

**CS673 Software Engineering**  
**Team 1 - BU Academic Navigator (BUAN)**  
**Software Test Document**



<u>Team Member</u>	<u>Role(s)</u>	<u>Signature</u>	<u>Date</u>
Natasya Liew	Team Leader	<u>Natasya Liew</u>	<u>September 2, 2024</u>
Natthaphon Foithong	Design and Implementation Lead	<u>Natthaphon Foithong</u>	<u>September 7, 2024</u>
Ananya Singh	Security Lead	<u>Ananya Singh</u>	<u>Sep 3, 2024</u>
Battal Cevik	QE Lead	<u>Battal Cevik</u>	<u>September 3, 2024</u>
Poom Chantarapornrat	Requirement Lead	<u>Chan P.</u>	<u>September 3, 2024</u>
Yu Jun Liu	Configuration Lead	<u>Yujun Liu</u>	<u>September 7, 2024</u>

#### Revision history

<u>Version</u>	<u>Author</u>	<u>Date</u>	<u>Change</u>
<u>2.0.0</u>	Battal Cevik	<u>Oct 5, 2024</u>	<u>Updated testing reports and test cases</u>
<u>2.0.1</u>	<u>Natasya Liew</u>	<u>Oct 6, 2024</u>	<u>Adding on to 'Who is involved' and 'Testing Technique Used', TC002 update</u>
<u>2.0.2</u>	<u>Battal Cevik</u>	<u>Oct 6, 2024</u>	<u>Added automation testing section and automation test cases, Added testing metrics</u>
<u>2.0.3</u>	<u>Natthaphon Foithong</u>	<u>Oct 6, 2024</u>	<u>Add manual testing report</u>

<u>2.0.4</u>	<u>Battal Cevik</u>	<u>Oct 13, 2024</u>	<u>Finished regression suite and added all test cases</u>
<u>2.0.5</u>	<u>Battal Cevik</u>	<u>Oct 14, 2024</u>	<u>Added automation test cases and reports</u>

[Testing Summary](#)

[Manuel Tests Reports](#)

[Automated Testing Reports](#)

[Testing Metrics](#)

[References](#)

[Glossary](#)

## Testing Summary

This section outlines the comprehensive testing strategies employed for the BUAN chatbot-driven web application. Our approach incorporated various testing methodologies to validate the system's adherence to the design goals and requirements specified in the SDD and SPPP sections. The BUAN Academic Advising Chatbot web application underwent a comprehensive testing process to ensure robust functionality, reliability, and performance across all system components, including the frontend, backend services, AI integration, and database layers. Our testing strategy encompassed unit, integration, system, acceptance, and regression testing methodologies. The following components were subject to rigorous testing:

### Frontend (React, Redux, and Fetch API)

- BUAN Chatbot Interface: Tested for smooth user interactions and appropriate response rendering. Automated tests were implemented to simulate user conversations and assess the accuracy of responses.
- User Interaction Flow: Verified the logical progression and intuitiveness of the user journey through the web application.
- Session Management: Verified the handling of user sessions, including session creation, validation, and expiration. Automated tests were set up to simulate user login/logout scenarios and assess session persistence.

### Backend (Spring Boot)

- RESTful APIs: Validated the functionality and reliability of APIs for course management operations (CRUD operations) and request processing from the frontend. Each API endpoint underwent unit testing to ensure they returned the expected functionality and handled edge cases appropriately.
- Data Processing: Ensured accurate transformation and handling of data between the front end, database, and Python3 AI service layers. Integration tests were conducted to validate data flows and the correctness of operations involving multiple components.

### Backend (Python3 AI Service using OpenAI 4.0 via Langchain API and Langchain for RAG, chat history generation, and session management tooling)

- Custom Course Builder Algorithm: Tested to ensure the generation of accurate course recommendations for the upcoming semester. We performed algorithmic testing with various input scenarios to validate the recommendation logic.
- Session Management: Verified the handling of chat sessions, including session creation, validation, and expiration. Automated tests were set up to simulate new chat creation scenarios and access session persistence.
- Chat History Generation: Validated the retrieval of chat history via Langchain. We conducted functional tests to ensure user interactions with the BUAN chatbot were accurately logged and retrievable in subsequent sessions.

## Databases (Postgresql)

- Chat History Storage: Verification of chat history storage was performed using unit tests to ensure that all interactions were saved correctly and that retrieval queries returned the expected results. We also conducted stress tests to assess performance under heavy loads.

## Authentication (JWT)

- User Login: Authentication mechanisms were thoroughly tested to ensure security and usability. This included testing various login scenarios (successful login, failed login, password recovery) and verifying the integration with the JWT API for OAuth 2.0 flows.

## Who is involved

### Development Team:

The development team comprised skilled software developers with specialized roles to ensure the effective construction of the application.

- **Frontend Developers (Natt, Poom, and Tash):** This trio was responsible for designing and implementing the user interface using React while using Jest to test their code within development. They focused on creating a seamless and intuitive user experience, translating design specifications into interactive elements while using JIRA to plan their acceptance tests for their functionalities. Their collaborative efforts included conducting regular code reviews, pair programming sessions, and ensuring that all components adhered to accessibility standards.
- **Spring Boot Developer (Ananya):** Ananya played a crucial role in building the backend services using Spring Boot using JUnit to test the existing code during development. He focused on developing RESTful APIs that facilitated communication between the frontend, Python AI service, and the database and conducting integration tests with the other developers. Ananya implemented best practices for security, data handling, and scalability to ensure the backend could handle varying loads and maintain performance and manually testing the functionality in a Docker environment.
- **Python AI Service Developers (Tash and Poom):** Tash and Poom worked together to create the AI decision algorithms that powered the application. They focused on implementing machine learning models and decision-making logic that provided personalized recommendations for users. By using Pytest and Unittest library during the development code testing before being pushed to Github for review, the team ensured that codes can be integrated with the other services smoothly with a performance that satisfies the acceptance test requirements. In order to achieve this, the team involved continuous iteration revisions, testing, and optimization of the algorithms to enhance accuracy, efficiency, and quality of chatbot responses.
- **Database Management (Tash and Ananya):** Tash and Ananya collaborated on managing the PostgreSQL database, ensuring data integrity and performance. During the process, Tash ensured that all code were executing correctly by manually testing the

table results after data was injected and call for retrieval. After Tash and Ananya tested the database and SB server connection in the integration test to ensure that data were moving through the two services correctly.

### Quality Assurance Engineers:

The Quality Assurance (QA) team was dedicated to ensuring the application met the highest quality standards through rigorous testing methodologies.

- **Lead QA Engineer (Battal):** As the lead, Battal oversaw the QA process, establishing testing strategies and methodologies. He coordinated testing efforts across the team, ensuring that all functionalities were rigorously evaluated. Battal's expertise in testing frameworks allowed him to design comprehensive test plans, covering both manual and automated testing approaches.
- **Supporting QA Engineers (AJ and Natt):** AJ and Natt supported Battal in executing the QA strategy. They were responsible for developing test cases and reporting defects back to the team. Their collaborative approach involved working closely with the development team to provide timely feedback on potential issues and verifying that resolved defects were correctly implemented.

The synergy between the development and QA teams was critical for the project's success. Regular communication and collaboration ensured that any challenges encountered during development were swiftly addressed, and that the application met user expectations and business requirements.

## Testing Techniques Used

### Unit Testing

Unit testing focused on evaluating individual components, including the React user interface, backend services built with Spring Boot, and Python decision algorithms. The following frameworks were employed: Jest for testing React components, JUnit for Spring Boot services, and Unittest and Pytest for Python services.

In addition to these testing frameworks, we leveraged GitHub and JIRA to enhance our code quality and adherence to best practices. GitHub's built-in Continuous Integration (CI) features allowed us to automate tests, ensuring that every commit triggered unit tests for functionality as well as linting checks for code formatting. This approach helped maintain coding standards and catch potential issues early in the development process. Within JIRA, we tracked issues and assigned tasks related to unit testing. Each user story linked to specific tests, allowing us to monitor progress and ensure comprehensive coverage. This integration facilitated better communication among team members, ensuring everyone was aligned on testing objectives.

We emphasized the importance of Test-Driven Development (TDD) among our development team. Before implementing any new features, developers wrote unit tests to define expected behaviors. This proactive approach ensured that code met specifications from the outset and

made it easier to identify and resolve bugs early in the development cycle. Once the development team completed their TDD process, the QA/testing team conducted further testing using the Cucumber framework. Cucumber allows for Behavior-Driven Development (BDD), enabling us to write tests in a natural language format that can be easily understood by non-technical stakeholders. This approach ensured that our acceptance criteria were met while also providing clarity on application behavior. By combining unit testing with TDD, CI practices, and the Cucumber framework, we aimed to create a robust testing environment that enhanced code reliability, improved collaboration among teams, and ultimately led to a higher-quality application.

## Example of Executions

Using Unittest library for unit testing within the Python AI service:



```

01 import unittest
# from coursebuilder import read_programs_csv, read_courses_csv, CourseTree # Make sure same folder
02
03 # Define the file paths
04 programs_file_path = './programs.csv'
05 courses_file_path = './courses.csv'
06
07 class TestCourseBuilder(unittest.TestCase):
08
09     def test_course_tree_recommendation(self):
10         """
11             Test the CourseTree class by simulating a student's course progress and path interest.
12         """
13         course_taken = ['521', '633', '673'] # Courses already taken
14         path_interest = 'AI/ML' # Student's elective path of interest
15         course_to_take = 5 # Number of courses to take next semester
16
17         # Create a CourseTree object
18         course_tree = CourseTree()
19
20         # Get recommended courses
21         recommended_courses = course_tree.recommend_courses(course_taken, path_interest, course_to_take)
22
23         # Assert that we receive the correct number of recommendations
24         self.assertEqual(len(recommended_courses), course_to_take)
25         self.assertIn(655, recommended_courses) # Core course 655 should be recommended
26         self.assertIn(677, recommended_courses) # AI/ML elective 677 should be recommended if not taken
27
28     # Run the tests
29 if __name__ == '__main__':
30     unittest.main(argv=['first-arg-is-ignored'], exit=False, verbosity=2)

```

Initially, without the correct implementation, we get this result:

```

=====
ERROR: test_course_tree_recommendation (__main__.TestCourseBuilder)
Test the CourseTree class by simulating a student's course progress and path interest.
-----
Traceback (most recent call last):
  File "<ipython-input-1-e86b5744a931>", line 19, in test_course_tree_recommendation
    course_tree = CourseTree()
NameError: name 'CourseTree' is not defined
=====
```

Ran 1 test in 0.034s

```
FAILED (errors=1)
=====
```

```
=====
And after the correct implementation, this is the result that we got:
```

```
=====
` test_course_tree_recommendation (__main__.TestCourseBuilder)
Test the CourseTree class by simulating a student's course progress and path interest.
... ok
```

```
-----
Ran 1 test in 0.006s
```

```
OK`
```

## Integration Testing

Integration testing assessed the interactions among the frontend, backend, AI service, and databases (PostgreSQL). This phase ensured that all system components functioned correctly in conjunction with one another, facilitating seamless data flow and interaction.

To streamline our testing process, we utilized Docker to create isolated environments for each service. This setup allowed us to simulate real-world deployment scenarios, ensuring that the application components interacted as expected under consistent conditions. By using Docker containers, we could easily replicate environments, which facilitated more reliable and reproducible tests. In conjunction with Docker, we employed Postman to validate the connections between various services. Postman allowed us to send requests to the API endpoints and observe the responses, ensuring that data was transmitted correctly between the frontend, backend, and AI service.

Through this process, we rigorously tested for latency issues, measuring the time taken for requests to be processed and responses to be returned. We also assessed the smoothness of these connections, focusing on whether there were any disruptions or delays during data transmission. This involved monitoring response times and analyzing the consistency of outputs across multiple test runs. Finally, we verified that the outputs generated by the AI service and the backend were accurate and aligned with the expected results. This comprehensive testing approach helped identify any discrepancies or bottlenecks in the system, ensuring a robust and efficient integration of all components.

## System Testing

System testing validated the comprehensive functionality of the web application by examining the integrated systems as a whole. This included testing the user interface, backend processes, database interactions, and the AI service, ensuring all components worked together as intended.

To ensure that the application functioned seamlessly, we conducted manual testing to verify the network connectivity among the various APIs and services integrated within our product. This involved checking the communication pathways between the frontend, backend, and third-party services to confirm that data was transmitted accurately and without interruption. Additionally, we assessed the latency of the system to ensure that response times were within acceptable limits. The end-to-end experience of using our product was thoroughly evaluated against the functionality acceptance test criteria. We verified that the user interactions, data retrieval, and overall responsiveness met the expected performance benchmarks, contributing to a smooth and satisfying user experience. Through this comprehensive system testing approach, we ensured that all components of the web application worked harmoniously, ultimately leading to a reliable and efficient product.

## Acceptance Testing

Acceptance testing aimed to confirm that the application met all functional requirements. Key areas of focus included user authentication, course recommendations, and chat history management. Tests were conducted to ensure that the user experience aligned with expected outcomes and usability standards.

To systematically manage our acceptance criteria, we created detailed user stories in JIRA. These user stories outlined the specific functionalities and expectations from the user's perspective, providing a clear roadmap for our testing efforts. Each acceptance test was mapped against these user stories to ensure comprehensive coverage and that all requirements were satisfied.

During the development phase, we encouraged our development team to engage in peer testing. Team members tested each other's functionalities to identify any issues early in the process. This collaborative approach not only fostered a culture of accountability but also allowed for immediate feedback and rapid iteration, enhancing the overall quality of the code before it reached the QA/testing team.

Once the initial peer testing was completed, the QA/testing team conducted thorough testing during the stage branch. This stage involved validating all functionalities against the acceptance criteria established in the JIRA user stories. The QA team focused on real-world scenarios to assess the application's performance and reliability, ensuring that the user experience met both expected outcomes and usability standards. By implementing this structured approach to acceptance testing, we aimed to ensure that the final product was robust, user-friendly, and aligned with the needs of our target audience.

## Regression Testing

Following the implementation of new features, such as chat history sharing, regression testing was performed to verify that existing functionality remained intact. This phase concentrated on critical areas, including authentication processes, recommendation accuracy, and chat history retrieval.

Regression testing Jira Epic can be found in this Document URL: [Regression Suite](#)

## Testing Results

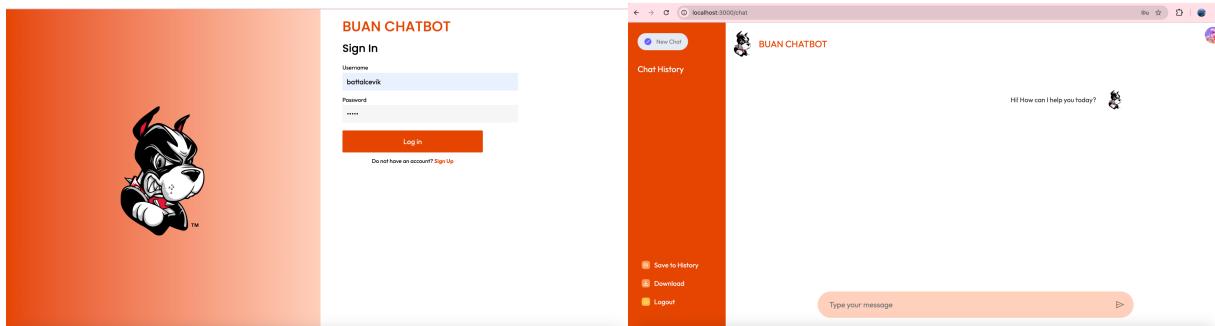
The majority of components successfully passed the testing phases, with minor issues identified, particularly concerning the integration between the courses and programs CSV files, the prompt library JSON file, the custom-built tree structure for course recommendations for the upcoming semester, and the OpenAI ChatGPT 4.0 service via the Langchain API. These issues were resolved through targeted bug fixes.

# Manual Testing Report

## Test Case ID: TC001, Name: User Login functionality

- **New or Old:** Old-Updated
- **Test items:** Login functionality updated
- **Test priority:** High
- **Dependencies:** None
- **Preconditions:** User should have e-mail account
- **Input data:** User credentials - e-mail and password
- **Test steps:**
  1. Launch the application.
  2. Navigate to the login page.
  3. Enter valid credentials.
  4. Click on the Login button.
- **Postconditions:** User should be logged in and redirected to the dashboard.
- **Expected output:** Successful login, user data fetched.
- **Actual output:** As expected.
- **Pass or Fail:** Pass

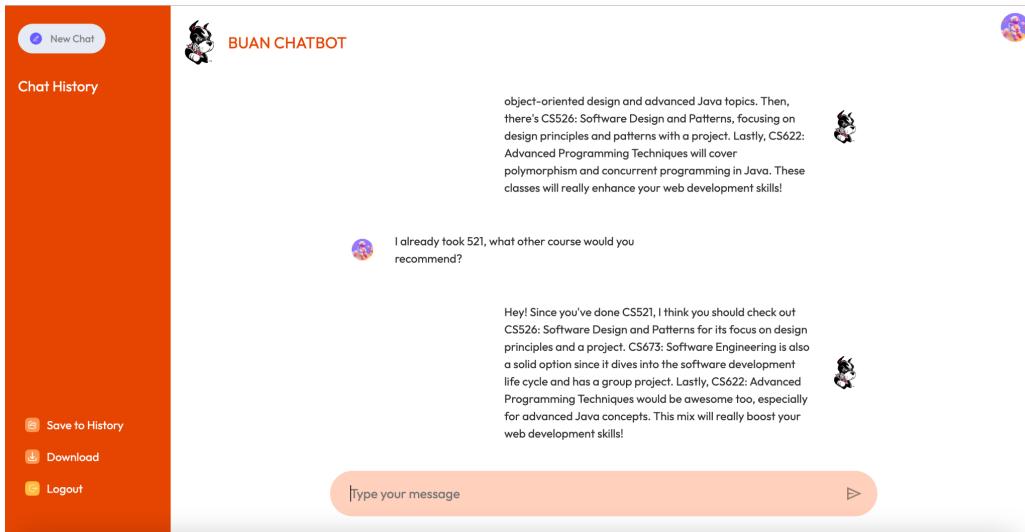
<https://seoprojects-cs673olf24team1.atlassian.net/browse/SCRUM-229>



## Test Case ID: TC002, Name: Course Recommendation Chatbot

- **New or Old:** Old-Updated
- **Test items:** Chatbot course recommendation system
- **Test priority:** High
- **Dependencies:** AI service, Langchain API, courses.csv, programs.csv, and Course Tree class object
- **Preconditions:** AI service running, documents readable by the Course Tree class object
- **Input data:** User input for course preferences: `program\_code`, `course\_to\_take`, `course\_taken`, and `path\_interest`.
- **Test steps:**
  1. Start a chat with the bot.

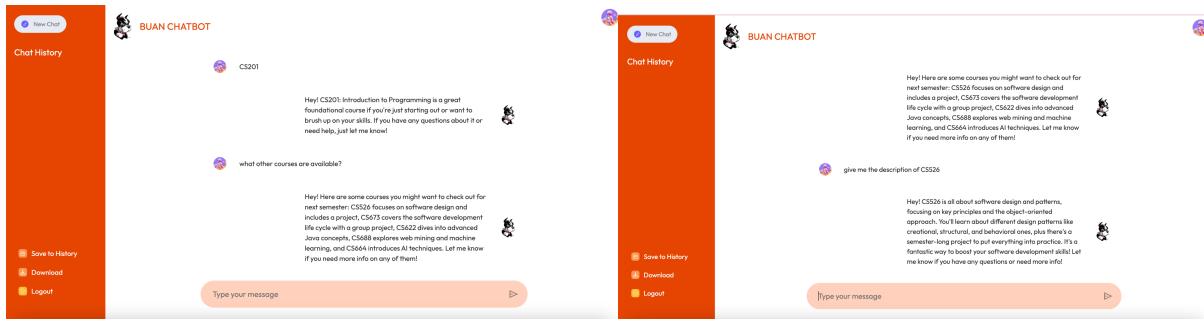
2. Enter course preferences (`program\_code`, `course\_to\_take`, `course\_taken`, and `path\_interest`).
  3. Receive course suggestions according to parameters of inputs given.
- **Postconditions:** Personalized course recommendations displayed with amount stated under the `course\_to\_take` parameter.
  - **Expected output:** The AI chatbot provides relevant course recommendations associated to their `path\_interest` parameter. If there are different options that the user can get, recommendations will change for different calls on the function.
  - **Actual output:** As expected.
  - **Pass or Fail:** Pass



## Test Case ID: TC003, Name: Provide Course Description

- **New or Old:** Old
- **Test items:** Chatbot response in regards to questions that can be retrieve from the courses.csv and programs.csv documents
- **Test priority:** High
- **Dependencies:** AI service, Langchain API, courses.csv, programs.csv
- **Preconditions:** AI service running, documents readable by the Langchain agents via the Langchain API
- **Input data:** User input for their query.
- **Test steps:**
  - Start a chat with the bot.
  - Enter their query.
  - Receive response as their expectation.
- **Postconditions:** Response generated satisfy their query with easy understanding.
- **Expected output:** The AI chatbot provides relevant response to the query with minimum follow-up query relevant to previous query.
- **Actual output:** As expected.
- **Pass or Fail:** Pass

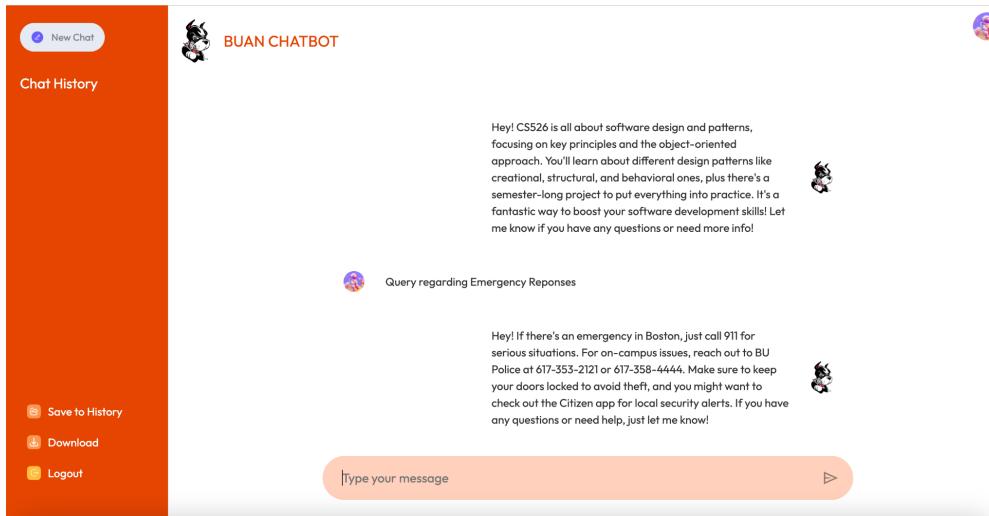
<https://seoprojects-cs673olf24team1.atlassian.net/browse/SCRUM-235>



## Test Case ID: TC003, Name: Query regarding Emergency Responses

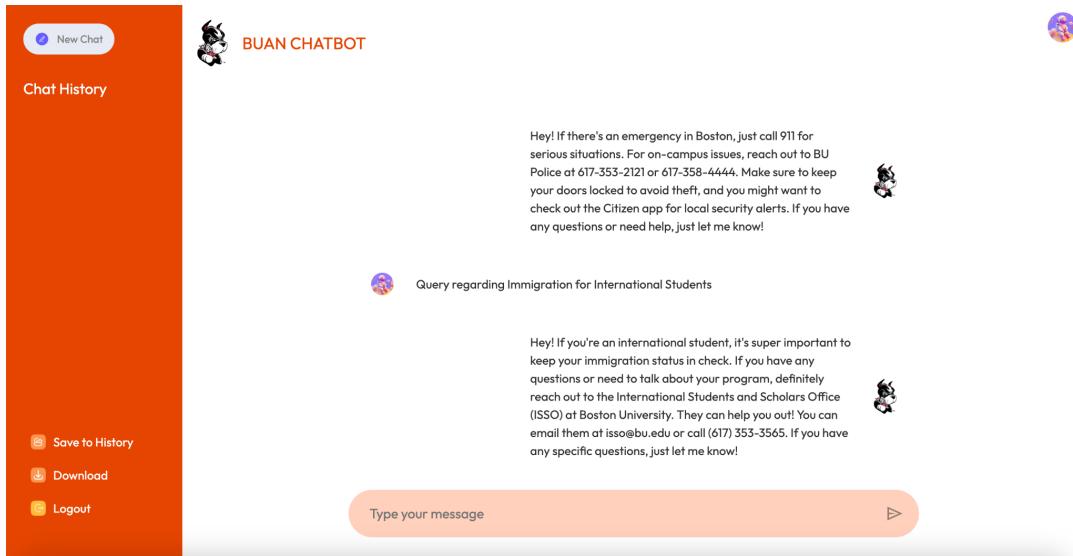
- **New or Old:** New Updated
- **Test items:** Chatbot response in regards to questions that can be retrieved from the promptlib.json
- **Test priority:** Mid
- **Dependencies:** AI service, Langchain API, promptlib.json
- **Preconditions:** AI service running, documents readable by the Langchain agents via the Langchain API
- **Input data:** User input for their query.
- **Test steps:**
  - Start a chat with the bot.
  - Enter regarding emergency services responses
  - Receive response as their expectation.
- **Postconditions:** Response generated satisfy their query with easy understanding.
- **Expected output:** The AI chatbot provides relevant response to the query with minimum follow-up query relevant to previous query.
- **Actual output:** As expected.
- **Pass or Fail:** Pass

<https://seoprojects-cs673olf24team1.atlassian.net/browse/SCRUM-255>



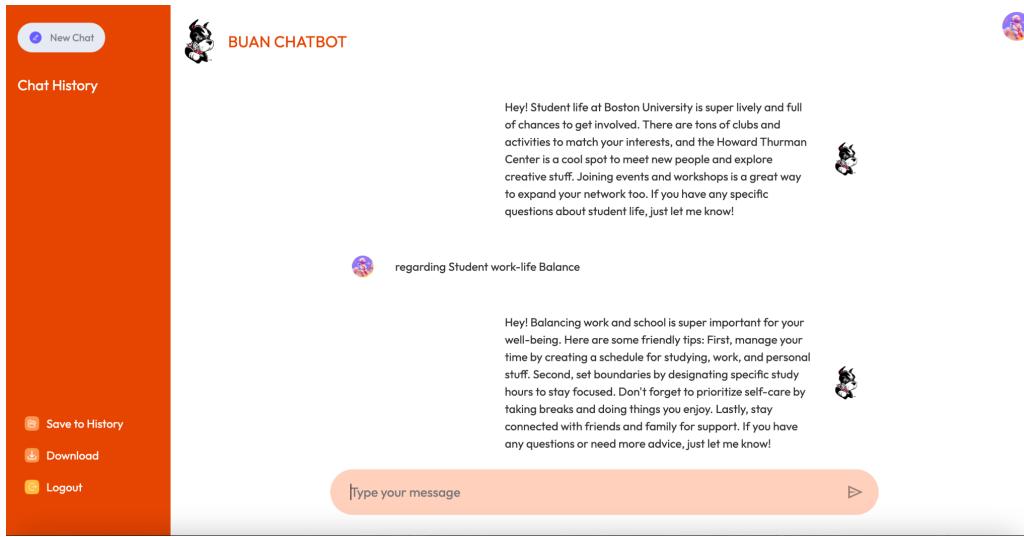
## Test Case ID: TC004, Name: Query regarding Immigration for International Students

- **New or Old:** New-Updated
- **Test items:** Chatbot response in regards to questions that can be retrieve from the promptlib.json document
- **Test priority:** Mid
- **Dependencies:** AI service, Langchain API, promptlib.json
- **Preconditions:** AI service running, documents readable by the Langchain agents via the Langchain API
- **Input data:** User input for their query.
- **Test steps:**
  - Start a chat with the bot.
  - Enter regarding immigration for international students.
  - Receive response as their expectation.
- **Postconditions:** Response generated satisfy their query with easy understanding.
- **Expected output:** The AI chatbot provides relevant response to the query with minimum follow-up query relevant to previous query.
- **Actual output:** As expected.
- **Pass or Fail:** Pass



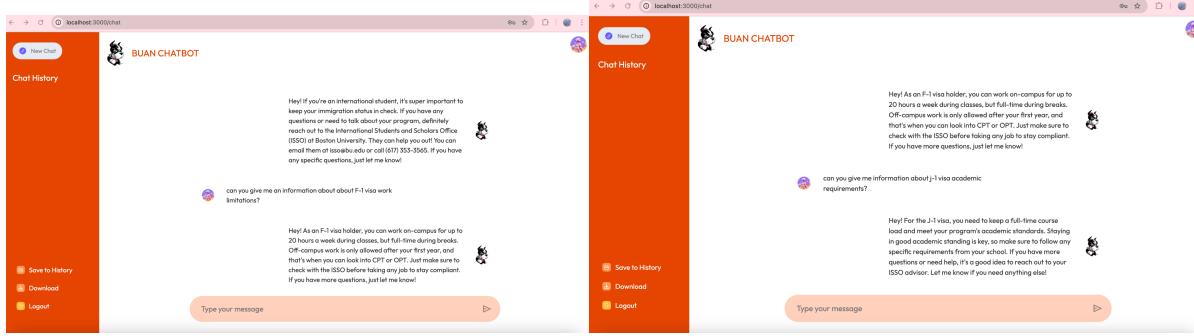
## Test Case ID: TC005, Name: Query regarding Student Work-Life Balance

- **New or Old:** New
- **Test items:** Chatbot response in regards to questions that can be retrieve from the promptlib.json document
- **Test priority:** Mid
- **Dependencies:** AI service, Langchain API, promptlib.json
- **Preconditions:** AI service running, documents readable by the Langchain agents via the Langchain API
- **Input data:** User input for their query.
- **Test steps:**
  - Start a chat with the bot.
  - Enter a query regarding student work-life balance.
  - Receive response as their expectation.
- **Postconditions:** Response generated satisfies their query with easy understanding.
- **Expected output:** The AI chatbot provides a relevant response to the query with minimum follow-up query relevant to previous query.
- **Actual output:** As expected.
- **Pass or Fail:** Pass



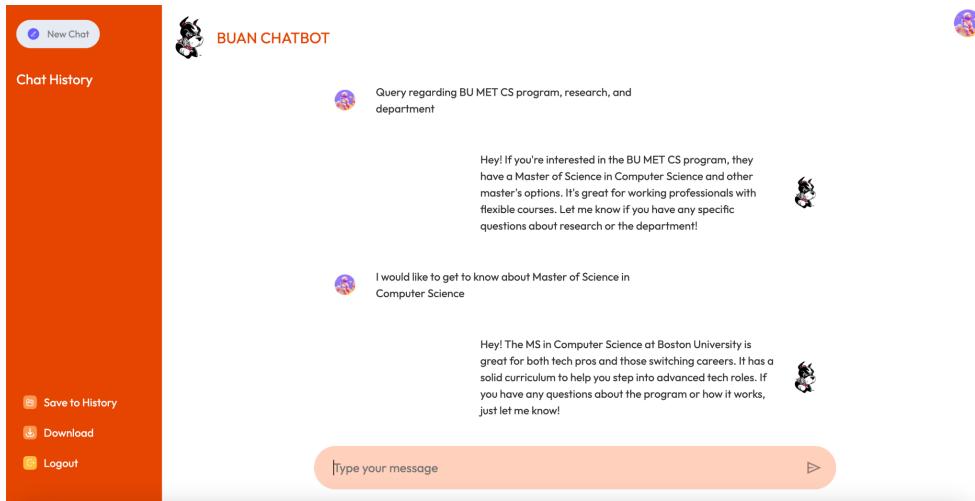
## Test Case ID: TC006, Name: Query regarding Student Status Inquiry

- **New or Old:** New-Updated
- **Test items:** Chatbot response in regards to questions that can be retrieve from the promptlib.json documents
- **Test priority:** Mid
- **Dependencies:** AI service, Langchain API, promptlib.json
- **Preconditions:** AI service running, documents readable by the Langchain agents via the Langchain API
- **Input data:** User input for their query.
- **Test steps:**
  - Start a chat with the bot.
  - Submit an inquiry about F-1 visa work limitations.
  - Submit an inquiry about J-1 visa academic requirements
  - Receive response as their expectation.
- **Postconditions:** Response generated satisfy their query with easy understanding.
- **Expected output:** The AI chatbot provides relevant response to the query with minimum follow-up query relevant to previous query.
- **Actual output:** As expected.
- **Pass or Fail:** Pass



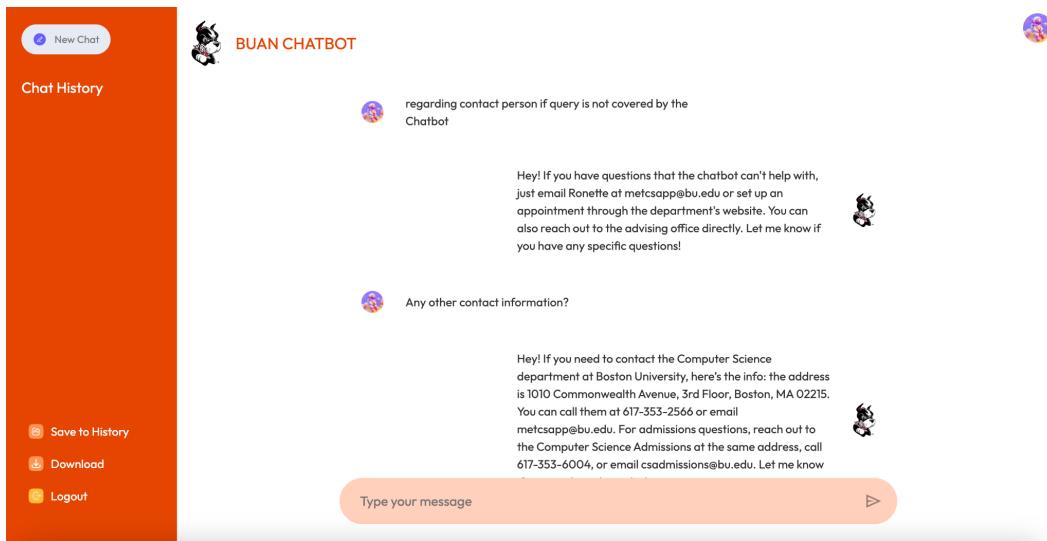
## Test Case ID: TC007, Name: Query regarding BU MET CS program, research, and department

- **New or Old:** New
- **Test items:** Chatbot response in regards to questions that can be retrieved from the courses.csv, programs.csv, promptlib.json documents
- **Test priority:** High
- **Dependencies:** AI service, Langchain API, courses.csv, programs.csv,promptlib.json
- **Preconditions:** AI service running, documents readable by the Langchain agents via the Langchain API
- **Input data:** User input for their query.
- **Test steps:**
  - Start a chat with the bot.
  - Enter their query.
  - Receive response as their expectation.
- **Postconditions:** Response generated satisfy their query with easy understanding.
- **Expected output:** The AI chatbot provides relevant response to the query with minimum follow-up query relevant to previous query.
- **Actual output:** As expected.
- **Pass or Fail:** Pass



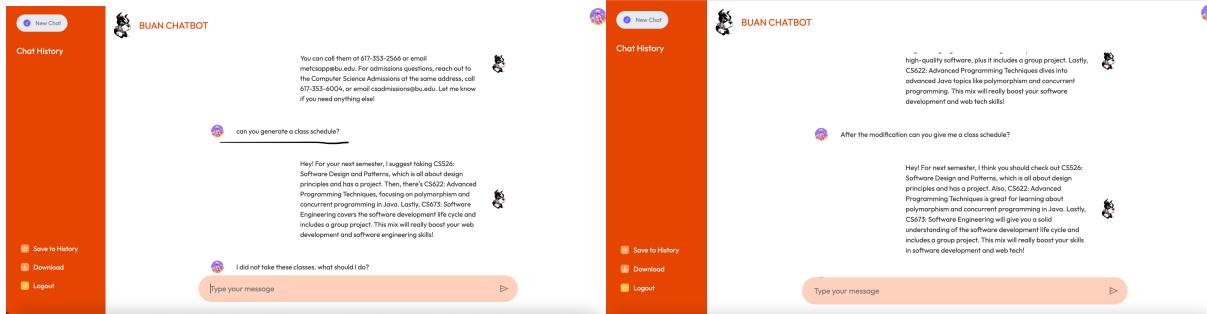
## Test Case ID: TC008, Name: Query regarding contact person if query is not covered by the Chatbot

- **New or Old:** New
- **Test items:** Chatbot response in regards to questions that can be retrieve from the promptlib.json document
- **Test priority:** High
- **Dependencies:** AI service, Langchain API, promptlib.json
- **Preconditions:** AI service running, documents readable by the Langchain agents via the Langchain API
- **Input data:** User input for their query.
- **Test steps:**
  - Start a chat with the bot.
  - Enter their query.
  - Receive response as their expectation.
- **Postconditions:** Response generated satisfy their query with easy understanding.
- **Expected output:** The AI chatbot provides relevant response to the query with minimum follow-up query relevant to previous query.
- **Actual output:** As expected.
- **Pass or Fail:** Pass



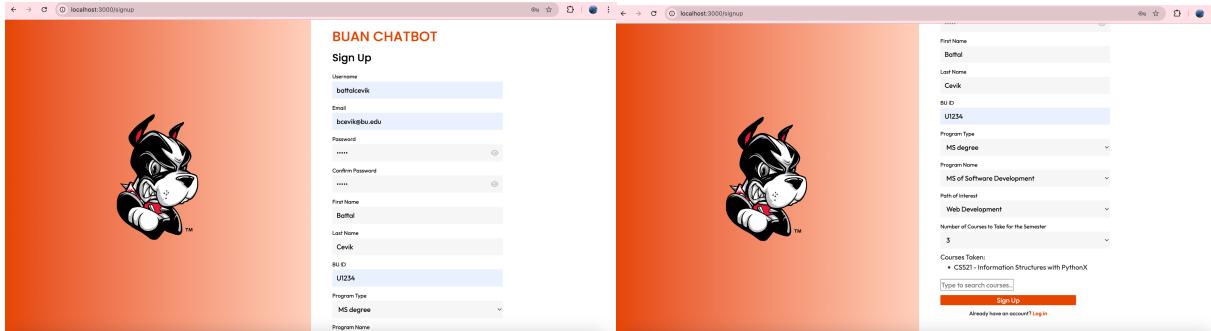
## Test Case ID: TC009, Name: Generate Class Schedule

- **New or Old:** New
- **Test items:** Chatbot response in regards to questions that can be retrieved from the courses.csv, programs.csv, and promptlib.json documents
- **Test priority:** High
- **Dependencies:** AI service, Langchain API, courses.csv, programs.csv, promptlib.json
- **Preconditions:** AI service running, documents readable by the Langchain agents via the Langchain API
- **Input data:** User input for their query.
- **Test steps:**
  - Log in and provide a list of completed courses.
  - Submit a request for a class schedule.
  - Verify that the system generates and displays the schedule.
  - Check that all courses in the schedule meet prerequisite requirements.
  - Request schedule modifications and verify that the system updates the schedule accordingly.
  - Request a course that is unavailable and verify that the system notifies the user.
  - Export the final schedule to a calendar format and verify successful export.
- **Postconditions:** Response generated satisfy their query with easy understanding.
- **Expected output:** The AI chatbot provides relevant response to the query with minimum follow-up query relevant to previous query.
- **Actual output:** As expected.
- **Pass or Fail:** Pass



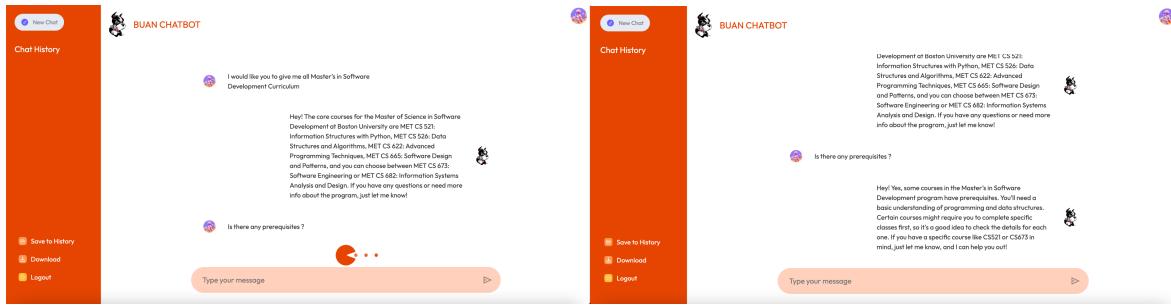
## Test Case ID: TC010, Name: User Signup Functionality

- **New or Old:** New
- **Test items:** Signup functionality
- **Test priority:** High
- **Dependencies:** Backend API should be available (<http://localhost:8080/api/signup>)
- **Preconditions:**
  - The user is not logged in
  - The signup page should be accessible from the front end
  - Database and backend services are running
  - All form fields should be visible and operational
- **Input data:** Username, Email, Password, Confirm Password, First Name, Last Name, BU ID, Program Type, Program Code, Path of Interest, Courses to Take
- **Test steps:**
  - Open the signup page
  - Input data in the corresponding form fields
  - Toggle password visibility to confirm passwords are correctly entered
  - Click the “Sign Up” button
  - Monitor the API response and confirm the user is redirected to the chat page
  - Validate that a success message (“Signup successful!”) is displayed after successful registration.
- **Postconditions:**
  - The form should reset, and the success message should be visible for a few seconds.
  - The user data should be stored in the backend database.
- **Expected output:** The user should be successfully registered, and the success message should be displayed.
- **Actual output:** As expected.
- **Pass or Fail:** Pass



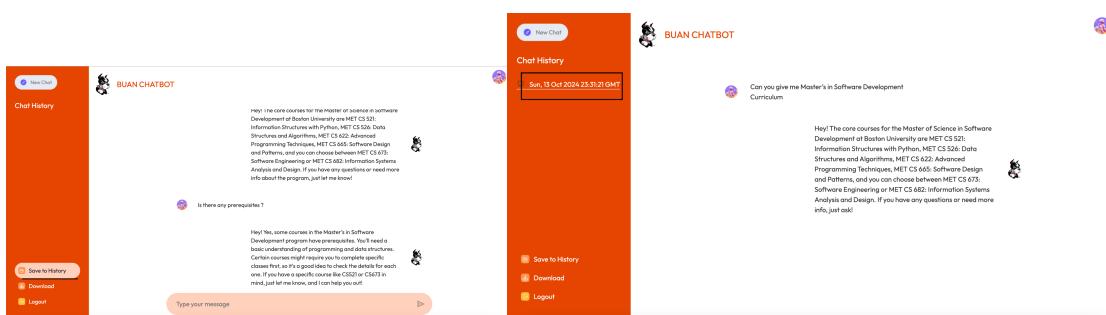
## Test Case ID: TC011, Name: Sending Messages in a Chat Application

- **New or Old:** New-Updated
- **Test items:** Input Field for Sending Messages
- **Test priority:** High
- **Dependencies:** None
- **Preconditions:**
  - The user is logged into the chat application.
  - The user is on a chat screen where the **InputField** component is rendered.
- **Input data:**
  - Valid message text
  - Empty message (when clicking the send button without any input)
- **Test steps:**
  - Navigate to the chat screen
  - Check that the input field is present on the screen with the placeholder text "Type your message"
  - Click inside the input field and Type a message
  - Verify that the text appears in the input field
  - Click the send button (send icon)
  - Verify that the message is sent to the parent component (ensure that the `onSend` function is called)
- **Postconditions:**
  - The message should be successfully sent if not empty
  - The input field should be cleared after sending a message.
- **Expected output:**
  - The input field is displayed correctly with the placeholder
  - Typed text is visible in the input field
  - Clicking the send button with valid input sends the message and clears the field
  - Clicking the send button with an empty input does not send a message and keeps the input field empty
- **Actual output:** As expected.
- **Pass or Fail:** Pass



## Test Case ID: TC012, Name: Chat History and Caching Consolidation

- **New or Old:** New
- **Test items:** Chat History
- **Test priority:** High
- **Dependencies:** None
- **Preconditions:**
  - The user is logged into the chat application.
- **Input data:**
  - Valid message text
  - Empty message (when clicking the send button without any input)
- **Test steps:**
  - Click on the Save to History button on the left side of window
  - Request the chat history from a previous session.
  - Verify that all previous interactions are displayed .
  - Confirm that the chat history loads.
- **Expected output:** The chat history should be retrieved accurately and cached for faster access in future requests.
- **Actual output:** As expected.
- **Pass or Fail:** Pass



## Test Case ID: TC013, Name: Courses Provided in the CS Department

- **New or Old:** New
- **Test items:** Chat History
- **Test priority:** High
- **Dependencies:** None
- **Preconditions:**
  - The user is logged into the chat application.
- **Input data:**

Verify that the system lists the courses offered by the Computer Science department along with their prerequisites.
- **Test steps:**
  - Search information about the CS department section.
  - Submit a request to view the course list.
  - Verify that the system displays the full list of CS courses.
  - Request the prerequisites for a specific course (e.g., CS201).
  - Verify that the system displays the correct prerequisites for each course.
- **Expected output:** The system should return a comprehensive list of CS courses along with accurate prerequisites..
- **Actual output:** As expected.
- **Pass or Fail:** Pass

The screenshots illustrate the interaction between a user and the BUAN CHATBOT to find courses and prerequisites.

**Screenshot 1: Initial Query**

User message: "Can you please give me more information about the CS department section?"

Bot response: "Hey! If you want to know more about the Computer Science department at Boston University, check out their official website at <https://www.bu.edu/cs/>. It has all the latest info on courses, faculty, and news. If you have any specific questions, just let me know!"

**Screenshot 2: Course List Response**

User message: "Do you have any course list?"

Bot response: "full list of CS courses." (The text is partially cut off in the screenshot.)

Bot detailed response: "Hey! Here's a list of Computer Science courses of Boston University: CS521: Information Structures with Python, CS526: Software Design and Patterns, CS601: Web Application Development, CS622: Advanced Programming Techniques, CS673: Software Engineering, CS688: Web Mining and Machine Learning, CS664: Artificial Intelligence, CS546: Introduction to Probability and Statistics, CS342: Data Structures with Java, and CS232: Programming with Java. If you want more info on any course or have other questions, just let me know!"

**Screenshot 3: Prerequisites Response**

User message: "what about prerequisites ?"

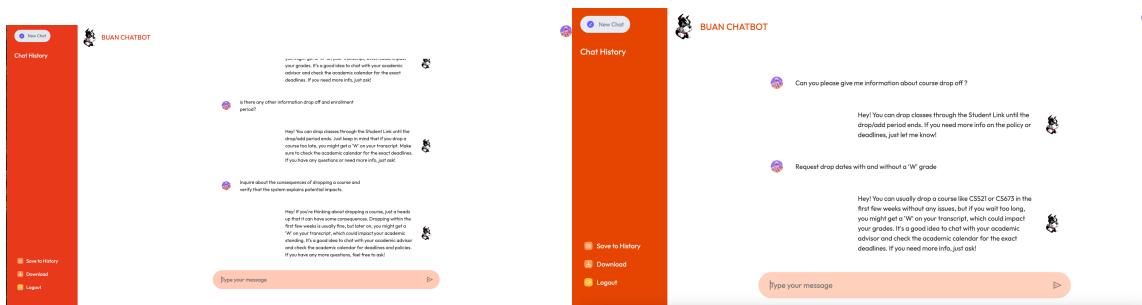
Bot response: "Hey! The prerequisites for Computer Science courses can differ. For instance, CS521 needs a basic programming background, while CS673 has its own requirements. It's a good idea to check the specifics for each course. If you have a particular course in mind, just let me know, and I can help you out!"

## Test Case ID: TC014, Name: Course Drop Dates

- **New or Old:** New
- **Test items:** Chat History
- **Test priority:** High
- **Dependencies:** None
- **Preconditions:**
  - The user is logged into the chat application.
- **Input data:**

Verify that the system retrieves and displays important course drop information

- **Test steps:**
  - Select the course drop inquiry section.
  - Request drop dates with and without a 'W' grade.
  - Verify that accurate dates are displayed.
  - Inquire about the consequences of dropping a course and verify that the system explains potential impacts.
- **Expected output:** The system should return a comprehensive list of CS courses along with accurate prerequisites..
- **Actual output:** As expected.
- **Pass or Fail:** Pass



## Test Case ID: TC015, Name: Graduation Timeline Inquiry

- **New or Old:** New
- **Test priority:** High
- **Dependencies:** None

- **Preconditions:**

- The user is logged into the chat application.

- **Input data:**

Verify that the system suggests a personalized graduation timeline based on user's completed courses.

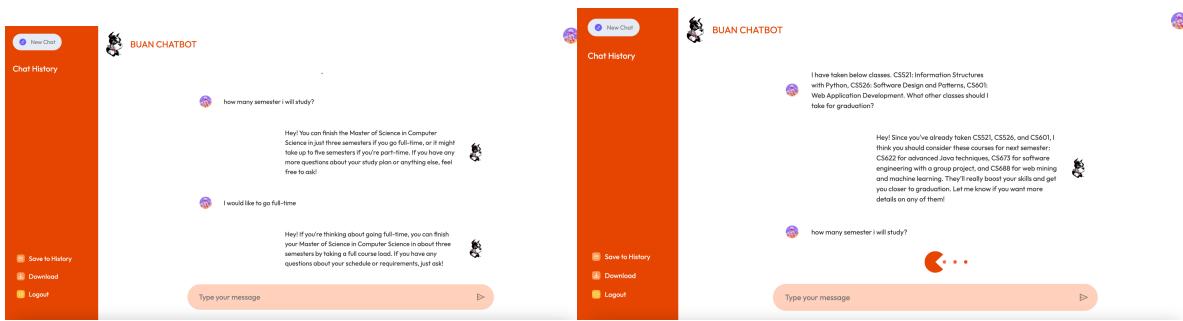
- **Test steps:**

- Submit a request for a graduation timeline after taking 3 courses.
- Verify that the system lists remaining course requirements.

- **Expected output:** The system should suggest a graduation timeline and classes

- **Actual output:** As expected.

- **Pass or Fail:** Pass



### Test Case ID: TC016, Name: Program Change and Course Transfer Inquiry

- **New or Old:** New

- **Test priority:** High

- **Dependencies:** None

- **Preconditions:**

- The user is logged into the chat application.

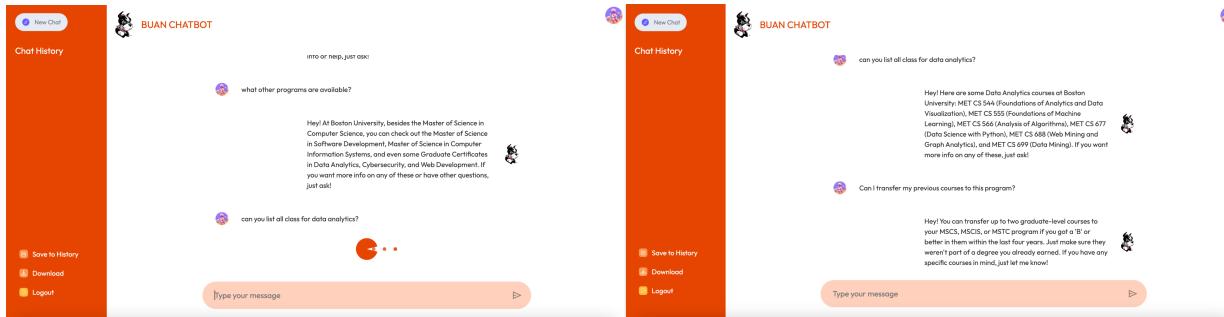
- **Input data:**

Verify that the system retrieves information about course transferability when a user inquires about changing programs.

- **Test steps:**

- Search if there are any other programs available.
- Select a new program from the available options.
- Submit the request to check course transferability.
- Verify that the system displays a list of courses that are applicable to the new program.

- **Expected output:** The system should correctly display applicable programs and courses, provide additional requirements
- **Actual output:** As expected.
- **Pass or Fail:** Pass



## Automated Testing Report

### Automated Testing Frameworks:

- **Frontend:** UI Automation with Selenium BDD framework.
- **Backend:** Test NG, Rest Assured API .

### Test Code Repository:

- Located in the `/tests` folder in the project repository. Separate test directories from main code for frontend, backend, and Python service.
- <https://github.com/BUMETCS673/seprojects-cs673olf24team1/tree/main/tests>

### Screenshots/Generated Reports:

- The generated test reports are stored in `/reports/` and can be viewed in cucumber report using cucumber.properties file and launching <https://reports.cucumber.io/> location.

### Automated Test cases:

### API Automation Test Cases:

#### Test Case 1: Valid Chatbot Request

- **Test Description:** Verify that the chatbot API returns a 200 status code for a valid request.
- **Preconditions:**
  1. The API endpoint is available.
- **Test Steps:**

1. Set the POST chatbot API endpoint.
  2. Send a POST HTTP request to the chatbot API.
- **Test Data:** A valid chatbot request message.
  - **Expected Result:** The server responds with HTTP status code 200.
  - **Actual Result:** As Expected.
  - **Status:** Pass.

## **Test Case 2: Invalid Chatbot Request with Empty Message**

- **Test Description:** Verify that the chatbot API returns a 422 status code when an empty message is sent.
- **Preconditions:**
  1. The API endpoint is available.
- **Test Steps:**
  1. Set the POST chatbot API endpoint.
  2. Send a POST HTTP request with an empty message.
- **Test Data:** Empty message.
- **Expected Result:** The server responds with HTTP status code 422 (Bad Request).
- **Actual Result:** As Expected.
- **Status:** Pass.

## **Test Scenario 3: Chatbot Request with Unauthorized Access**

- **Test Description:** Verify that the chatbot API returns a 401 status code when credentials are missing.
- **Preconditions:**
  1. The API endpoint is available.
- **Test Steps:**
  1. Set the POST chatbot API endpoint.
  2. Send a POST HTTP request without credentials.
- **Test Data:** No credentials.
- **Expected Result:** The server responds with HTTP status code 401 (Unauthorized).
- **Actual Result:** As Expected.
- **Status:** Pass.

## **UI Automation Test Cases:**

### **Test Case 4: User Can Log In with Valid Credentials**

- **Test Description:** Verify that a user can successfully log in using valid credentials.
- **Preconditions:**
  1. The user is on the login page.

2. Valid credentials exist (e.g., username: `bcevik@bu.edu`, password: `admin`).
- **Test Steps:**
    1. Navigate to the login page.
    2. Enter valid username `bcevik@bu.edu`.
    3. Enter valid password `admin`.
    4. Click on the "Login" button.
  - **Expected Result:** The user is redirected to the homepage after successful login.
  - **Actual Result:** As Expected.
  - **Status:** Pass

## Test Case 5: User Cannot Log In with Invalid Credentials

- **Test Description:** Verify that a user cannot log in with incorrect credentials.
- **Preconditions:**
  1. The user is on the login page.
- **Test Steps:**
  1. Navigate to the login page.
  2. Enter an invalid username `wrong@bu.edu`.
  3. Enter an invalid password `wrongpass`.
  4. Click on the "Login" button.
- **Expected Result:** The user sees an error message "Invalid credentials" and remains on the login page.
- **Actual Result:** As expected.
- **Status:** Pass.

## Test Scenario 6: User Can Log Out Successfully

- **Test Description:** Verify that a user can successfully log out.
- **Preconditions:**
  1. The user is logged in with valid credentials (username: `bcevik@bu.edu`, password: `admin`).
- **Test Steps:**
  1. Log in using valid credentials (`bcevik@bu.edu / admin`).
  2. Click on the "Logout" button.
- **Expected Result:** The user is redirected to the login page after logging out.
- **Actual Result:** As Expected.

- **Status:** Pass.

## Test Case7: User Can Sign Up with Valid Credentials

### Test Description:

Verify that a user can successfully sign up using valid credentials.

### Preconditions:

- The user is on the login page.
- The "Sign Up" link is available.

### Test Steps:

1. **Launch the login page.**
  - Navigate to the login page.
2. **Click on the "Sign Up" link.**
  - Verify that the "Sign Up" page is displayed.
3. **Enter valid user credentials:**
  - Enter username: "testuser12311".
  - Enter email: "testuser11@example.com".
  - Enter password: "TestPassword1231!".
  - Confirm the password: "TestPassword1231!".
4. **Enter personal information:**
  - Enter first name: "John".
  - Enter last name: "Doe".
  - Enter BU ID: "U12345678".
5. **Select academic program information:**
  - Select program type: "MS degree".
  - Select program name: "MS of Software Development".
  - Select path of interest: "Web Development".
6. **Submit the form:**
  - Click on the "Sign Up" button.

### Expected Result:

- The user is successfully registered and redirected to the account dashboard or receives a success message indicating the account creation was successful.

### Actual Result: Pass

## Test Case 8: Retrieve Sessions for a User

### Test Description:

Verify that an authenticated user can retrieve sessions using a valid user ID.

### Preconditions:

- The user is authenticated.
- A valid user ID exists.
- The Sessions API is available.

### Test Steps:

1. **Authenticate the user.**
  - Ensure the user is authenticated using valid credentials.
2. **Prepare the GET request:**
  - Set the API endpoint to "`/api/v1/sessions/user/2`".
  - Include the valid user ID "`2`" in the request.
3. **Send the GET request:**
  - Send a GET HTTP request to the endpoint.

### Expected Result:

- The server responds with a valid session data for the user with ID `2`.
- The response status should be HTTP `200 OK`.

### Actual Result: Pass

## Test Case 9: Save a Conversation for a User

### Test Description:

Verify that an authenticated user can save a conversation using a valid user ID, and the response is successful but the body is null.

### Preconditions:

- The user is authenticated.
- A valid user ID exists (e.g., `2`).

- There is conversation data to save.
- The Sessions API is available.

#### **Test Steps:**

1. **Authenticate the user.**
  - Ensure the user is authenticated using valid credentials.
2. **Prepare the POST request:**
  - Set the API endpoint to "`/api/v1/sessions/user/2/conversation`".
  - Include the valid user ID "`2`" and the conversation data in the request.
3. **Send the POST request:**
  - Send a POST HTTP request to the endpoint with the conversation data.

#### **Expected Result:**

- The server responds with a success status for saving the conversation.
- The response body should be null.
- The response status should be HTTP `200 OK` or another success code.

#### **Actual Result: Pass**

### **Test Case 10: Perform All Chatbot Interactions in One Session**

#### **Test Description:**

Verify that a user can successfully log in to the chatbot and perform all chatbot interactions (highlighting the message box, saving to history, and highlighting "New Chat") in one session.

#### **Preconditions:**

- The user is on the chatbot login page.
- The chatbot application is available.

#### **Test Steps:**

1. **Log in to the chatbot:**
  - Enter username: "`testuser123`".
  - Enter password: "`TestPassword123!`".
  - Click on the "Login" button.
2. **Verify the message box is highlighted:**
  - After successful login, ensure the message box in the chatbot interface is highlighted.
3. **Click on the "Save to History":**
  - Locate and click the "Save to History" button.
4. **Verify that "Save to History" is highlighted:**
  - Ensure the "Save to History" button is highlighted after it is clicked.

**5. Verify that "New Chat" is highlighted:**

- Ensure that the "New Chat" button is highlighted after the "Save to History" operation is completed.

**Expected Result:**

- The user successfully logs in.
- The message box is highlighted.
- Clicking "Save to History" successfully highlights it.
- The "New Chat" button is highlighted after saving to history.

**Actual Result: Pass**

## **Test Case 11: Successfully Create a New User While Authenticated**

**Test Description:**

Verify that an authenticated user can successfully create a new user by sending a POST request with valid user data.

**Preconditions:**

- The user is authenticated.
- The API endpoint is available.
- New user data exists and is valid.

**Test Steps:**

**1. Authenticate the user:**

- Ensure the user is authenticated using valid credentials.

**2. Prepare the new user data:**

- Gather valid data for the new user (e.g., username, email, password, role, etc.).

**3. Prepare the POST request:**

- Set the API endpoint to "[/api/v1/users/user](#)".
- Include the new user data in the request body.

**4. Send the POST request:**

- Send a POST HTTP request to the specified endpoint with the new user data.

**5. Verify the user is created:**

- Check that the server responds with a successful status code (e.g., HTTP 201 Created).
- Confirm the new user is listed in the system or database (optional if applicable).

**Expected Result:**

- The server responds with HTTP status code **201 Created**.
- The new user is successfully created and stored in the system.

**Actual Result: Pass****Test Case 12: Valid Chatbot Request Asking for Class Schedule****Test Description:**

Verify that the chatbot API returns a 200 status code when requesting a class schedule for CS web development.

**Preconditions:**

- The API endpoint is available.

**Test Steps:**

1. **Set the POST chatbot API endpoint.**
  - Ensure the chatbot API endpoint is correctly set.
2. **Send a POST request with the message:**
  - Send the following message: "Can you create a class schedule for CS web development?".

**Expected Result:**

- The server responds with an HTTP status code **200 OK**.

**Actual Result: Pass****Test Case 13: Valid Chatbot Request Asking for Project Ideas****Test Description:**

Verify that the chatbot API returns a 200 status code when requesting project ideas for web development.

**Preconditions:**

- The API endpoint is available.

**Test Steps:**

1. **Set the POST chatbot API endpoint.**
  - Ensure the chatbot API endpoint is correctly set.
2. **Send a POST request with the message:**
  - Send the following message: "Can you suggest project ideas for Web development?".

**Expected Result:**

- The server responds with an HTTP status code 200 OK.

**Actual Result: Pass**

### **Test Case 14: Valid Chatbot Request Asking for Career Advice**

**Test Description:**

Verify that the chatbot API returns a 200 status code when requesting career paths for web development.

**Preconditions:**

- The API endpoint is available.

**Test Steps:**

1. **Set the POST chatbot API endpoint.**
  - Ensure the chatbot API endpoint is correctly set.
2. **Send a POST request with the message:**
  - Send the following message: "What career paths are available for web development?".

**Expected Result:**

- The server responds with an HTTP status code 200 OK.

**Actual Result: Pass**

### **Test Case 15: Valid Chatbot Request Asking for Prerequisites**

**Test Description:**

Verify that the chatbot API returns a 200 status code when requesting prerequisites for advanced web development courses.

**Preconditions:**

- The API endpoint is available.

**Test Steps:**

1. **Set the POST chatbot API endpoint.**
  - Ensure the chatbot API endpoint is correctly set.
2. **Send a POST request with the message:**
  - Send the following message: "What are the prerequisites for advanced web development courses?".

**Expected Result:**

- The server responds with an HTTP status code 200 OK.

**Actual Result: Pass****Test Case 16: Valid Chatbot Request Asking for Course Recommendations****Test Description:**

Verify that the chatbot API returns a 200 status code when requesting course recommendations for a specialization in web development.

**Preconditions:**

- The API endpoint is available.

**Test Steps:**

1. **Set the POST chatbot API endpoint.**
  - Ensure the chatbot API endpoint is correctly set.
2. **Send a POST request with the message:**
  - Send the following message: "Which courses should I take for a specialization in web development?".

**Expected Result:**

- The server responds with an HTTP status code 200 OK.

**Actual Result: Pass**

## Testing Metrics

- **Number of test cases:** 32
- Number of manual test cases: 16 - All Passed
- Number of automated test cases: 16 - All Passed
- **Test coverage:** 100% (All test scenarios related to the chatbot and login functionalities are covered).
- Test Report: <https://reports.cucumber.io/reports/9666f6b7-cc1a-4746-b900-b2be50c07041>

## References

- BU MET CS Team 1.** (2024). Project documentation (SPPP, SPPP risk management, Progress Report, SDD, Readme.md). N. Liew, N. Foithong, A. Singh, B. Cevik, Y. Liu, P. Chantarapornrat (Authors).
- Okta.** (2023). Authentication API documentation. Retrieved from <https://developer.okta.com/docs/reference/api-overview/>
- Langchain.** (2023). API documentation for OpenAI ChatGPT 4.0, chat history generation, RAG, and session management. Retrieved from <https://docs.langchain.com/docs/>
- CS673 Course Team.** (2024). Notes from CS673 slides. Blackboard Course MS.
- Spring.io.** (2023). Spring Boot documentation. Retrieved from <https://spring.io/projects/spring-boot>
- Docker, Inc.** (2023). Docker documentation. Retrieved from <https://docs.docker.com/>
- Axios.** (2023). Axios documentation. Retrieved from <https://axios-http.com/docs/intro>
- Atlassian.** (2023). JIRA documentation. Retrieved from <https://support.atlassian.com/jira-software-cloud/docs/>
- GitHub.** (2023). GitHub documentation. Retrieved from <https://docs.github.com/en>
- PostgreSQL Global Development Group.** (2023). PostgreSQL documentation. Retrieved from <https://www.postgresql.org/docs/>
- Foithong, N.** (2024, September 18). Chat AI Bot - 673ONE. Figma. Retrieved from <https://www.figma.com/design/gjNG1bADwnFxgDqclMwQVQ/Chat-AI-Bot---673ONE?node-id=0-1&node-type=canvas>
- Braude, E., & Bernstein, M. E.** (2016). Software engineering: Modern approaches (2nd ed.). Waveland Press, Inc.
- Martin, R. C.** (2003). Agile software development: Principles, patterns, and practices.
- Bruegge, B., & Dutoit, A. H.** (2010). Object-oriented software engineering: Using UML, patterns, and Java.
- Pfleeger, S. L., & Atlee, J. M.** (2010). Software engineering: Theory and practice.
- Pressman, R. S.** (2014). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill.
- Van Vliet, H.** (2008). Software engineering: Principles and practice.
- Sommerville, I.** (2016). Software engineering (10th ed.).

- Sommerville, I.** (2011). *Engineering software products: An introduction to modern software engineering*.
- Farley, D.** (2022). *Modern software engineering: Doing what works to build better software faster*.
- Brooks, F. P., Jr.** (1995). *The mythical man month: Essays on software engineering* (2nd ed.). Addison-Wesley.
- Freeman, E., Freeman, E., Bates, B., & Sierra, K.** (2004). *Head first design patterns*. O'Reilly Media.
- Fowler, M., Beck, K., & Roberts, D.** (2019). *Refactoring: Improving the design of existing code* (2nd ed.). Addison-Wesley.
- McConnell, S.** (2004). *Code complete: A practical handbook of software construction* (2nd ed.). Microsoft Press.
- Martin, R. C.** (2008). *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.
- Thomas, D., & Hunt, A.** (2019). *The pragmatic programmer: Your journey to mastery* (20th Anniversary ed.). Addison-Wesley.
- Winters, T., Mansreck, T., & Wright, H.** (2020). *Software engineering at Google: Lessons learned from programming over time*. O'Reilly Media.
- Humble, J., & Farley, D.** (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley.
- Kim, G., Behr, K., Spafford, G., & Ruen, C.** (2018). *The phoenix project: A novel about IT, DevOps, and helping your business win* (3rd ed.). IT Revolution Press.
- Forsgren, N., Humble, J., & Kim, G.** (2018). *Accelerate: The science of lean software and DevOps: Building and scaling high-performance organizations*. IT Revolution Press.
- Kim, G., Humble, J., Debois, P., Willis, J., & Forsgren, N.** (2016). *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. IT Revolution Press.
- Farley, D.** (2021). *Continuous delivery pipelines: How to build better software faster*. O'Reilly Media.

## Glossary of Terms

**Academic Advisor:** A faculty or staff member who guides students on academic courses, degree requirements, and career planning.

**AI (Artificial Intelligence):** The simulation of human intelligence by machines, specifically algorithms that allow computers to perform tasks like understanding natural language and making decisions. This project integrates OpenAI's ChatGPT 4.0 and Llama 2 for real-time course recommendations.

**API (Application Programming Interface):** A set of protocols and tools that allow different software components to communicate and share data. In this project, APIs connect the front-end application to back-end services, enabling data exchange between the chatbot, databases, and AI models.

**Application:** A software program designed to perform a specific function for the user.

**Authentication:** The process of verifying the identity of a user to ensure secure access. Okta is used in the project for handling user authentication, ensuring only authorized users can log in.

**Axios:** A JavaScript library for making HTTP requests from Node.js or the browser.

**AWS (Amazon Web Services):** A cloud computing platform offering services like storage, databases, and AI tools.

**Backend:** The part of the system responsible for connecting databases, tools, APIs, and services to frontend components.

**Branches:** Versions of a codebase used for managing features or changes before merging into the main code.

**Bugs:** Errors in code causing malfunction or failure in the system.

**Caching:** A mechanism to temporarily store data for quick access, reducing load times. This project considers using caching for faster page loads by retaining frequently accessed data.

**Chat:** A platform for real-time communication via text, voice, or video.

**Chatbot:** An AI-driven system designed to engage in conversations with users, providing information and assistance based on user queries.

**ChatGPT 4.0:** The AI model used in the web application.

**Components:** Reusable parts of an application, such as UI elements or modular code.

**Database:** An organized collection of data stored electronically.

**Docker:** A platform that uses containers to package and deploy applications, ensuring that software runs the same way regardless of the environment. Docker is an optional tool in this project for maintaining consistent development and production environments.

**Fetch API:** A JavaScript API for making network requests to servers.

**Figma:** A cloud-based design tool for interface design and prototyping.

**Framework:** A collection of tools and libraries to streamline software development.

**Frontend:** The user-facing side of the web application, built using React.js. It provides the interface through which users interact with the chatbot and access course recommendations.

**CI/CD (Continuous Integration/Continuous Deployment):** A development practice where code changes are automatically tested and deployed, ensuring that new features or bug fixes are regularly integrated into the project. GitHub Actions is used for this purpose.

**Configuration/Config:** A set of parameters to customize a program's behavior.

**GitHub:** A file management system for collaborative software development.

**HTTPs:** Hypertext Transfer Protocol Secure, a secure version of HTTP.

**Issues:** Problems or tasks tracked in project management tools.

**Java:** A programming language used for building applications.

**JIRA:** A project management tool for tracking issues, bugs, and tasks.

**Langchain:** A company offering AI tooling for Retrieval-Augmented Generation (RAG) and session management.

**LLM (Large Language Model):** A type of AI model trained on vast amounts of text data to understand and generate human-like text responses.

**Management:** Coordinating resources and tasks to achieve objectives.

**Microservices:** A software architecture where applications are structured as independent, deployable services.

**Okta:** A cloud-based identity and access management service.

**OpenAI:** A company that provides the ChatGPT AI model.

**PostgreSQL:** An open-source relational database management system used for storing structured data efficiently and chat history and other user data in this project.

**Python3:** A version of the Python programming language.

**RAG (Retrieval-Augmented Generation):** A technique that combines retrieval of relevant information and generative responses to improve the accuracy of chatbot replies.

**React.js:** A JavaScript library used to build the front end of the application. It allows for the creation of interactive user interfaces and handles the chat interaction between the user and the AI.

**Redux:** A state management library for JavaScript applications, often used with React.

**RESTful API:** A set of guidelines for building APIs using standard HTTP methods.

**Regression Testing:** A software testing practice that ensures new code changes don't adversely affect existing functionality. The project implements regression testing to ensure that updates to the AI chatbot don't disrupt other features.

**REST Assured:** A Java library used for testing RESTful APIs. The project uses REST Assured to validate that the backend APIs respond correctly and efficiently.

**Selenium:** A testing framework used to automate web browsers, validating the chatbot's user interface. Selenium ensures the front end functions properly after each code change.

**Spring Boot:** A Java-based framework used for building and deploying back-end services, handling API requests, and managing interactions with the databases.

**TensorFlow:** A machine learning framework used to develop and train AI models. TensorFlow may be used to enhance the chatbot's learning capabilities.

**Unit Testing:** A type of software testing where individual components of the application are tested in isolation. This project uses unit testing to ensure each module (React components, Java services, and AI models) works correctly before integrating them.

**WebSockets:** A protocol enabling real-time, two-way communication between a client and server, allowing for instant updates during chat sessions.

**Workflow:** A sequence of steps to complete a task or achieve an objective.