Concordia University

**Department of Computer Science**

**and Software Engineering**

**Software Process**

**SOEN 341 --- Fall 2015 Section H**

**Project Design Document**

|  |  |
| --- | --- |
| **Team information** | |
| **Team :** | |
| **Name** | **SID** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**~~Grading Sheet~~**

|  |  |  |
| --- | --- | --- |
| **Section** | **Evaluation criteria (see instructions in the template for details)** | **Grading** |
| all | 10 marks are allocated for excellence, professionalism and quality of work above and beyond the correct meeting of specifications.. | /10 |
| 1 | Introduction of the document | /1 |
| 2.1 .  2.2 | Validity and clarity of the architectural diagrams, as well as of the textual rationale.  Validity, completeness, and clarity of description of each component interface. | /4  /4 |
| 3.1  3.2  3.3 | Validity and clarity of the (UML) class diagrams or equivalent for each subsystem, as well as of the textual rationale.  Validity and clarity and completeness of the class descriptions for each subsystem.  Clarity and completeness of each module interface. | /10  /4  /3 |
| 4 | Validity and clarity of the dynamic design diagrams and contracts for each scenario.  Compatibility of the scenarios with the components presented in section 2 and 3, as well as of the textual rationale. | /9 |
| 5 | Revised cost estimation of each individual artifact, validity of explanation of cost estimation, total cost estimate | /2 |
| 6 | Rapid Prototyping and Risk Report | /3 |
| Total |  | /50 |

**DO NOT REMOVE THIS PAGE WHEN SUBMITTING YOUR DOCUMENT**

**Table of Contents**

Table of Contents

[1. Introduction to part 2 5](#_Toc466445394)

[2. Architectural Design 5](#_Toc466445395)

[2.1 Architecture Diagram 5](#_Toc466445396)

[2.1.1 Logical View 5](#_Toc466445397)

[2.1.2 Physical View 7](#_Toc466445398)

[2.1.3 Process View 8](#_Toc466445399)

[2.2 Subsystem Interfaces Specifications 9](#_Toc466445400)

[3. Detailed Design 11](#_Toc466445401)

[3.1 Detailed Design Diagram 11](#_Toc466445402)

[3.2 Unit Descriptions 13](#_Toc466445403)

[3.3 Module Interface Specifications 16](#_Toc466445404)

[3.3.1 Admission Class 17](#_Toc466445405)

[3.3.2 Information Class 18](#_Toc466445406)

[3.3.3 Preference Class 21](#_Toc466445407)

[3.3.4 Generator Class 22](#_Toc466445408)

[3.3.5 Course Class 24](#_Toc466445409)

[3.3.6 Database Class 24](#_Toc466445410)

[3.3.7 Course section Class 26](#_Toc466445411)

[3.3.8 Program Class 27](#_Toc466445412)

[4. Dynamic Design Scenarios 29](#_Toc466445413)

[4.1 Case Scenario: UC11–Generate Course Schedule 29](#_Toc466445414)

[4.1.1 System Sequence Diagram 29](#_Toc466445415)

[4.1.2 System Sequence Diagram 30](#_Toc466445416)

[4.2 Case Scenario: UC21-Add Section 31](#_Toc466445417)

[4.2.1 System Sequence Diagram 31](#_Toc466445418)

[4.2.2 System Sequence Diagram 32](#_Toc466445419)

[5. Estimation 33](#_Toc466445420)

[6. Rapid Prototyping and Risk 35](#_Toc466445421)

[6.1 User Interface during Login and Registration phases 35](#_Toc466445422)

[6.2 Interface after the login was successful 36](#_Toc466445423)

[6.3 Algorithm for the sequence generation 36](#_Toc466445424)

[6.4 Backend 36](#_Toc466445425)

[6.5 Analysis on risks impact and estimations for the Deliverable 2 Prototype 39](#_Toc466445426)

[6.5.1 Risk impact Analysis 39](#_Toc466445427)

[6.5.2 Estimate impact Analysis 40](#_Toc466445428)

[7. Appendix A 41](#_Toc466445429)

[7.1 Modified Use Case Diagrams from Deliverable 1 41](#_Toc466445430)

[7.2 Sequence generation algorithm in Pseudocode 42](#_Toc466445431)

# Introduction to part 2

In this document, we will describe the architecture and the design of our project, Fate Generator. The goal at this stage of the project is to describe the way the project is being made, how the system will be divided up to make smaller subsystems, as well as the design of the project as a whole. To do this, we will describe the architectural design in detail, using class diagrams, deployment diagrams and activity diagrams, as well as the dynamic design scenarios, using sequence diagrams. The estimation done in deliverable 1 will also be revised and the prototyping section will be updated in accordance to the advancement of the project.

# Architectural Design

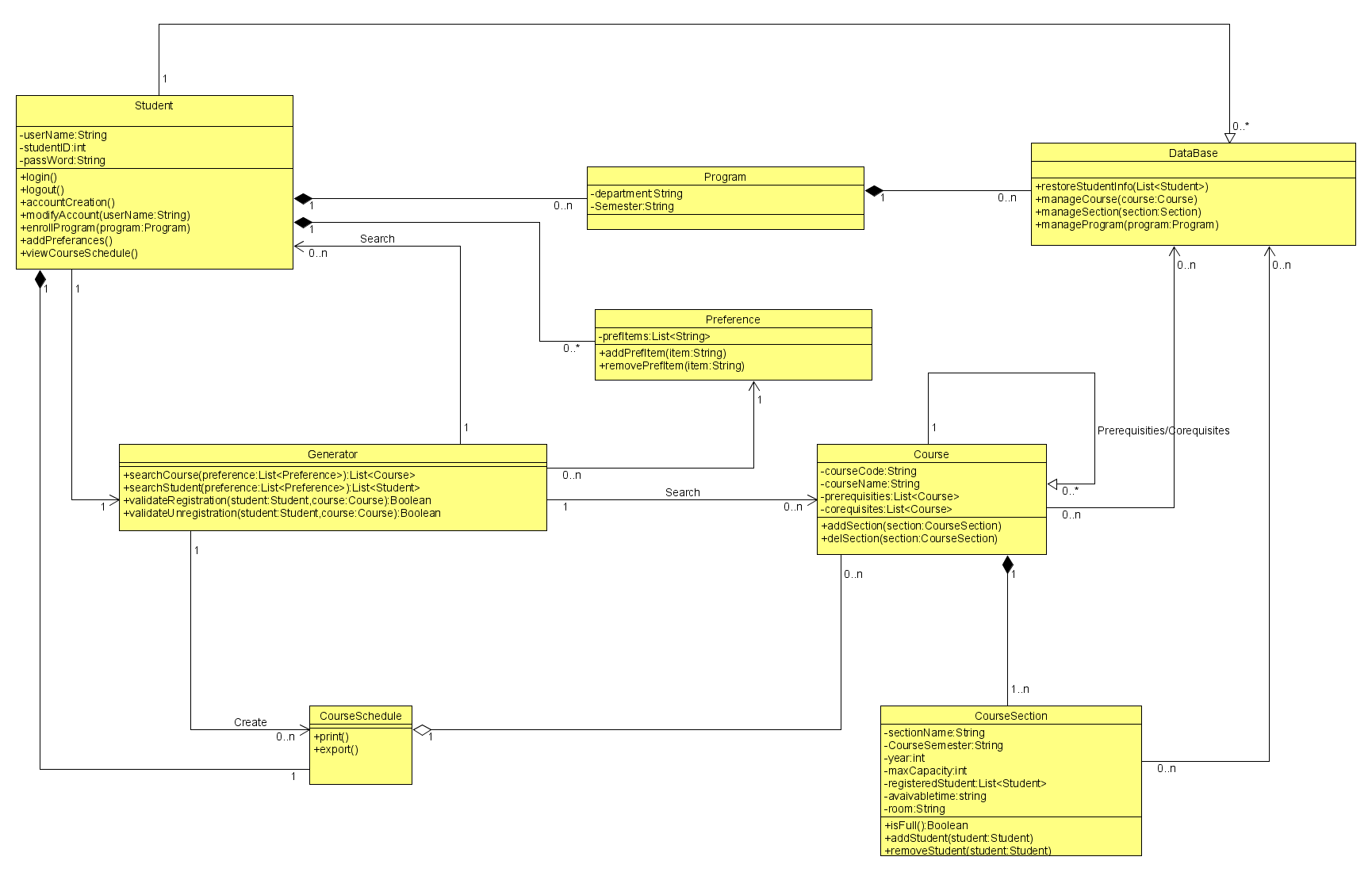
## Architecture Diagram

The Architectural design of the system shows the way the system is built and why we decided to do it in this particular way. Through the logical view, the physical view and the process view we will see how the system will be divided into several subsystems.

### Logical View

The logical view of the system shows the system in its entirety. All the classes with their attributes and methods, as well as the relationships between all the classes, are shown in the diagram below.

The class diagram is centered on the student class who is the main actor of the system. The student can put in his preferences and choose a program, before the system generates a schedule based on the information it collects. Information on the courses and students will be stored in a database and used when needed. The final goal is for the student to view his personalized course schedule.



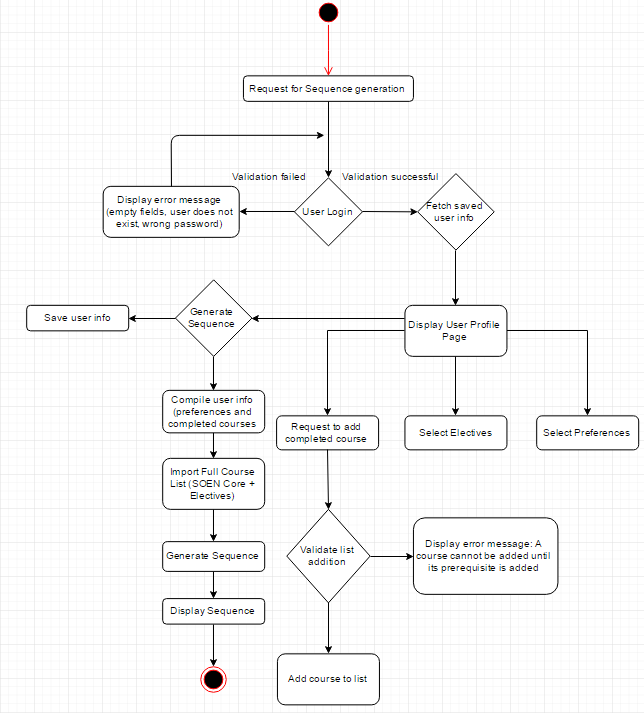
### Physical View

The project is designed with the MVC architecture. The user uses a device with a web access to get onto the application. This connects the user to the database and allows the user to perform the actions wanted.

Physical view

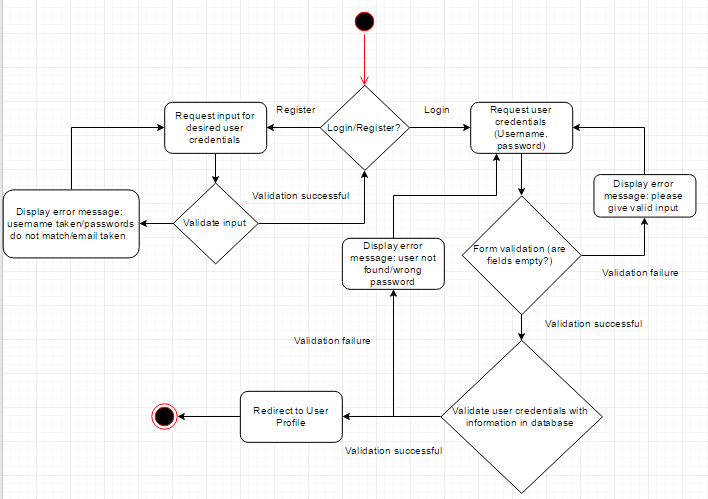
### Process View

#### Sequence Generation



This diagram shows the process of a sequence generation. The user, a student, logs into the system and requests for a sequence generation. The student will need to enter their preferences, electives and completed courses. The system will then compile all this information to generate a sequence which will then be displayed to the student.

#### User Login & Registration



This diagram shows the process of a user login and registration. If a user doesn’t have an account, they will be taken to a registration page to register for a new account. They will need to provide information to create this account. When the user has created an account, they will go to a login page where they will need to enter their credentials. If a user already has an account, they would go directly to the login page. If the credentials aren’t valid, an error message will be displayed and the the user will be redirected to the login page. If the credentials are valid, the user will be logged in and directed to their profile.

## Subsystem Interfaces Specifications

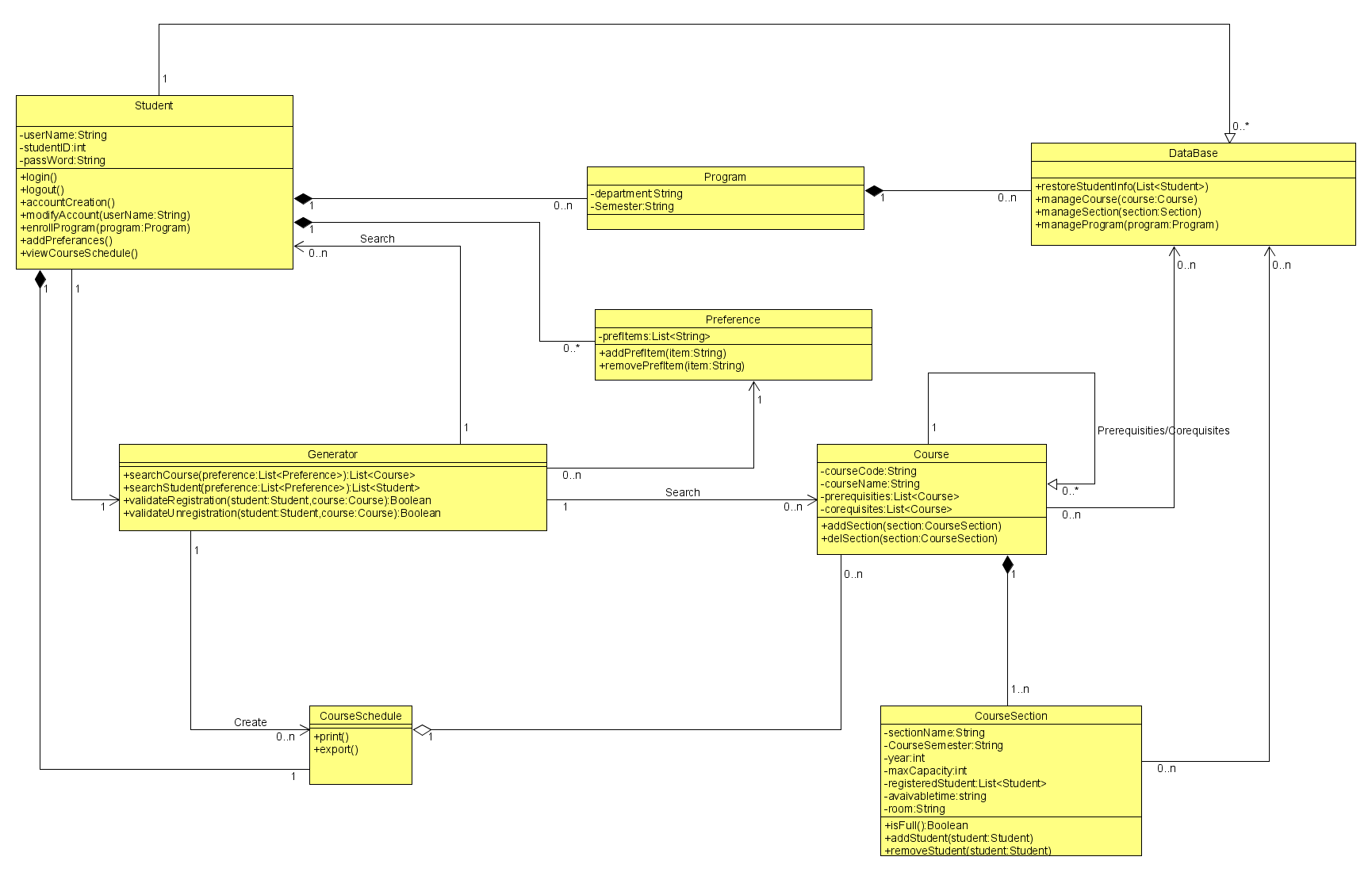
|  |  |  |
| --- | --- | --- |
| Class | Method | Description |
| Student | enrollProgram(program :Program) | This method communicates with the Program class, linking a Student object with a Program object. |
| Student | addPreferences() | This method links a Student object with a Preference object which contains all the preference details for that student |
| Student | viewCourseSchedule() | As the name of the method implies, this method accesses CourseSchedule object that is generated for this particular Student object |
| DataBase | restoreStudentInfo(List<Student>) | This method accesses the database directly and extracts the information pertinent to the list of Student objects that were passed to it |
| DataBase | manageCourse(course:Course) | Invokes the Course object specified in the method call body from the table, so that operations can be performed on it |
| DataBase | manageSection(section:Section) | Invokes the Section (Course Section) object specified in the method call body from the table, so that operations can be performed on it |
| DataBase | manageProgram(program:Program) | Invokes the Program object specified in the method call body from the table, so that operations can be performed on it |
| Generator | searchCourse(preference:List<Preference>) | Invokes a search on the list of Course objects extracted via the DataBase class, based on Preference list passed in the method body |
| Generator | searchStudent(preference:List<Preference>) | Invokes a search on the list of Student objects extracted via the DataBase class, based on Preference list passed in the method body. |
| Generator | validateRegistration(student: Student, course:Course) | Verifies that the student object passed in the method body has a matching course linked to it as successfully registered |
| Generator | validateRegistration(student: Student, course:Course) | Verifies that the student object passed in the method body does NOT have a matching course linked to it as successfully registered |

# Detailed Design

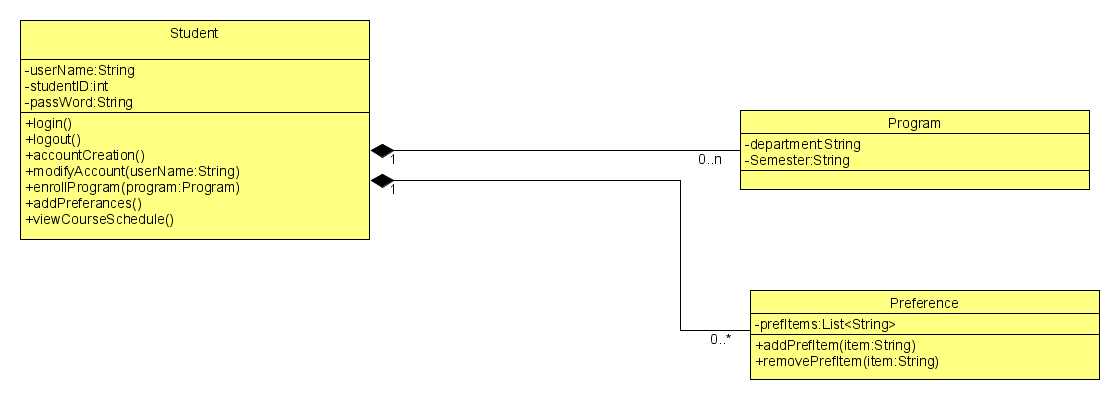
In this part, the system will be divided into subsystems and will be described more in detail.

## Detailed Design Diagram

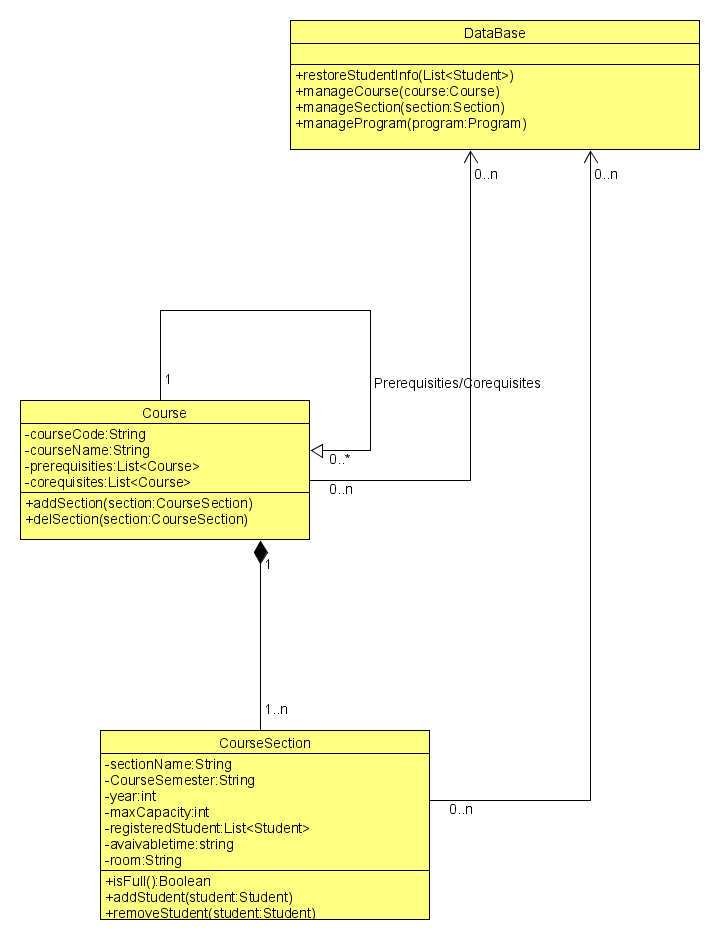
This is the logical view of the System. A student must be registered in the system to use this application. A registered student may choose preferences if wanted. Once the student has determined the semester and given information about completed courses, the generator will generate a schedule for the student by going through the Course and CourseSection classes.



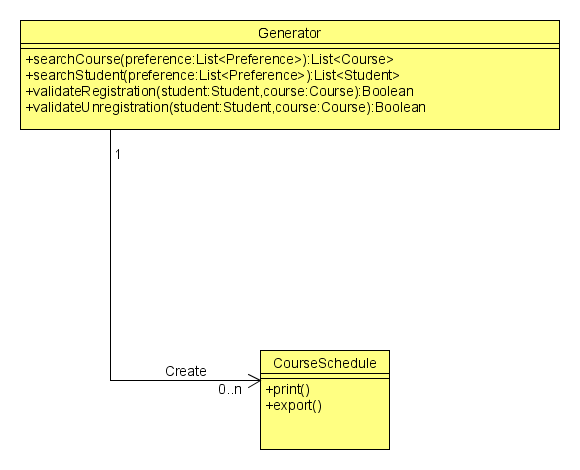
Here is the detailed information on each part of the diagram, broken down:



The Student class has a one-to-many relationship to Preference and Program. This means that a student can either be enrolled in none or more than one program. Such situation is an edge case, but it is possible for it to occur. A Student can also have no preferences to his schedule, or have many different ones.



The DataBase is a Controller class that is used to pull information from our Heroku Server. This diagram depicts the controller’s relation to Course and CourseSection classes, which are what our DB should contain.



The final piece of the diagram consists of a Generator class that has a Create relationship with the CourseSchedule class, which is the final product of our application. The Generator class takes care of validating the registration of the student, as well as the organization of time conflicts and prerequisites conflicts.

## Unit Descriptions

The following is detailed information for each class designed above.

|  |  |
| --- | --- |
| **Class** | **Generator** |
| Description | Core of the application, the Generator class generates a course sequence/schedules for the student to follow. |
| Attributes |  |
| Methods | * searchCourse(preference: List<Preference>): List<Student>   + Search for Courses id * searchStudent(preference: List<Preference>): List<Student>   + Search for Student id * validateRegistration(student,course): Boolean   + Permit Student enrollment * validateUnregistration(student,course): Boolean   + Verify Student/Course Statues |

|  |  |
| --- | --- |
| **Class** | **Database** |
| Description | This controller class is responsible for the information extraction from our Heroku Server. The information extracted is Courses, as well as the Course Sections |
| Attributes |  |
| Methods | * restoreStudentInfo(List<Student>)   + Search the List of Student * manageCourse(course: Course)   + Open Course Data * manageSection(section: Section)   + Open Section Data * manageProgram(program: Program)   + Open Program Data |

|  |  |
| --- | --- |
| **Class** | **Program** |
| Description | This data class is responsible for storing a student’s choice of program for the semester |
| Attributes | * Semester: String * Course: String |
| Methods |  |

|  |  |
| --- | --- |
| **Class** | **CourseSection** |
| Description | CourseSection is a data class that is pulled from the Database, containing all the pertinent information to a specific section of a course |
| Attributes | * sectionName: String * CourseSemester: String * Year: int * maxCapacity: int * registeredStudentList<Student> * availableTime: String * room: String |
| Methods | * isFull(): Boolean   + Verify class available space * addStudent (student: Student)   + Enroll student to class * removeStudent (student: Student)   + Remove student from class |

|  |  |
| --- | --- |
| **Class** | **CourseSchedule** |
| Description | CourseSchedule Object is the final product of schedule generation, containing the final schedule for the student user |
| Attributes | * courseCode: String * courseName: String * preRequisites:List<Course> * coRequisites:List<Course> |
| Methods | * print()   + Prints the course schedule generated * export()   + Exports the Course Schedule generated |

|  |  |
| --- | --- |
| **Class** | **Student** |
| Description | The student class is contains all information of the students and allow new student to register |
| Attributes | * username: String * studentID: Integer * password:String |
| Methods | * login()   + Log in system – username/password * logout()   + Sign out of the system * accountCreation()   + Register an account to the system * modifyAccount(userName: String)   + Change username ID * enrollProgram(program: Program)   + Specify Program * addPreferences()   + Specify Preferences wanted – morning class/ Afternoon class * viewCourseSchedule()   + Opens the Course Schedule |

|  |  |
| --- | --- |
| **Class** | **Preference** |
| Description | The Preference List allows the user to set a more specific schedule generation |
| Attributes | * prefItems: List<String> |
| Methods | * addPrefitem (item: String)   + Specify Preference wanted * removePrefitem(item: String)   + Remove Preference wanted |

|  |  |
| --- | --- |
| **Class** | **Course** |
| Description | Course Objects are obtained from database via the DataBase class Controller. Contains all information pertinent to the schedule generation |
| Attributes | * courseCode: String * courseName: String * preRequisites:List<Course> * coRequisites: List<Course> |
| Methods | * addSection(section: CourseSection)   + Add a course section in the schedule * delSection(section: CourseSection   + Remove a course section in the schedule |

## Module Interface Specifications

### Student Class

#### Detailed Description

The Student class is the module created for creating, validating and modifying students’ account.

#### Student Functions

The following are the student functions for the Student class:

* ValidateUsername (getUsername)
* ValidatePassword (getPassword)
* Login (getUsername, getPassword)
* Logout (getUsername)
* accountCreation (addUsername, addPassword, addFirstName, addLastName)
* modifyAccount (getUsername, getOriginalPassword, addNewPassword)

#### Student Function Documentation

|  |
| --- |
| **Student::accountCreation(addUsername, addPassword, addFirstName, addLastName)** |
| Description:   * To register an account in the System.   Parameters:   * string addUsername * string addPassword * string addFirstName * string addLastName   Returns:   * Boolean. True if successful, false otherwise. |

|  |
| --- |
| **Student::login (getUsername, getPassword)** |
| Description:   * To log into a registered account in the System.   Parameters:   * string getUsername * string getPassword   Returns:   * Boolean. True if successful, false otherwise. |

|  |
| --- |
| **Student::logout (getUsername)** |
| Description:   * To log out of the System.   Parameters:   * string getUsername   Returns:   * Void |

|  |
| --- |
| **Student::modifyAccount(getUsername, getOriginalPassword, addNewPassword)** |
| Description:   * To change the original password in the student’s account.   Parameters:   * string getUsername * string getOriginalPassword * string addNewPassword   Returns:   * Boolean. True if successful, false return original password. |

### Course Section Class

#### Detailed Description

The Course Section class is used to check which program the student has been enrolled in as well as the courses in which he is registered and has validated. It is also used to view the course information.

#### Student functions

The following are the student functions for the Course Section class:

* getRegisteredCourse( string)
* getRegisteredSection(string )
* getValidateCourse(string)
* getValidateSection(string)
* getValidatePrerequisities(string)
* EnrollProgram(string)
* getViewDetailsofCourses(string)

#### Student functions documentation

|  |
| --- |
| **Course Section::enrollProgram(string)** |
| Description:   * .To enroll in program in the System.   Parameters:   * string enrollProgram * Invalid values: NULL. In case the program is not exists in the system, an exception will be thrown.   Returns:   * Void |

|  |
| --- |
| **Course Section::getRegisteredCourse( string)** |
| Description:   * To view the courses that has already been registered.   Parameters:   * string getRegisteredCourse   + Invalid values: NULL. In case the student has not registered any courses in the System.   Returns:   * Void |

|  |
| --- |
| **Course Section:: getRegisteredSection(string )** |
| Description:   * To view the course section that have already been registered.   Parameters:   * string getRegisteredSection   Returns:   * Void |

|  |
| --- |
| **Course Section: getValidateCourse(string)** |
| Description:   * To look through all the courses that are validated in the program.   Parameters:   * string getValidateCourse   Returns:   * Boolean. True if exists, false otherwise. |

|  |
| --- |
| **Course Section::getValidateSection(string)** |
| Description:   * To view all the sections that are validated in the course.   Parameters:   * string getValidateSection   Returns:   * Boolean. True if exists, false otherwise. |

|  |
| --- |
| **Course Section::getValidatePrerequisites(string)** |
| Description:   * To view the prerequisites in the program.   Parameters:   * string getValidatePrerequisites   Returns:   * Boolean. True if exists, false otherwise. |

|  |
| --- |
| **Course Section::viewDetailsofCourses(string)** |
| Description:   * To look through the course detail in the program.   Parameters:   * string viewDetailsofCourses   Returns:   * Void |

### Preference Class

#### Detailed Description

The preference module is intended for students to choose the section and the year they want based on their own preferences.

#### Student function

The following are the student functions for the Preference class:

* addPrefItem(item:String)
* removePrefItem(item:String)

#### Student function documentation

|  |
| --- |
| **Preference::addPrefItem(item:String)** |
| Description:   * To specify the term and the section of the course.   Parameters:   * string addPrefItem(item)   + Invalid values: NULL.   Returns:   * Void |

|  |
| --- |
| **Preference::removePrefItem(item:String)** |
| Description:   * To delete preferences added by the student.   Parameters:   * string removePrefItem(item)   + Invalid values: NULL. If there are no preferences added, an exception will be thrown.   Returns:   * Boolean. True if changing successfully, false otherwise. |

### Generator Class

#### Detailed Description

The Generator module is used for collecting the student’s preferences and courses based on the Master Schedule and program.

#### Generator function

The following are the generator functions for the generator class:

* searchCourse(preference:List<Preference>):List<Course>
* searchStudent(preference:List<Preference>):List<Student>
* validateRegistration(student:Student,course:Course):Boolean
* validateUnregistration(student:Student,course:Course):Boolean

#### Generator function documentation

|  |
| --- |
| **Generator:: searchStudent(preference:List<Preference>):List<Student>** |
| Description:   * To collect preferences according to student’s specification.   Parameters:   * searchStudent(preference:List<Preference>):List<Student>   Returns:   * List<Student>- The timetable list of student who is willing to meet. |

|  |
| --- |
| **Generator: :searchCourse(preference:List<Preference>):List<Course>** |
| Description:   * To collect course from the Master Schedule. To match course with student’s preferences.   Parameters:   * searchCourse(preference:List<Preference>):List<Course>   Returns:   * List<Course>- The course list of student who is willing to meet. |

|  |
| --- |
| **Generator::validateRegistration(student:Student,course:Course):Boolean** |
| Description:   * Verify if the student can register the course.   Parameters:   * Student - The student to be validated. * Course - The course to be validated.   Returns:   * Boolean. True if the student be registered, false otherwise. |

|  |
| --- |
| **Generator: :validateUnregistration(student:Student,course:Course)** |
| Description:   * Verify if the student can drop the course.   Parameters:   * Student - The student to be validated. * Course - The course to be validated.   Returns:   * Boolean. True if the student can cancel the registration, false otherwise. |

### Course Class

#### Detailed Description

The Course module is created for adding and deleting sections of the course based on the result from Course Section module.

#### Course functions

The following are the course functions for the course class:

* addSection(section:CourseSection)
* delSection(section:CourseSection)

#### Course function documentation

|  |
| --- |
| **Course::addSection(section:CourseSection)** |
| Description:   * To add sections of the course.   Parameters:   * string addSection(section)   Return:   * Boolean. True if the adding of the section is successful, false otherwise. |

|  |
| --- |
| **Course::delSection(section:CourseSection)** |
| Description:   * To delete section of the course.   Parameters   * string delSection(section)   Return:   * Boolean. True if the removal of the section is successful, false otherwise. |

### Database Class

#### Detailed Description

The Database Module is intended for storing student information, managing course and section, and managing program.

#### Database function

The following are the database functions for the database class:

* restoreStudentInfo(List<Student>)
* manageCourse(course:Course)
* manageSection(section:Section)
* manageProgram(program:Program)

#### Database function documentation

|  |
| --- |
| **Database::restoreStudentInfo(List<Student>)** |
| Description:   * To store the student information in the system.   Parameters:   * restoreStudentInfo(List<Student>)   Return:   * Void |

|  |
| --- |
| **Database::manageCourse(course:Course)** |
| Description:   * To modify course schedule in the system.   Parameters:   * string: manageCourse(course)   Return:   * Boolean. True if modification of the section is successful, false otherwise. |
| **Database::manageSection(section:Section)** |
| Description:   * To change course section in the course schedule.   Parameters:   * string: manageSection(section)   Return   * Boolean. True if section change is successful, false otherwise. |

|  |
| --- |
| **Database::manageProgram(program:Program)** |
| Description：   * To modify student’s program in the system.   Parameters:   * string: manageProgram(program)   + Invalid values: NULL. In case the student has not enrolled the program.   Return:   * Boolean. True if the program management is successful, false otherwise. |

### Course section Class

#### Detailed Description

The Course Section Module is used for recording the capacity of the class and displaying the full information of the class. If full capacity is not reached, new students can be added.

#### Course section functions

The following are the course section functions for the course section class:

* isFull():Boolean
* addStudent(student:Student)
* removeStudent(student:Student)

#### Course section function documentation

|  |
| --- |
| **CouseSection::isFull():Boolean** |
| Description:   * To collect the number of students enrolled for the course, and comparing it with the maximum capacity of the class.   Parameters:   * Boolean::isFull()   Return:   * Boolean. True if the capacity is full, false otherwise. |

|  |
| --- |
| **CouseSection: addStudent(student:Student)** |
| Description:   * To enroll the student in the course.   Parameters:   * string: addStudent(student)   Return:   * Void |

|  |
| --- |
| CourseSection: :removeStudent(student:Student) |
| Description:   * To remove the student from the course.   Parameters:   * string: removeStudent(student:Student)   Return:   * Void |

### Program Class

#### Detailed Description

The Program Module is created to get registration information on the student’s program and storing it in the Database.

#### Program function

The following are the program functions for the program class:

* getDepartment:String
* getSemester:String

#### Program function documentation

|  |
| --- |
| **Program:: getDepartment:String** |
| Description:   * To collect information on the enrolled department of the student.   Parameters:   * string: getDepartment()   Return;   * Void |

|  |
| --- |
| **Program::getSemester:String** |
| Description:   * To record the semester that the student has enrolled in.   Parameters:   * string: getSemester()   Return:   * Void |

# Dynamic Design Scenarios

## Case Scenario: UC11–Generate Course Schedule

### System Sequence Diagram

**4.1SystemDiagram**

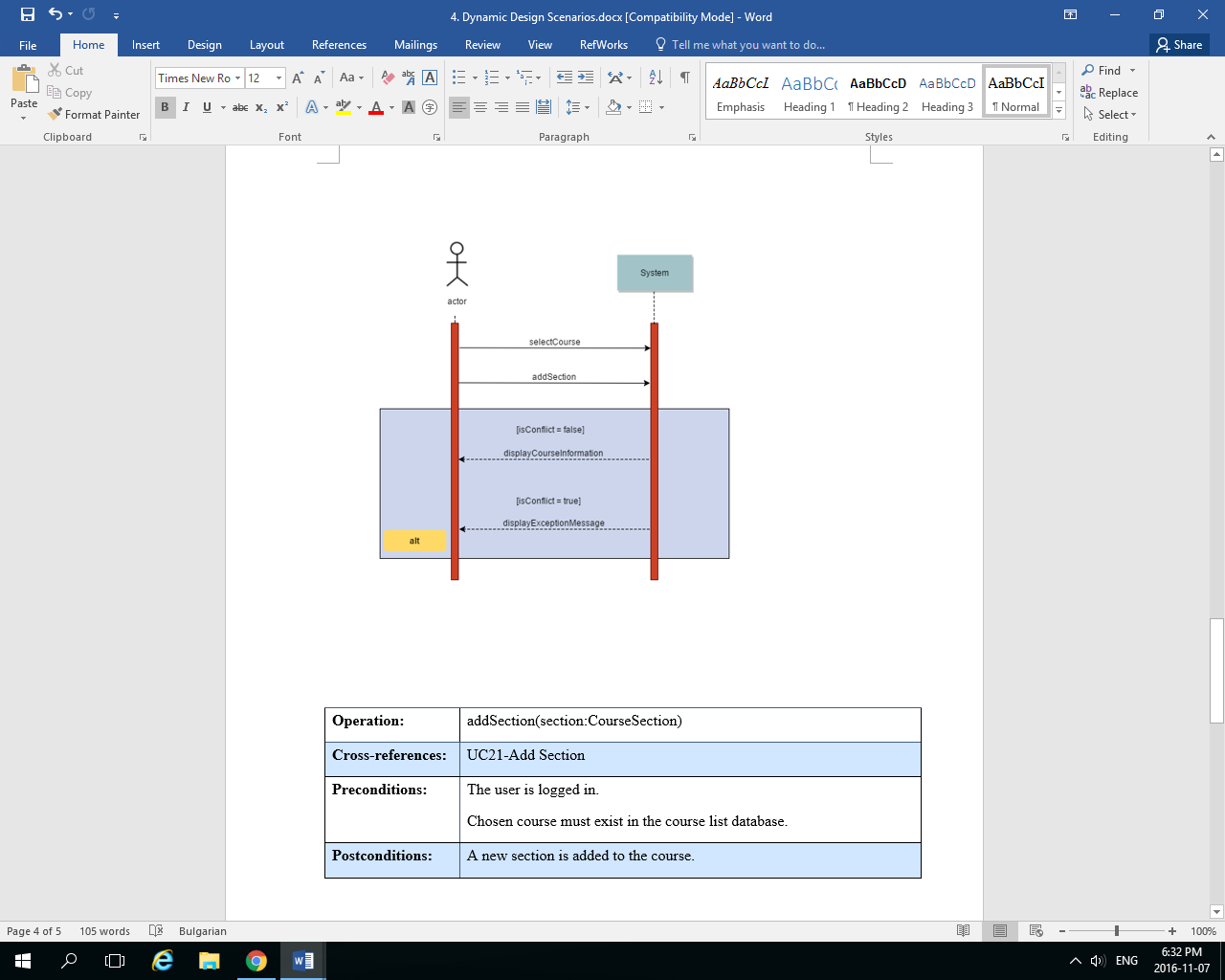
|  |  |
| --- | --- |
| **Operation** | **requestScheduleGeneration()** |
| **Cross-reference** | UC11-View Course Schedule |
| **Preconditions** | The user is logged in.  The system has access to the student’s academic record.  The system has access to the student’s preferences.  The user must be enrolled in a program. |
| **Postconditions** | The student can consult his course schedule. |

### System Sequence Diagram

4.2

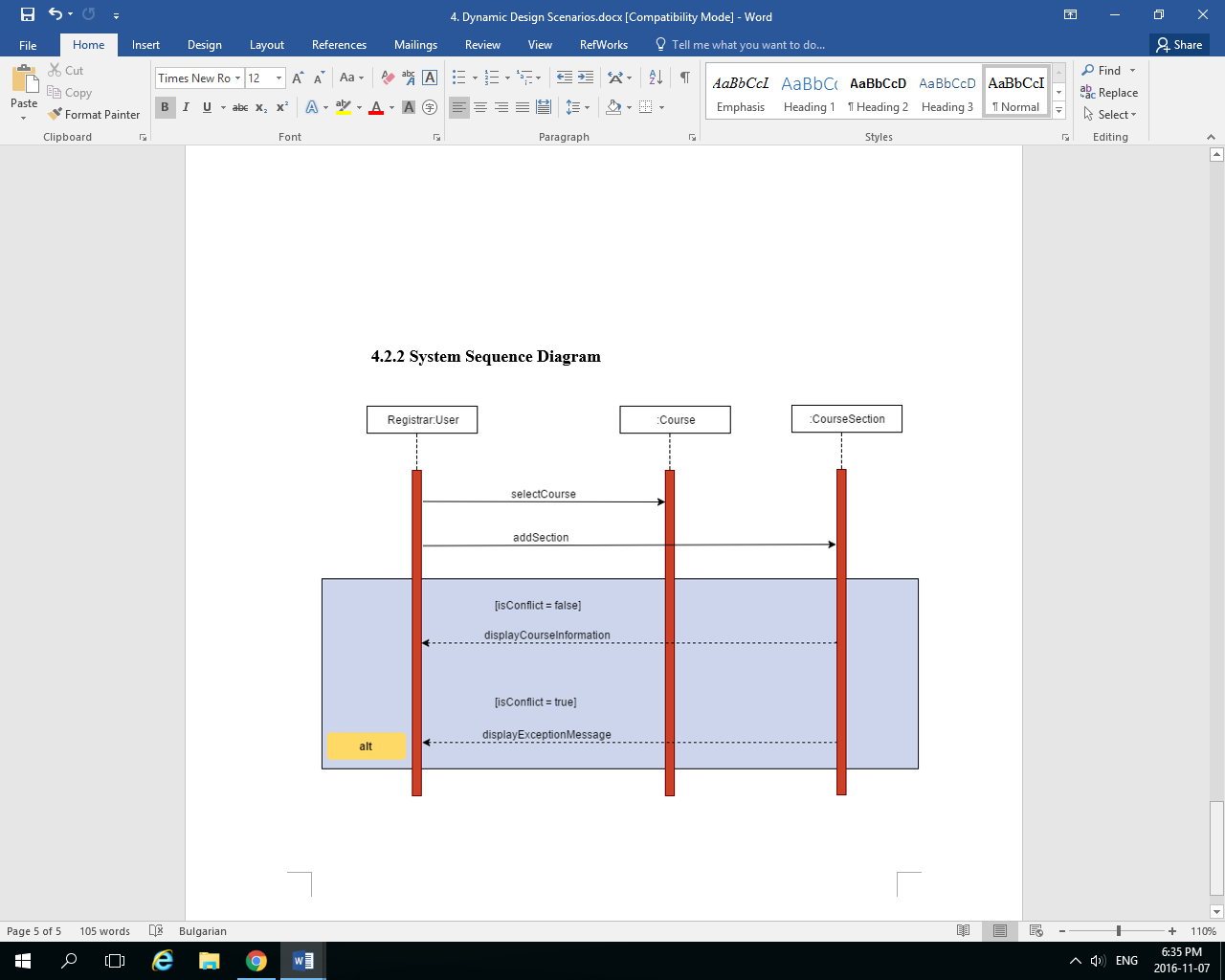
## Case Scenario: UC21-Add Section

### System Sequence Diagram



|  |  |
| --- | --- |
| **Operation** | **addSection(section:CourseSection)** |
| **Cross-reference** | UC21-Add Section |
| **Preconditions** | The user is logged in.  The chosen course must exist in the course list database. |
| **Postcondition** | A new section is added to the course. |

### System Sequence Diagram



# Estimation

|  |  |  |
| --- | --- | --- |
| **Activity** | **Artefacts** | **Estimation** |
| System overview | Project description, domain model | 5 hours  Cost: 100 |
| Role distribution | Team members and roles | 1 hour  Cost: 20 |
| Project definition | Introduction, project definition | 2 hours  Cost: 40 |
| Goals and constraints | Use Cases, constraints and qualities | 7 hours  Cost: 140 |
| Resource evaluation | Human and technical resources | 2 hours  Cost: 40 |
| Scoping and Prototyping | Features, Prototype | 18 hours  Cost: 360 |
| Solution sketching | Technologies used | 1 hour  Cost: 20 |
| Plan | Activities artifacts estimates, schedule, risks | 6 hours  Cost: 120 |
| Architectural design | Architecture design, Class diagram | 8 hours  Cost: 160  **Revised: 10 hours**  **Cost: 200** |
| Subsystems Interfaces | Module Interface Specifications, Unit descriptions | 8 hours  Cost: 160  **Revised: 10 hours**  **Cost: 200** |
| Requirements implementation | Configured Server, Database, View implementation, Model implementation | 55 hours  Cost: 1100  **Revised: 80 hours**  **Cost: 1600** |
| Dynamic design scenarios | System sequence diagrams | 6 hours  Cost: 120  **Revised: 8 hours**  **Cost: 160** |
| Unit tests | Tested and untested items | 20 hours  Cost: 400 |
| Requirements tests | Tested and untested items | 5 hours  Cost: 100 |
| Stress tests | Tested and untested items | 8 hours  Cost: 160 |
| Write manuals | Installation and User Manual | 40 hours  Cost: 800 |

|  |  |
| --- | --- |
| Total number of hours | 192 hours |
| Total cost | 3840 |
| **Total number of hours (revised)** | **223** |
| **Total cost (revised)** | **4460** |

Each subsystem corresponds to a part of the software that need to be developed in order to create the schedule generator. The software is made up of 3 principal subsystems that will interact with each other. The estimations of the development, design and implementation are already integrated in the activities defined above.

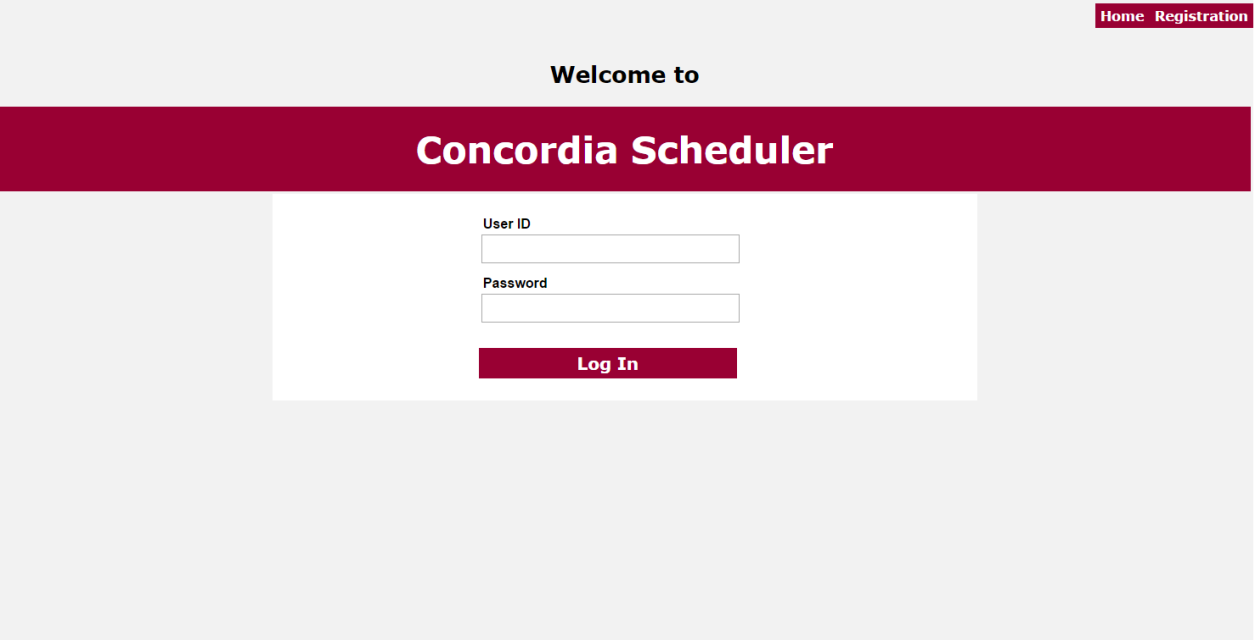
|  |  |  |
| --- | --- | --- |
| **Subsystem** | **Summary** | **Estimation** |
| User | The student can interact with the system by choosing their program and their preferences. | 30 hours  Cost: 600 |
| Courses | The list of courses available with their sections according to the program chosen. All this information is taken from a database. | 20 hours  Cost: 400 |
| Schedule | The schedule generator that creates the student’s schedule with all the information collected. | 30 hours  Cost: 600 |

|  |  |
| --- | --- |
| **Total number of hours** | **80 hours** |
| **Total cost** | **1600** |

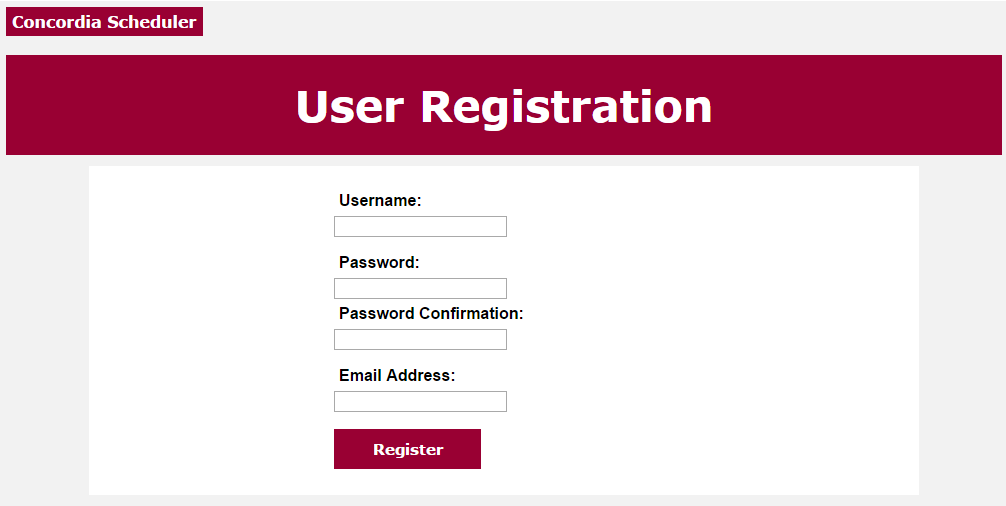
# Rapid Prototyping and Risk

After having discussed the drawbacks of the prototype presented in Deliverable 1, the following prototype is now presented by the team:

## User Interface during Login and Registration phases



When User visits the page without existing session



Page presented to the User after the user clicked the “Registration” button in the top right corner.

## Interface after the login was successful

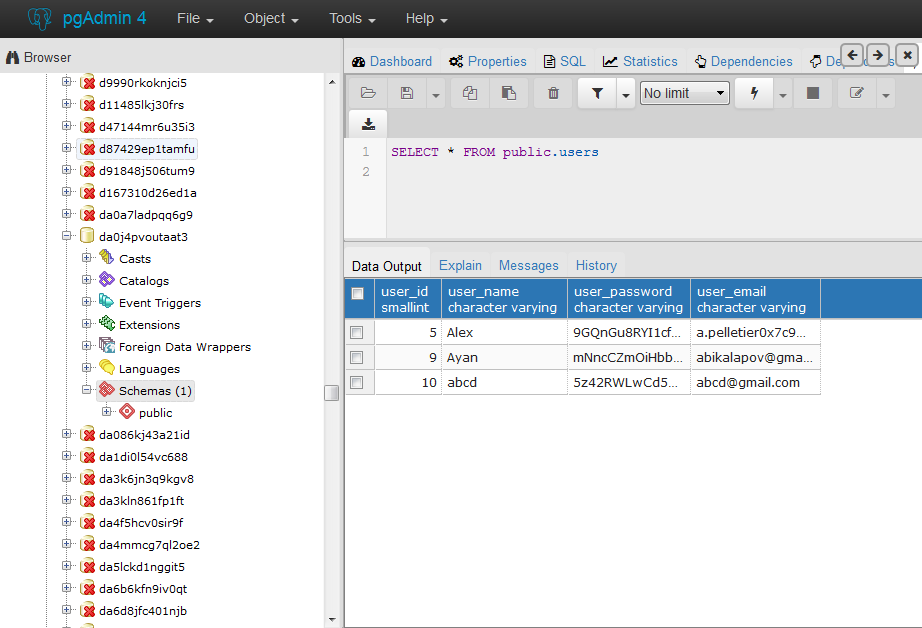
This part is currently under development, since the algorithm for the Schedule generation is being implemented. We expect to have “Generate Sequence” and “Edit preferences” actions to be available on this dashboard.

## Algorithm for the sequence generation

Please find the algorithm in pseudocode format attached in part 7.2 of Appendix A

## Backend

For our backend, we are using PostgreSQL Database, as presented below:



Our final database configuration design will consist of two tables, one for all the user information, and another one for the course information that is needed for successful sequence generation:





Source code example of page navigation and form submission below:



Source code example of database communication and credentials validation:



## Analysis on risks impact and estimations for the Deliverable 2 Prototype

### Risk impact Analysis

The following risks were identified in Deliverable 1:

* Time Constraint
* Team Cohesion and Communication
* Team Experience
* Uncooperative Members
* Poor Implementation
* Unforeseen Risks

After having worked on Deliverable 2, here is the risk analysis for each risk identified in Deliverable 1:

* Time Constraint

Prioritization played a great role in Deliverable 2, since we had to prioritize the tasks for the new deliverable over the back-fixes for deliverable 1. Overall the risk remains minimal, but team coordination can still be improved.

* Team Cohesion and Communication

During the design discussion of the product, we have run into some disagreements in the team in the coding division. This can be attributed to inexperience in management and lack of communication. The issue has been resolved by reassigning conflicting parties on different tasks. Current team configuration presents minimal risk to the project.

* Team Experience

The team experience in coding in JavaScript currently presents the highest risk to the project. However, there is a continuous learning process by the members of the coding team, and the remainder of the work should only involve syntax practice, since the main algorithm has been thought out.

* Uncooperative Members

No member of the team has been uncooperative during the work done on Deliverable 2. This risk is currently nil.

* Poor Implementation

This risk is related to the Team Experience mentioned earlier. This risk will be fully assessed during part 3 of the project, where extensive testing will be done on the system to fully test the limits of our implementation.

* Unforeseen Risks

No member of the team is currently planning on dropping the class, this risk should be minimal with proper team communication.

### Estimate impact Analysis

Deliverable 2 has modified our Estimates towards larger time constraints and spending. As the team progresses with the project, new difficulties arise and cause us to reevaluate the effort necessary for certain parts of the project, thus modifying the estimated time and resources spent on that task. Refer to part 5 of this document for more details.

# Appendix A

## Modified Use Case Diagrams from Deliverable 1

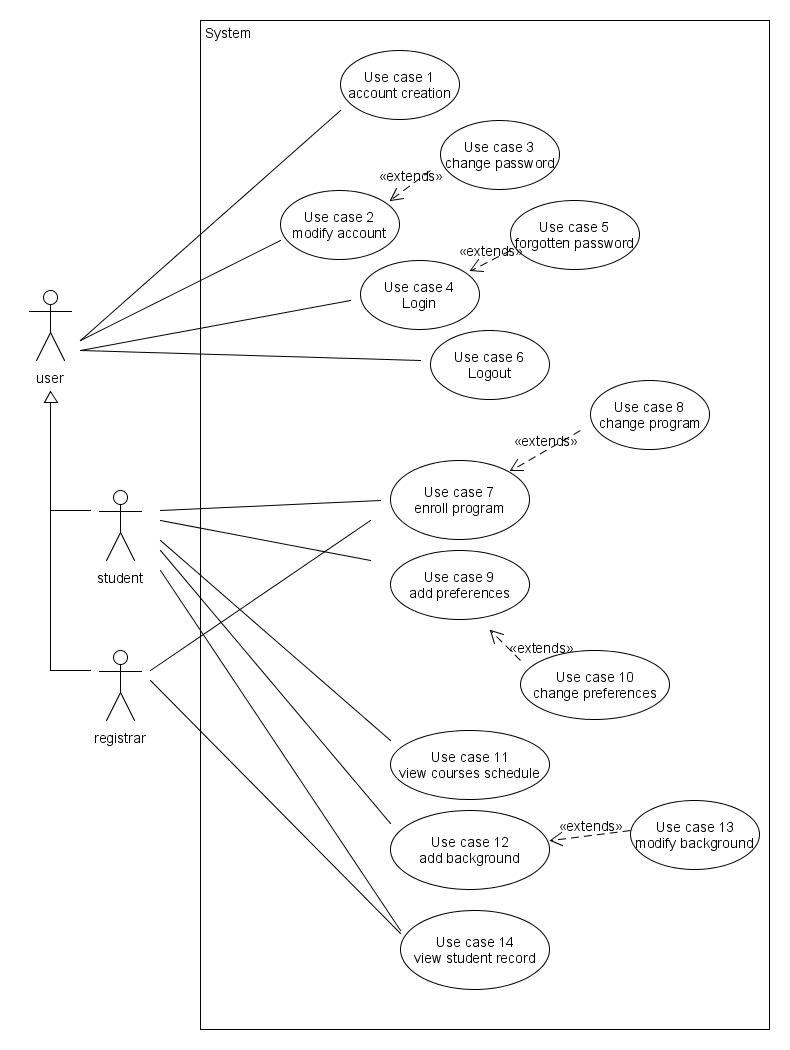


Figure 3.1: Use Case Diagram for the Users and the Student

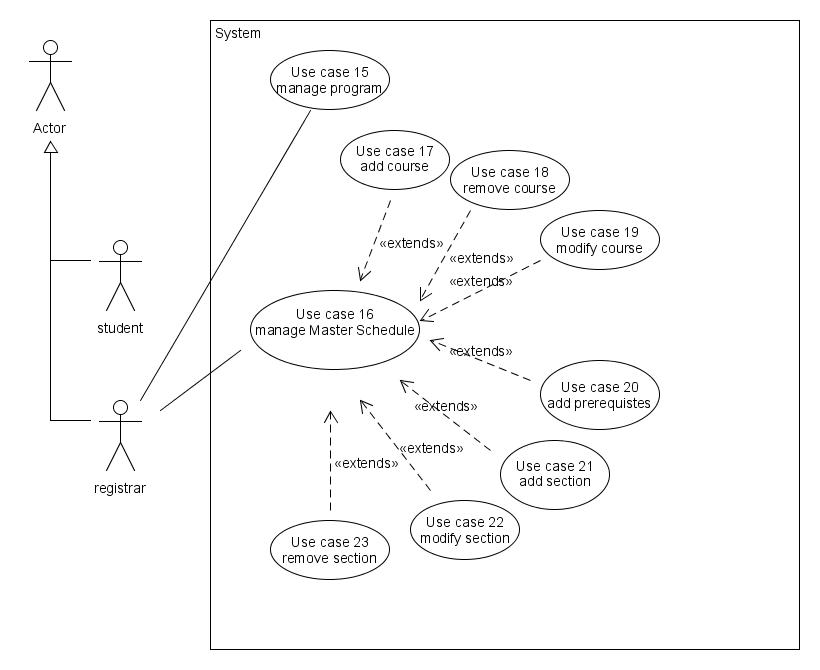


Figure 3.2: Use Case Diagram for the Registrar

## Sequence generation algorithm in Pseudocode