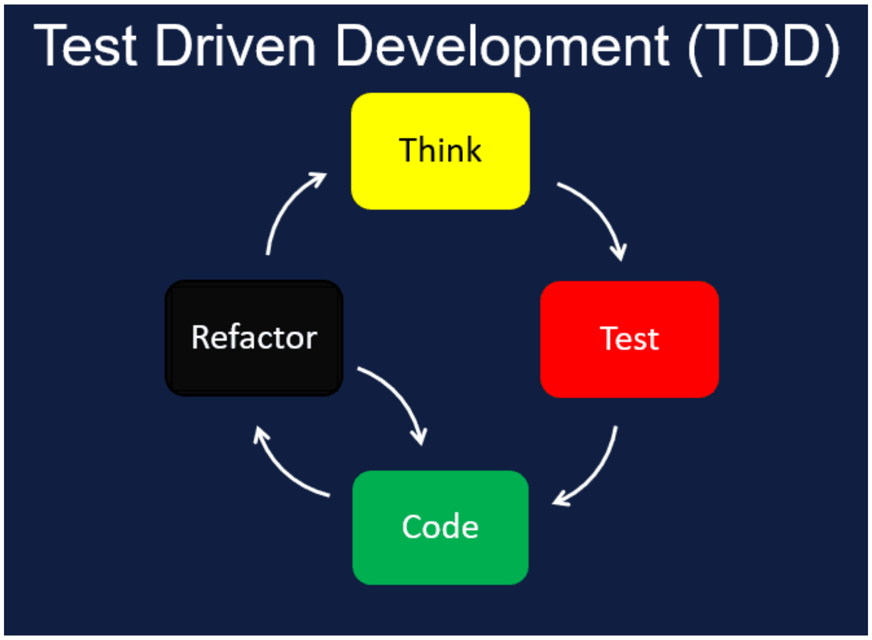
Test Driven Development (TDD) Report for “Smart Class Routine Management System”

Author: Akila Nipo

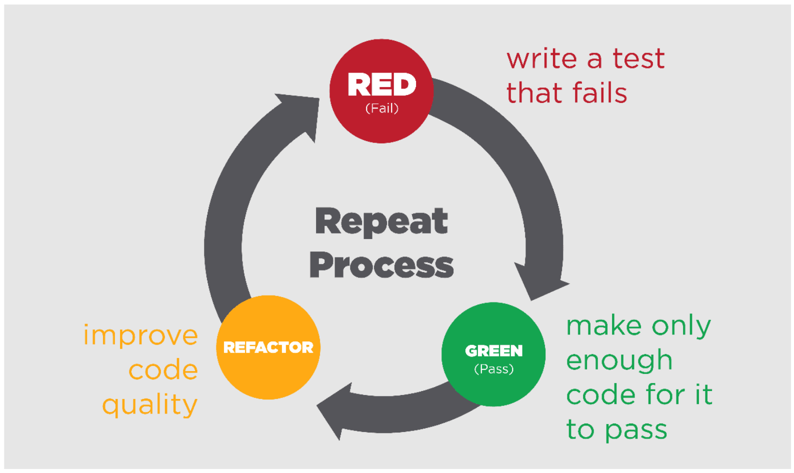
📚 Introduction

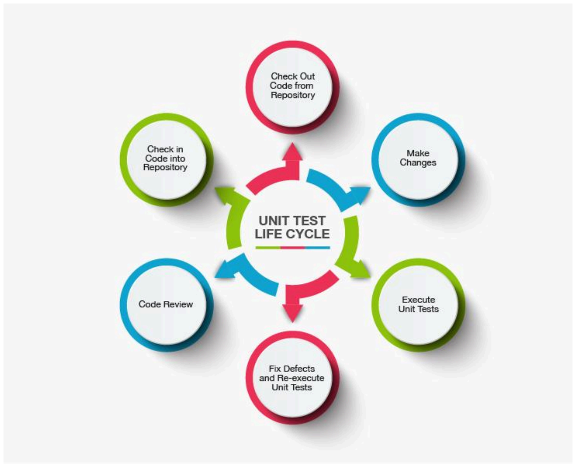
Test Driven Development (TDD) is a software development methodology where test cases are written before the actual code that fulfills them. It follows a short, repetitive development cycle to ensure correctness and precise design.



The main steps include:
➡️ 1. Add a Test: Write a test case based on function requirements.
➡️ 2. Run Tests: Confirm that the new test fails.
➡️ 3. Write Code: Develop the code required to pass the test.
➡️ 4. Run Tests Again: Ensure all tests pass.
➡️ 5. Refactor Code: Remove redundancy and optimize.
➡️ 6. Repeat: Continue the cycle to build up functionality.
🔁 *TDD focuses on improving the design of code while ensuring that functionality is always tested through iteration.*

📜 TDD vs. Unit Testing




UNIT TEST LIFE CYCLE
Check Out Code from Repository
Make Changes
Execute Unit Tests
Fix Defects and Re-execute Unit Tests
Code Review
Check in Code into Repository

Aspect	TDD	Unit Testing
Purpose	Drives design by ensuring tests are in place before coding	Validates individual units of code after implementation
Development Cycle	Test-first approach with continuous refactoring	Test-after approach with static test cases
Code Reliability	Ensures high reliability through iterative testing	Primarily confirms correctness without guiding design
Ideal Use Cases	Complex, evolving projects needing strong design alignment	Smaller projects or specific features within larger projects

---

⚡ Summary

- TDD: Ideal for projects requiring robust code design and maintainability.
- Unit Testing: Suitable for simple validation tasks and focused testing of specific sections.

🔧 Pre-requisites

➡️ Tools: Visual Studio Code
➡️ Framework: Node.js with Express
➡️ Unit Testing Tool: Mocha (with Chai for assertions)
➡️ Test Coverage Tool: `npm install nyc --save-dev`

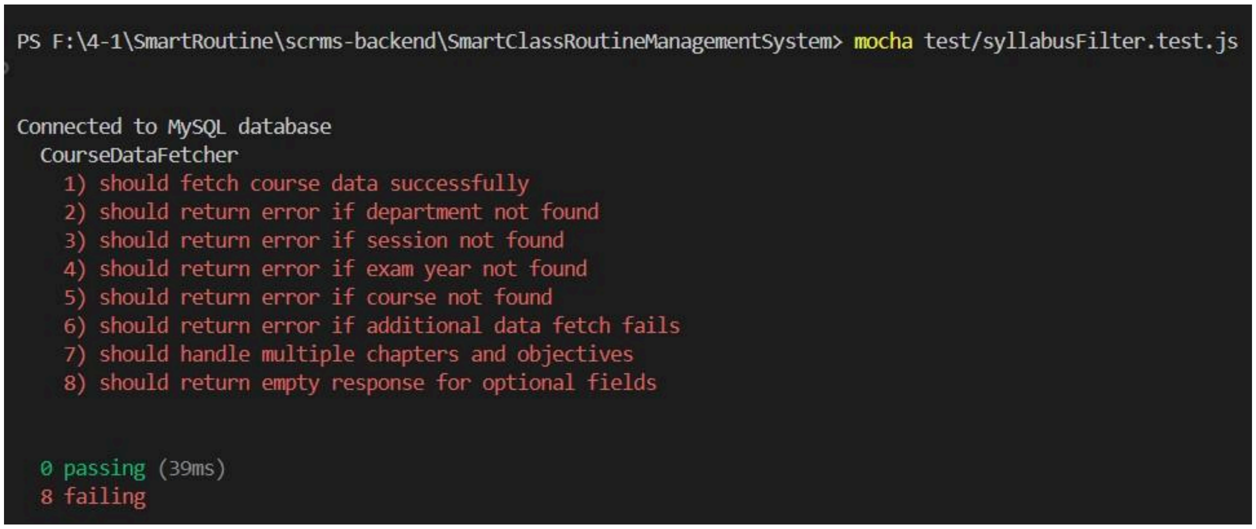
📋 Procedure

1. Setup Mocha: Create a `test` folder and add the test file (e.g., `syllabusFilter.test.js`).
2. Write Initial Test Cases: Design tests to verify each function's expected behavior. *Unit Testing Guide*
3. Run Initial Tests: Execute Mocha tests:
   `mocha test/syllabusFilter.test.js`
4. Implement Code: Write code to meet failing test cases.
5. Refactor and Repeat: Optimize code to ensure efficiency.

🚨 Test Case Fails

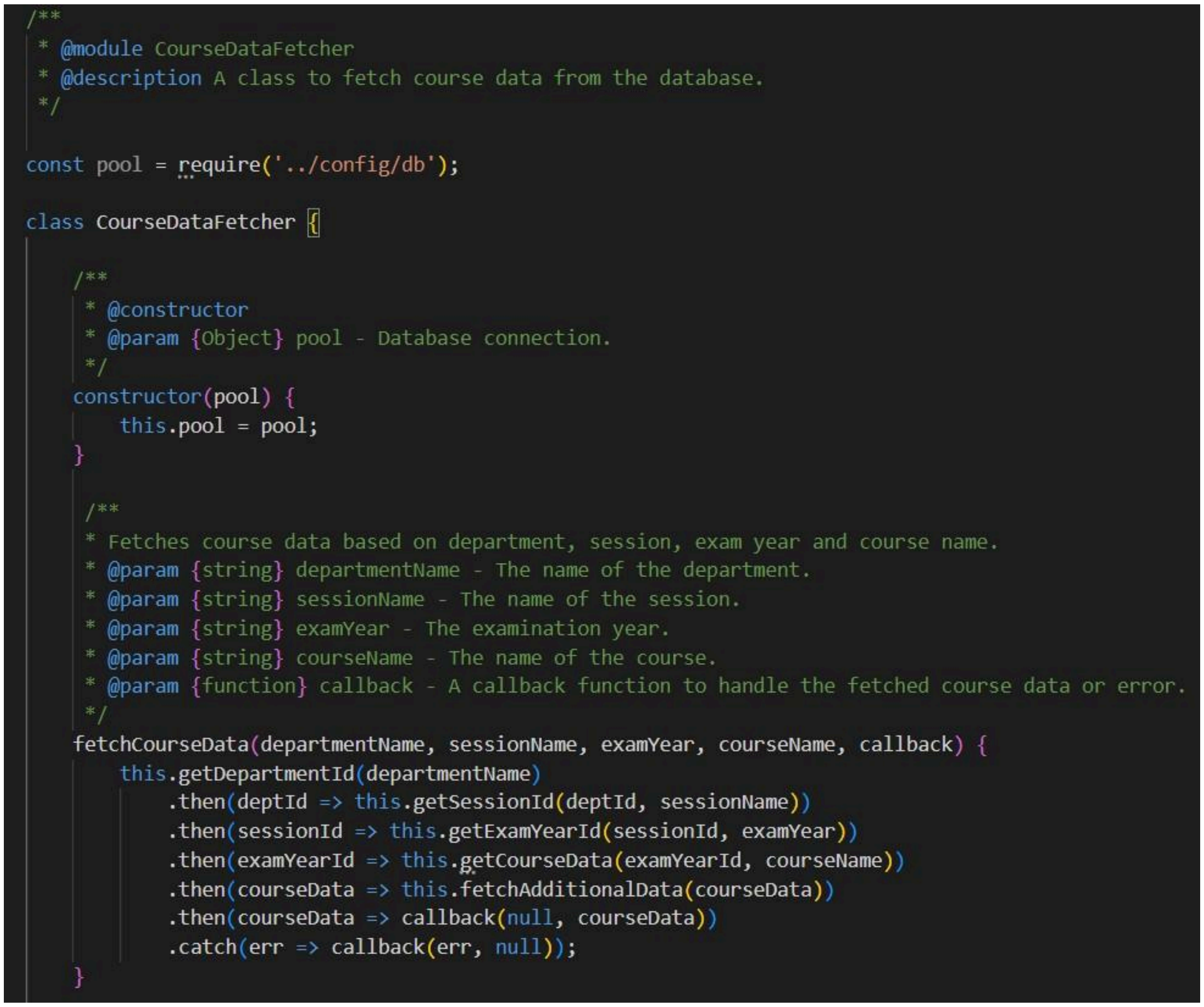
Example Feature: Filter Syllabus

- Initial Test Case Failure
- Feature: Filtering Syllabus
- Observation: Out of *8 test cases*, *all of them failed* due to missing conditions for fetching syllabus.



```
PS F:\4-1\SmartRoutine\scrms-backend\SmartClassRoutineManagementSystem> mocha test/syllabusFilter.test.js


Connected to MySQL database
  CourseDataFetcher
    1) should fetch course data successfully
    2) should return error if department not found
    3) should return error if session not found
    4) should return error if exam year not found
    5) should return error if course not found
    6) should return error if additional data fetch fails
    7) should handle multiple chapters and objectives
    8) should return empty response for optional fields


  0 passing (39ms)
  8 failing
```

At the time of failure, the function to filter syllabus was:

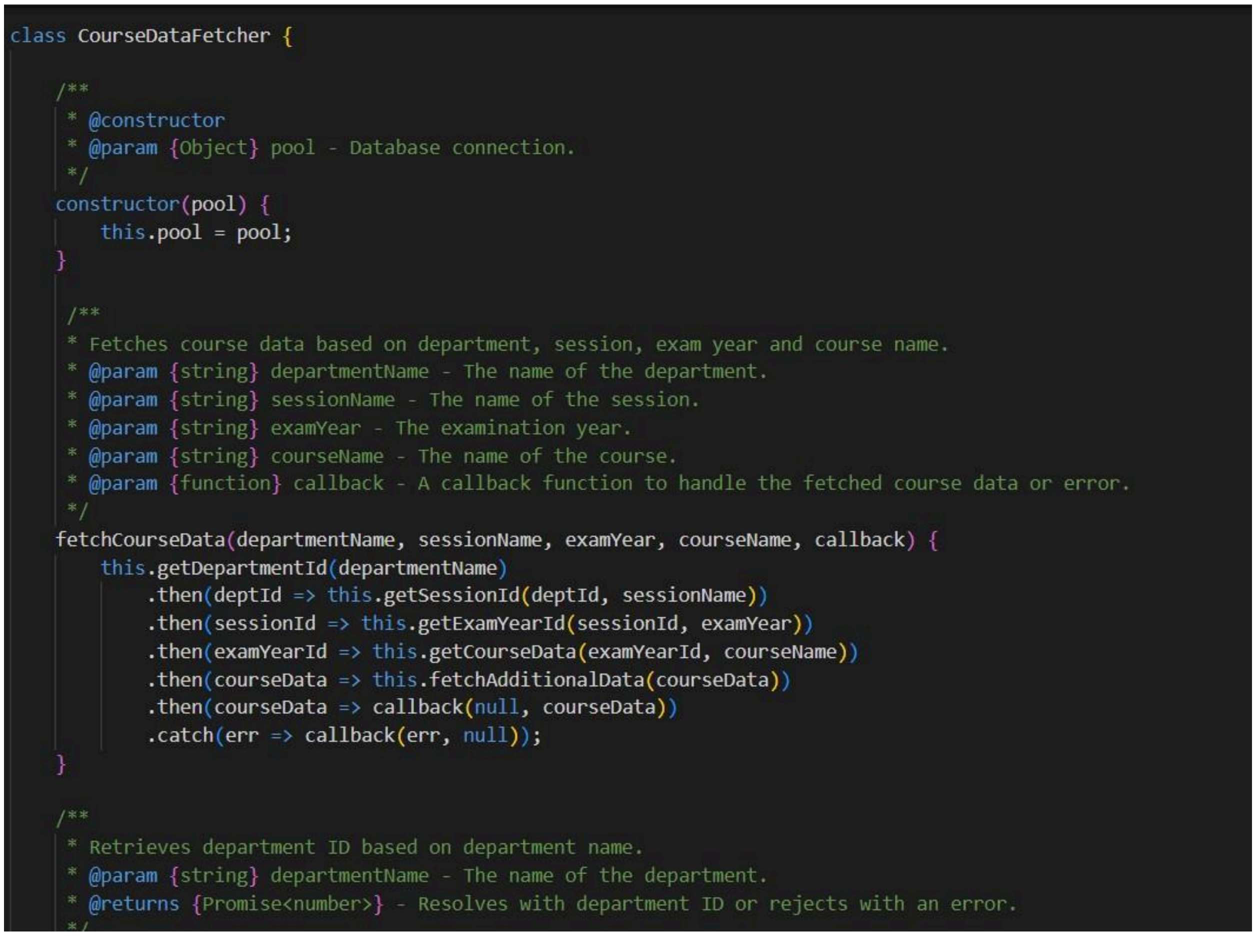


```
/**
 * @module CourseDataFetcher
 * @description A class to fetch course data from the database.
 */

const pool = require('../config/db');

class CourseDataFetcher {

    /**
     * @constructor
     * @param {Object} pool - Database connection.
     */
    constructor(pool) {
        this.pool = pool;
    }

     /**
     * Fetches course data based on department, session, exam year and course name.
     * @param {string} departmentName - The name of the department.
     * @param {string} sessionName - The name of the session.
     * @param {string} examYear - The examination year.
     * @param {string} courseName - The name of the course.
     * @param {function} callback - A callback function to handle the fetched course data or error.
     */
    fetchCourseData(departmentName, sessionName, examYear, courseName, callback) {
        this.getDepartmentId(departmentName)
            .then(deptId => this.getSessionId(deptId, sessionName))
            .then(sessionId => this.getExamYearId(sessionId, examYear))
            .then(examYearId => this.getCourseData(examYearId, courseName))
            .then(courseData => this.fetchAdditionalData(courseData))
            .then(courseData => callback(null, courseData))
            .catch(err => callback(err, null));
    }
```

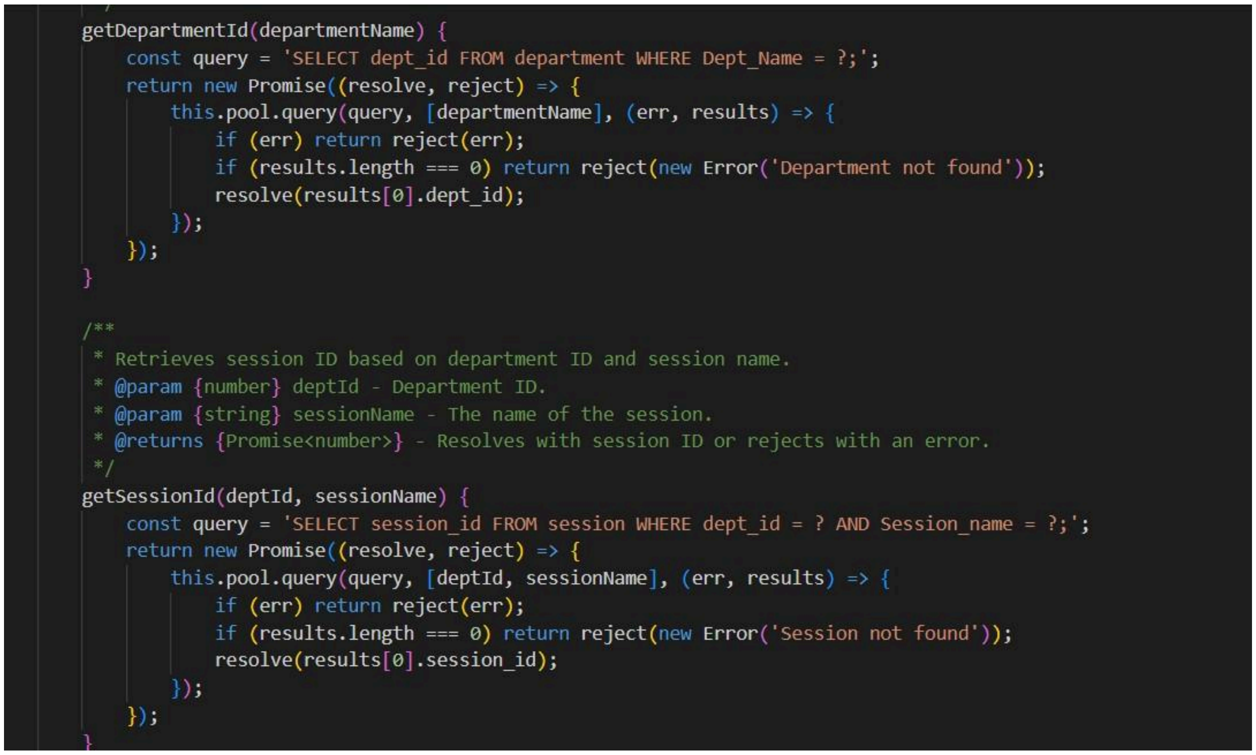
✅ Test Case Passes

Resolution

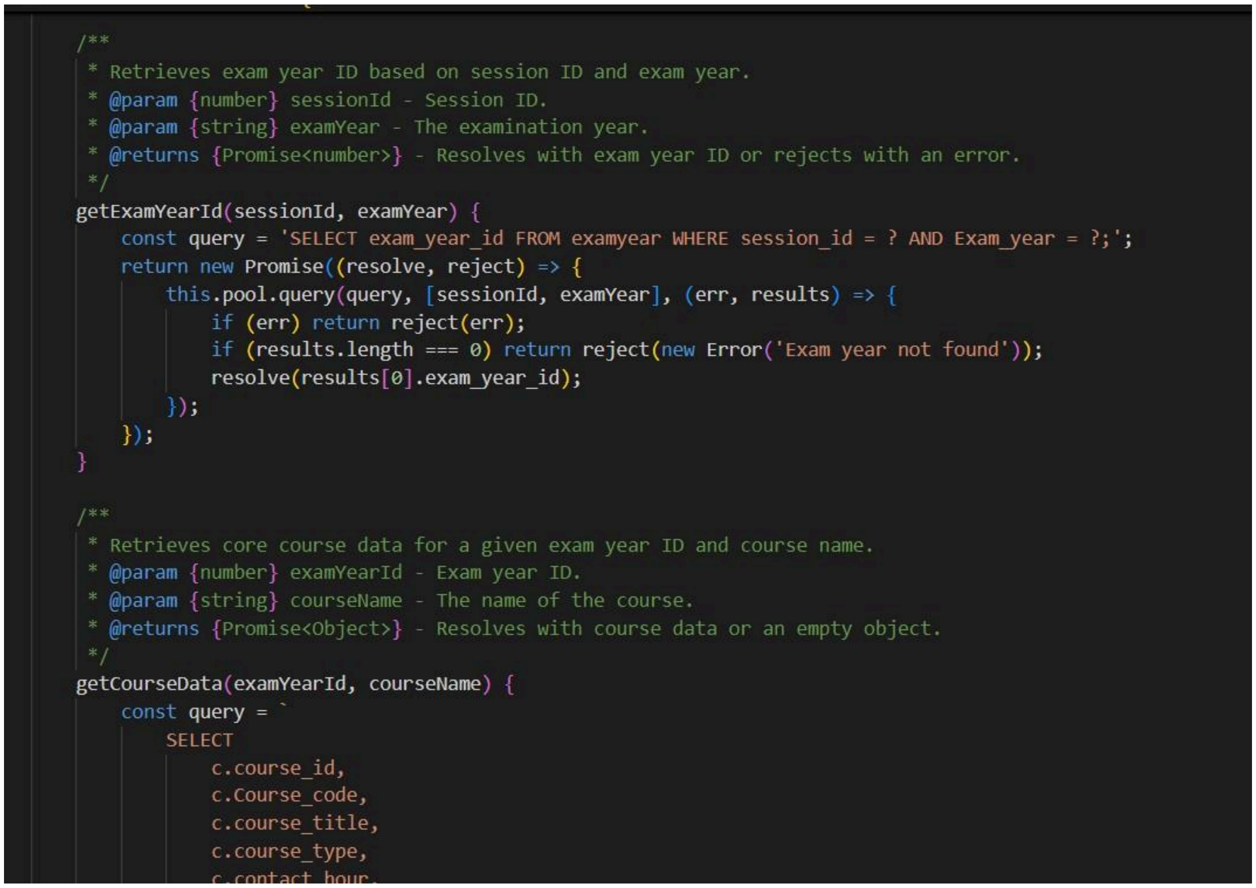
- Completed Function Logic: The logic for fetching the syllabus was implemented successfully, *passing all tests*.



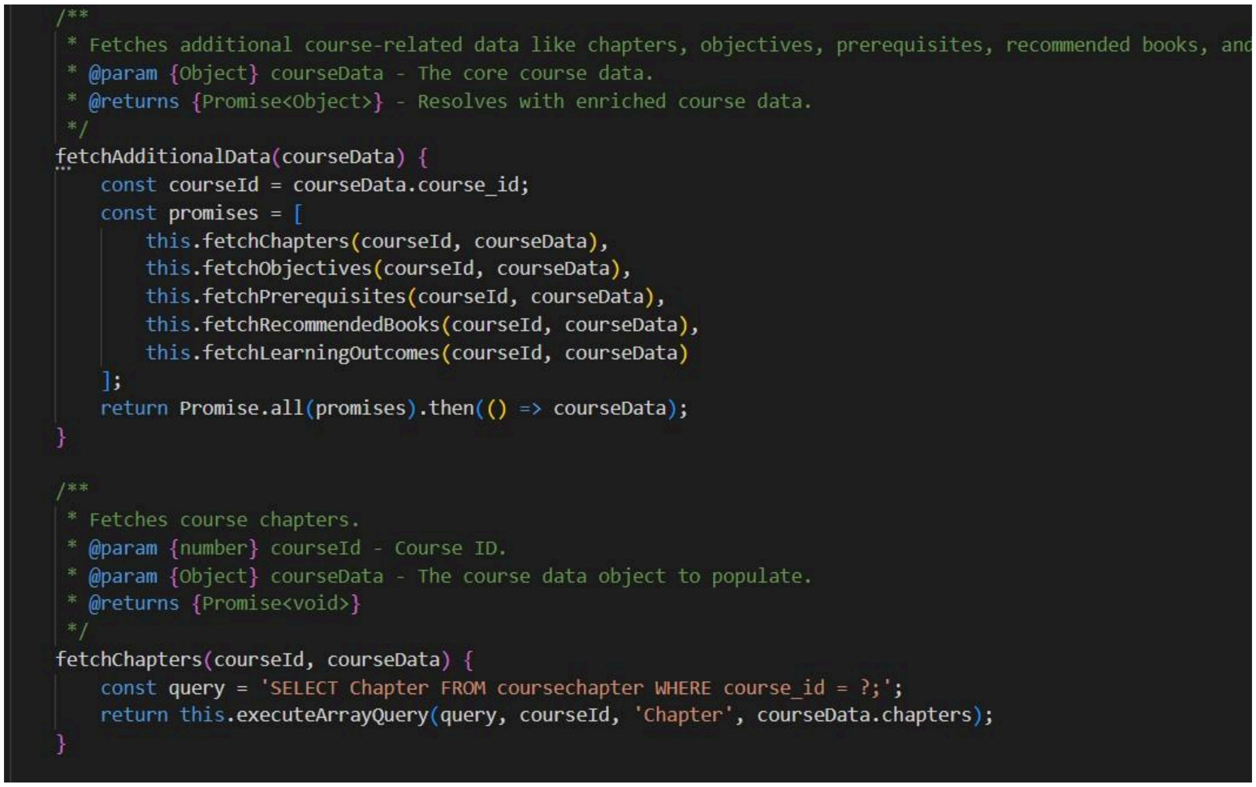
```javascript
class CourseDataFetcher {

    /**
     * @constructor
     * @param {Object} pool - Database connection.
     */
    constructor(pool) {
        this.pool = pool;
    }

     /**
     * Fetches course data based on department, session, exam year and course name.
     * @param {string} departmentName - The name of the department.
     * @param {string} sessionName - The name of the session.
     * @param {string} examYear - The examination year.
     * @param {string} courseName - The name of the course.
     * @param {function} callback - A callback function to handle the fetched course data or error.
     */
    fetchCourseData(departmentName, sessionName, examYear, courseName, callback) {
        this.getDepartmentId(departmentName)
            .then(deptId => this.getSessionId(deptId, sessionName))
            .then(sessionId => this.getExamYearId(sessionId, examYear))
            .then(examYearId => this.getCourseData(examYearId, courseName))
            .then(courseData => this.fetchAdditionalData(courseData))
            .then(courseData => callback(null, courseData))
            .catch(err => callback(err, null));
    }

    /**
     * Retrieves department ID based on department name.
     * @param {string} departmentName - The name of the department.
     * @returns {Promise<number>} - Resolves with department ID or rejects with an error.
```



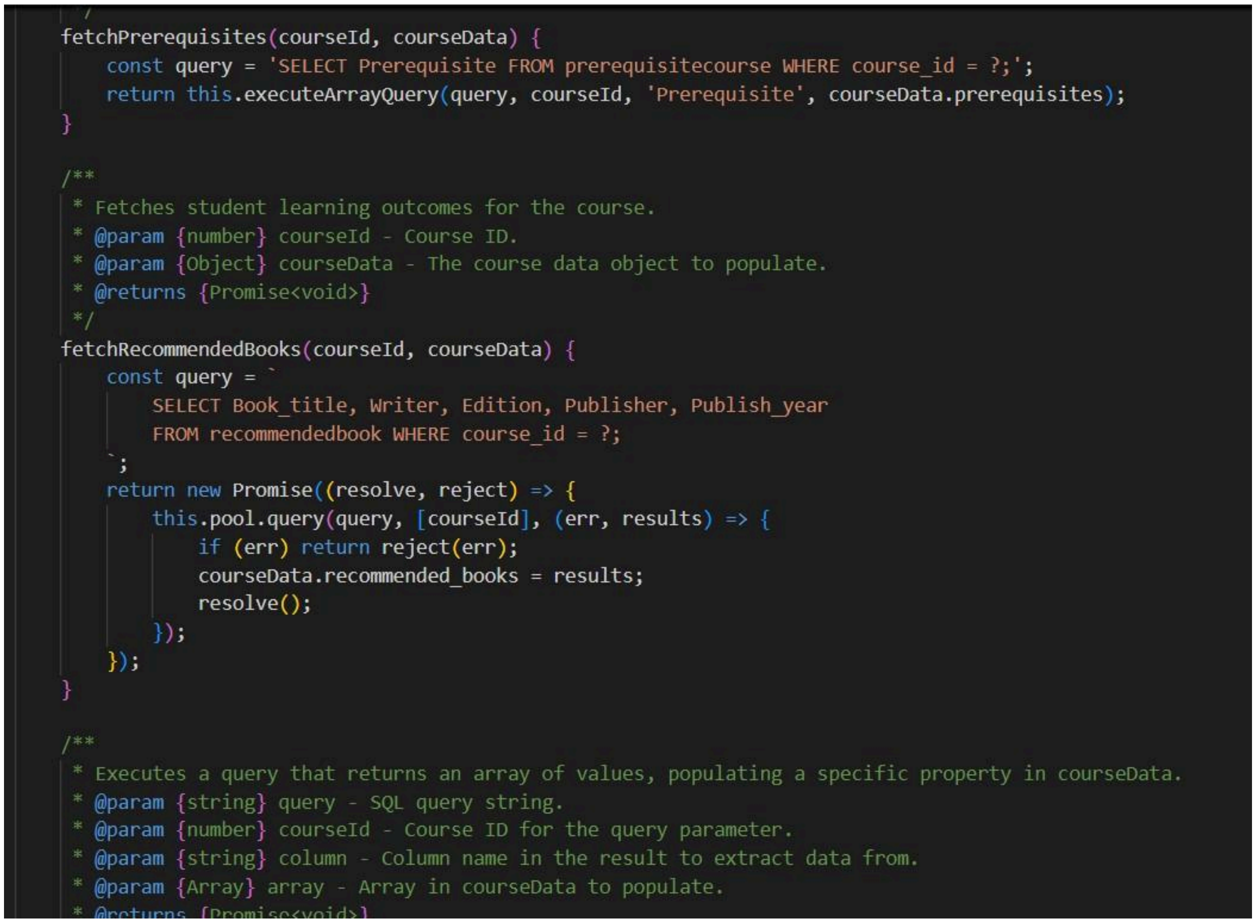
```javascript
getDepartmentId(departmentName) {
    const query = 'SELECT dept_id FROM department WHERE Dept_Name = ?;';
    return new Promise((resolve, reject) => {
        this.pool.query(query, [departmentName], (err, results) => {
            if (err) return reject(err);
            if (results.length === 0) return reject(new Error('Department not found'));
            resolve(results[0].dept_id);
        });
    });
}

/**
 * Retrieves session ID based on department ID and session name.
 * @param {number} deptId - Department ID.
 * @param {string} sessionName - The name of the session.
 * @returns {Promise<number>} - Resolves with session ID or rejects with an error.
 */
getSessionId(deptId, sessionName) {
    const query = 'SELECT session_id FROM session WHERE dept_id = ? AND Session_name = ?;';
    return new Promise((resolve, reject) => {
        this.pool.query(query, [deptId, sessionName], (err, results) => {
            if (err) return reject(err);
            if (results.length === 0) return reject(new Error('Session not found'));
            resolve(results[0].session_id);
        });
    });
}
```



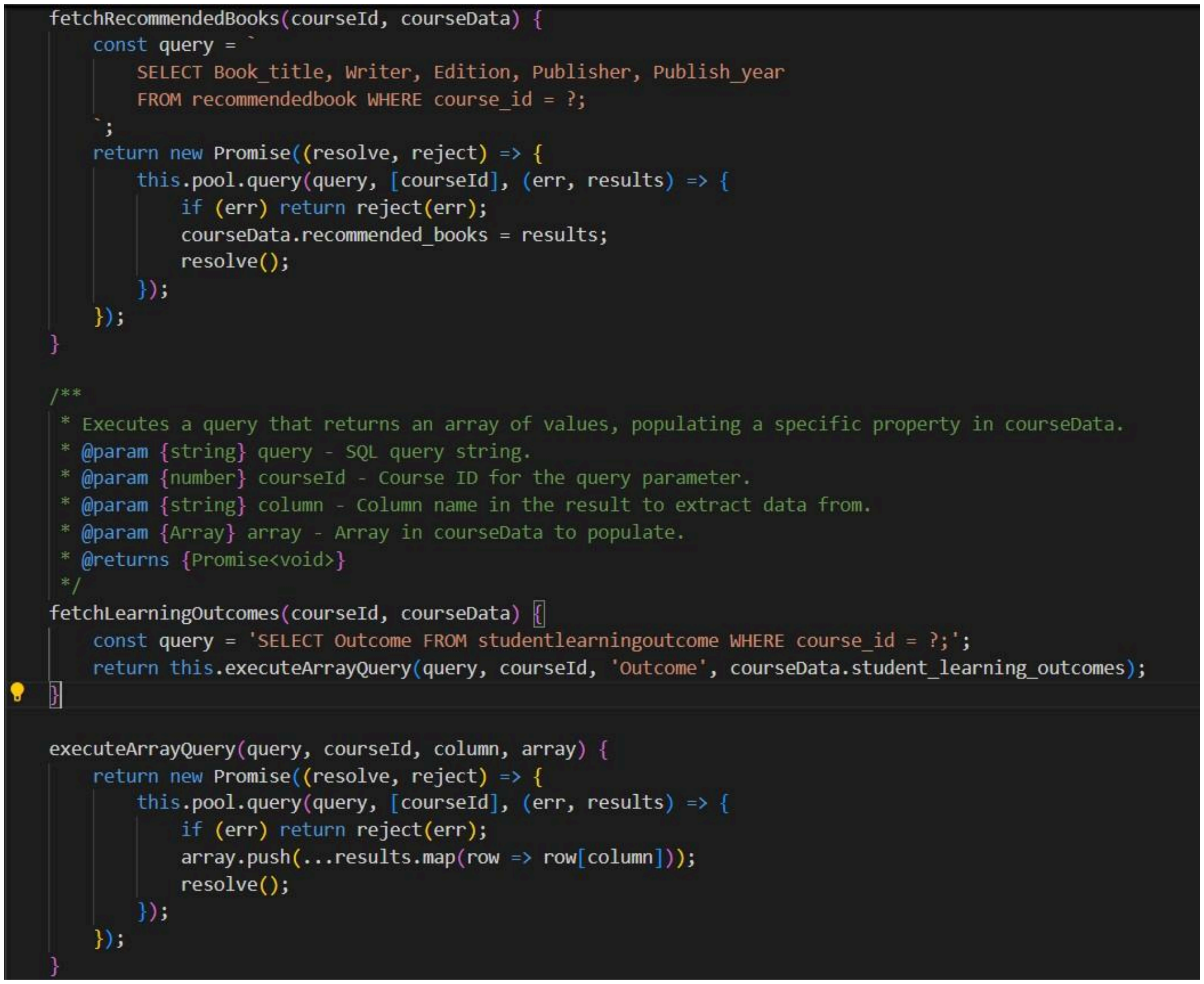
```javascript
/**
 * Retrieves exam year ID based on session ID and exam year.
 * @param {number} sessionId - Session ID.
 * @param {string} examYear - The examination year.
 * @returns {Promise<number>} - Resolves with exam year ID or rejects with an error.
 */
getExamYearId(sessionId, examYear) {
    const query = 'SELECT exam_year_id FROM examyear WHERE session_id = ? AND Exam_year = ?;';
    return new Promise((resolve, reject) => {
        this.pool.query(query, [sessionId, examYear], (err, results) => {
            if (err) return reject(err);
            if (results.length === 0) return reject(new Error('Exam year not found'));
            resolve(results[0].exam_year_id);
        });
    });
}

/**
 * Retrieves core course data for a given exam year ID and course name.
 * @param {number} examYearId - Exam year ID.
 * @param {string} courseName - The name of the course.
 * @returns {Promise<Object>} - Resolves with course data or an empty object.
 */
getCourseData(examYearId, courseName) {
    const query = `
        SELECT
            c.course_id,
            c.Course_code,
            c.course_title,
            c.course_type,
            c.contact_hour,
```



```
/**
 * Fetches additional course-related data like chapters, objectives, prerequisites, recommended books, and
 * @param {Object} courseData - The core course data.
 * @returns {Promise<Object>} - Resolves with enriched course data.
 */
fetchAdditionalData(courseData) {
    const courseId = courseData.course_id;
    const promises = [
        this.fetchChapters(courseId, courseData),
        this.fetchObjectives(courseId, courseData),
        this.fetchPrerequisites(courseId, courseData),
        this.fetchRecommendedBooks(courseId, courseData),
        this.fetchLearningOutcomes(courseId, courseData)
    ];
    return Promise.all(promises).then(() => courseData);
}

/**
 * Fetches course chapters.
 * @param {number} courseId - Course ID.
 * @param {Object} courseData - The course data object to populate.
 * @returns {Promise<void>}
 */
fetchChapters(courseId, courseData) {
    const query = 'SELECT Chapter FROM coursechapter WHERE course_id = ?;';
    return this.executeArrayQuery(query, courseId, 'Chapter', courseData.chapters);
}
```



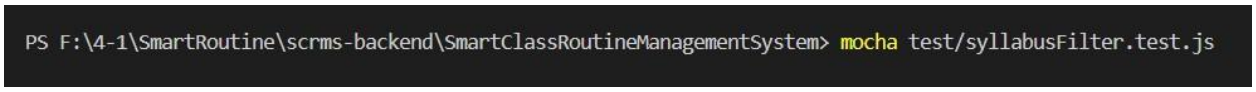
```
    fetchPrerequisites(courseId, courseData) {
        const query = 'SELECT Prerequisite FROM prerequisitecourse WHERE course_id = ?;';
        return this.executeArrayQuery(query, courseId, 'Prerequisite', courseData.prerequisites);
    }

    /**
     * Fetches student learning outcomes for the course.
     * @param {number} courseId - Course ID.
     * @param {Object} courseData - The course data object to populate.
     * @returns {Promise<void>}
     */
    fetchRecommendedBooks(courseId, courseData) {
        const query = `
            SELECT Book_title, Writer, Edition, Publisher, Publish_year
            FROM recommendedbook WHERE course_id = ?;
        `;
        return new Promise((resolve, reject) => {
            this.pool.query(query, [courseId], (err, results) => {
                if (err) return reject(err);
                courseData.recommended_books = results;
                resolve();
            });
        });
    }

    /**
     * Executes a query that returns an array of values, populating a specific property in courseData.
     * @param {string} query - SQL query string.
     * @param {number} courseId - Course ID for the query parameter.
     * @param {string} column - Column name in the result to extract data from.
     * @param {Array} array - Array in courseData to populate.
```



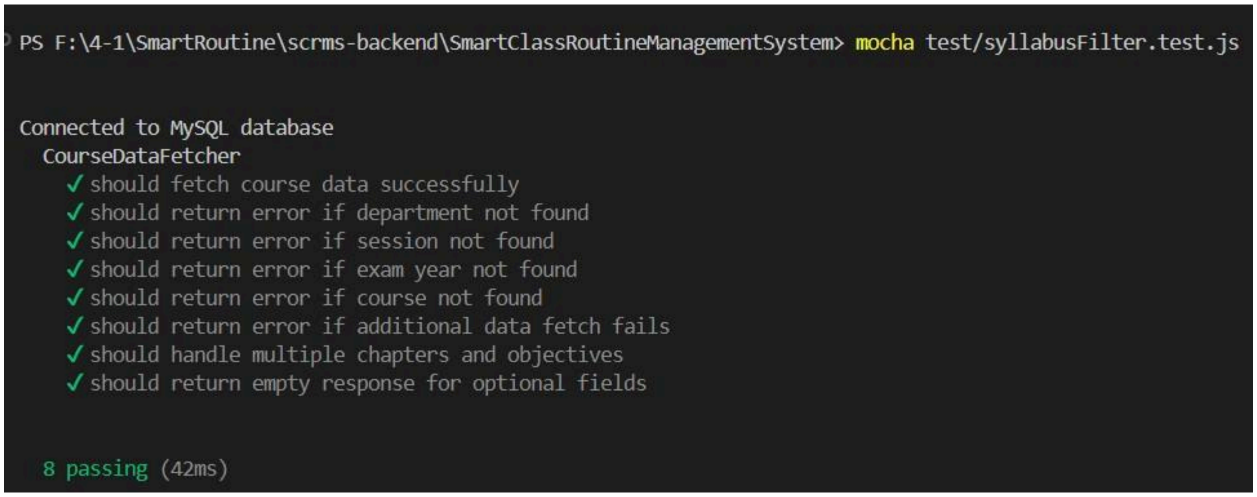
```
fetchRecommendedBooks(courseId, courseData) {
    const query = `
        SELECT Book_title, Writer, Edition, Publisher, Publish_year
        FROM recommendedbook WHERE course_id = ?;
    `;
    return new Promise((resolve, reject) => {
        this.pool.query(query, [courseId], (err, results) => {
            if (err) return reject(err);
            courseData.recommended_books = results;
            resolve();
        });
    });
}

/**
 * Executes a query that returns an array of values, populating a specific property in courseData.
 * @param {string} query - SQL query string.
 * @param {number} courseId - Course ID for the query parameter.
 * @param {string} column - Column name in the result to extract data from.
 * @param {Array} array - Array in courseData to populate.
 * @returns {Promise<void>}
 */
fetchLearningOutcomes(courseId, courseData) {
    const query = 'SELECT Outcome FROM studentlearningoutcome WHERE course_id = ?;';
    return this.executeArrayQuery(query, courseId, 'Outcome', courseData.student_learning_outcomes);
}

executeArrayQuery(query, courseId, column, array) {
    return new Promise((resolve, reject) => {
        this.pool.query(query, [courseId], (err, results) => {
            if (err) return reject(err);
            array.push(...results.map(row => row[column]));
            resolve();
        });
    });
}
```

- Verification: We checked the function's performance for the test case with the following command:

```
mocha test/syllabusFilter.test.js
```



```
PS F:\4-1\SmartRoutine\scrms-backend\SmartClassRoutineManagementSystem> mocha test/syllabusFilter.test.js
```

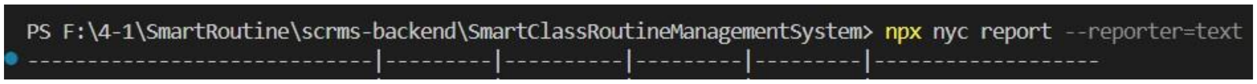


```
PS F:\4-1\SmartRoutine\scrms-backend\SmartClassRoutineManagementSystem> mocha test/syllabusFilter.test.js


Connected to MySQL database
  CourseDataFetcher
    ✓ should fetch course data successfully
    ✓ should return error if department not found
    ✓ should return error if session not found
    ✓ should return error if exam year not found
    ✓ should return error if course not found
    ✓ should return error if additional data fetch fails
    ✓ should handle multiple chapters and objectives
    ✓ should return empty response for optional fields


  8 passing (42ms)
```

📊 Results

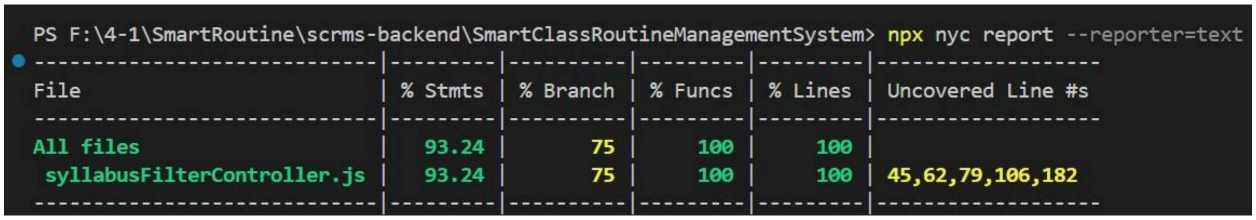
Test Coverage Report

To generate a test coverage report, run the following commands:
npm install nyc --save-dev
npx nyc report --reporter=text



```
PS F:\4-1\SmartRoutine\scrms-backend\SmartClassRoutineManagementSystem> npx nyc report --reporter=text
-----------------------------|---------|----------|---------|---------|-------------------
```

📊 Report Analysis



```
PS F:\4-1\SmartRoutine\scrms-backend\SmartClassRoutineManagementSystem> npx nyc report --reporter=text
-----------------------------|---------|----------|---------|---------|-------------------
File                         | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------------|---------|----------|---------|---------|-------------------
All files                    |   93.24 |       75 |     100 |     100 |
 syllabusFilterController.js |   93.24 |       75 |     100 |     100 | 45,62,79,106,182
-----------------------------|---------|----------|---------|---------|-------------------
```

✅ Statement Coverage: *93.24%* – Most lines are tested, with a few uncovered lines.
⚙️ Branch Coverage: *75%*
✔️ Function Coverage: *100%* – All functions in the file are covered by tests.
📜 Line Coverage: *100%* – All lines of code are executed during tests, except for a few uncovered lines mentioned below.

🔍 **Uncovered Lines:**

- Lines: 45, 62, 79, 106, 182

📈 Total Coverage: 93.24%