID, e.g., "001"
    - component (String): Class type, e.g., "LEC"
    - session (String): Session description, e.g., "Regular Academic Session"
    - startDate (Date): Start date of the course for the term.
    - endDate (Date): End date of the course for the term.
    - meetings (Array of Objects):
        - days (String): Meeting days, e.g., "TuTh"
        - startTime (String): Start time, e.g., "2:00PM"
        - endTime (String): End time, e.g., "3:15PM"
        - buildingCode (String): Building code, e.g., "HP"
        - room (String): Room number
        - facilityDescr (String): Facility description
    - instructors (Array of Objects):
        - name (String): Instructor's name
        - email (String): Instructor's email
    - enrollmentInfo (Object): Enrollment details.
        - status (String): Enrollment status, e.g., "Open"
        - classCapacity (Number): Class capacity
        - enrollmentTotal (Number): Current enrollment
        - enrollmentAvailable (Number): Seats available
        - waitListCapacity (Number): Waitlist capacity
        - waitListTotal (Number): Waitlist total
    - location (String): Location, e.g., "On Campus"
    - locationCode (String): Foreign Key reference to LocationData.location
- **Relationships**:
    - courseBridgeId: Links each active course section entry back to the general course information in CourseData.
    - locationCode: Links to LocationData.location
    - subject: Links to SubjectData.subjectCode

CampusData

- **Description**: Stores campus information
- **Fields**:
    - _id (ObjectId): Primary Key
    - campus (String): Campus code
    - descr (String): Description of the campus

LocationData

- **Description**: Stores location details, including whether courses are online or at a specific campus
- **Fields**:
    - _id (ObjectId): Primary Key

    - location (String): Location code
    - descr (String): Location description

OpenAIRequests

- **Description**: Logs OpenAI API requests for tracking
- **Fields**:
    - _id (ObjectId): Primary Key
    - studentId (String): References StudentData.studentId
    - requestData (Object): Data sent to OpenAI
    - responseData (Object): Response from OpenAI
    - timestamp (Date): Request timestamp

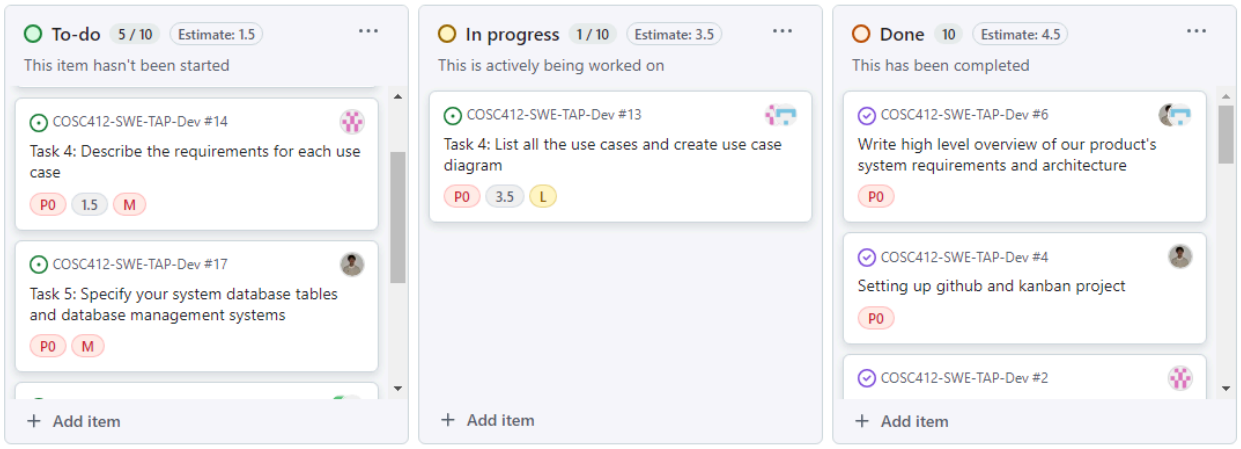
Relationships

- **Primary Keys (PK)**: _id field in each collection.
- **Foreign Keys (FK)**:
    - StudentData.studentId can be referenced in **OpenAIRequests**
    - CourseData.campusCode and locationCode link to **CampusData** and **LocationData**
    - `CourseBridge.courseId`, `CourseData._id`, and `CourseSectionData._id` to associate each active course section with its main course details.

Database Management System (DBMS)

TAP will be using **MongoDB Atlas**. This cloud-based Database-as-a-Service (DBaaS) enhances and manages MongoDB's NoSQL flexibility through schemaless support, scalability, real-time data handling, and monitoring and management tools.

Appendix


To-do 5 / 10 Estimate: 1.5
This item hasn't been started
COSC412-SWE-TAP-Dev #14
Task 4: Describe the requirements for each use case
P0 1.5 M
COSC412-SWE-TAP-Dev #17
Task 5: Specify your system database tables and database management systems
P0 M
Add item
In progress 1 / 10 Estimate: 3.5
This is actively being worked on
COSC412-SWE-TAP-Dev #13
Task 4: List all the use cases and create use case diagram
P0 3.5 L
Add item
Done 10 Estimate: 4.5
This has been completed
COSC412-SWE-TAP-Dev #6
Write high level overview of our product's system requirements and architecture
P0
COSC412-SWE-TAP-Dev #4
Setting up github and kanban project
P0
COSC412-SWE-TAP-Dev #2
Add item