# DegreeFlow

## V&V document

## Group 04

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

| Version Number | Authors | Description | Date |
|---|---|---|---|
| 0 | ● Anderson Zhou<br>● Aniruddh Arora<br>● Anupam Raj<br>● Harshit Sharma<br>● Sarah Neelands<br>● Yanchun Wang | Initial Draft | Feb 9th, 2025 |
| **1** | ● Anderson Zhou<br>● Aniruddh Arora<br>● Anupam Raj<br>● Harshit Sharma<br>● Sarah Neelands<br>● Yanchun Wang | Final Submission | April 3,2025 |

**1.0 Table of Contents**

2.0 Project Description ..... 4
3.0 Component Test Plan ..... 4
3.1 PDF Parsing Component ..... 4
3.1.1 Unit Tests ..... 4
3.1.1.1 Transcript Parsing and Data Extraction ..... 4
3.1.1.2 Transcript Recency Validation ..... 4
3.1.1.3 Course Recommendation Accuracy ..... 4
3.1.1.4 Error Handling and Security ..... 4
3.1.2 Performance Tests and Metrics ..... 4
3.2 Seat Alert & Real-Time Notifications ..... 5
3.2.1 Unit Tests ..... 5
3.2.1.1 Seat Availability Query and Subscription Handling .....5
3.2.1.2 Notification Delivery Accuracy .....5
3.2.1.3 Subscription Management .....5
3.2.2 Performance Tests and Metrics ..... 6
3.3 Ratings for Each Course ..... 7
3.3.1 Unit Tests ..... 7
3.3.1.1 Rating Submission and Validation ..... 8
3.3.1.2 Rating Display and UI Updates ..... 8
3.3.1.3 Rating Storage and Retrieval ..... 8
3.3.2 Performance Tests and Metrics ..... 8
3.4 Automatic Scheduling ..... 9
3.4.1 Unit Tests ..... 9
3.4.1.1 Schedule Generation Accuracy ..... 9
3.4.1.2 Rating Display and UI Updates ..... 9
3.4.1.3 Timetable Storage and Retrieval ..... 9
3.4.2 Performance Tests and Metrics ..... 9
3.5 What If Scenarios ..... 10
3.5.1 Unit Tests ..... 10
3.5.1.1 Degree Plan Generation Correctness ..... 10
3.5.1.2 UI Generation Correctness ..... 10
3.5.2 Performance Tests and Metrics ..... 10

## 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 (v2) ; Link to Design document (v2)

## 3.0 Component Test Plan

### 3.1 PDF Parsing Component

We are currently testing the PDF Parsing component, which is responsible for extracting structured academic data from student transcripts. The testing is being carried out using **TestNG** within a **Java Spring Boot** environment.

---

#### 3.1.1 Unit Tests

##### 3.1.1.1 Transcript Parsing and Data Extraction

We are verifying that the PDF parsing logic (using **Apache PDFBox**) correctly extracts course codes, grades, academic terms, and GPAs. Unit tests also ensure:

- Co-op terms are properly detected.
- Invalid or malformed PDFs are flagged.
- Extracted data is correctly structured into JSON for downstream use.

##### 3.1.1.2 Transcript Recency Validation

Tests confirm that:

- The issue date of the transcript is correctly extracted.
- Transcripts older than 3 months trigger appropriate warnings while remaining processable.
- Transcripts missing an issue date are flagged for manual review.

##### 3.1.1.3 Course Recommendation Accuracy

We are validating that:

- Extracted courses align with McMaster's official degree requirements (via the Academic Calendar

API).

- Generated course recommendations respect prerequisites, co-requisites, and co-op term requirements.

#### 3.1.1.4 Error Handling and Security

We ensure the parser handles corrupted or non-standard PDFs gracefully without crashing the system. We also validate that:

- Unauthorized access is blocked.
- Sensitive student data remains protected at all times.

---

### 3.1.2 Performance Tests and Metrics

Performance testing is actively being conducted using **k6** for load simulation and **TestNG** for benchmarking. Our targets:

- **Transcript parsing speed**: Less than 3 seconds per page.
- **Data extraction accuracy**: At least 99.9%, measured as
  Accuracy = (Correctly extracted fields / Total expected fields) * 100
- **Concurrent user handling**: Support for 500+ simultaneous users uploading transcripts.
- **API latency**: Course recommendation retrieval within 2 seconds.

These performance tests replicate real-world scenarios—such as multiple concurrent uploads—to ensure the system remains efficient and responsive under load.

## 3.2 Seat Alert & Real-Time Notifications

### 3.2.1 Unit & Integration Tests

This component ensures students receive real-time notifications when seats open up in full courses. The system integrates:

- McMaster's Create My TimeTable API for live seat data
- PostgreSQL for subscription storage
- Email service for notifications
- A React-based frontend

Tests are written using **TestNG** (Java Spring Boot) for backend services and **Jest + Selenium** for automated UI interactions.

---

#### 3.2.1.1 Backend Unit Tests (TestNG)

Backend service tests validate core functionality:

- subscribeToSeatAlert(...) stores a subscription and sends confirmation email.
- getSubscriptions() fetches all subscriptions.
- checkSeatAvailability(course, term) correctly detects seat availability using mocked API responses.
- deleteSubscription(id) deletes user subscriptions.
- getSubscriptionById(id) fetches specific subscriptions by ID.
- Controller endpoints handle correct and incorrect inputs (e.g. subscribing to non-existent course).

---

#### 3.2.1.2 Frontend UI Tests (Selenium + Jest)

You've implemented a full Selenium test that:

- Navigates to the Seat Alert page: http://localhost:3000/seat-alert
- Fills out the form with:
    - Course Code: "COMPSCI 1MD3"
    - Email: "test@example.com"
    - Term: "Winter-2025"
- Clicks the Subscribe button.
- Waits for a confirmation alert and checks that it contains **"Subscribed successfully"**.

This test verifies:

- The form elements are interactable.
- Submissions trigger correct state changes.
- Success message is shown after form submission.

---

#### 3.2.1.3 Notification Delivery

Covered via backend unit tests:

- Emails are only triggered when seats are available.
- Correct recipient, message content (course/term), and delivery logic is validated using mock JavaMailSender.

---

#### 3.2.1.4 Subscription Management

- Subscriptions can be created, modified, or deleted using the REST API.
- UI form allows subscriptions.
- Unsubscribed users do not receive further alerts.
- Controller ensures only valid course/term pairs are processed.

---

### 3.2.2 Performance Tests and Metrics

These are our **target metrics**, which can be tested using **k6** and observed during heavy usage:

| Metric | Target |
|---|---|
| Seat availability response time | ≤ 2 seconds |
| Email delivery latency | ≤ 5 seconds post-detection |
| Subscription CRUD time | ≤ 1 second |
| Frontend subscription E2E flow | ≤ 3 seconds per attempt |
| Concurrent user support | 500+ users without slowdowns |

---

## 3.3 Course Filter Search & Difficulty Ratings for Each Course

### 3.3.1 Unit & Integration Tests

These components allow students to filter courses by various criteria and view/submit difficulty ratings. The system integrates:

- McMaster's Academic Calendar API for course data
- PostgreSQL for storing rating data
- React-based frontend with interactive filtering and rating UI
- API endpoints for course filtering and rating management

Tests are implemented using TestNG (Java Spring Boot) for backend services and Jest + Selenium for automated UI interactions.

#### 3.3.1.1 Backend Unit Tests (TestNG)

Backend service tests validate core functionality:

- *submitRating(email, courseCode, stars)* correctly stores new ratings and updates existing ones
- *getCourseSummary(courseCode)* calculates accurate average ratings and difficulty categories
- *getRatingByStudentAndCourse(email, courseCode)* retrieves individual student ratings
- Course filtering by subject code and level returns appropriate results

The RatingService tests verify:

- New ratings are properly saved to the database
- Existing ratings are updated rather than duplicated
- Average calculation and difficulty categorization are accurate
- Repository integration functions as expected

#### 3.3.1.2 Frontend UI Tests (Selenium + Jest)

The implemented Selenium test verifies the end-to-end user experience:

- Navigates to the FilterSelection page
- Expands the SUBJECT CODE filter and selects "COMPSCI"
- Opens the COURSE LEVEL filter and selects "Level 1"
- Clicks the "Apply Filters" button to filter courses
- Expands a specific course (COMPSCI 1XD3) to view details
- Opens the rating modal by clicking the "Rating" button

- Selects a 3-star difficulty rating
- Submits the rating and verifies the thank you message appears
- Confirms the rating statistics update with the correct difficulty category

This comprehensive test validates:

- Filter selection work correctly
- Course expansion shows detailed information
- Rating submission flow functions as expected
- UI updates appropriately after submission

#### 3.3.1.3 Rating Submission and Retrieval

Covered via backend unit tests:

- Ratings are properly stored with student email, course code, and star value
- Update logic prevents duplicate ratings from the same student
- Difficulty categories are assigned based on defined thresholds
- Controllers return appropriate HTTP responses for all cases

#### 3.3.1.4 Course Filtering

- Filtering combines two criteria (subject code, level)
- Backend controller supports subject and level-based filtering
- Frontend correctly displays filtered course results in a list

### 3.3.2 Performance Tests and Metrics

These are your **target metrics**, which can be tested using **k6** and observed during heavy usage:

| **Metric** | **Target** |
|---|---|
| Course filter response time | ≤ 2 seconds |
| Rating submission latency | ≤ 1.5 seconds |
| Rating statistics retrieval | ≤ 1.5 seconds |
| Concurrent user support | 200+ users without slowdown |

---

## 3.4 Automatic Scheduling

### 3.4.1 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).

#### 3.4.1.1 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.

#### 3.4.1.3 Timetable Storage and Retrieval

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

- Saving Schedules: Verify that generated schedules are correctly stored in the PostgreSQL database under each student's profile.
- Viewing and Editing: Ensure that previously saved schedules can be retrieved, updated, or deleted without data conflicts..

### 3.4.2 Performance Tests and Metrics

The primary metrics for the Automatic Scheduling component involve schedule generation speed, system capacity, and timely updates. Testing uses k6 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.

## 3.5 What if scenarios

### 3.5.1 Unit Tests

The What if scenarios component generates possible undergraduate degree plans based on the student's major(s), gap years and desired course load. It will utilize the McMaster Academic Calendar thus the testing will be for the completeness of the degree plan generation, the completeness 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).

#### 3.5.1.1 Api Parsing Completeness

The unit tests verifies

- That all the information from the Mosaic api is parsed completely and correctly
- That all api calls are made correctly and errors are caught
- That all front end submissions into the database are stored and retrieved correctly

#### 3.5.1.2 UI Generation Correctness

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

- Navigates to the select degree name button
- Click on the button
- Wait for list of all degrees to generate
- Select "BTec1-Automt Sys EngTech CO-OP" degree
- Submit the degree and wait for the schedules to generate

This test validates

- List of all degrees are retrieved properly
- Requirements of degree are generated properly
- UI buttons react dynamically to user input

### 3.5.3 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. k6 will be used for load testing and TestNG will be used for performance benchmarking.

- Degree plan generation < 15 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 < 3 second
- The k6 test doesn't pass the majority of the tests due to an empty database.