DegreeFlow

V&V document

Group 04

Supervisor	
Name: **Email:** **Organization:**	Luke Schuurman schuurml@mcmaster.ca McMaster Engineering Society
Team Member 1	
Name: **Email:** **Student Number:**	Anderson Zhou zhoua35@mcmaster.ca 400391994
Team Member 2	
Name: **Email:** **Student Number:**	Aniruddh Arora aroraa33@mcmaster.ca 400318688
Team Member 3	
Name: **Email:** **Student Number:**	Anupam Raj raja5@mcmaster.ca 400351087
Team Member 4	
Name: **Email:** **Student Number:**	Harshit Sharma sharmh17@mcmaster.ca 400349986
Team Member 5	
Name: **Email:** **Student Number:**	Sarah Neelands neelands@mcmaster.ca 400389835
Team Member 6	
Name: **Email:** **Student Number:**	Yanchun Wang wang146@mcmaster.ca 400369124

Version Number	Authors	Description	Date
0	● Anderson Zhou ● Aniruddh Arora ● Anupam Raj ● Harshit Sharma ● Sarah Neelands ● Yanchun Wang	Initial Draft	Feb 9th, 2024

1.0 Table of Contents

2.0 Project Description

DegreeFlow is a web-based academic planning tool designed to help McMaster University students efficiently track their degree progress, manage course selections, and receive personalized recommendations. The system integrates with university resources such as the Academic Calendar and Mosaic to provide accurate course requirements, schedule conflict detection, and real-time seat availability notifications. By allowing students to upload their unofficial transcripts, DegreeFlow extracts and analyzes academic data to generate tailored course suggestions and ensure timely graduation. This document builds upon previous project documentation, including the Software Requirements Specification (SRS) and Design Document.

Link to SRS document ; Link to Design document

3.0 Component Test Plan

3.1 PDF Parsing Component

3.1.1 Unit Tests

The PDF Parsing component extracts academic data from transcripts. Unit tests validate extraction accuracy, recency validation, and course recommendations. Testing is conducted using TestNG in Java Spring Boot.

3.1.1.1 Transcript Parsing and Data Extraction

- Verify that Apache PDFBox extracts course codes, grades, terms, and GPA. Ensure co-op terms are detected, invalid PDFs are flagged, and extracted data is structured in JSON before integration.

3.1.1.2 Transcript Recency Validation

- Check that issue dates are extracted and outdated transcripts (older than three months) trigger warnings but remain processable. Transcripts missing dates are flagged for review.

3.1.1.3 Course Recommendation Accuracy

- Ensure extracted courses align with McMaster's degree requirements using the Academic Calendar API. Validate that recommendations follow prerequisites, co-requisites, and co-op adjustments.

3.1.1.4 Error Handling and Security

- Ensure corrupted PDFs do not crash the system. Validate security measures to prevent unauthorized access and ensure sensitive student data is not exposed.

3.1.2 Performance Tests and Metrics

Performance testing evaluates extraction speed, data processing time, and scalability using Locust for load testing and TestNG for benchmarking.

- **Transcript processing time:** <3 seconds per page.
- **Data extraction accuracy:** ≥ 99.9%, calculated as:
  $Accuracy = (\frac{Number\ of\ correctly\ extracted\ fields}{Total\ number\ of\ expected\ fields}) * 100$
- **System load handling:** Supports 500+ concurrent users.
- **API response time:** Course recommendation retrieval <2 seconds.

These tests simulate real-world scenarios where multiple students upload transcripts simultaneously to ensure efficiency.

3.2 Seat Alert & Real-Time Notifications

3.2.1 Unit Tests

The Seat Alert & Real-Time Notifications component ensures students receive timely alerts when seats become available in previously full courses. Since it involves querying McMaster's Mosaic API, handling database storage, and triggering real-time notifications, our tests focus on data accuracy, API responsiveness, and notification delivery reliability. Testing will be conducted using TestNG for backend unit testing (Java Spring Boot) and Selenium for UI testing (React.js frontend).

3.2.1.1 Seat Availability Query and Subscription Handling

The unit testing ensures that seat availability queries return accurate results and that student subscriptions are correctly processed.

- Verifying that the system correctly retrieves seat availability data from the Mosaic API at the scheduled 20–30 second intervals.
- Ensuring that student subscription requests are stored in the database with accurate course information (course code, term, section).
- Validating that students receive confirmation when successfully subscribing to seat alerts..

3.2.1.2 Notification Delivery Accuracy

Unit tests ensure that notifications are sent only when a seat is genuinely available and reach the correct students promptly.

- Testing that the system triggers a notification only when a seat is confirmed as available.
- Ensuring that email notifications are sent to the correct McMaster email address associated with the student.
- Confirming that notifications contain accurate course details, including course code, section, and a direct link to Mosaic for enrollment..

3.2.1.3 Subscription Management

Unit tests ensure that students can successfully subscribe, modify, and remove seat alerts through the system.

- Checking that students can update their notification preferences (e.g., preferred contact method).
- Verifying that students can delete or modify subscriptions without errors.

- Ensuring that only authorized users can manage their subscriptions.
- Testing that unsubscribed students no longer receive seat availability alerts.

3.2.2 Performance Tests and Metrics

The primary performance metrics for the Seat Alert system focus on API query response time, database efficiency, and notification delivery speed. Using **Locust** for load testing and **TestNG** for performance testing, we will measure:

- **Seat availability query response time:** Must complete within **2 seconds** to ensure real-time data retrieval.
- **Notification delivery time:** Email alerts should be sent within **5 seconds** of detecting an available seat.
- **Subscription processing time:** Adding or updating a subscription should take less than **1 second**.
- **System load handling:** The platform should support at least **500 concurrent users** without degradation.

3.3 Degree Progress Tracker

3.3.1 Unit Tests

The Degree Progress Tracking component accurately extracts, categorizes, and displays students' academic progress. Since it involves PDF parsing, database storage, and real-time visualization, our tests focus on data extraction, categorization accuracy, and UI responsiveness. Testing will be conducted using: TestNG for backend unit testing (Java Spring Boot) and Selenium for UI testing (React.js frontend).

3.3.1.1 Degree Transcript Parsing and Course Extraction

The unit testing ensures that the PDF parser extracts course codes, units, and grades correctly from the student's unofficial transcript.

- Verifying that the PDF parser correctly extracts course codes, credit values, and grades from the unofficial transcript.
- Ensuring that extracted courses are categorized accurately into requisite, elective, technical elective, and minor courses (if applicable).
- Validating that outdated transcripts are detected and rejected using timestamp verification.
- Confirming that incorrectly formatted transcripts are flagged as errors and do not update student records.
- Ensuring that the categorized courses match the degree requirements retrieved from the McMaster Academic Calendar API.

3.3.1.2 Course Categorization Accuracy

The unit testing ensures that extracted courses from the unofficial transcript are correctly categorized into the appropriate degree requirement sections: requisite, elective, technical elective, and minor courses (if applicable). The system must accurately match courses against McMaster’s Academic Calendar.

- Verifying that courses extracted from the transcript are categorized correctly into requisite, elective, technical elective, or minor categories.
- Ensuring that the categorization logic correctly matches course codes with McMaster's degree requirements using the Academic Calendar API.
- Validating that minor courses are properly categorized if the student has declared a minor.
- Ensuring that changes in the Academic Calendar reflect correctly in the categorization process.
- Checking that all categorized courses are correctly stored in the PostgreSQL database without duplication or missing information.

3.3.1.3 Progress Calculation and Display

The unit testing ensures that degree progress calculations are accurate and correctly reflected in the UI. The system must dynamically update the progress bars for total degree completion, electives, technical electives, and minor requirements.

- Verifying that the system correctly calculates the percentage of total degree completion based on completed course credits.
- Ensuring that elective, technical elective, and minor progress percentages are calculated separately and accurately.
- Confirming that progress calculations adjust dynamically when new transcripts are uploaded.
- Verifying that degree progress updates are stored and retrieved correctly from the PostgreSQL database.
- Checking that outdated transcripts do not overwrite previously stored progress data.

3.3.2 Performance Tests and Metrics

The main metrics we want to test for the Degree Progress Tracking component are latency, load balancing, and data accuracy. We will use Locust (https://locust.io) to simulate multiple concurrent users. This will specifically be used to monitor how our application handles load as the number of concurrent users increases and ensure that progress updates remain accurate and responsive under peak usage conditions. for performance metrics:

- Degree progress update time < 2 seconds after transcript upload
- API response time (Mosaic data retrieval)< 3 seconds
- System load handling Supports 500+ concurrent users without degradation
- The number of correctly matched courses divided by the total number of courses in a student's advisement report times 100 must be > 99.9%

3.4 Ratings for Each Course

3.4.1 Unit Tests

The Course Rating component enables students to submit ratings for courses they have taken or are currently enrolled in and display the accurate average rating of each course. Since it involves user input validation, database operations, and real-time UI updates, our tests focus on rating submission accuracy, data persistence, and display responsiveness. Testing will be conducted using TestNG for backend unit testing (Java Spring Boot) and Selenium for UI testing (React.js frontend).

3.4.1.1 Rating Submission and Validation

The unit testing ensures that the rating system properly validates and processes student submissions:

- Verifying that only enrolled students can submit ratings for courses they are taking or have taken
- Ensuring that rating values are restricted to the valid range (1-5 stars)
- Confirming that students cannot submit multiple ratings for the same course
- Validating that submission timestamps and student information are accurately recorded
- Ensuring that rating updates properly overwrite previous submissions from the same student
- Checking that invalid submissions are rejected with appropriate error messages

3.4.1.2 Rating Display and UI Updates

The unit testing ensures that ratings are displayed accurately and the UI updates dynamically:

- Verifying that the star rating display correctly reflects the current rating value
- Ensuring that average ratings are displayed with the correct precision
- Confirming that rating counts and distributions are accurately presented
- Testing the responsive design of rating components across different screen sizes
- Checking that rating submission feedback is clearly displayed to users

3.4.1.3 Rating Storage and Retrieval

The unit testing ensures that course ratings are properly and accurately stored in the database:

- Verifying that new ratings are correctly stored in the PostgreSQL database
- Ensuring that rating updates do not create duplicate entries
- Validating that the average rating calculation is accurate and updates in real-time
- Confirming that course rating history is maintained properly
- Verifying that rating data is correctly associated with specific courses and terms

3.4.2 Performance Tests and Metrics

The main metrics we want to test for the Course Rating component are response time, data accuracy, and system load handling. We will use **Locust** for load testing and **TestNG** for performance testing to create test scenarios that mimic real-world usage patterns. This will specifically monitor how our application handles load during peak usage periods like end of term when many students submit ratings simultaneously.

Performance metrics include:

- Rating submission time < 2 seconds after clicking submit
- Average rating calculation < 1 second per course
- System handling: 100+ concurrent users without degradation
- Data accuracy: the number of correctly stored ratings divided by the total number of submitted ratings times 100 must be > 99.9%

These tests will ensure the rating system remains responsive and accurate even during high-usage periods.

3.5 Automatic Scheduling

3.5.1 Unit Tests

The Automatic Scheduling component generates personalized timetables based on a student's academic history, degree requirements, and user-defined constraints (credit limits, preferred times, no Friday classes, etc.). Since it integrates data from PDF Parsing, the McMaster Mosaic API, and seat alerts, our tests focus on the correctness of schedule generation, conflict detection, and the responsiveness of UI updates. Testing will be conducted using TestNG for backend unit testing (Java Spring Boot) and Selenium for UI testing (React.js frontend).

3.5.1.1 Schedule Generation Accuracy

The unit testing ensures that the scheduling system accurately builds course timetables based on all required data:

- Prerequisite Validation - Validate that only courses for which prerequisites are met appear in the generated schedule.
- Time-Conflict Avoidance: Ensure that overlapping class/lab times are detected and excluded.
- Error Handling: Invalid requests (e.g., non-existent course codes) should return descriptive error messages without crashing.

3.5.1.2 Conflict Detection and Constraints

The unit testing ensures that the generated schedules meet user-defined constraints and avoid conflicts:

- Time-Conflict Avoidance: Ensure that overlapping class/lab times are detected and excluded.
- User Preferences: Validating that credit cap, no-morning-class requests, or free-day requirements are integrated.
- Alternate Schedule Options: Confirm that multiple valid timetables are suggested if a single solution cannot fulfill all constraints.

3.5.1.3 Timetable Storage and Retrieval

The unit testing ensures that schedules can be properly saved, accessed, and modified:

- Saving Schedules: Verify that generated timetables are correctly stored in the PostgreSQL database under each student's profile.
- Viewing and Editing: Ensure that previously saved timetables can be retrieved, updated, or deleted without data conflicts.
- Data Consistency: Check that updated seat availability or changes in degree requirements prompt a re-check of stored schedules.

3.5.2 Performance Tests and Metrics

The primary metrics for the Automatic Scheduling component involve schedule generation speed, system capacity, and timely updates. Testing uses Locust for load simulation and TestNG for performance benchmarking. This testing will assess the application's performance under heavy load, when many students submit ratings concurrently :

- Schedule Generation Time < 3 seconds for each new schedule request under normal load.
- Concurrency Handling: The system should handle at least 500 concurrent scheduling requests from students without critical slowdowns or crashes.
- Real-Time Updates: Changes in course seats or prerequisites should reflect in newly generated schedules in under 2 seconds of detection.
- Data Accuracy: All proposed timetables must remain 100% conflict-free, respecting prerequisites and user constraints.

These performance tests ensure that Automatic Scheduling runs efficiently and responsively at scale while consistently providing error-free and user-appropriate timetables.

3.6 What if scenarios

3.6.1 Unit Tests

The What if scenarios component generates possible undergraduate degree plans based on the student's major(s), minor(s), co-op terms, gap years and desired course load. It will utilize the McMaster Academic Calendar thus the testing will be for the correctness of the degree plan generation, the correctness of the UI generation and the response time of the degree plan generation. TestNG for backend unit testing (Java Spring Boot) and Selenium for UI testing (React.js frontend).

3.6.1.1 Degree Plan Generation Correctness

The unit testing ensures that the degree plan generation is correct based on the required data:

- Verify that all courses are listed within the terms that they are offered in
- Verify that for all courses their listed term is after the completion all of their prerequisites
- Verify that if the user specified degree plan is possible, ensure that all of their required restraint (required courses, credit, timespan, courseload … etc) are met.

3.6.1.2 UI Generation Correctness

This unit testing ensures that the generated UI remains correct with the degree plan:

- Verify that all courses are presented in the terms they are offered in
- Verify that the schedule is dynamically updated with user input submissions
- Verify that all the courses listed within the degree plan is presented in the UI

3.6.3 Performance Tests and Metrics

The performance metrics for the What If component includes McMaster Academic Calendar access speed, degree plan generation speed, data accuracy and system capacity. Locust will be used for load testing and TestNG will be used for performance benchmarking.

- Degree plan generation < 3 seconds for each new request under normal traffic
- The system should handle at least 500 concurrent requests without performance impacts
- Each access to the McMaster Academic Calendar should take < 1 second
- All data generated must be 100% conflict free, respecting university course enrollment policies.