Table of Contents for EnrollEase

Contents

Scope

EnrollEase is a system designed to simplify the student enrollment process by generating multiple schedule options for one semester based on the student's preferences. The student can review these options, select a preferred schedule, click register, and seamlessly enroll in the chosen courses. The system will focus on providing an easy and efficient enrollment experience for students.

The system does not provide authentication services (provided by Shibboleth) and does not contain the offered courses or course enrollment functionality (provided by Jenzabar).

Authorship: Winter 2024 - CSE 430 “Architectural Design” Class

Status and Change History

Latest version on 04/09/2024.
Expected completion date on 04/09/2024

Change History	02/02/2024 – Updated Diagram from simple component to a more complex component diagram.
	02/12/2024 – Changed where Shibboleth, Jenzabar, Interactive Map, and Rate My Professor are located to show they are external parts.
	02/22/2024 – Added an Assembly Connection between Controller and Algorithm that was missing.
	02/27/2024 – Removed all unnecessary numbers and simplified diagram.
	02/28/2024 – Described the scope of EnrollEase
	03/18/2024 – Added most of the Diagrams and put TBDs for the rest.
	03/28/2024 – Replaced all TBDs with Diagrams, Charts, and similar stuff.

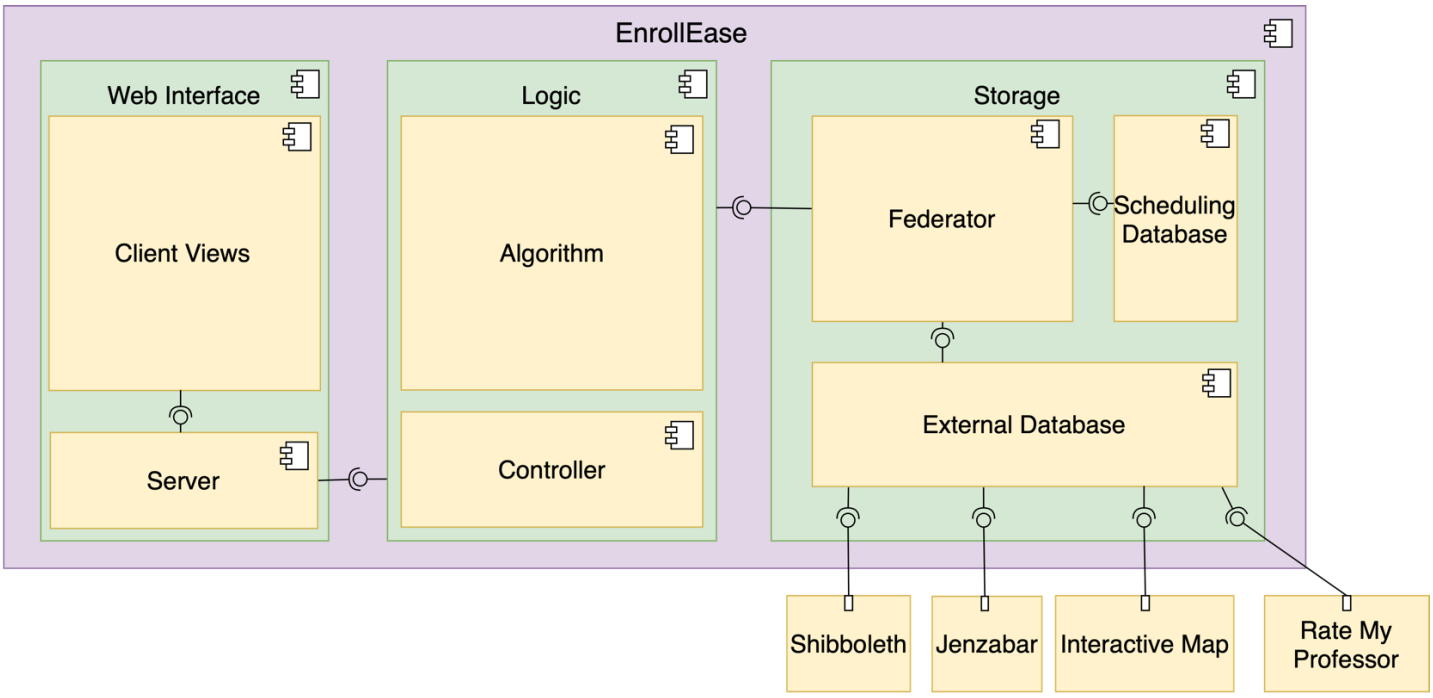
Stakeholders

EnrollEase serves as an intuitive enrollment system with stakeholders including the University, students, advisors, and the development team. For the University, EnrollEase seamlessly integrates with existing systems like Shibboleth for secure authentication and Jenzabar for course registration. Students benefit from a user-friendly interface, enabling them to easily customize and register for semester plans based on personal preferences. Advisors utilize the system to assist students in navigating course requirements and optimizing schedules. The development team ensures the continuous improvement and functionality of EnrollEase to meet the evolving needs of all stakeholders.

Viewpoints

Viewpoints	Component Diagram - Software Design Textbook - Helfrich, James. Software Design. Available from: VitalSource Bookshelf, Kendall Hunt Publishing, 2023.
	Data Flow Diagram - Software Design Textbook - Helfrich, James. Software Design. Available from: VitalSource Bookshelf, Kendall Hunt Publishing, 2023.
	Structure Chart - Software Design Textbook – Helfrich, James. Software Design. Available from: VitalSource Bookshelf, Kendall Hunt Publishing, 2023.
	Class Diagram - Software Design Textbook – Helfrich, James. Software Design. Available from: VitalSource Bookshelf, Kendall Hunt Publishing, 2023.
	Site Plan - https://www.digitalsilk.com/digital-trends/website-planning/
	Entity Relationship Diagram - https://drawio-app.com/blog/entity-relationship-diagrams-with-draw-io/
	Flow Chart - Software Design Textbook – Helfrich, James. Software Design. Available from: VitalSource Bookshelf, Kendall Hunt Publishing, 2023.
	Pseudocode - Software Design Textbook – Helfrich, James. Software Design. Available from: VitalSource Bookshelf, Kendall Hunt Publishing, 2023.
	JSON Schema - https://json-schema.org/learn/miscellaneous-examples
	SQL - https://www.sqlite.org/docs.html

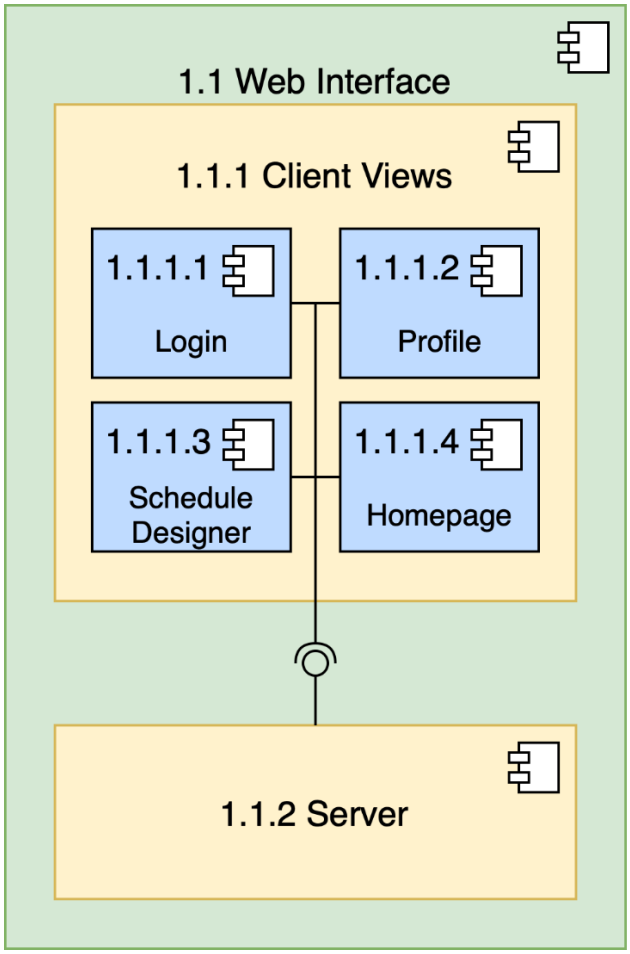
1.0 EnrollEase



Name	**1.0 EnrollEase**
Purpose	Minimize the time required for students to register for classes. Minimize walk-in distance from class to class. Empower students to generate customized schedules.
Description	This program presents the required classes for a given major, allows the students to place the required class in a semester plan, and sends the semester plan to the registration system. It also allows the student to auto-generate a plan considering the location of a class, the ratings of a professor, and other student-specified preferences.
Requirements	All
Elements	1.1 Web Interface
	1.1.1 Client Views
	1.1.2 Server
	1.2 Logic
	1.2.1 Algorithm
	1.2.2 Controller
	1.3 Storage
	1.3.1 Federator
	1.3.2 Scheduling Database
	1.3.3 External Database
	Shibboleth – See Glossary
	Jenzabar – See Glossary
	Interactive Map – See Glossary
	Rate My Professor – See Glossary
Referenced by	N/A
Viewpoint	Component Diagram

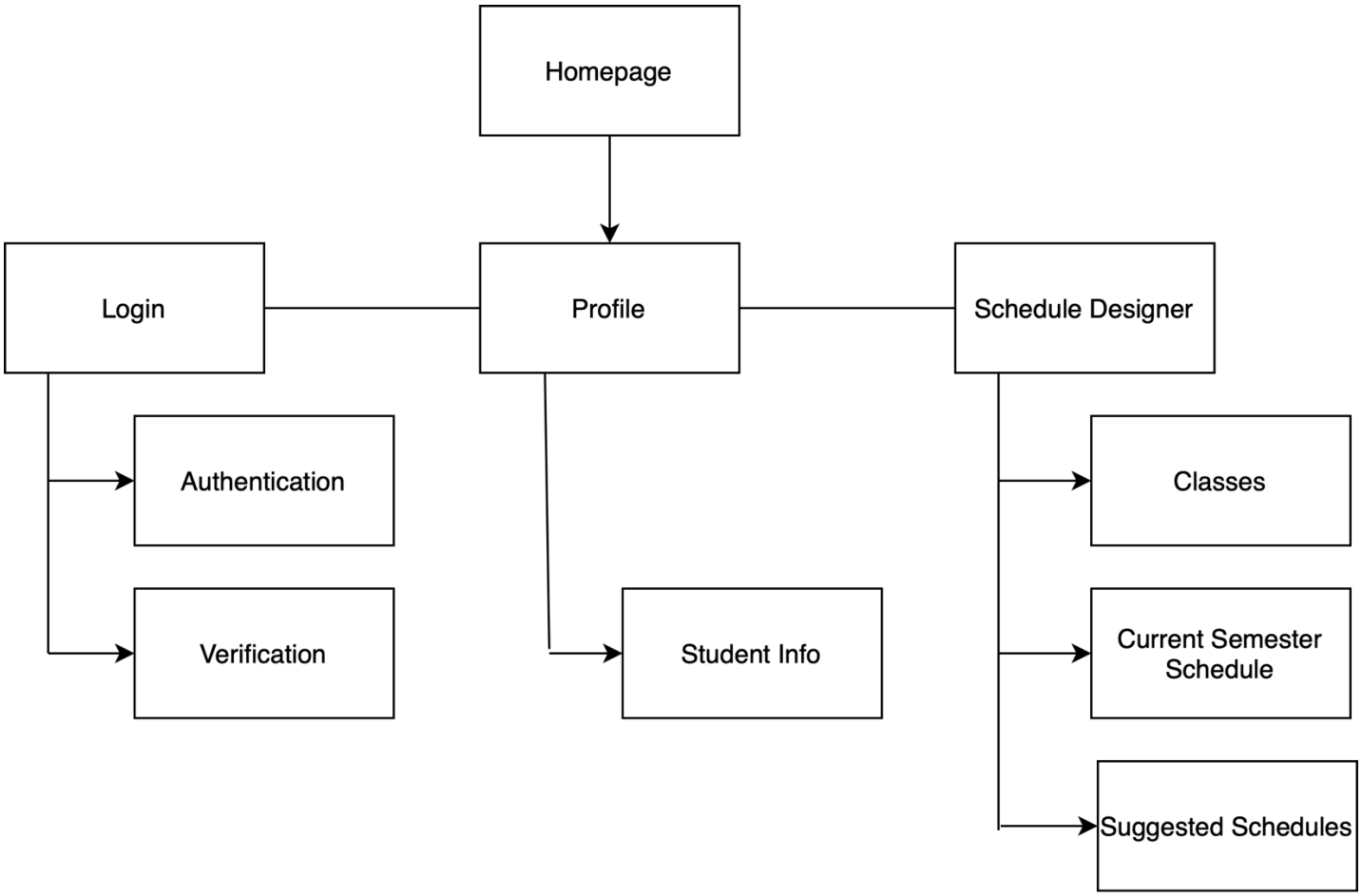
Web Interface

1.1 Web Interface



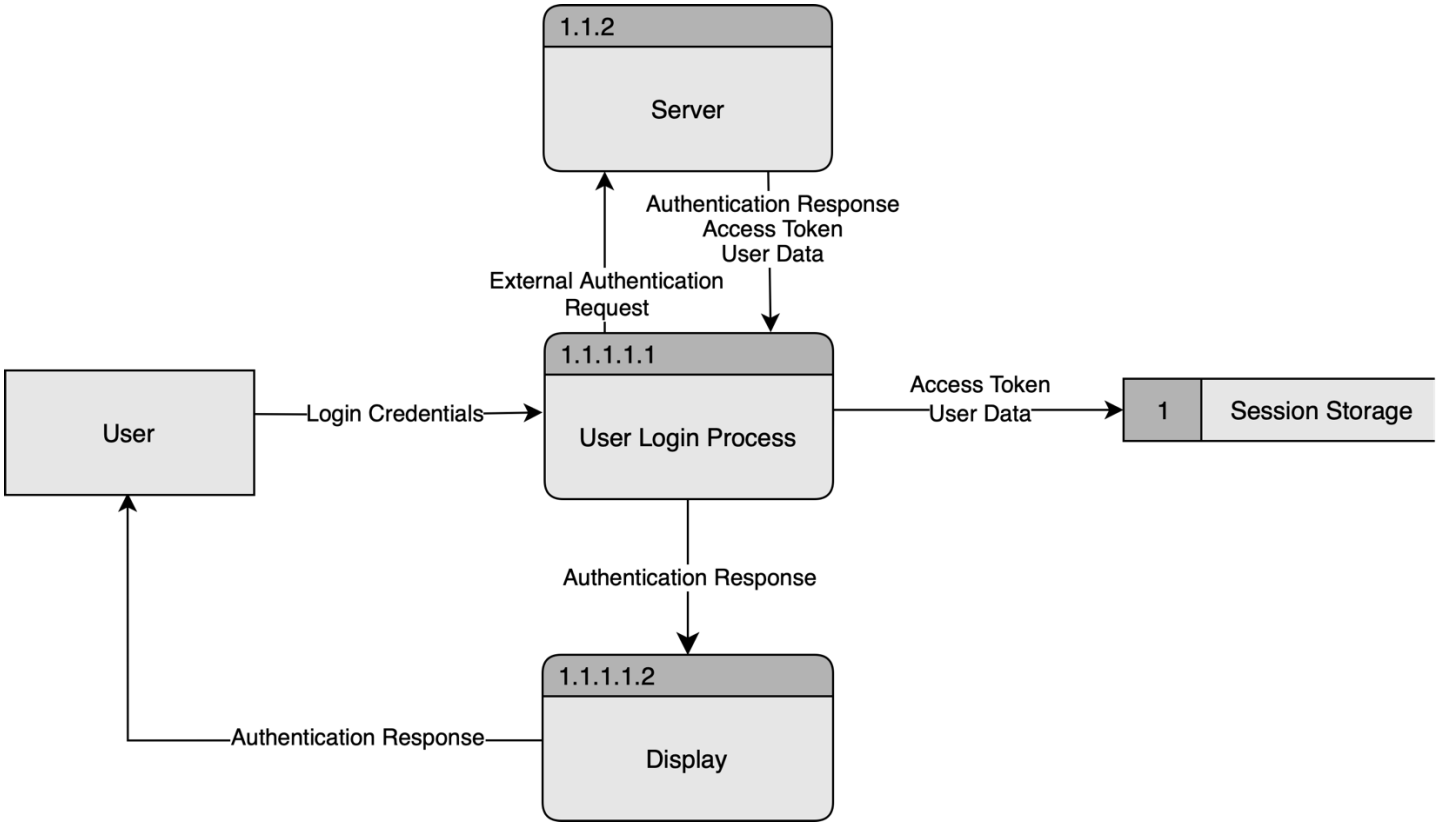
Name	**1.1 Web Interface**
Purpose	Represent the client-facing portion of the software.
Description	This portion of the program will handle client-facing code as well as the server which will act as the mediator between client requests and program logic and storage.
Requirement(s)	18
Elements	1.1.1 Client Views
	1.1.1.1 Login
	1.1.1.2 Profile
	1.1.1.3 Schedule Designer
	1.1.1.4 Homepage
	1.1.2 Server
Referenced by	1.0 EnrollEase
Viewpoint	Component Diagram

1.1.1 Client Views



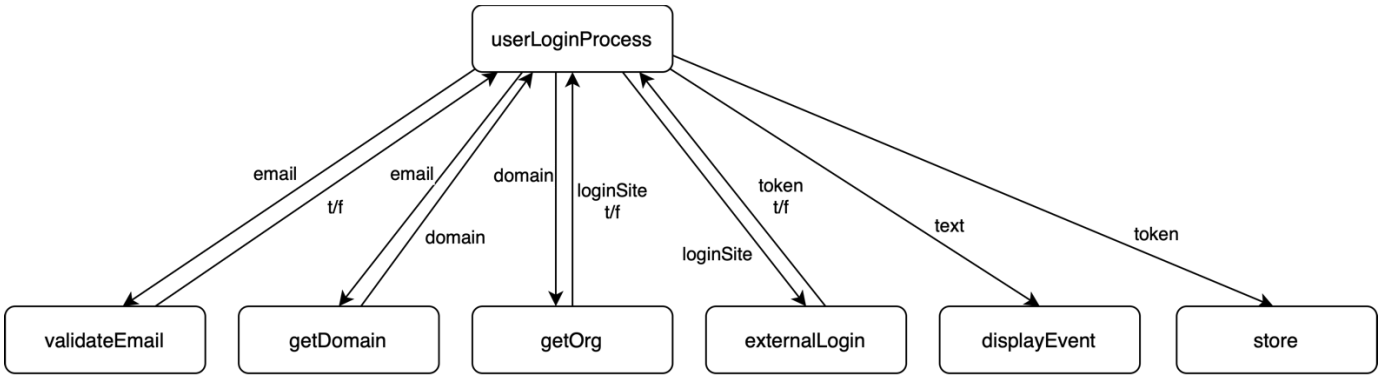
Name	**1.1.1 Client Views**
Purpose	The User’s view of the webpages
Description	The overview of what the user sees when they visit the website. It takes them to a login page where they can then go to their profile or design their schedule. The website is then stored as session storage to make it so that they can log back in as needed.
Requirements	15-19
Elements	1.1.1.1 Login
	1.1.1.2 Profile
	1.1.1.3 Schedule Designer
	1.1.1.4 Homepage
	Authentication: Checking the user's credentials
	Verification: Verifying that the user is who he or she says he or she is whether that be through text or email
	Student Info: The student's major, preferences, ID, contact info, GPA, and expected graduation.
	Classes: Class Sections available for the student.
	Current Semester Schedule: The schedule for the current semester the student has selected
	Suggested Schedules: Schedule suggestions for the student.
Referenced by	1.0 EnrollEase
	1.1 Web Interface
	1.1.2 Server
Viewpoint	Site plan

1.1.1.1 Login


1.1.2
Server
Authentication Response
Access Token
User Data
External Authentication
Request
1.1.1.1.1
User
Login Credentials
User Login Process
Access Token
User Data
1
Session Storage
Authentication Response
1.1.1.1.2
Authentication Response
Display

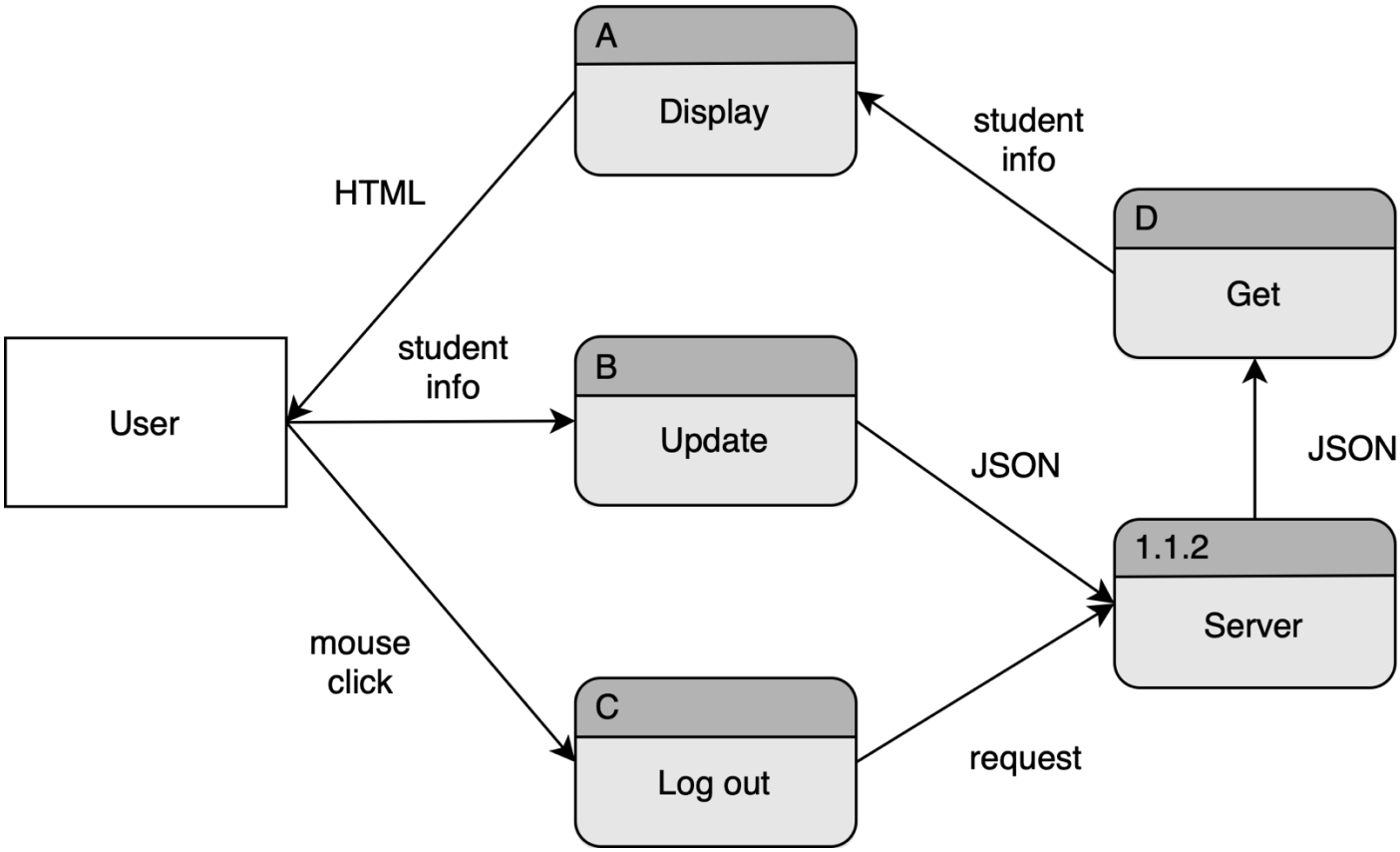
Name	**1.1.1.1 Login**
Purpose	Provide Authentication for users accessing system.
Description	The user will give user credentials which will be sent to the campus' external authentication server. The response will be shown to the user and if it goes through the server will receive a token that will be used to get the profile information related to that user.
Requirements	Requirements 15 and 21
Elements	1.1.1.1.1 User Login Process
	1.1.2 Server
	1 Session Storage - a browser- and session- based storage for the user's data.
	Display - Display Authentication Response to the user, whether it is successful or if and where it failed.
	Access Token - Token received as proof of successful login, stored for future authorization attempts for a set amount of time.
	Login Credentials - User's email address.
	External Authorization Request - Request to 3rd party program for authorization
	Authorization Response - Success or failure, to be displayed to the user.
	User Data - Data pertaining to the user's profile, received from the application's database.
	User - The user external to the system.
Referenced by	1.1 Web Interface
	1.1.1 Client Views
Viewpoint	Data Flow Diagram

1.1.1.1.1 User Login Process



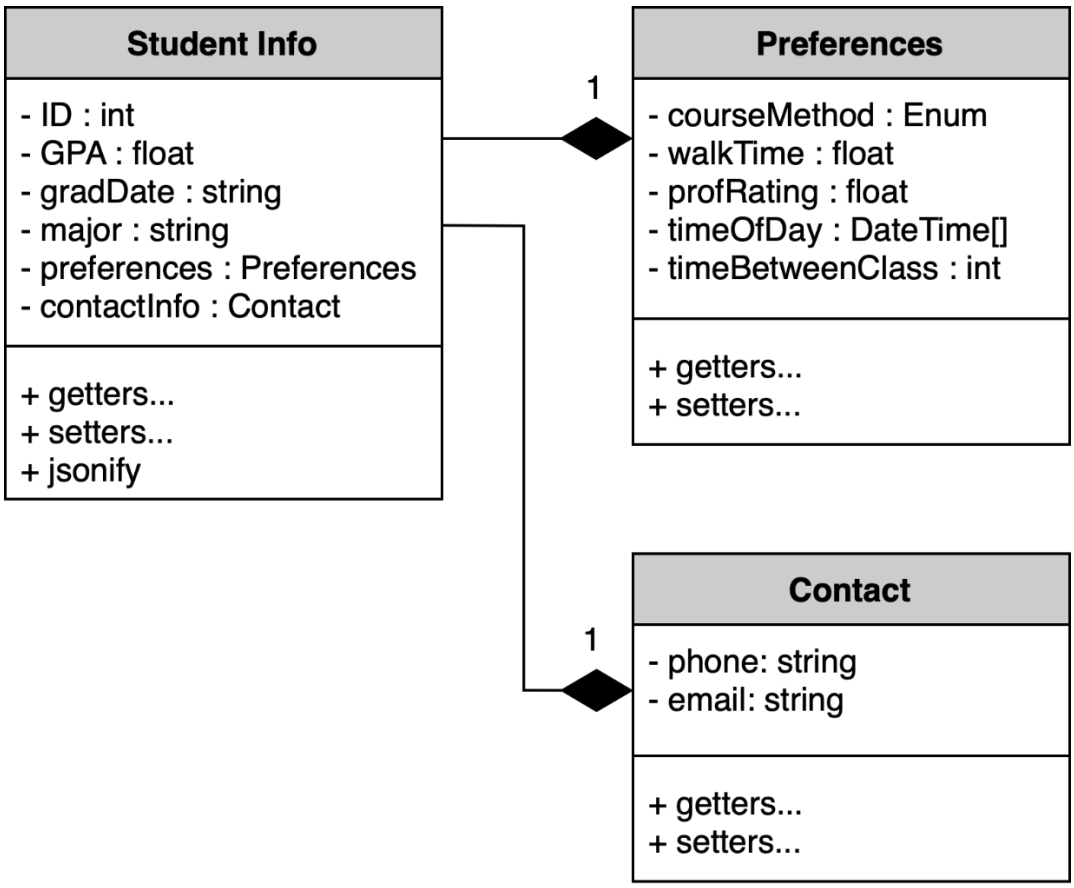
Name	**1.1.1.1.1 User Login Process**
Purpose	Initiate the third-party authentication based on user input and storing the token in storage if successful.
Description	The process will take an email as input from the user, parse it to receive the organization and reference the program's database to get the organization specific authentication website. Then it will initiate the third-party authentication and store the token if successful. If a stage fails it will let the user know.
Requirements	Requirements 15 and 21
Elements	User Login Process: see 1.1.1.1.1
	validateEmail - Enforce a structure of (text)@(text).(text) on user input.
	getDomain - Parse through the string email and return everything after the @ symbol.
	getOrg - Send domain to Server and request data regarding third-party authentication attached to the domain if included in the database.
	externalLogin - Send user to external website to do third-party authentication with itself as a callback website.
	displayEvent - Display if an action fails.
	store - give token to Server to send to Storage.
	t/f - Boolean - Received at multiple points to determine success. If false, the failure will be displayed to the user and the process will terminate.
	Token - Token received as proof of successful login, stored for future authorization attempts for a set amount of time.
	Email - String - User's email address gotten as input.
	Domain - String - Everything right of the @ of the email which determines the organization.
	loginSite - The organization specific authentication website stored in 1.3 Storage
	Text - String - Message explaining a failure in the process or a successful login.
Referenced by	1.1.1.1 Login
Viewpoint	Structure Chart

1.1.1.2 Profile



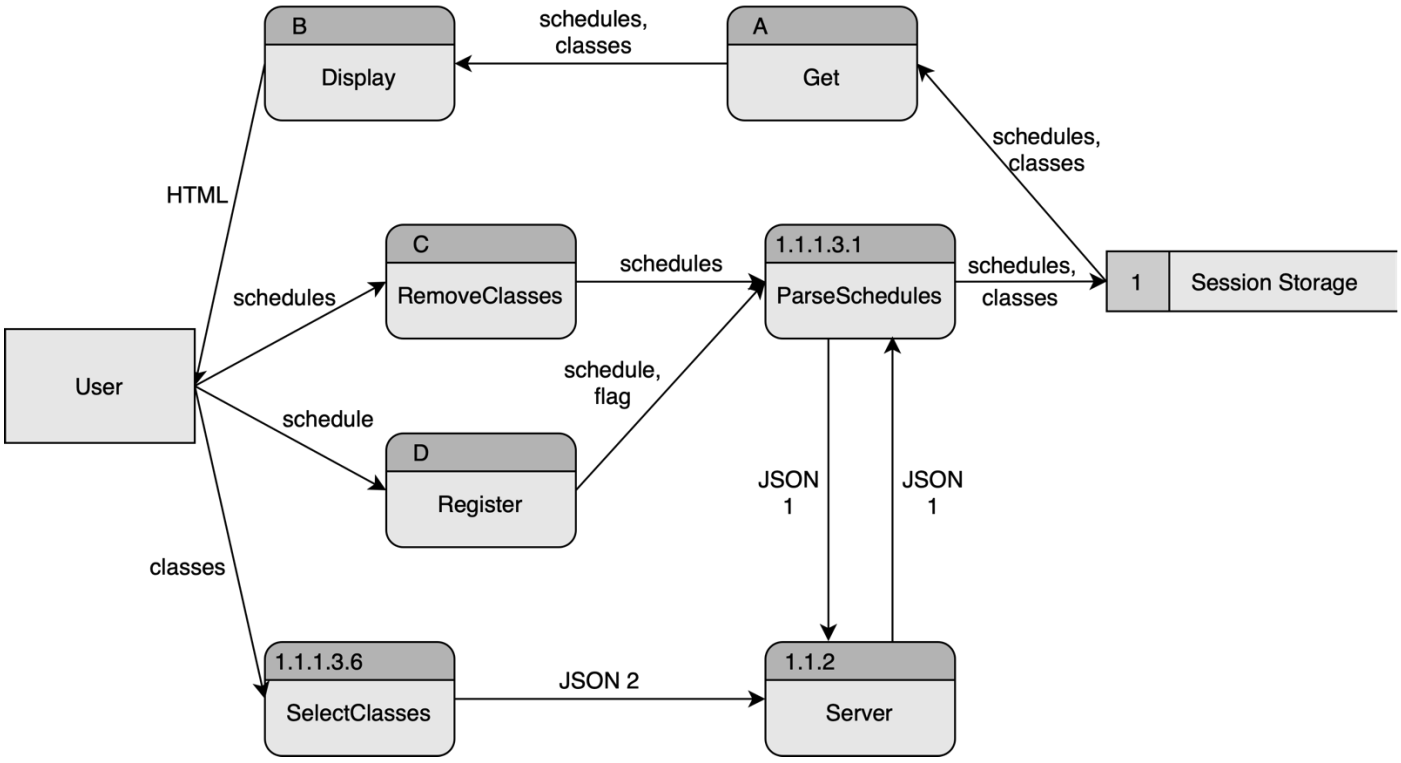
Name	**1.1.1.2 Profile**
Purpose	Display the User's Profile Page.
Description	Having already logged in, the user will be redirected to the profile page upon clicking a link to the page.
Requirements	4, 16
Elements	User: The user external to the system.
	A Display: Displays contents to the user.
	B Update: When user clicks the "save" button when editing preferences, the request and associated info is sent to server for handling.
	C Log Out: User clicks the logout button to end their session.
	D Get: Receives information regarding student's preferences to display to user.
	Server: see #1.1.2
	JSON: see #1.1.2.1 Users Schema
	Student Info: see #1.1.1.2.1
	HTML: What is presented to the user.
	Request: Logout request to server.
	Mouse Click: user clicking the "log out" button.
Referenced by	1.1 Web Interface
	1.1.1 Client Views
Viewpoint	Data Flow Diagram

1.1.1.2.1 Student Info



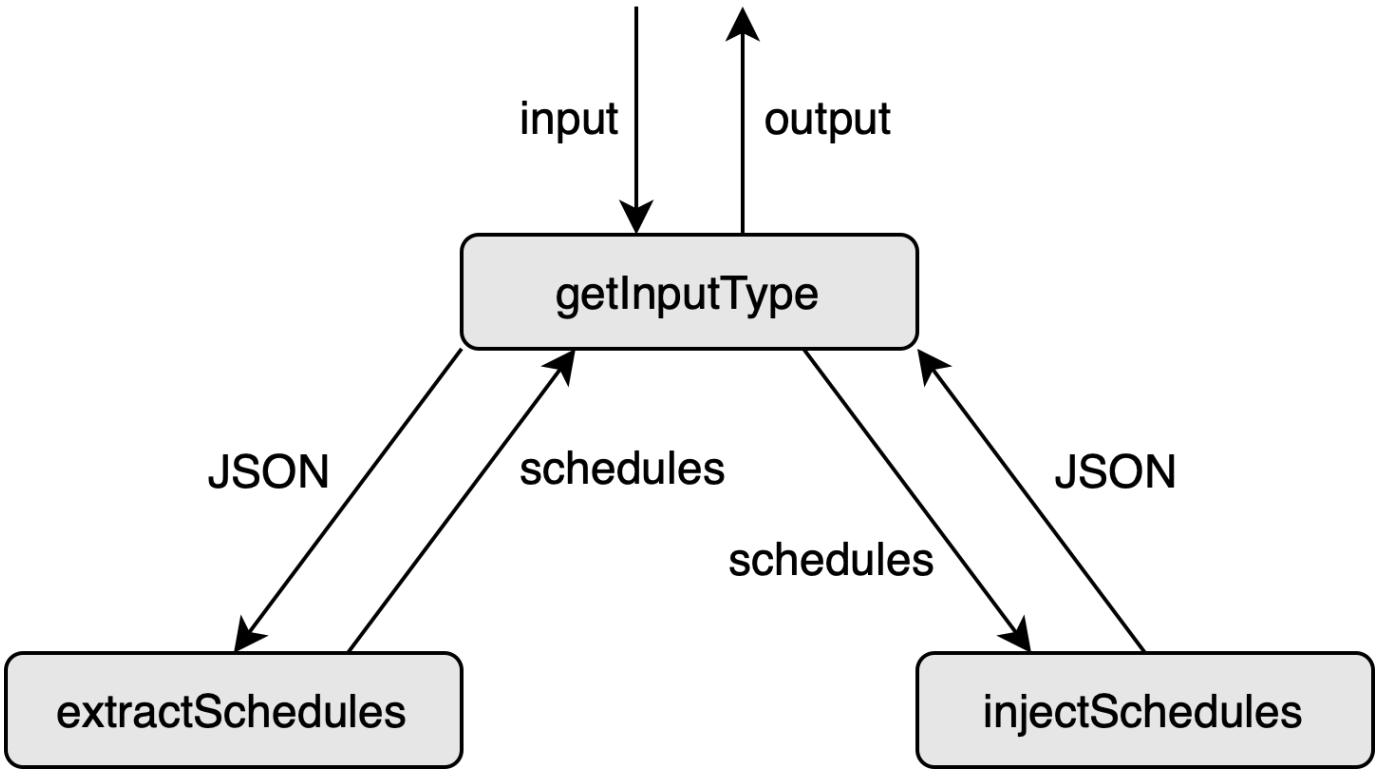
Name	**1.1.1.2.1 Student Info**
Purpose	Object representing the student's information.
Description	Detailed representation of how student's information is stored as it moves through the program.
Requirements	4, 16
Elements	ID: Student's unique ID.
	GPA: Student's GPA.
	gradDate: Student's expected graduation date.
	Major: Students declared major.
	Preferences: Student's preferences that act as filters when generating schedules.
	contactInfo: Student's contact info.
	courseMethod: ENUM = Virtual live, online, in-person
	walkTime: Preferred walk distance between classes.
	profRating: Desired professor rating for class sections.
	timeOfDay: Collection of preferred time for classes.
	timeBetweenClass: Desired "break" time between classes.
	Phone: Student's phone number.
	Email: Student's email.
	Getters & Setters: Simple methods to manipulate private attributes.
	Jsonify: Method to convert class elements into a Json construct for passing to server for handling.
Referenced by	1.1.1.2 Profile
Viewpoint	Class Diagram

1.1.1.3 Schedule Designer



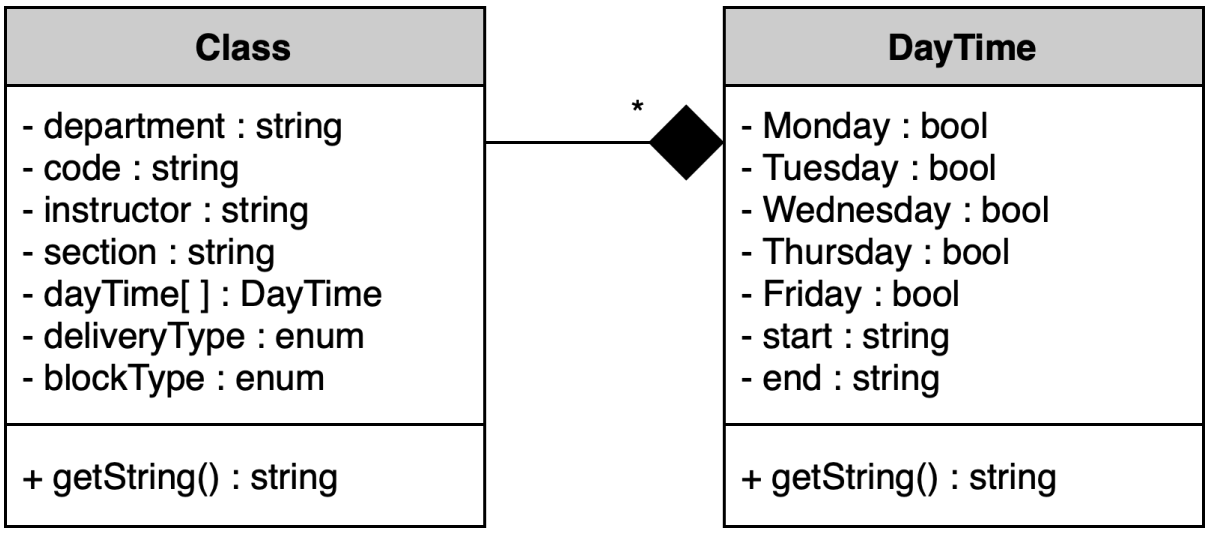
Name	**1.1.1.3 Schedule Designer**
Purpose	Provides a page for the User to input classes, view and edit suggested schedule plans, and register for the desired schedule.
Description	Pulls schedule data from the server, stores it in session storage, and presents it to the User, who can then make edits and save. The schedules are then sent back to the server. There is other functionality included, which allows the User to register for classes.
Requirements	2, 5, 13, 14, 17, 18
Elements	User: The user external to the system
	Server: see #1.1.2
	1.1.1.3.1 ParseSchedules
	A Get: Retrieves data from Session Storage and sends it to Display
	B Display: Converts the data given by Get into HTML.
	C RemoveClasses: Manipulates the user's schedule(s) to remove classes.
	D Register: Sends the classes to be registered along with a flag.
	1.1.1.3.6 Select Classes
	1 Session Storage: a browser- and session- based storage for the user's data.
	JSON 1: see #1.1.2.1
	JSON 2: see #1.1.1.3.9
	Schedules: see #1.1.1.3.8
	Classes: see #1.1.1.3.7
	HTML: What is presented to the user. Includes forms and inputs.
	Updates: The deletions the user makes to their schedules.
	Schedule: The schedule the student chooses to register for. see #1.1.1.3.8
	Flag: A key-value pair indicating that the schedule passed to #1.1.1.3.1 should be registered.
Referenced by	1.1 Web Interface
	1.1.1 Client Views
Viewpoint	DFD

1.1.1.3.1 Parse schedules



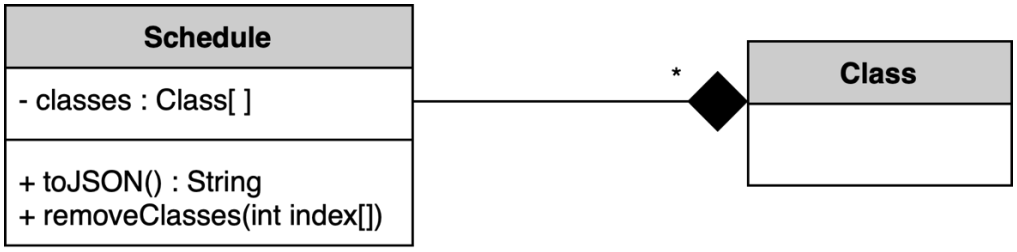
Name	**1.1.1.3.1 Parse Schedules**
Purpose	Isolate just the schedules from the JSON object and insert them back in the right place once updates are made.
Description	This component first decides whether an update or JSON object is being passed in. It then calls the function necessary to complete its purpose.
Requirements	18
Elements	getInputType: determines whether the incoming data is a JSON object or a Schedule
	extractSchedules: turns the schedule info in the JSON object into a Schedule object
	injectSchedules: puts the schedule info back into the JSON object exactly as it was found
	input: either the JSON object or the updated schedules
	output: either the updated JSON object or the schedules
	JSON: see #1.1.2.1
	schedules: see #1.1.1.3.8
Referenced by	1.1.1.3 Schedule Designer
Viewpoint	Structure Chart

1.1.1.3.7 SelectClasses



Name	**1.1.1.3.7 Class**
Purpose	Represent a student's Class in the frontend.
Description	The frontend needs to know all the details about a Class, so this class diagram will temporarily represent that.
Requirements	2, 20
Elements	department: the prefix to the course code, i.e. "CSE"
	code: the number to the course code, i.e. "430"
	instructor: the instructor for this specific section
	section: the number that distinguishes this specific class
	dayTime: an array of the days and times this section is offered
	deliveryType: can be one of three types: online, in-person, or remote
	blockType: can be one of three types: semester, first-block, or second block
	DayTime: a helper class to hold days and their associated times
	getString: a function utilized by both Class and DayTime to get the string interpretation of the object
Referenced by	1.1.1.3 Schedule Designer
	1.1.1.3.8 Schedule
Viewpoint	Class Diagram

1.1.1.3.8 Schedule



Name	**1.1.1.3.8 Schedule**
Purpose	Represents a schedule in the front end, allowing for viewing and editing of schedules.
Description	A schedule is a collection of classes, which can be sent back to the server after the user has made any adjustments.
Requirements	13
Elements	classes: a collection of Classes, see #1.1.1.3.7
	toString(): gets the string representation of the schedule, for use in the JSON object
	removeClasses(): takes in the indices of which classes to delete from the classes member variable
Referenced by	1.1.1.3 Schedule Designer
Viewpoint	Class Diagram

1.1.1.3.9 Classes JSON Schema

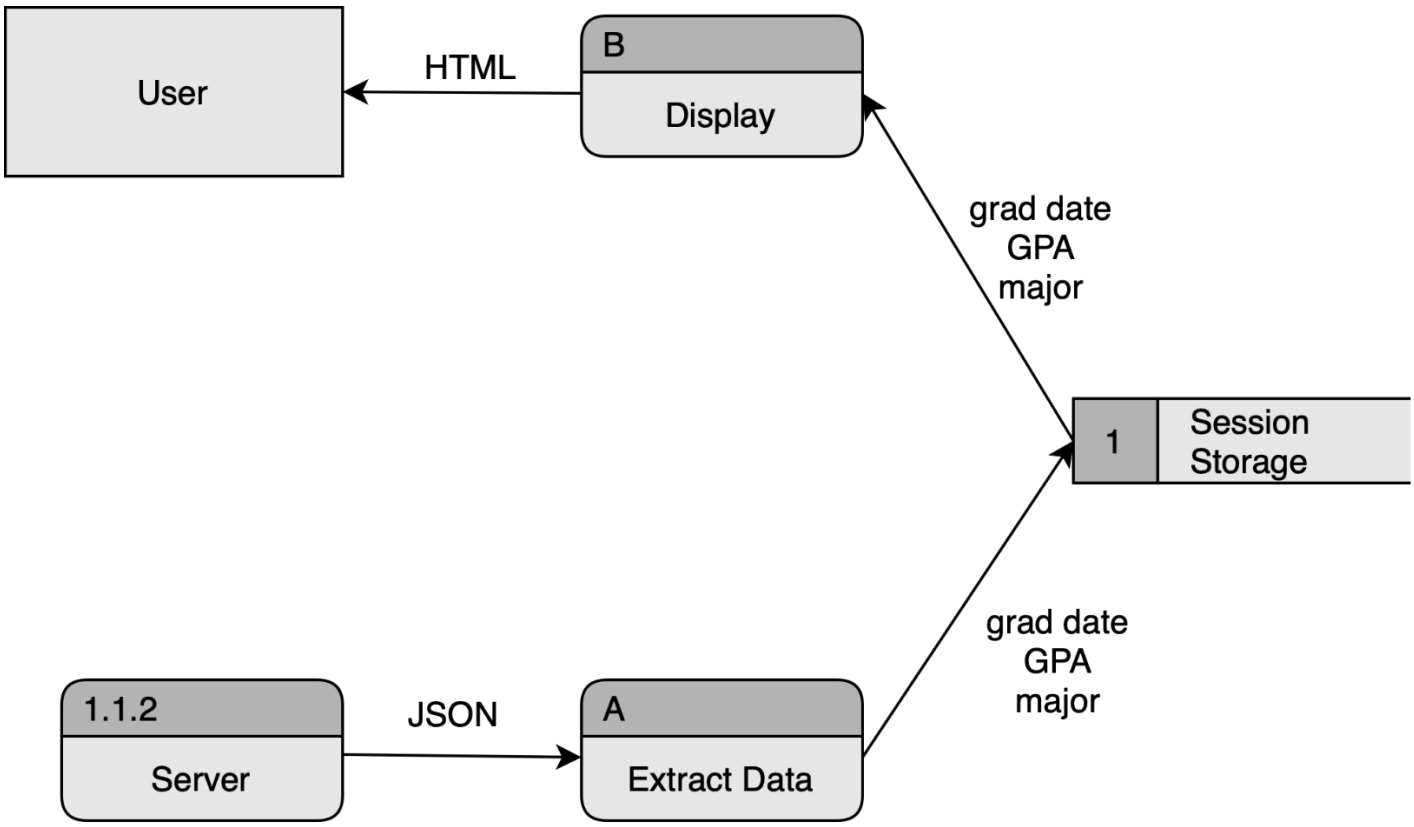
```
{
    "$id": "https://enrollease.com/schema/schedule/1.1",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "Schedule",
    "type": "object",
    "description": "A class object that contains a list of classes.",
    "properties": {
        "section": {
            "type": "object",
            "description": "A section object that contains information about
a course section.",
            "properties": {
                "section_id" : {
                    "type": "string",
                    "description": "The unique identifier for the section."
                },
                "course" : {
                    "type": "object",
                    "description": "The course object that the section
belongs to.",
                    "properties": {
                        "course_id" : {
                            "type": "string",
                            "description": "The unique identifier for the
course."
                        },
                        "course_title" : {
                            "type": "string",
                            "description": "The title of the course."
                        }
                    },
                    "required": ["course_id", "course_title"]
                },
                "location" : {
                    "type": "object",
                    "description": "The location object where the section is
located.",
                    "properties": {
                        "building" : {
                            "type": "string",
                            "description": "The building where the section
is located."
                        },
                        "room" : {
```

```
                            "type": "string",
                            "description": "The room number where the
section is located."
                        }
                    },
                    "required": ["building", "room"]
                },
                "professor" : {
                    "type": "object",
                    "description": "The professor object teaching the
section",
                    "properties": {
                        "professor_name" : {
                            "type": "string",
                            "description": "The name of the professor
teaching the section."
                        },
                        "professor_score" : {
                            "type": "float",
                            "description": "The score of the professor
teaching the section."
                        }
                    },
                    "required": ["professor_name", "professor_score"]
                },
                "start_time" : {
                    "type": "datetime",
                    "description": "The start time of the section."
                },
                "end_time" : {
                    "type": "datetime",
                    "description": "The end time of the section."
                },
                "seats_max" : {
                    "type": "integer",
                    "description": "The maximum number of seats available in
the section."
                },
                "seats_open" : {
                    "type": "integer",
                    "description": "The number of open seats in the
section."
                },
                "open" : {
                    "type": "boolean",
```

```
                    "description": "Indicates if the section is open for
enrollment."
                },
                "delivery_method" : {
                    "type": "object",
                    "properties": {
                        "data": {
                            "enum": ["in-person", "online", "hybrid"],
                            "description": "The delivery method of the
section."
                        }
                    },
                    "required": ["data"]
                },
                "status" : {
                    "type": "object",
                    "properties": {
                        "data": {
                            "enum": ["open", "closed", "waitlist"],
                            "description": "The status of the section."
                        }
                    },
                    "required": ["data"]
                }
            },
            "required": ["section_id", "course", "location", "professor",
"start_time", "end_time", "seats_max", "seats_open", "open",
"delivery_method", "status"]
        }
    }
}
```

Name	**1.1.1.3.9 Classes JSON Schema**
Purpose	Validates that the data is encoded in JSON and is known for the system.
Description	Object and attribute declarations used by Schedule Designer to prepare a JSON response for Server.
Requirements	13, 17
Referenced By	1.1.1.3 Schedule Designer
Viewpoint	JSON Schema

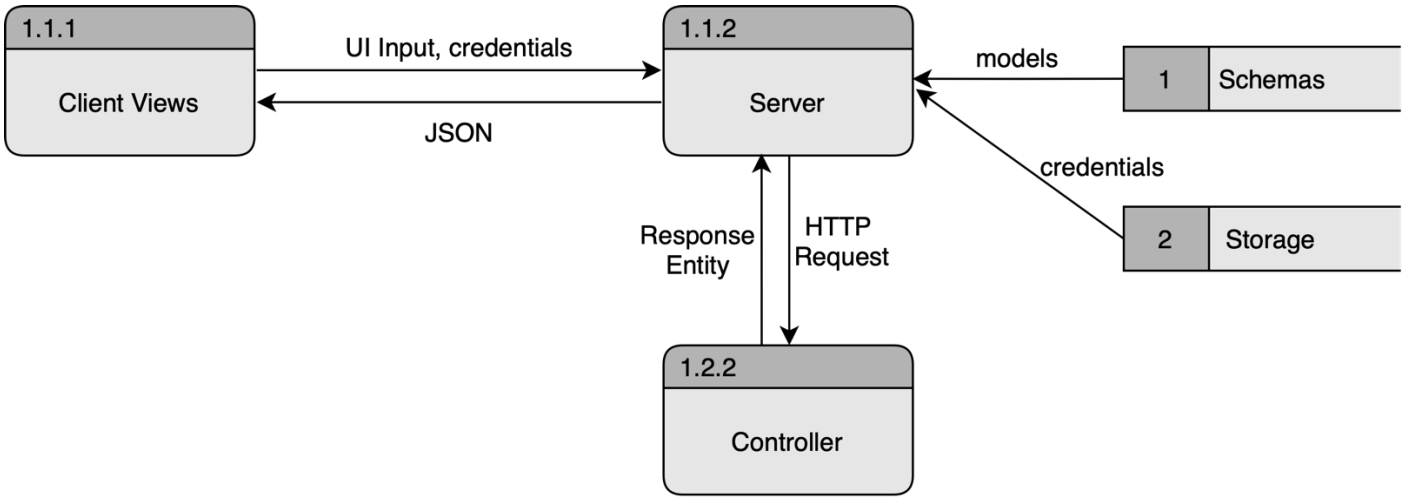
1.1.1.4 Homepage



Name	**1.1.1.4 Homepage**
Purpose	The User’s view of the Homepage
Description	The overview of what the user sees when they manage to login and visit the homepage.
Requirements	22
Elements	Extract Data: This extracts the data from the server using JSON
	Display: This displays the homepage to the user
	1 Session Storage: a browser- and session- based storage for the user's data.
	Grad Date: The user's graduation date
	GPA: The user's GPA
	Major: The user's major
	HTML: What is presented to the user
	JSON: see #1.1.2.1
	Server: see # 1.1.2
	User: The user external to the system
Referenced by	1.1 Web Interface
	1.1.1 Client Views
Viewpoint	DFD

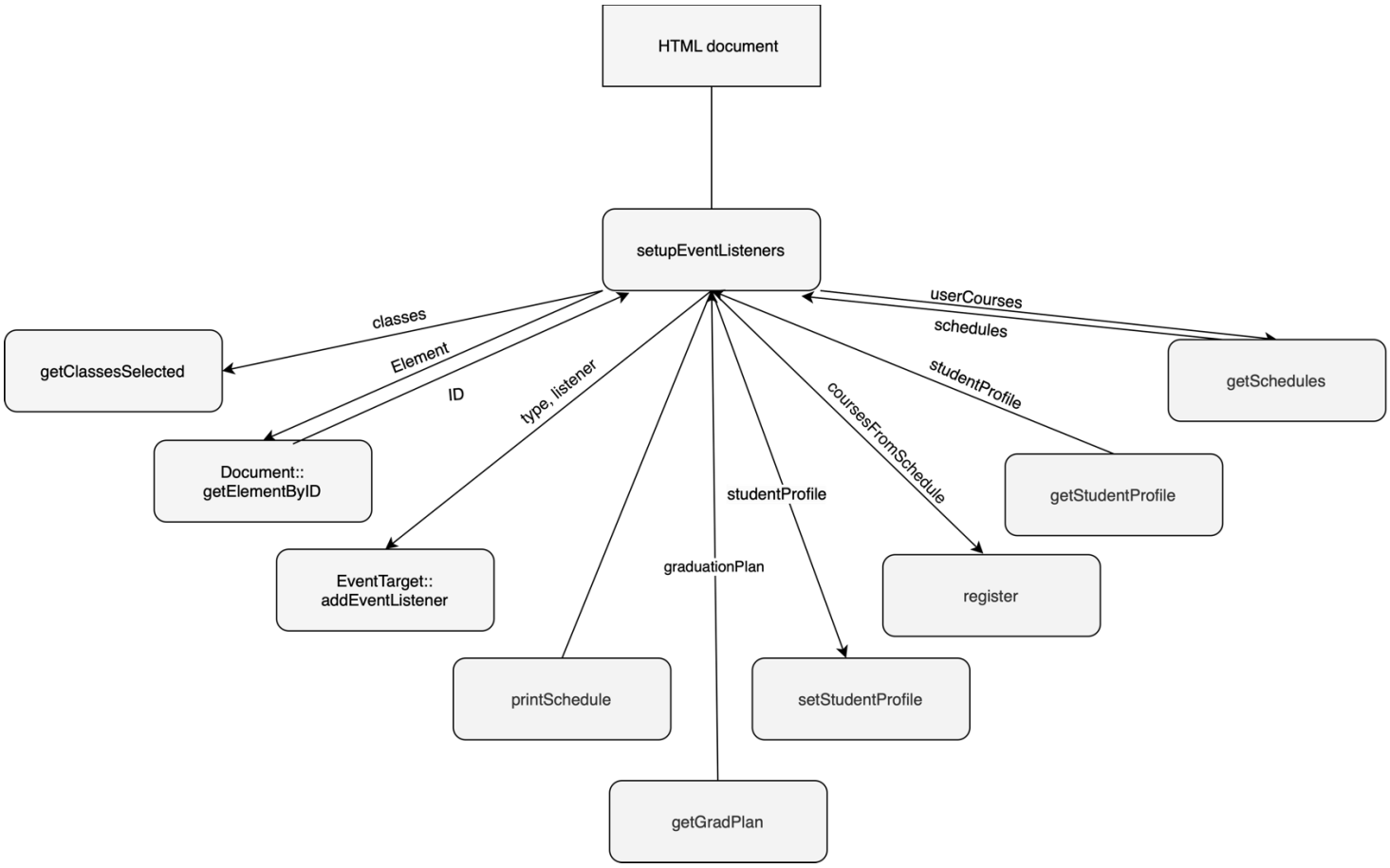
Server

1.1.2A Server



Name	**1.1.2 Server**
Purpose	Mediator between client and business logic.
Description	Converts an action from the web interface into an HTTP request if authorized, receives a response entity back, and serves an answer that the front end can interpret.
Requirements	1-5, 13, 21
Elements	1.1.1 Client View
	1.2.2 Controller
	1 Schemas: #1.1.2.1
	Storage: see #1.3
	UI Input: Events listened by the server such as clicks or submissions.
	Credentials: Secure user ID.
	Models: Architecture that describes the shape of an object in the database.
	HTTP Request: Authorized message with specific action and body.
	Response Entity: Package of requested data with a request status code.
	JSON: Lightweight text for storing and transporting data.
Referenced By	1.1 Web Interface
	1.1.1 Client Views
	1.1.1.1 Login
	1.1.1.2 Profile
	1.1.1.3 Schedule Designer
	1.1.1.4 Homepage
	1.2 Logic
	1.3 Storage
Viewpoint	Data Flow Diagram

1.1.2B Server



Name	**1.1.2 Server**
Purpose	Mediator between client and business logic.
Description	Converts an action from the web interface into an HTTP request if authorized, receives a response entity back, and serves an answer that the front end can interpret.
Requirements	1-5, 13, 21
Elements	setupEventListeners: Pays attention to any action in the web interface.
	getCoursesSelected: Returns a list of courses from a select HTML element.
	Document::getElementByID: Returns an HTML element given its ID.
	EventTarget::addEventListener: Pays attention to any action on a specific HTML element.
	printSchedule: Trigger the browser's print function.

Name	**1.1.2 Server**
	getGradPlan: Retrieves a graduation plan.
	getStudentProfile: Retrieves a student profile.
	setStudentProfile: Modifies a student profile.
	register: Send the order to register for all the given courses.
	getSchedules: Retrieves a set of optimized schedules
	ID: HTML element ID.
	Element: HTML Element.
	graduationPlan: A graduation plan in JSON format.
	studentProfile: A student profile in JSON format.
	coursesFromSchedule: Courses from an optimized schedule.
	userCourses: Courses chosen by the user.
	HTML document: HTML document.
Referenced By	1.1 Web Interface
	1.1.1 Client Views
	1.1.1.1 Login
	1.1.1.2 Profile
	1.1.1.3 Schedule Designer
	1.1.1.4 Homepage
	1.2 Logic
	1.3 Storage
Viewpoint	Structure Chart

1.1.2.1 JSON Schema

Schedule Schema

```
{
    "$id": "https://enrollease.com/schema/schedule/1.1",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "Schedule",
    "type": "object",
    "description": "A schedule object that contains a list of sections.",
    "properties": {
        "schedule_id": {
            "type": "integer",
            "description": "The unique identifier for the schedule."
        },
        "walk_distance": {
            "type": "integer",
            "minimum": 0,
            "description": "The distance in meters for walking to the
location."
        },
        "walk_time": {
            "type": "integer",
            "minimum": 0,
            "description": "The time in minutes for walking to the
location."
        },
        "sections": {
            "type" : "array",
            "items": { "$ref": "#/$defs/section" },
            "description": "An array of sections within the schedule."
        }
    },
    "required": ["schedule_id", "walk_distance", "walk_time", "sections"],
    "$defs": {
        "section": {
            "type": "object",
            "description": "A section object that contains information about
a course section.",
            "properties": {
                "section_id" : {
                    "type": "string",
                    "description": "The unique identifier for the section."
                },
                "course" : {
                    "type": "object",
```

```
                    "description": "The course object that the section
belongs to.",
                    "properties": {
                        "course_id" : {
                            "type": "string",
                            "description": "The unique identifier for the
course."
                        },
                        "course_title" : {
                            "type": "string",
                            "description": "The title of the course."
                        }
                    },
                    "required": ["course_id", "course_title"]
                },
                "location" : {
                    "type": "object",
                    "description": "The location object where the section is
located.",
                    "properties": {
                        "building" : {
                            "type": "string",
                            "description": "The building where the section
is located."
                        },
                        "room" : {
                            "type": "string",
                            "description": "The room number where the
section is located."
                        }
                    },
                    "required": ["building", "room"]
                },
                "professor" : {
                    "type": "object",
                    "description": "The professor object teaching the
section",
                    "properties": {
                        "professor_name" : {
                            "type": "string",
                            "description": "The name of the professor
teaching the section."
                        },
                        "professor_score" : {
                            "type": "float",
```

```
                            "description": "The score of the professor
teaching the section."
                        }
                    },
                    "required": ["professor_name", "professor_score"]
                },
                "start_time" : {
                    "type": "datetime",
                    "description": "The start time of the section."
                },
                "end_time" : {
                    "type": "datetime",
                    "description": "The end time of the section."
                },
                "seats_max" : {
                    "type": "integer",
                    "description": "The maximum number of seats available in
the section."
                },
                "seats_open" : {
                    "type": "integer",
                    "description": "The number of open seats in the
section."
                },
                "open" : {
                    "type": "boolean",
                    "description": "Indicates if the section is open for
enrollment."
                },
                "delivery_method" : {
                    "type": "object",
                    "properties": {
                        "data": {
                            "enum": ["in-person", "online", "hybrid"],
                            "description": "The delivery method of the
section."
                        }
                    },
                    "required": ["data"]
                },
                "status" : {
                    "type": "object",
                    "properties": {
                        "data": {
                            "enum": ["open", "closed", "waitlist"],
                            "description": "The status of the section."
```

```
                        }
                    },
                    "required": ["data"]
                }

            },
            "required": ["section_id", "course", "location", "professor",
"start_time", "end_time", "seats_max", "seats_open", "open",
"delivery_method", "status"]

        }
    }
}
```

User Schema

```
{
    "$id": "https://enrollease.com/schema/user/1.0",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "User",
    "type": "object",
    "description": "A user object that contains information about a user.",
    "properties": {
        "user_id": {
            "type": "integer",
            "description": "The unique identifier for the user."
        },
        "first_name": {
            "type": "string",
            "description": "The first name of the user."
        },
        "last_name": {
            "type": "string",
            "description": "The last name of the user."
        },
        "student_info": {
            "type": "object",
            "description": "Information about the student.",
            "properties": {
                "ID": {
                    "type": "integer",
                    "description": "The student's unique identifier."
                },
                "GPA": {
                    "type": "number",
                    "description": "The student's grade point average."
```

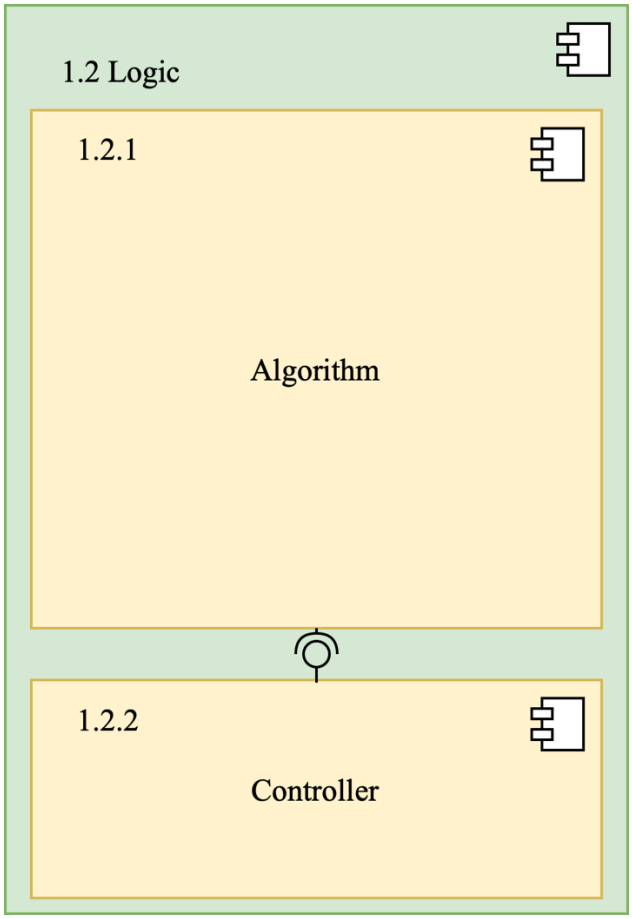
```
                },
                "gradDate": {
                    "type": "string",
                    "description": "The expected graduation date of the
student."
                },
                "major": {
                    "type": "string",
                    "description": "The major field of study of the
student."
                },
                "preferences": {
                    "$ref": "#/definitions/preferences"
                },
                "contactInfo": {
                    "$ref": "#/definitions/contact"
                }
            },
            "required": [
                "ID"
            ]
        }
    },
    "definitions": {
        "preferences": {
            "type": "object",
            "description": "The student's course and scheduling
preferences.",
            "properties": {
                "courseMethod": {
                    "type": "string",
                    "enum": [
                        "Online",
                        "In-person",
                        "Hybrid"
                    ],
                    "description": "The preferred method of course
delivery."
                },
                "walkTime": {
                    "type": "number",
                    "description": "The preferred walking time to class in
minutes."
                },
                "profRating": {
                    "type": "number",
```

```
                    "description": "The preferred minimum professor rating."
                },
                "timeOfDay": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "date-time",
                        "description": "Preferred class times."
                    },
                    "description": "The preferred times of day for classes."
                },
                "timeBetweenClass": {
                    "type": "integer",
                    "description": "The preferred time between classes in
minutes."
                }
            }
        },
        "contact": {
            "type": "object",
            "description": "The student's contact information.",
            "properties": {
                "phone": {
                    "type": "string",
                    "description": "The student's phone number."
                },
                "email": {
                    "type": "string",
                    "description": "The student's email address."
                }
            }
        }
    },
    "required": [
        "user_id"
    ]
}
```

Name	**1.1.2.1 JSON Schema**
Purpose	Validates that the data is encoded in JSON and is known for the system.
Description	Object and attribute declarations used by Server to prepare a JSON response for Client Views
Requirements	13, 17
Referenced By	1.1.1.2 Profile
	1.1.1.3 Schedule Designer
	1.1.1.4 Homepage
	1.1.2 Server
Viewpoint	JSON Schema

Logic

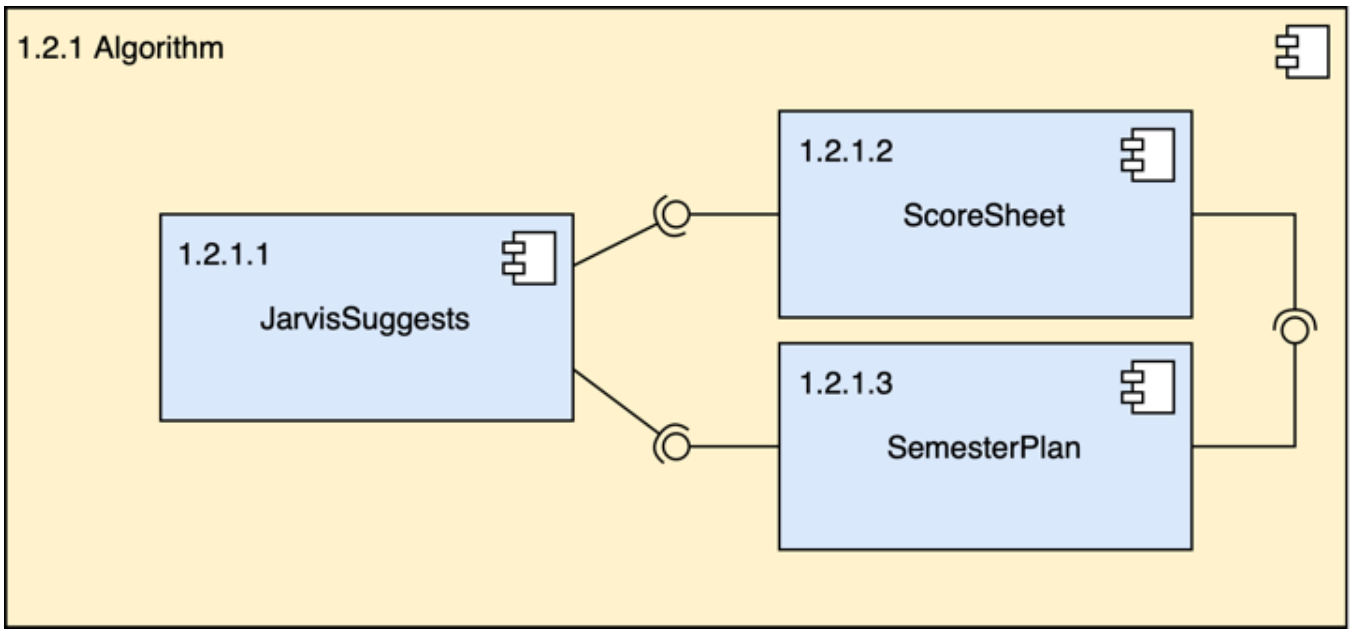
1.2 Logic



Name	1.2 Logic Component
Purpose	Back-facing element that is responsible for making decisions and validating output.
Description	It ensures that the software executes specific instructions or algorithms depending on the input or current state.
Requirements	Requirements 2, 9-16
Elements	1.2.1 Algorithm
	1.2.2 Controller
Referenced by	1.0 EnrollEase
Viewpoint	Flowchart

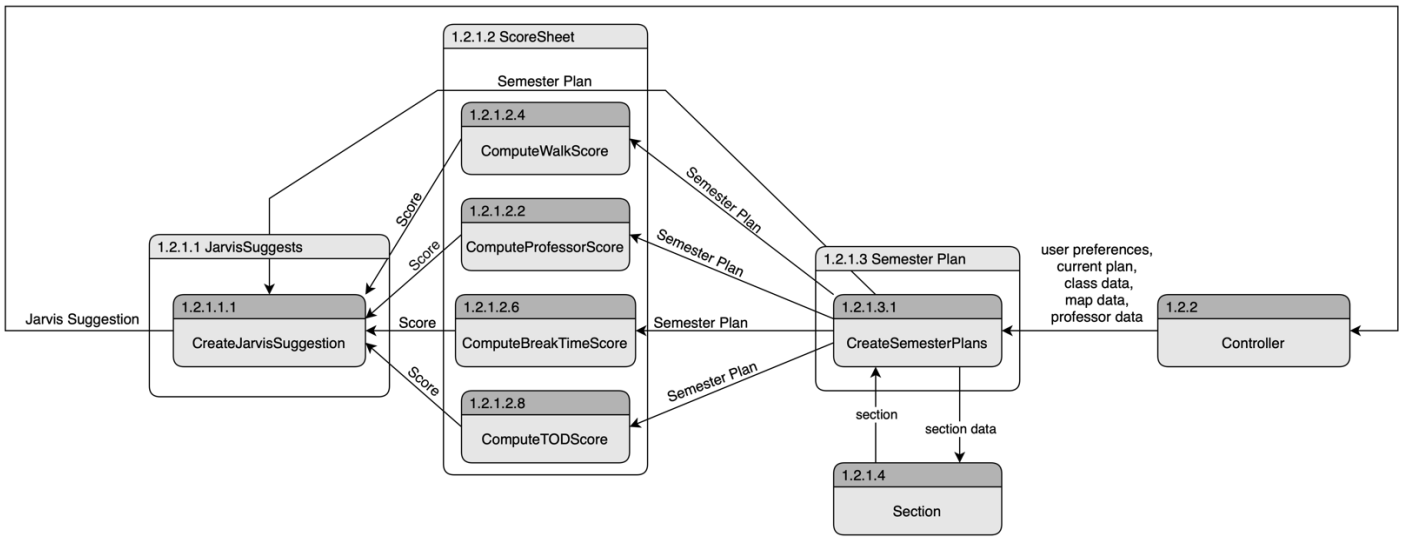
Algorithm

1.2.1A Algorithm



Name	1.2.1A Algorithm Component
Purpose	Minimize the time required for students to register for classes. Minimize walk-in distance from class to class. Empower students to generate customized schedules.
Description	This program presents the required classes for a given major, allows the students to place the required class in a semester plan, and sends the semester plan to the registration system. It also allows the student to auto-generate a plan considering the location of a class, the ratings of a professor, and other student-specified preferences.
Requirements	9-16
Elements	1.2.1.1 Jarvis Suggests
	1.2.1.2 Score Sheet
	1.2.1.3 Semester plan
Referenced by	1.2 Logic Component
Viewpoint	Component Diagram

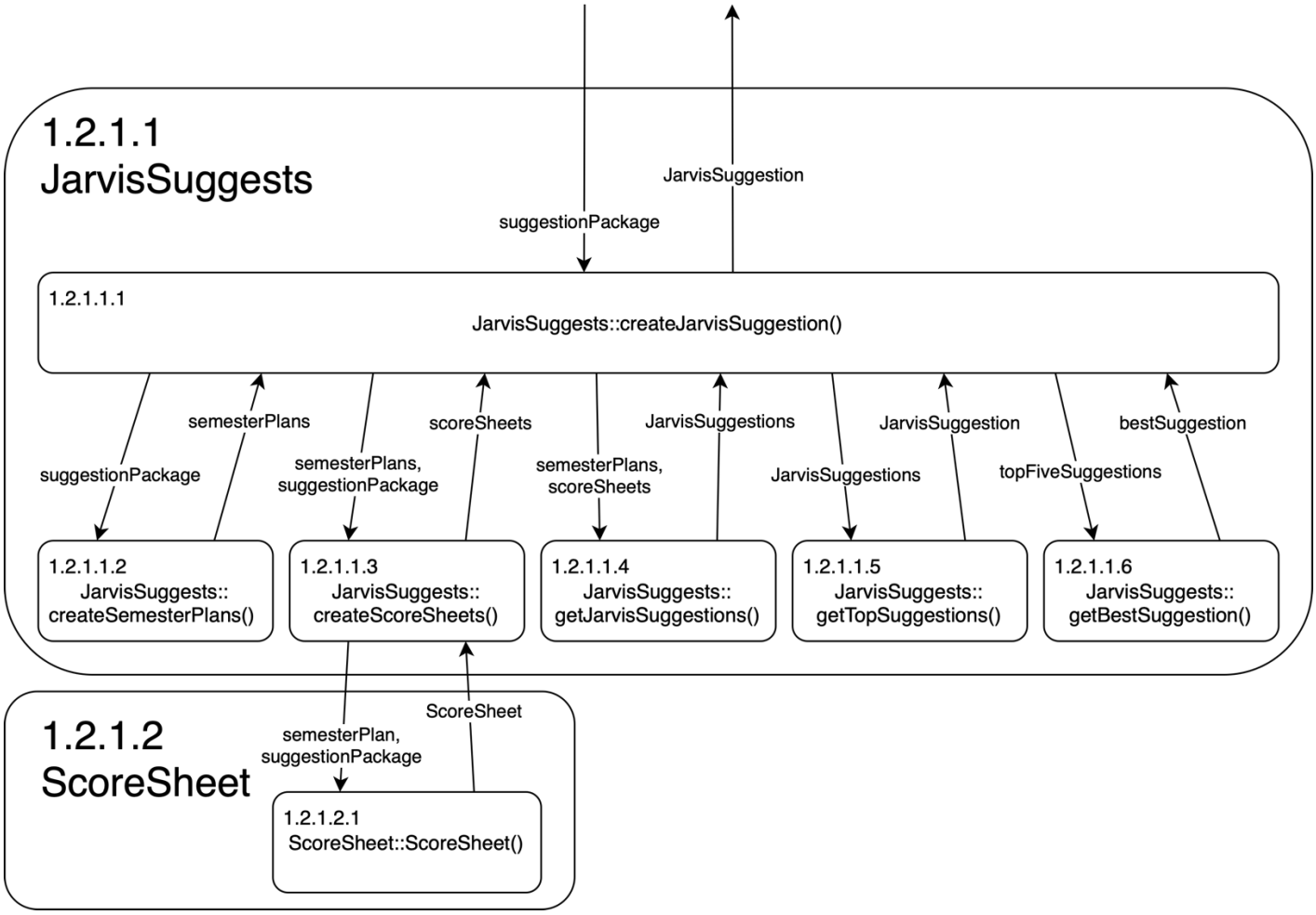
1.2.1B Algorithm



Name	1.2.1 Algorithm Component
Purpose	Creates and scores the semester plans that will be handed to the user.
Description	The logic that handles the creation, scoring, and suggestions of semester plans that will be delivered to the user consisting of 3 main components and their sub-components.
Requirements	Requirements 9-16
Elements	1.2.1.1 Jarvis Suggests: Suggests the plan with its attached scores including scores such as Professor Score, Walk Score, etc.
	1.2.1.2 Score Sheet: Collection of subcomponents that score a plan in a variety of ways.
	1.2.1.3 Semester Plan: Creates semester plans that align with in the closest way with the user's preferences while utilizing class data, distance data, professor data, and more.
	1.2.1.4 Section: Holds all the data related to a specific section of a class including professor, location, and time info.
	1.2.1.3.1 Create Semester Plans: Creates a collection of closely scheduled plans that closely align with user preferences.
	1.2.1.2.4 Compute Walk Score: Calculates a numerical value to represent how closely the plan aligns with a user's walking preferences.
	1.2.1.2.2 Compute Professor Score: Calculates a numerical value to represent the quality of professors represented in the plan.
	1.2.1.2.6 Compute Break Time Score: Calculates a numerical value to represent how closely the plan aligns with a user's break preferences.

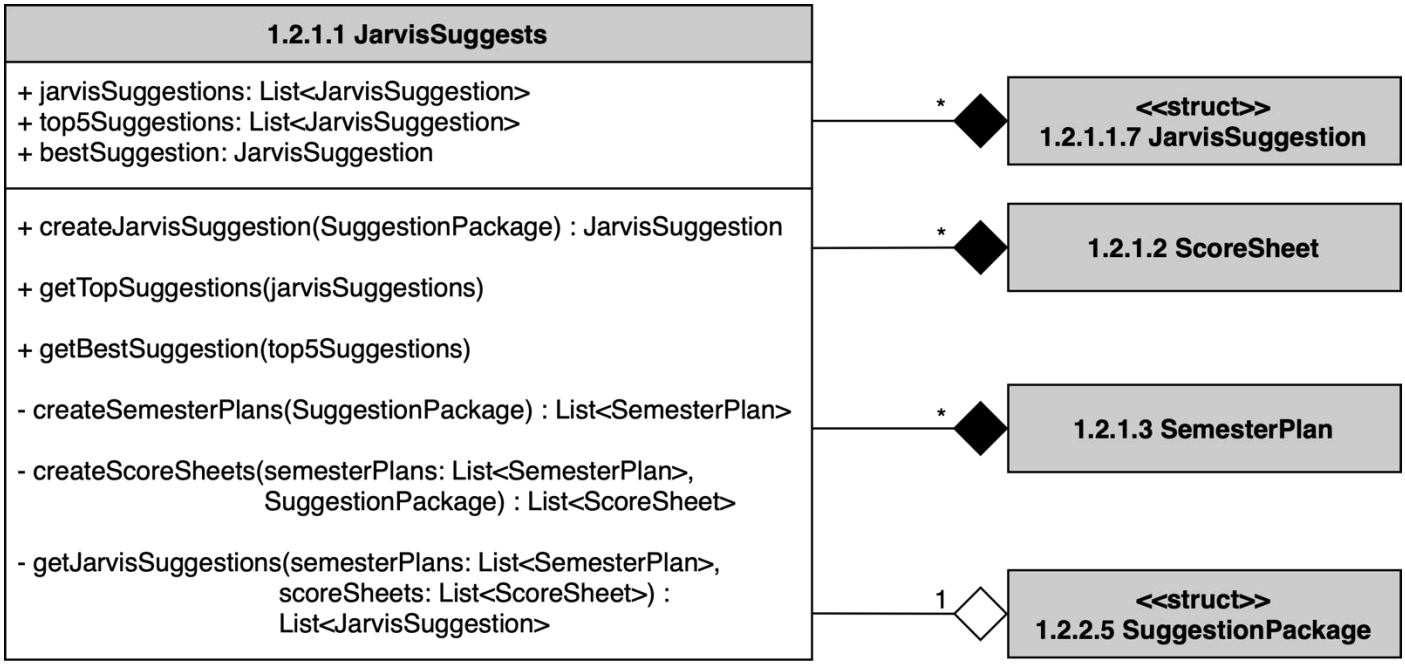
Name	**1.2.1 Algorithm Component**
	1.2.1.2.8 Compute TOD Score: Calculates a numerical value to represent how closely the plan aligns with a user's time of day preferences.
	1.2.1.1.1 Create Jarvis Suggestions: Combines the plan with its associated scores to create a Jarvis Suggestion (See 1.2.1.1.7).
	user preferences: The user's preferences for their schedule, including but not limited to preferences such as Time Of Day (TOD) preference.
	current plan: The user's current plan which includes data such as previously taken classes.
	class data: All of the data related to a class or section. Will contain data such as Time Of Day, Location, Professor, etc.
	map data: The data concerning the BYU-I map. This will allow distance between classes to be calculated.
	professor data: All of the data concerning a professor including name, department, rate my professor scores, etc.
	section data: All of the data related to a specific section such as professor, location, time info, etc.
	Semester Plan: A plan containing and compiling all of the data related to the created semester plan. This will include classes, professors, and more.
	Score: A score object holding all of the related scores and other metrics associated with the scoring of a semester plan.
	1.2.1.1.7 Jarvis Suggestion: A combination of the semester plan and it's associated scores that will be delivered to the user via the controller.
Referenced By	1.2 Logic Component
Viewpoint	UML Data Flow Diagram and Component Diagram

1.2.1.1 Jarvis Suggests



Name	**1.2.1.1 Jarvis Suggests**
Purpose	Suggests a semester plan that best aligns with the user's preferences and degree requirements.
Description	Takes the created semester plans and corresponding score sheets, creates an overall score and then after evaluating the best semester plan returns that semester plan.
Requirements	Requirements 15, 16
Elements	1.2.1.1.1 createJarvisSuggestion: Will return the semester plan coupled with the score sheet (JarvisSuggestion) that most closely matches the user's preferences.
	1.2.1.1.2 createSemesterPlans: Will take all the data from the suggestionPackage and will create and return a collection of all the possible semester plans.
	1.2.1.1.3 createScoreSheets: Will take the collection of possible semester plans and the suggestionPackage data to score the plans. This will return a collection of scoreSheets containing the scores.
	1.2.1.1.4 getJarvisSuggestions: Will take the collections of possible semesterPlans and scoreSheets and return a collection of SemesterPlan and ScoreSheet pairs called a JarvisSuggestion.
	1.2.1.1.5 getTopSuggestion: Will take the Suggestions and will return the top 5 best scoring suggestions in order of total score.
	1.2.1.1.6 getBestSuggestion: Will take the top 5 JarvisSuggestions and returns the best scoring one.
	suggestionPackage: A collection of data required to make a JarvisSuggestion.
	semesterPlans: A collection of SemesterPlan objects.
	scoreSheets: A collection of ScoreSheet objects.
	1.2.1.2 ScoreSheet: An object that holds all the scores for a semester plan.
	1.2.1.2.1 ScoreSheet()::ScoreSheet: The score sheet class constructor.
	1.2.1.1.7 JarvisSuggestion: A struct containing both the SemesterPlan and ScoreSheet. This is the best plan that will be presented to the user later.
Referenced by	1.2.1 Logic Component
Viewpoint	Structure Chart

1.2.1.1A Jarvis Suggests



Name	**1.2.1.1 Jarvis Suggests Class Diagram**
Purpose	Suggests a semester plan that best aligns with the user's preferences and degree requirments.
Description	Takes the created semester plans and correspoding score sheets, creates an overall score and then after evaluating the best semester plan returns that semester plan.
Requirements	Requirements 12, 13, 14
Elements	1.2.1.1.7 JarvisSuggestion: A pair of ScoreSheet and SemesterPlan that represents a suggestion.
	1.2.1.2 Score Sheet: Scores and holds all the scores associated with a plan.
	1.2.1.3 Semester Plan: Holds all the sections that make up a semester plan.
	1.2.2.5 SuggestionPackage: A package holding all the data needed to create plans and make suggestions.
	jarvisSuggestions: A collection of JarvisSuggestions.
	topFiveSuggestions: The top 5 scoring Jarvis Suggestions.
	bestSuggestion: The top scoring Jarvis Suggestion.
	1.2.1.1.1 createJarvisSuggestion(): A function that returns the JarvisSuggestion from a suggestion package.
	1.2.1.1.2 createSemesterPlans(): A function that creates semester plans from a suggestion package.
	1.2.1.1.3 createScoreSheets(): A function that creates score sheets for semester plans using the suggestion package.
	1.2.1.1.4 getJarvisSuggestions(): A function that combines score sheets and semester plans to create Jarvis Suggestions
	1.2.1.1.5 getTopSuggestions(): Takes the Jarvis Suggestions and returns the top 5 suggestions.
	1.2.1.1.6 getBestSuggestions(): Returns the best suggestion from the top 5 suggestions.
Referenced by	1.2.1 Logic Component
Viewpoint	Class Diagram

1.2.1.1.1 CreateSuggestion()

```
createJarvisSuggestion(suggestionPackage):

      semesterPlans <- createSemesterPlans(suggestionPackage)
      scoreSheets <- createScoreSheets(semesterPlans, suggestionPackage)

      suggestions <- getJarvisSuggestions(semesterPlans, scoreSheets)

      topFive <- getTopSuggestions(suggestions)

      RETURN getBestSuggestion(topFive)
```

Name	**1.2.1.1.1 createJarvisSuggestion**
Purpose	Creates a JarvisSuggestion from a suggestion package.
Description	Uses a suggestion package to create semester plans, score them, and then narrows them down before returning a JarvisSuggestion.
Requirements	Requirements 12, 13, 14
Elements	1.2.1.1.2 createSemesterPlans: Will take all the data from the suggestionPackage and will create and return a collection of all the possible semester plans.
	1.2.1.1.3 createScoreSheets: Will take the collection of possible semester plans and the suggestionPackage data to score the plans. This will return a collection of scoreSheets containing the scores.
	1.2.1.1.4 getJarvisSuggestions: Will take the collections of possible semesterPlans and scoreSheets and return a collection of SemesterPlan and ScoreSheet pairs called a JarvisSuggestion.
	1.2.1.1.5 getTopSuggestion: Will take the Suggestions and will return the top 5 best scoring suggestions in order of total score.
	1.2.1.1.6 getBestSuggestion: Will take the top 5 JarvisSuggestions and returns the best scoring one.
	suggestionPackage: A collection of data required to make a JarvisSuggestion.
	semesterPlans: A collection of SemesterPlan objects.
	scoreSheets: A collection of ScoreSheet objects.
	suggestions: A collection of JarvisSuggestion objects.
	topFive: The top five suggestions.
Referenced by	1.2.1.1 JarvisSuggests
Viewpoint	Pseudocode

1.2.1.1.2 ComputeOverallScore

```
createSemesterPlans(suggestionPackage)

      major <- suggestionPackage.userPreferences.major.major
      currentPlan <- suggestionPackage.currentPlan
      coursesTaken <- suggestionPackage.userPreferences.major.courseRequired

      neededCourseCodes <- getNewClassCodes(userMajor, coursesTaken,
suggestionPackage)

      combinations <- [[]]  // Initialize with an empty combination

      FOR course IN neededCourseCodes:
            new_combinations <- []

            FOR section IN course["sections"]:
                  FOR combination IN combinations:
                        new_combinations.append(combination + [section]) //
Combination: [cse430-1] Section: [cse251-2] Output: [cse430-1, cse251-2]

            combinations <- new_combinations

      FOR combination IN combinations:
            semesterPlans.append(semesterPlan(combination,
suggestionPackage))

      RETURN semesterPlans
```

Name	**1.2.1.1.2 createSemesterPlans**
Purpose	Creates all the possible semester plans that a user could take.
Description	Takes the suggestionPackage and retrieves the userData, userMajor, and courses taken and uses that data to calculate all possible plans.
Requirements	Requirements 11
Elements	1.2.1.3.2 getNewClassCodes: A function that gets the new classes that need taken based on what classes have already been taken.
	userData: The data related to a user.
	major: The user's major.
	coursesTaken: The courses a user has already taken.
	neededCourseCodes: The course codes that a user still has to take.
	combinations: The combinations of sections that could be a semester plan.
	section: A section of a class.
	new_combinations: A temporary place to store newly calculated combinations.
	1.2.1.3 SemesterPlan: A class holding the sections that make a semester plan.
	semesterPlans: A collection of semester plan objects.
Referenced by	1.2.1.1 Jarvis Suggests
Viewpoint	Pseudocode

1.2.1.1.3 EvaluateBestPlan

```
createScoreSheets(semesterPlans, suggestionPackage)

    scoreSheets <- []

    FOR semesterPlan IN semesterPlans
        newSheet <- ScoreSheet(semesterPlan, suggestionPackage)
        scoreSheets.append(newSheet)

    RETURN scoreSheets
```

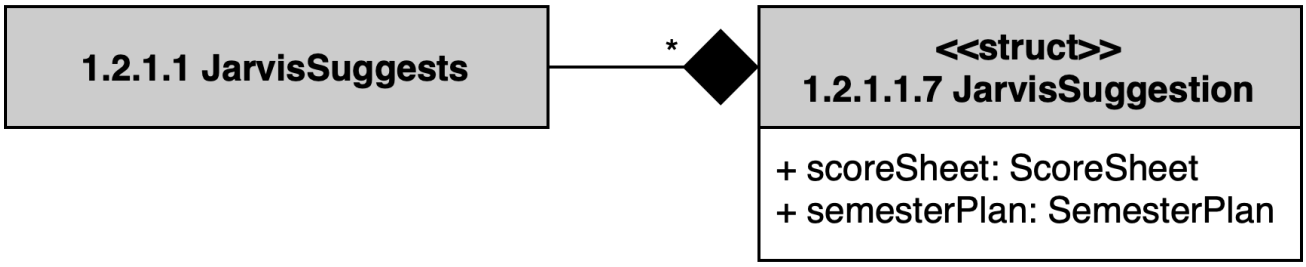
Name	**1.2.1.1.3 createScoreSheets**
Purpose	To create score sheets that based on a list of semester plans.
Description	A function that will create a score sheet for every semester plan it is given. This list of score sheets is returned.
Requirements	Requirement 11
Elements	semesterPlans: A collection of semesterPlans.
	1.2.2.5 suggestionPackage: A package of data containing everything needed to score a semester plan.
	scoreSheets: A collection of score sheets.
	1.2.1.3 semesterPlan: Hold all the data and sections that make up a user's semester plan.
	newSheet: A temporary score sheet before it is stored in the collection.
Referenced by	1.2.1.1 JarvisSuggests
Viewpoint	Pseudocode

1.2.1.1.4 getJarvisSuggestions

```
getJarvisSuggestions(semesterPlans, scoreSheets)

      jarvisSuggestions <- []

      FOR i_suggestion IN range(0, len(semesterPlans)):

            JarvisSuggestion suggestion

            suggestion.semesterPlan <- semesterPlans[i_suggestion]
            suggestion.scoreSheet <- scoreSheets[i_suggestion]

            jarvisSuggestions.append(suggestion)

      RETURN jarvisSuggestions
```

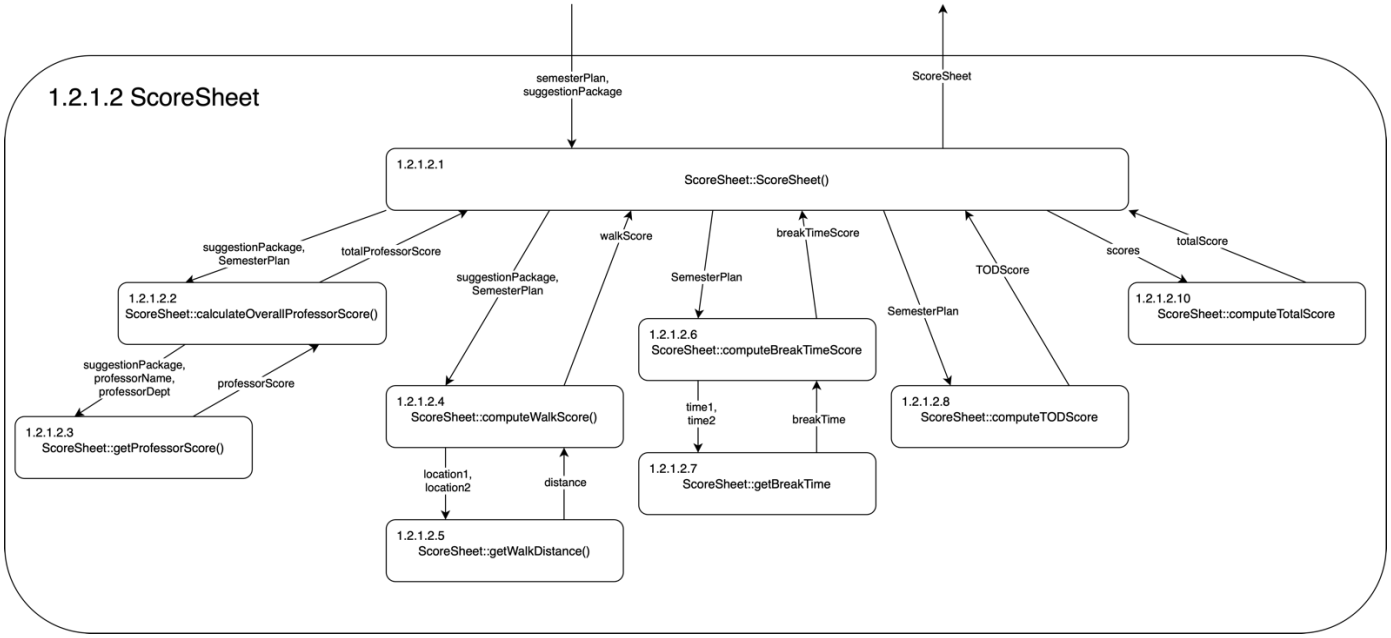
Name	**1.2.1.1.4 getJarvisSuggestions**
Purpose	Creates a list of jarvisSuggestions from lists of semester plans and score sheets.
Description	Function that combines matching scoreSheet and semesterPlan pairs into one suggestion.
Requirements	Requirements 12, 13
Elements	semesterPlans: A collection of semester plans.
	scoreSheets: A collection of score sheets.
	jarvisSuggestions: A collection of JarvisSuggestions.
	i_suggestion: The index of the current suggestion being made.
	1.2.1.1.7 JarvisSuggestion: Struct that hold a score sheet and it's semester plan.
	suggestion: The current suggestion being built.
Referenced by	1.2.1.1 JarvisSuggests
Viewpoint	Pseudocode

1.2.1.1.7 JarvisSuggestion



Name	**1.2.1.1.7 JarvisSuggestions**
Purpose	Holds the data that makes a Jarvis Suggestion.
Description	Structure containing a semester plan and a score sheet.
Requirements	Requirement 14
Elements	scoreSheet: The score sheet holding all the scores for the semester plan.
	semesterPlan: The plan containing all the information for the semester like section, professors, locations, etc.
Referenced by	1.2.1.1 JarvisSuggests
Viewpoint	Class Diagram

1.2.1.2 Score Sheet



Name	1.2.1.2 Score Sheet
Purpose	Scores and holds all the scores associated with a plan.
Description	Holds and calculates all the scores from a semester plan. Scores include walking score, professor score, break score, time of day (TOD) score.
Requirements	Requirements 10-13
Elements	1.2.1.2.1 ScoreSheet::ScoreSheet(): Constructor for the Score Sheet that initializes all the score computations.
	1.2.1.2.2 ScoreSheet::calculateOverallProfessorScore(): Takes a semester plan and related data to compute an overall professor score.
	1.2.1.2.4 ScoreSheet::computeWalkScore(): Takes a semester plan and relevant data to compute a score representing how well the plan aligns with a user's walking preference.
	1.2.1.2.5 ScoreSheet::getWalkDistance(): Returns a walk distance between two given locations.
	1.2.1.2.6 ScoreSheet::computeBreakTimeScore(): Takes a semester plan and computes an overall break time score that represents how closely break times match with the user's preference.

Name	**1.2.1.2 Score Sheet**
	1.2.1.2.7 ScoreSheet::getBreakTime(): Returns a break time between two given class objects.
	1.2.1.2.8 ScoreSheet::computeTODScore: Takes a semester plan and computes an overall TOD score that represents how closely the classes times of day match with the user's preference.
	1.2.1.2.10 ScoreSheet::computeTotalScore: Takes all of the previously calculated scores and returns a overall score that represents how well the plan aligns with all the user's preferences.
	suggestionPackage: A collection of data required to make a JarvisSuggestion(See 1.2.1.1.7) and subsequently SemesterPlans and ScoreSheets.
	SemesterPlan: An object holding all of data, such as classes, that makes up a semester plan.
	totalProfessorScore: The overall professor score for the semester plan.
	walkScore: A score representing how closely the SemesterPlan's walking distance aligns with the users walking preference.
	breakTime: The time between two classes.
	breakTimeScore: A score representing how closely a SemesterPlan's breaks line up with the users break preference.
	TODScore: A score representing how closely a SemesterPlan's Time of Day (TOD) aligns with the users TOD preference.
	scores: All the scores.
	totalScore: A score calculated from all previous scores that represents how well the plan aligns with all the user's preferences.
	ScoreSheet: An object consisting of all the calculated scores and the total scores of a SemesterPlan.
Referenced by	1.2.1 Algorithm
Viewpoint	UML Structure Chart

1.2.1.2A Score Sheet

1.2.1.2A ScoreSheet
- totalProfessorScore : int - walkScore : int - breakTimeScore : int - TODScore : int - totalScore : int
+ calculateOverallProfessorScore(suggestionPackage, semesterPlan) : totalProfessorScore + getProfessorScore(suggestionPackage, professorName, professorDept) : professorScore + computeWalkScore(suggestionPackage, semesterPlan) : walkScore + getWalkDistance(location1, location2) : distance + computeBreakTimeScore(semesterPlan) : breakTimeScore + getBreakTime(time1, time2) : breakTime + computeTODScore(semesterPlan) : TODScore + computeTotalScore()

Name	**1.2.1.2A Score Sheet Class**
Purpose	Scores and holds all the scores associated with a plan.
Description	Holds and calculates all the scores from a semester plan. Scores include walking score, professor score, break score, and time of day (TOD) score.
Requirements	Requirement 10-13
Elements	1.2.1.2A ScoreSheet: The class representing a score sheet.
	totalProfessorScore: An int that represents the total professor score.
	walkScore: An int that represents the walk score.
	breakTimeScore: An int that represents the break time score.
	TODScore: An int that represents the time-of-day score.
	totalScore: an Int that represents the overall score of the score sheet.
	calculateOverallProfessorScore(suggestionPackage, semesterPlan): Method that calculates the overall professor score.
	getProfessorScore(suggestionPackage, professorName, professorDept): Method that returns the professor score.
	computeWalkScore(suggestionPackage, semesterPlan): Method that computes the walk score.
	getWalkDistance(location1, location 2): Method that gets the walking distance between classes.
	computeBreakTimeScore(semesterPlan): Method that computes the time between classes.
	getBreakTime(time1, time2): Method that returns the break time between two times.
	computeTODScore(semesterPlan): Method that computes the time of day score based on user preference.
	computeTotalScore(): Method that takes all scores into consideration to create a total score.
Referenced By	1.2.1.2 Score Sheet
Viewpoint	UML Class Diagram

1.2.1.2.1 ScoreSheet::ScoreSheet()

```
ScoreSheet::ScoreSheet(semesterPlan, suggestionPackage)

     totalProfessorScore <-
calculateOverallProfessorScore(suggestionPackage, semesterPlan)
     walkScore <- computeWalkScore(suggestionPackage, semesterPlan)
     breakScore <- computeBreakScore(semesterPlan)
     TODScore <- computeTODScore(semesterPlan)

     totalScore <- computeTotalScore()
```

Name	**1.2.1.2.1 Score Sheet Constructor**
Purpose	Initializes all of ScoreSheets member values.
Description	Constructor that takes the semesterPlan and suggestionPackage to calculate and set all scores.
Requirements	7, 8, 9, 10, 11
Elements	1.2.1.2.2 calculateOverallProfessorScore(suggestionPackage, semesterPlan): Method that calculates the overall professor score.
	1.2.1.2.4 computeWalkScore(suggestionPackage, semesterPlan): Method that computes the walk score.
	1.2.1.2.6 computeBreakTimeScore(semesterPlan): Method that computes the time between classes.
	1.2.1.2.8 computeTODScore(semesterPlan): Method that computes the time of day score based on user preference.
	1.2.1.2.10 computeTotalScore(): Method that takes all scores into consideration to create a total score.
	totalProfessorScore: An int that represents the total professor score.
	walkScore: An int that represents the walk score.
	breakTimeScore: An int that represents the break time score.
	TODScore: An int that represents the time-of-day score.
	totalScore: an Int that represents the overall score of the score sheet.
Referenced By	1.2.1.2 Score Sheet
Viewpoint	Pseudocode

1.2.1.2.2 computeProfessorScore

```
calculateOverallProfessorScore()

   overallScore <- 0
   totalProfessors <- 0

   for professor in suggestionPackage.professorData
      overallScore += professor.rating
      totalProfessors += 1

   overallScore <- overallScore / totalProfessors

   return overallScore
```

Name	**1.2.1.2.2 calculateOverallProfessorScore**
Purpose	Represent the average score of all the professors in the semester plan.
Description	Function that takes all the professor ratings and averages them out to return a total professor score that can be used to compare with other semester plans.
Requirements	Requirements 10-13
Elements	overallScore: The average rating of all the professors.
	totalProfessors: A count of the number of professors on the semester plan.
	professorData: The name, department, and rating of each professor.
Referenced by	1.2.1.2 Score Sheet
Viewpoint	Pseudocode

1.2.1.2.4 computeWalkScore

```
computeWalkScore(SemesterPlan)

   walkScore <- 0

   for i_course in range(0, len(semesterPlan.sections) - 1)
      walkScore += getWalkDistance(semesterPlan.sections[i_course].location,
semesterPlan.sections[i_course + 1].location)

   walkScore <- walkScore / len(semesterPlan.sections)

   return walkScore
```

Name	**1.2.1.2.4 computeWalkScore**
Purpose	Returns a score based on how much walking a user will have to do between classes.
Description	Function takes all the class sections and then returns a score based on the average distance between each class. The lower the score the less walking.
Requirements	Requirements 10-13
Elements	walkScore: the score that will be returned based on the averaging walking distance between classes.
	1.2.1.2.5 getWalkDistance: Function that returns the distance between two buildings.
	i_course: Index for the loop that allows us to keep track of the sections we are tracking.
	semesterPlan: The current semester plan that is being scored.
Referenced by	1.2.1.2 Score Sheet, 1.2.1.2.1 ScoreSheetConstructor
Viewpoint	Pseudocode

1.2.1.2.5 getWalkDistance

```
getWalkDistance(suggestionPackage, location1, location2)

   for location in suggestionPackage.mapData.distance

      // Search for the first location and then return the distance that it
takes to get to the second location
      if location == location1
         return location[location2]

      // If the distance is not found then return null
      return null
```

Name	**1.2.1.2.5 getWalkDistance**
Purpose	Returns the distance between two buildings on campus
Description	Function takes two buildings on campus and then pulls the distance between the two that was returned in the suggestionsPackage.
Requirements	Requirements 10-13
Elements	suggestionPackage: A collection of data required to make a JarvisSuggestion.
	location1: The first building which the distance returned will be between this and the second building.
	location2: The second building which the distance returned will be between this and the first building.
	distance: A list of dictionaries that holds the distance between each building.
Referenced by	1.2.1.2 Score Sheet, 1.2.1.2.1 ScoreSheetConstructor
Viewpoint	Pseudocode

1.2.1.2.6 computeBreakScore

```
computeBreakTimeScore(semesterPlan)

   timeScore <- 0

   for i_course in range(0, len(semesterPlan.sections) - 1)
      timeScore += getBreakTime(semesterPlan.sections[i_course],
semesterPlan.sections[i_course + 1])

   timeScore <- timeScore / len(semesterPlan.sections)

   return timeScore
```

Name	**1.2.1.2.6 computeBreakTimeScore**
Purpose	Returns a score based on the break time between classes.
Description	Function that computes the average break time between classes. Returns that average so that the user can either prefer a lower score for shorter breaks or a higher score for longer breaks.
Requirements	Requirements 10-13
Elements	semesterPlan: The current semester plan that is being scored.
	i_course1: Index for the loop that allows us to keep track of the sections we are tracking.
	1.2.1.2.7 getBreakTime: Function that takes in two class sections and returns the break time between them.
	timeScore: The score based off the time between classes.
Referenced by	1.2.1.2 Score Sheet, 1.2.1.2.1 ScoreSheetConstructor
Viewpoint	Pseudocode

1.2.1.2.7 getBreakTime

```
getBreakTime(time1, time2)
    IF time1 > time2
        break = time2 – time1
    ELSE
        break = time1 – time2
    RETURN break
```

Name	**1.2.1.2.7 getBreakTime**
Purpose	getBreakTime tells how much time two different times is in between given
Description	The function receives two times and after figuring out which time is larger, subtracts the smaller time from the larger one.
Requirements	9
Elements	time1: the first time given to the function
	time2: the second time given to the function
	break: the difference between time1 and time2
Referenced by	1.2.1.2.6 computerBreakTimeScore, 1.2.1.2 scoreSheet
Viewpoint	Psuedocode

1.2.1.2.8 computeTODScore

```
computeTODScore(semesterPlan, suggestionPackage)
    IF suggestionPackage.userPreferences.TOD == “Morning”
        startTime = 8
        endTime = 12
    ELSE IF suggestionPackage.userPreferences.TOD == “Afternoon”
        startTime = 12
        endTime = 16
    ELSE IF suggestionPackage.userPreferences.TOD == “Evening”
        startTime = 16
        endTime = 20
    counter = 0 #a counter meant to count the number of times the for loop
occurs
    FOR course in semesterPlan.sections
        counter += 1
        IF course.startTime > startTime and course.startTime < endTime:
            TODScore += 100
        ELSE
            IF course.startTime < startTime
                TODScore += 100 – ( (startTime – course.startTime) * 10)
            ELSE IF course.startTime > endTime
                TODScore += 100 – ( (course.startTime – endTime) * 10)
    TODScore = TODScore / counter
    #The counter is used to take the total score and make it an Average.
    RETURN TODScore
```

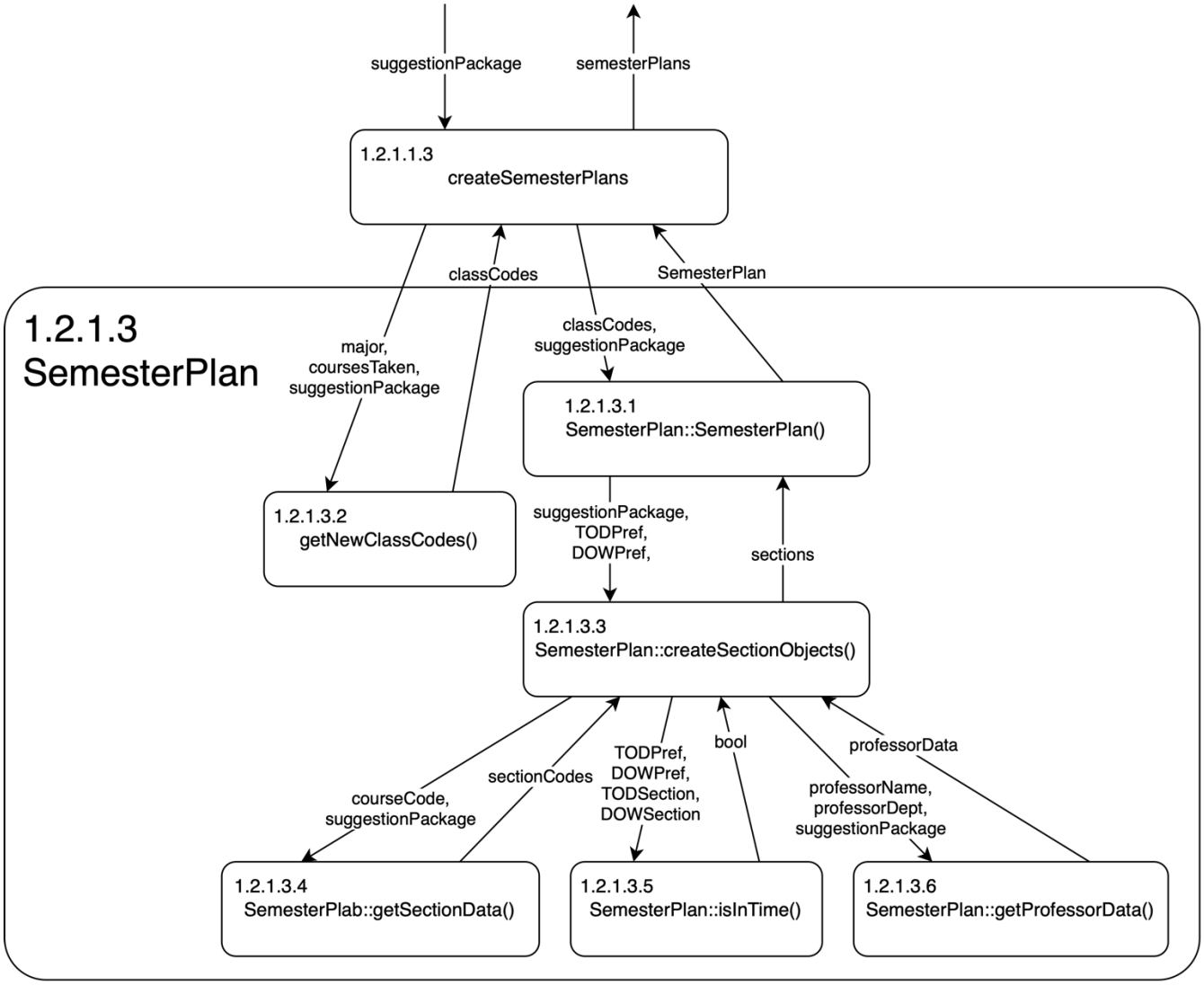
Name	**1.2.1.2.8 computeTODScore**
Purpose	computerTODScore is a function to give a semesterPlan a score based on its TOD (Time of Day)
Description	The function is given a semesterPlan and suggestionPackage to then score the semesterPlan based on the user's preferences in the suggestionPackage
Requirements	num. 10
Elements	semesterPlan: a series of courses placed in a list in this class.
	suggestionPackage: a group of preferences the user has made all placed into this one class.
	suggestionPackage.userPreferences: a class that has its own variables for different user made preferences.
	suggestionPackage.userPreferences.TOD: A string variable. It's the Time-of-Day preference inside userPrefences.
	semesterPlan.sections: a list of all the classes inside the semesterPlan
	course: a variable in the for loop that represents each element in the semesterPlan.sections
	startTime: the first time available in the users preferred times.
	endTime: the last time available in the users preferred times.
	counter: a counter meant to count the number of times the for loop occurs.
	TODScore: the variable stored inside the scoreSheet class
Referenced by	1.2.1.2 scoreSheet
Viewpoint	Pseudocode

1.2.1.2.10 ScoreSheet::computeTotalScore

```
computeTotalScore()
    totalScore = totalProfessorScore + walkScore + breakTimeScore + TODScore
    RETURNre totalScore
```

Name	1.2.1.2.10 computeTotalScore
Purpose	computeTotalScore is a function to give a semesterPlan a score based on every score. It is a total score, not an average score
Description	The function goes through every score and adds them into a single score
Requirements	11
Elements	totalScore: a combined score of every other score on the scoreSheet.
	totalProfessorScore: a score based on the professor
	walkScore: a score based on walking distance
	breakTimeScore: a score based on time between classes
	TODScore: score based on Time of Day
Referenced by	1.2.1.2 scoreSheet
Viewpoint	Pseudocode

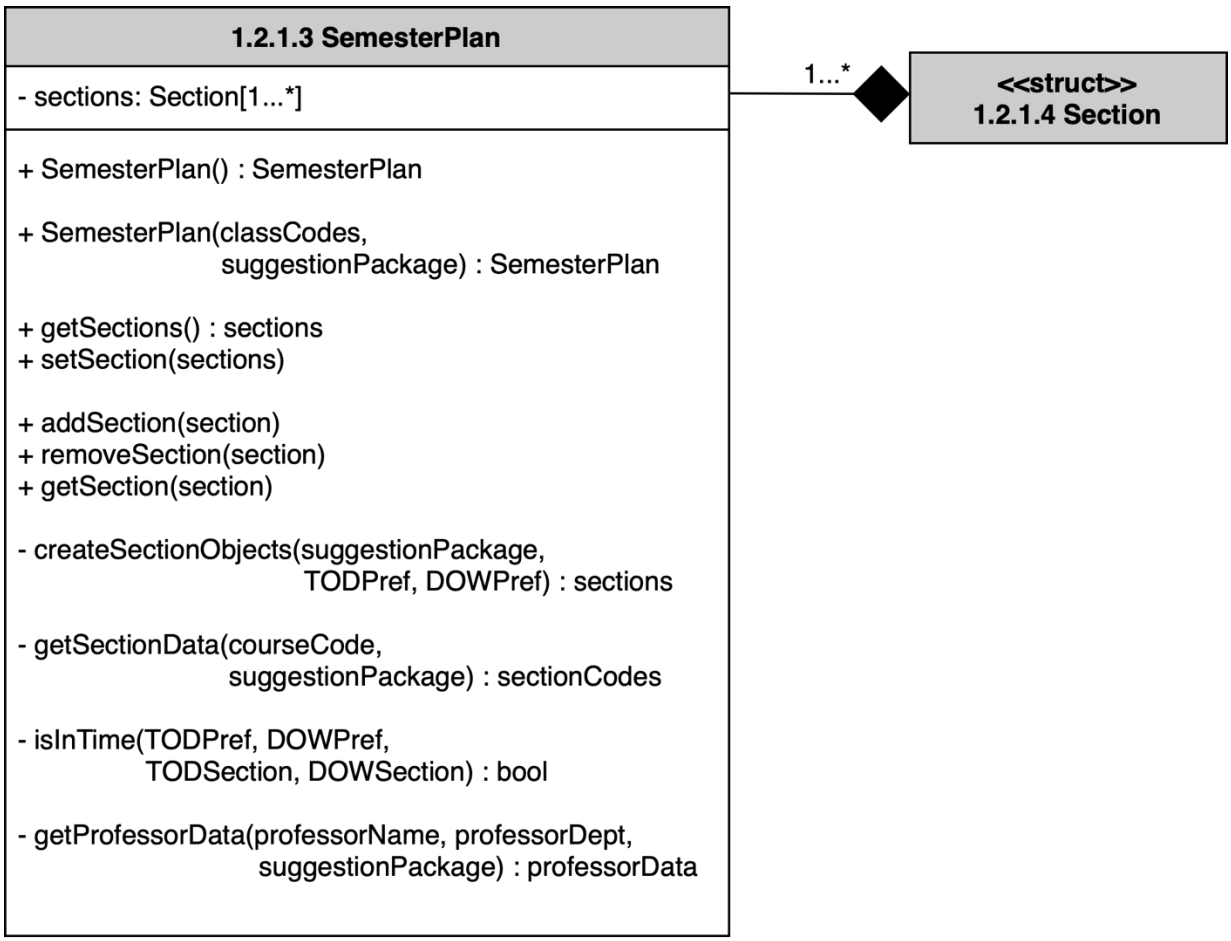
1.2.1.3 Semester Plan



Name	**1.2.1.3A Semester Plan**
Purpose	Creates and collects data for semester plans from a user's major and taken classes.
Description	Creates semester plans that align in the closest way with the user's Time of Day (TOD) and Day of Week (DOW) preferences while capturing class data, location data, professor data, and more.
Requirements	Requirements 9, 14, 16
Elements	1.2.1.3.1 SemesterPlan::SemesterPlan(): Constructor for SemesterPlan that creates and holds the classes that have been created from classCodes and the suggestionPackage.

Name	**1.2.1.3A Semester Plan**
	1.2.1.3.2 getNewClassCodes(): Function that will determine the classes that still need to be taken and return those course codes.
	1.2.1.3.3 SemesterPlan::createSectionObjects(): Function that creates the section objects that match Time of Day (TOD) and Day of Week (DOW) preferences. Returns a collection of sections.
	1.2.1.3.4 SemesterPlan::getSectionData(): Function that gathers sectionCodes from a class code and the suggestionPackage of data.
	1.2.1.3.5 SemesterPlan::isInTime(): Function that returns a bool representing whether or not a sections TOD and DOW match the user preferences.
	1.2.1.3.6 SemesterPlan::getProfessorData(): Function that takes a professor name and department to extract professor data from the suggestionPackage.
	suggestionPackage: A collection of data required to make a JarvisSuggestion(See 1.2.1.1.7) and subsequently SemesterPlans and ScoreSheets.
	major: The user's major that is used to find their required courses.
	coursesTaken: The courses that a user has already taken.
	classCodes: The codes representing a course. EX: CSE430.
	SemesterPlan: An object holding all of data, such as classes, that makes up a semester plan.
	TODPref: The users Time of Day preference.
	DOWPref: The users Day of Week preference.
	sections: A collection of section objects.
	sectionCodes: Codes representing a class section. EX: CSE430-1
	TODSection: The sections Time of Day. The time the section occurs.
	DOWSections: The sections Days of the Week. The days the section occurs.
	professorName: The name of the professor teaching.
	professorDept: The department the professor works in.
	professorData: The data related to a professor including ratings.
Referenced by	1.2.1 Logic Component
Viewpoint	Structure Chart

1.2.1.3B Semester Plan



Name	**1.2.1.3B Semester Plan Class**
Purpose	Holds and handles all of the data assigned to a course section.
Description	The class representing a section of a course. Holds data such as class time, location, professor and more.
Requirements	Requirement 13
Elements	1.2.1.3 SemesterPlan: The class representing a semester plan.
	1.2.1.4 Section: A struct holding section information.
	sections: A collection of section objects.
	SemesterPlan(): The default constructor.
	SemesterPlan(classCodes, suggestionPackage): The constructor that initializes the semester plan variables.
	getSections(): Trivial getter.
	setSection(sections): Trivial setter.
	addSection(sections): Method to add section to the semester plan.
	removeSection(section): Method to remove section from semester plan.
	getSection(section): Method to get a section from the semester plan.
	createSectionObjects(suggestionPackage, TODPref, DOWPref,): Method to create the sections that make up the semester plan.
	getSectionData(courseCode, suggestionPackage): Method that gets the section data required to create a section class.
	isInTime(): Simple method that determines if a section is in the user's preferred time range.
	getProfessorData(professorName, professorDept, suggestionPackage): Method to pull related professor data from the suggestionPackage
Referenced By	1.2.1 Logic Component & 1.2.1.3A Semester Plan
Viewpoint	Class Diagram

1.2.1.3.1 SemesterPlan::SemesterPlan()

```
SemesterPlan(courseCodes, SuggestionPackage)
    FOR code in courseCodes
        section = createSectionObjects(code, SuggestionPackage)
        sections.addItem(section)
```

Name	**1.2.1.3.1 semesterPlan Constructor**
Purpose	Creates the semesterPlan class when given parameters.
Description	This function receives the parameters and then adds everything into the class
Requirements	num. 6
Elements	courseCodes: a collection of courses codes that will be used to find courses.
	suggestionPackage: a group of preferences the user has made all placed into this one class.
	code: a variable in the for loop that represents each element in the courseCodes
	section: a class object that represents all the information of a course
	sections: a list of all the classes inside the semesterPlan
Referenced by	1.2.1.2 scoreSheet
Viewpoint	Pseudocode

1.2.1.3.2 getNewClassCodes()

```
     coursesRequired <-
suggestionPackage.userPreferences.major.coursesRequired
     FOR index in 0..coursesRequired.length
          course <- coursesRequired[index]
          IF course IN coursesTaken
               coursesRequired.remove(course)
     return coursesRequired
```

Name	**1.2.1.3.2 getNewClassCodes**
Purpose	Provide information on classes still to be taken.
Description	Function that will determine the classes that still need to be taken and return those course codes.
Requirements	Requirements 9, 14, 16
Elements	coursesRequired: a collection of required courses, associated with the major of the user.
	course: the current course being evaluated.
	coursesTaken: a parameter denoting what classes have been taken by the user.
	suggestionPackage: a collection of data necessary to suggest to the user.
Referenced by	1.2.1.3A SemesterPlan
Viewpoint	Pseudocode

1.2.1.3.3 SemesterPlan::createSectionObjects()

```
SemesterPlan::createSectionObjects(classCode, suggestionPackage)
      section <- Section
      result <- []
      TODPref <- suggestionPackage.userPreferences.TOD
      sectionCodes <- getSectionData(courseCode, suggestionPackage)
      WHILE NOT sectionCodes.isEmpty()
            section.courseName <-
suggestionPackage.classData.get(classCode).courseName
            section.department <-
suggestionPackage.classData.get(classCode).department
            section.courseCode <- classCode
            section.sectionNumber <- sectionCodes[0]
            professorData <-
getProfessorData(suggestionPackage.professorData)
            section.professorName <- professorData.name
            section.professorDept <- professorData.dept
            section.startTime <-
suggestionPackage.classData.get(classCode).startTime
            section.endTime <-
suggestionPackage.classData.get(classCode).endTime
            section.location <- suggestionPackage.mapData.get(classCode)
            sectionCodes.remove(0)
            IF startTime < 12
                  TODSection <- suggestionPackage.userPreferences.TOD.morning
            ELSE IF startTime < 17
                  TODSection <-
suggestionPackage.userPreferences.TOD.afternoon
            ELSE
                  TODSection <- suggestionPackage.userPreferences.TOD.evening
            IF isInTime(TODPref, TODSection)
                  result.add(section)
      RETURN result
```

Name	**1.2.1.3.3 createSectionObjects**
Purpose	Create Section Objects that match TOD.
Description	Function that creates the section objects that match Time of Day (TOD) preferences. Returns a collection of sections.
Requirements	9, 14, 16
Elements	section: a single section of a course being evaluated. See 1.2.1.4 Section.
	result: a collection of sections that meet user preferences for timing.
	TODPref: the user preference about time of day.
	sectionCodes: all the section codes for a particular course.
	professorData: necessary data pertaining to the professor of a course.
	TODSection: the time of day a particular section occurs.
	1.2.1.3.5 isInTime: a function that determines if a section meets preferences.
Referenced By	1.2.1.3A Semester Plan, 1.2.1.3B Semester Plan
Viewpoint	Pseudocode

1.2.1.3.4 SemesterPlan::getSectionData()

```
SemesterPlan::getSectionData(courseCode, suggestionPackage)
      sectionCodes <- []
      FOR EACH key IN suggestionPackage.classData
            IF key = courseCode

      sectionCodes.add(suggestionPackage.classData.get(key).sectionNumber)
      RETURN sectionCodes
```

Name	**1.2.1.3.4 getSectionData**
Purpose	Obtaining section codes for a particular class, indicated by class code.
Description	Function that gathers sectionCodes from a class code and the suggestionPackage of data.
Requirements	Requirements 9, 14, 16
Elements	courseCode: the code identifying a particular course.
	suggestionPackage: all necessary data acquired for suggesting to the user.
	sectionCodes: the collection of sections associated with a particular course. Returned by the function upon completion.
Referenced by	1.2.1.3A Semester Plan, 1.2.1.3B Semester Plan
Viewpoint	Pseudocode

1.2.1.3.5 SemesterPlan::isInTime()

```
SemesterPlan::isInTime(TODPref, TODSection)
      RETURN TODPref == TODSection
```

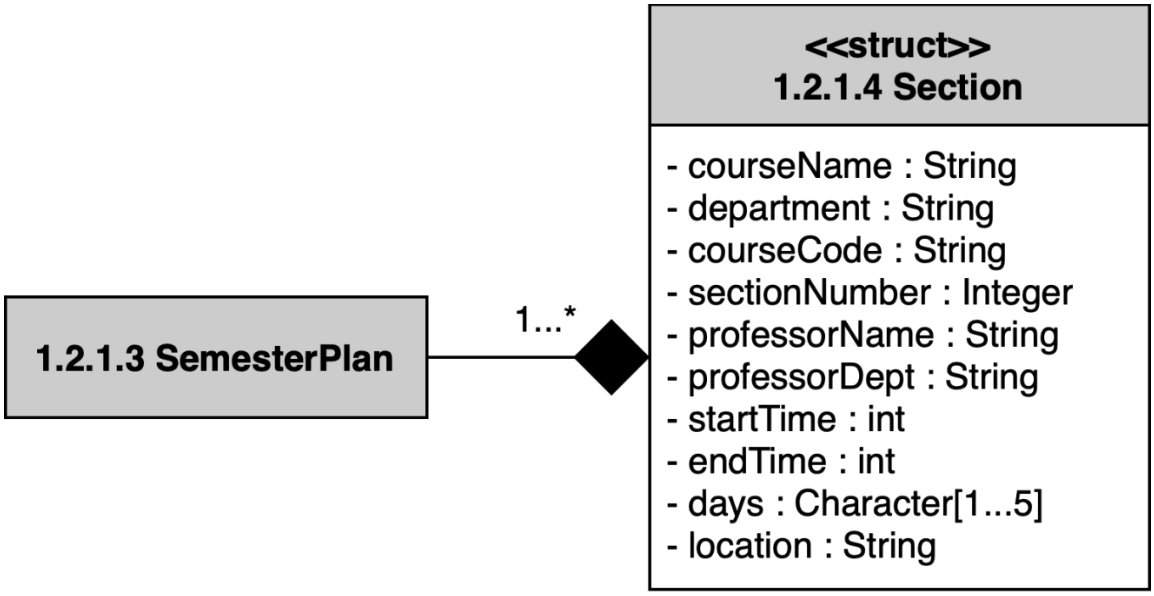
Name	**1.2.1.3.5 isInTime**
Purpose	Verify that a section meets user preferences for timing.
Description	Simple method that determines if a section is in the user's preferred time range.
Requirements	9, 14, 16
Elements	TODPref: the user preference concerning time of day for selected courses.
	TODSection: the time of day a specific section occurs.
Referenced by	1.2.1.3A Semester Plan, 1.2.1.3B Semester Plan, 1.2.1.3.3 createSectionObjects
Viewpoint	Pseudocode

1.2.1.3.6 SemesterPlan::getProfessorData()

```
getProfessorData(suggestionPackage, professorName, professorDept)
      allProfessorData <- suggestionPackage.professorData

      FOR professorData IN allProfessorData:
            IF professorData["professorName"] == professorName:
                  IF professorData["professorDept] == professorDept:
                        // Data found
                        RETURN professorData

      // Data not found
      RETURN NULL
```

Name	**1.2.1.3.6 getProfessorData**
Purpose	Retrieves the relevant professor data from the suggestion package.
Description	A function that takes a professor's name and department and uses that to find their data in the suggestion package.
Requirements	Requirement 8
Elements	1.2.2.5 suggestionPackage: A struct holding all the data needed to make and score a suggestion.
	professorName: The professor's name.
	professorDept: The professor's department at the school.
	professorData: The data relating to a specific professor.
Referenced by	1.2.1.3 Semester Plan
Viewpoint	Pseudocode

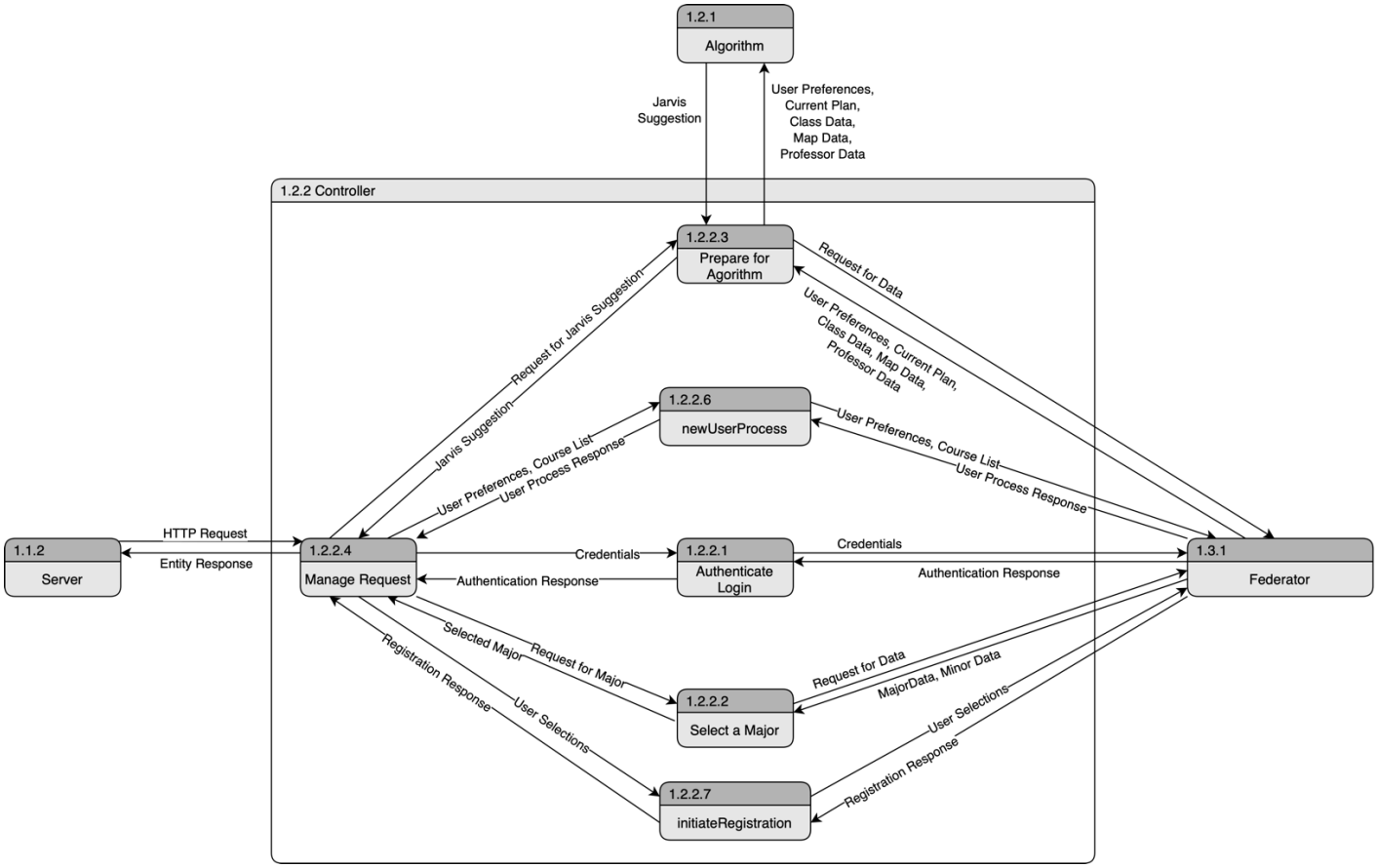
1.2.1.4 SemesterPlan



Name	**1.2.1.4 Section**
Purpose	Holds and handles all the data assigned to a user's semester plan.
Description	The class holding the semester plan sections. Will handle the creation and deletion of section objects within itself.
Requirements	Requirement 13
Elements	1.2.1.4 Section: The class representing a section of a course.
	courseName: The name of a course.
	department: The department a given course is a part of.
	courseCode: The code assigned to a give course. Example: CSE430.
	sectionNumber: The number representing which section of the course the section is.
	professorName: The name of the professor teaching the section.
	professorDept: The department the professor is a part of.
	startTime: The time in military time that the class starts.
	endTime: The time in military time that the class ends.
	days: The days of the week the class occurs on.
	location: The room and building the section takes.
Referenced By	1.2.1 Algorithm
Viewpoint	Class Diagram

Controller

1.2.2 Controller

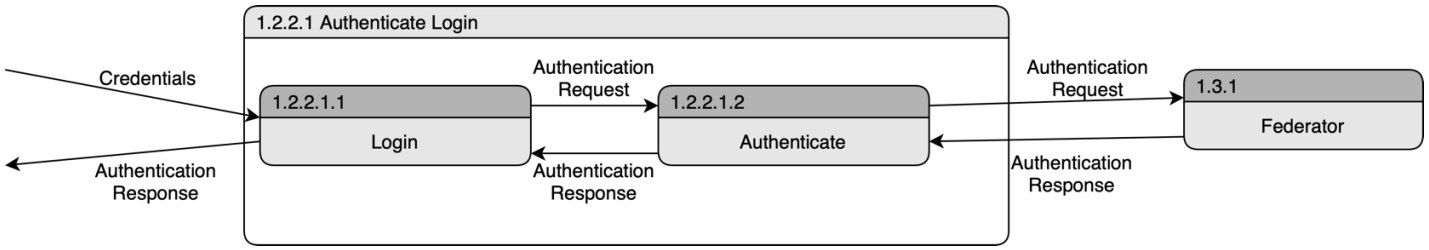


Name	1.2.2 Controller Component
Purpose	Tool for validating system state, initiating requests to appropriate connected components.

Name	**1.2.2 Controller Component**
Description	The coordinator, decision-maker, and manager of interactions between different components within the system.
Requirements	Requirements 2
Elements	1.2.2.1 Authenticate Login - Given user credentials, deliver an authentication response
	1.2.2.2 Select a Major - Deliver a selected major to the user based on data requested from Storage through 1.3.1 Federator
	1.2.2.3 Prepare for Algorithm - aggregate and send off the necessary data to receive a Jarvis Suggestion from 1.2.1 Algorithm
	1.2.2.4 Manage Request - Given an HTTP Request from 1.1.2 Server, route to appropriate utility, and deliver response
	1.2.2.6 newUserProcess - Function that sends user preferences and a course list to the Federator to create a new user.
	1.2.2.7 initiateRegistration - Function that takes user selections and routes them to the Federator to initiate registration.
	HTTP Request - Request from 1.1.2 Server
	Entity Response - Response to 1.1.2 Server
	Request for Jarvis Suggestion - Request to 1.2.2.3 Prepare for Algorithm
	1.2.1.1.7 Jarvis Suggestion - A suggested Plan for the User
	Credentials - Login Credentials
	Authentication Response - Success or Failure to Authenticate
	Request for Major - Request to 1.3.1 Federator for pertinent data
	Selected Major - The Declared Major of the User
	Request for Data - Data to be received through 1.3.1 Federator
	MajorData - Details of a given Major
	MinorData - Details of a given Minor
	User Preferences - Preferences within EnrollEase
	Current Plan - Currently Selected Graduation Plan
	Class Data - Details on a specific Class at BYUI
	Map Data - Sourced through 1.3.1 Federator and external services

Name	**1.2.2 Controller Component**
	Professor Data - Sourced through 1.3.1 Federator and RateMyProfessor, an external service
	User Selections - The user's selections that are necessary to initiate a registration.
	Registration Response - Success or Failure to register.
	User Preferences - The user's preferences that are required to make a new user.
	Course List - A list of the user's courses.
	User Process Response - Success or Failure to create a new user.
Referenced by	1.2 Logic Component
Viewpoint	UML Data Flow Diagram

1.2.2.1 Authenticate Login



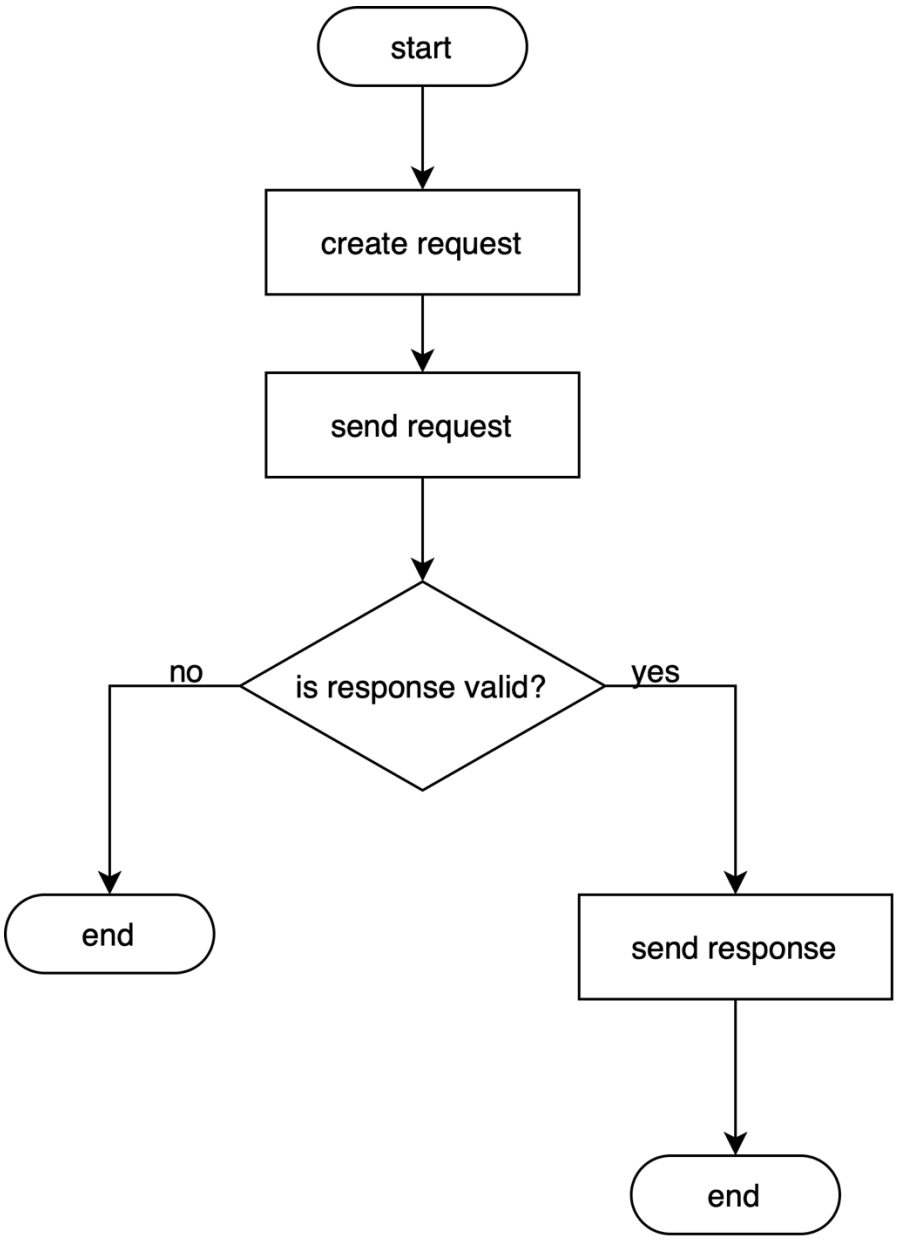
Name	**1.2.2.1 Login Authentication**
Purpose	To receive User Data, send it through the Authentication process, check credentials using Shibboleth. If credentials pass then it will send back a SSO token for the user, so it will sign them in.
Description	This diagram represents the flow of data during the login authentication process with single sign-on functionality which is Shibboleth.
Requirements	Requirement 17
Elements	1.2.2.1.1 Login Interface: The user provides their login credentials through the login interface.
	1.2.2.1.2 Authentication Process: The "Authentication Request" data flow (going to Shibboleth) carries the user's credentials to authenticate and through that to Shibboleth. The "Authentication Response" data flow (going back from Shibboleth) indicates whether the user's credentials were successfully authenticated.
	1.3.1 Federator
	UserData: The username and password just typed in.
	Authentication Request/Check: Going into Shibboleth with the Userdata
	Authentication Response: Going out of Shibboleth with the Userdata and SSO Token.
	SSO Token: Proof that the user is part of our system.
Referenced by	1.2.2 Controller
Viewpoint	Data Flow Diagram

1.2.2.1.1 Login Interface

```
login(credentials)

      IF authenticate(credentials)
            RETURN successfulAuthenticationResponse
      else
            RETURN failedAuthenticationResponse
```

Name	**1.2.2.1.1 Login**
Purpose	Receives credentials and returns a response indicating if authentication was successful.
Description	A simple component that calls authenticate and returns a response based on the authentication.
Requirements	Requirement 15
Elements	1.2.2.1.2 Authenticate: Simple component that retrieves an authentication and determines if it is valid.
	credentials: The user's login credentials.
	successfulAuthenticationResponse: A successful response that will be sent out.
	failedAuthenticationResponse: A failed response that will be sent out.
Referenced by	1.2.2.1 Login Authentication
Viewpoint	Pseudocode

1.2.2.1.2 Authentication Process



Name	**1.2.2.1.2 Authenticate**
Purpose	Receives authentication from Shibboleth through the storage federator and sends it to Login.
Description	A simple component that retrieves an authentication and determines if it is valid to be sent through logic.
Requirements	Requirement 15
Elements	start: The beginning of the decisions.
	response: The response received from storage.
	create request: Create a request to send to federator to receive response.
	send request: Send the request to the federator.
	send response: Pass the response to Logic.
	end: The end of the decisions.
Referenced by	1.2.2.1 Login Authentication
Viewpoint	Flow Chart

1.2.2.2 Select a Major

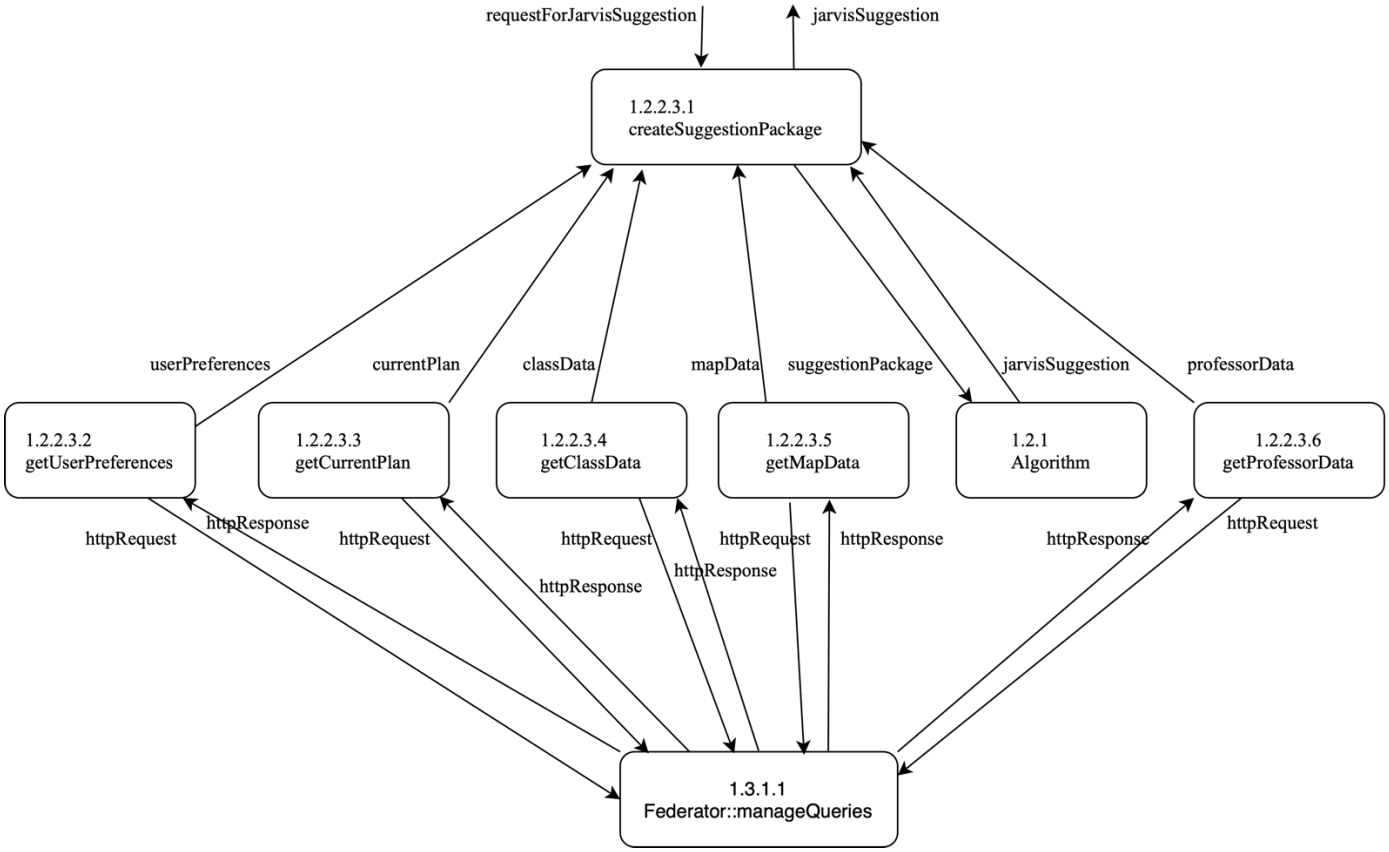
Name	1.2.2.2 Major Selection
Purpose	Provides information to the User based on the selected major and/or minor.
Description	Receives a requested major and/or minor, then searches through a provided database of majors and minors before returning information on the requested major and/or minor.
Requirements	2
Elements	1.2.2.2.1 Find Major and Minor
	1.2.2.2.3 Major/Minor Database
	1.1.2 Server
	1.3.1 Federator
Referenced by	1.2.2 Controller
Viewpoint	Data Flow Diagram

1.2.2.2.1 Find Major and Minor

```
findMajorAndMinor(major, minor, majorData, minorData)

   fullMajorData
   fullMinorData

   if major in majorData
      fullMajorData <- majorData[major]

   if minor in minorData
      fullMinorData <- minorData[minor}

   return fullMajorData, fullMinorData
```

Name	**1.2.2.2.1 findMajorAndMinor**
Purpose	Return information about the selected Major and Minor
Description	Receives the selected major and minor as well as information for all majors and minors and then searches for the indicated ones and returns the pertinent information.
Requirements	Requirement 2
Elements	fullMajorData: Information about the selected major.
	fullMinorData: Information about the selected minor.
	major: The major selected by the user.
	minor: The minor selected by the user.
	majorData: Data for all the offered majors.
	minorData: Data for all the offered minors.
Referenced by	1.2.2.2 MajorSelection
Viewpoint	Pseudocode

1.2.2.3 Prepare for Algorithm



Name	**1.2.2.3 Prepare For Algorithm**
Purpose	The utility to collect necessary data, send to 1.2.1 Algorithm, receive Jarvis Suggestion, and relay received Suggestion to 1.2.2.4 Manage Request.
Description	Utility within 1.2.2 Controller for sending valid data to 1.2.1 Algorithm, and relaying received suggestion.
Requirements	Requirement 6
Elements	1.2.2.3.1 createSuggestionsPackage: requests and aggregates necessary data for a suggestionPackage to be sent to the Algorithm
	1.2.2.3.2 getUserPreferences: retrieves the current userPreferences
	1.2.2.3.3 getCurrentPlan: retrieves the current user's plan
	1.2.2.3.4 getClassData: retrieves the necessary class data
	1.2.2.3.5 getMapData: retrieves the necessary map data
	1.2.2.3.6 getProfessorData: retrieves the necessary professor data
	requestForJarvisSuggestion: a request for a jarvisSuggestion (See 1.2.1.1.7)
	1.2.1.1.7 jarvisSuggestion: a suggested plan for the user
	userPreferences: preferences for scoring plans based on selected user criteria
	currentPlan: the current plan being acted on by the user
	classData: details on classes within the plan (section, time, etc)
	mapData: relevant details on locations for classes
	professorData: data retrieved on professor ratings, name, etc
	suggestionPackage: details necessary for 1.2.1 Algorithm to deliver a jarvisSuggestion. Consisting of userPreferences, currentPlan, classData, mapData, and professorData
	httpRequest: a request for a specific set of data, sent to 1.3.1.1 Federator::manageRequest
	httpResponse: the response provided by 1.3.1.1 Federator::manageRequest
	1.3.1.1 Federator::manageRequest: see 1.3.1.1
Referenced by	1.2.2 Controller Component
Viewpoint	Structure Chart

1.2.2.3.1 createSuggestionPackage

```
createSuggestionPackage()

   // The package is created.
   SuggestionPackage suggestionPackage

   suggestionPackage.userPreferences <- getUserPreferences()
   suggestionPackage.currentPlan <- getCurrentPlan()
   suggestionPackage.classData <- getClassData()
   suggestionPackage.mapData <- getMapData()
   suggestionPackage.professorData <- getProfessorData

   // The suggesstion is created from the package
   jarvisSuggests <- createJarvisSuggestion(suggestionPackage)

   // The suggesstion is sent back.
   RETURN JarvisSuggests
```

Name	**1.2.2.3.1 createSuggestionPackage**
Purpose	The function to create and pass a suggestion package to the logic to receive a Jarvis Suggestion.
Description	Recieves neccesary data to fill a suggestionPackage and then passes it to createJarvisSuggestion before returning the suggestion received.
Requirements	Requirements 6
Elements	1.2.2.5 suggestionPackage: The structure holding all the data needed for suggestion creation. Contains many different data types and data from user preferences to professor data.
	1.2.2.5.1 userPreferences: A structure containing values representing the user's preferences.
	currentPlan: A collection of semesterPlans that make up the user's previously taken plans.
	classData: All the data related to classes like sections, professors, locations, and more.
	mapData: All the data related to the campus map including things like building locations.
	professorData: All the data related to the professors including name, department, and ratings.
	1.2.1.1 jarvisSuggests: The suggestion object holding the best semester plan and its score sheet.
Referenced by	1.2.2.3 Prepare For Algorithm
Viewpoint	Pseudocode

1.2.2.3.2 getUserPreferences

```
getUserPreferences( )

      UserPreferences userPreferences

      raw_preferences <- federator.manageQueries(httpUserPrefRequest)

      userPreferences.classType <- raw_preferences["classType"]
      userPreferences.walkTime <- raw_preferences["walkTime"]
      userPreferences.professorScore <- raw_preferences["professorScore]
      userPreferences.TOD <- raw_preferences["TOD"]
      userPreferences.breakTime <- raw_preferences["breakTime"]
      userPreferences.major <- raw_preferences["major"]

      RETURN userPreferences
```

Name	**1.2.2.3.2 getUserPreferences**
Purpose	The function will return the userPreferences from the storage in a known format.
Description	Retrieves userPreferences from the storage and puts it into a UserPreferences struct before returning it.
Requirements	Requirements 6
Elements	1.2.2.5.1: A struct that holds the values associated with each user preference.
	1.3.1: federator: Component that facilitates data transfer between Logic and Storage.
	1.3.1.1 manageQueries: Method of federator that allows data to be requested from storage.
	httpUserPrefRequest: A http request sent to the federator that requests userPreference data.
	classType: The type of class the user prefers including online, virtual, in person.
	walkTime: The user's prefered time for walking between classes.
	professorScore: The user's preferred score for their professors to have.
	TOD: The user's preferred Time Of Day for their classes. Morning, Afternoon, or Evening.
	breakTime: The user's preferred time between classes and walking to classes.
	major: The user's major.
Referenced by	1.2.2.3 Prepare For Algorithm
Viewpoint	Pseudocode

1.2.2.3.3 getCurrentPlan

```
GetCurrentPlan()
   currentPlan <- Federator.manageQueries(planHTTP)

   SemesterPlan semesterPlan

   semesterPlan.section.sectionCode <- currentPlan.sectionCode

   return SemesterPlan
```

Name	**1.2.2.3.3 getCurrentPlan**
Purpose	The function to get the Users current plan.
Description	Calls the Federator to request and then pass on the Users current plan.
Requirements	Requirements 6
Elements	currentPlan: The Users current plan
Referenced by	1.2.2.3 Prepare For Algorithm
Viewpoint	Pseudocode

1.2.2.3.4 getClassData

```
GetClassData()
   classData <- Federator.manageQueries(classDataHTTP)
   return classData
```

Name	**1.2.2.3.4 getClassData**
Purpose	The function to get the necessary class data
Description	Calls the Federator, to request, and then pass on the relevant class data
Requirements	Requirements 6
Elements	classData: The relevant class data
Referenced by	1.2.2.3 Prepare For Algorithm
Viewpoint	Pseudocode

1.2.2.3.5 getMapData

```
getMapData()

    jsonMapdata <- federator.manageQueries(mapDataHttpRequest);

    dictionaryMapData <- jsonMapData

    RETURN dictionaryMapData
```

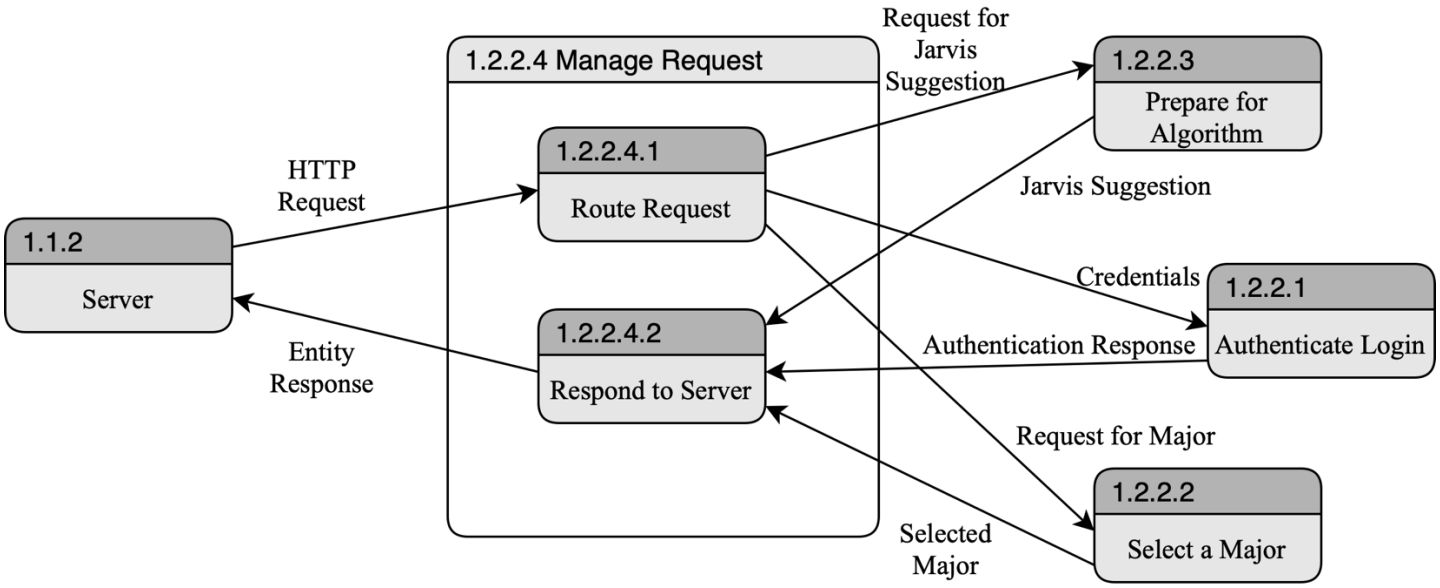
Name	**1.2.2.3.5 getMapData**
Purpose	Function to retrieve Map data from federator and return it in a known format.
Description	This function will request the map data from storage, covert it from JSON to a dictionary, and then returns the dictionary.
Requirements	Requirement 7
Elements	jsonMapData: The map data received from federator.
	1.3.1 federator: Component that facilitates data transfer between Logic and Storage.
	1.3.1.1 manageQueries: Method of federator that allows data to be requested from storage.
	mapDataHttpRequest: The http request formatted to request map data from the federator.
	dictionaryMapData: The map data after being converted to a dictionary.
Referenced By	1.2.2.3 Prepare For Algorithm
Viewpoint	Pseudocode

1.2.2.3.6 getProfessorData

```
createSuggestionPackage()

   // The package is created.
   SuggestionPackage suggestionPackage

   suggestionPackage.userPreferences <- getUserPreferences()
   suggestionPackage.currentPlan <- getCurrentPlan()
   suggestionPackage.classData <- getClassData()
   suggestionPackage.mapData <- getMapData()
   suggestionPackage.professorData <- getProfessorData

   // The suggesstion is created from the package
   jarvisSuggests <- createJarvisSuggestion(suggestionPackage)

   // The suggesstion is sent back.
   RETURN JarvisSuggests
```

Name	**1.2.2.3.1 createSuggestionPackage**
Purpose	The function to create and pass a suggestion package to the logic to receive a Jarvis Suggestion.
Description	Receives necessary data to fill a suggestionPackage and then passes it to createJarvisSuggestion before returning the suggestion received.
Requirements	Requirements 6
Elements	1.2.2.5 suggestionPackage: The structure holding all the data needed for suggestion creation. Contains many different data types and data from user preferences to professor data.
	1.2.2.5.1 userPreferences: A structure containing values representing the user's preferences.
	currentPlan: A collection of semesterPlans that make up the user's previously taken plans.
	classData: All the data related to classes like sections, professors, locations, and more.
	mapData: All the data related to the campus map including things like building locations.
	professorData: All the data related to the professors including name, department, and ratings.
	1.2.1.1 jarvisSuggests: The suggestion object holding the best semester plan and its score sheet.
Referenced by	1.2.2.3 Prepare For Algorithm
Viewpoint	Pseudocode

1.2.2.4 Manage Request



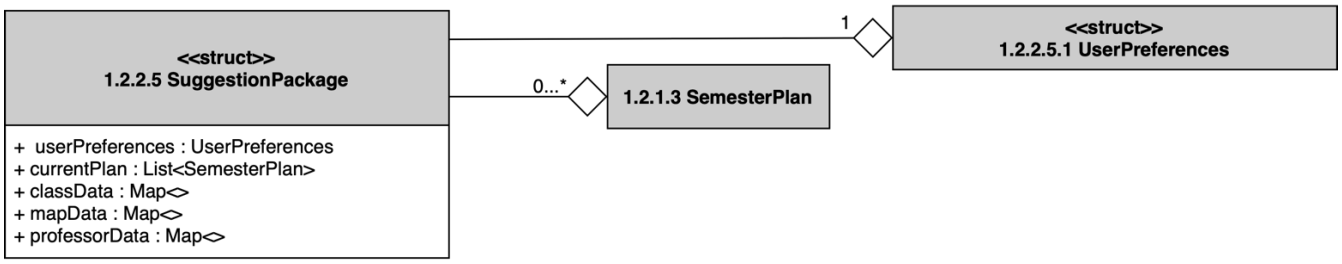
Name	**1.2.2.4 Manage Request Component**
Purpose	Used to handle routing and responding to requests received from 1.1.2 Server
Description	Manager of Requests entering the Controller and Responses exiting the Controller
Requirements	Requirements 2, 4
Elements	1.2.2.4.1 Route Request - Delivers HTTP Request to appropriate utility
	1.2.2.4.2 Respond to Server - Delivers Entity Response to 1.1.2 Server
	HTTP Request - Request from 1.1.2 Server
	Entity Response - Response to 1.1.2 Server
	Request for Jarvis Suggestion - Request to 1.2.2.3 Prepare for Algorithm
	1.2.1.1.7 Jarvis Suggestion - A suggested Plan for the User
	Credentials - Login Credentials
	Authentication Response - Success or Failure to Authenticate
	Request for Major - Request to 1.2.2.2 Select a Major for pertinent data
	Selected Major - The Declared Major of the User
Referenced By	1.2.2 Controller Component
Viewpoint	Data Flow Diagram

1.2.2.4.1 Manage Request

```
manageRequest()
   authentication <- authenticateLogin()
   IF (authentication)
      jarvisSuggestion <- prepareForAlgorithm()
      major <- selectAMajor()
      return jarvisSuggestion, major
```

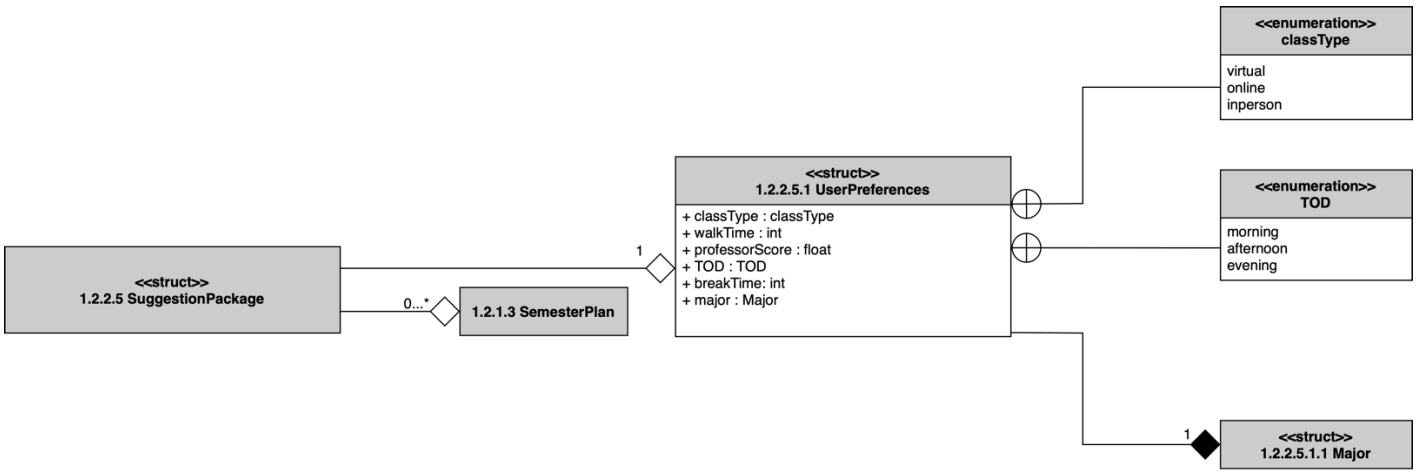
Name	**1.2.2.4.1 manageRequest**
Purpose	Used to handle routing and responding to requests received from 1.1.2 Server.
Description	Manager of Requests entering the Controller and Responses exiting the Controller
Requirements	Requirements 2, 4
Elements	N/A
Referenced by	1.2.2 Controller Component
Viewpoint	Pseudocode

1.2.2.5 Suggestion Package



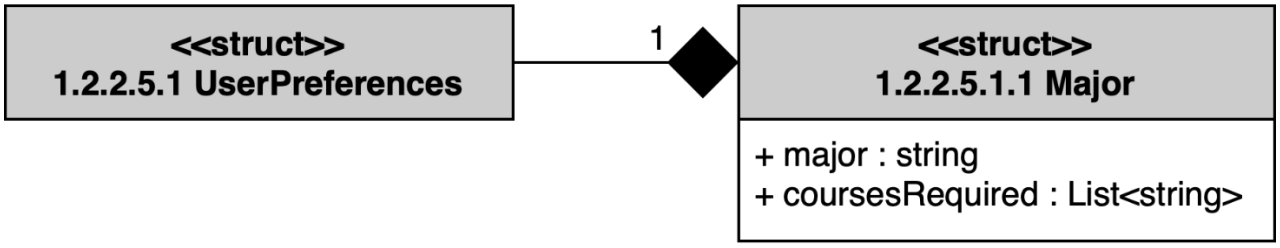
Name	**1.2.2.5 Suggestion Package**
Purpose	Holds and handles all the data needed to create a suggestion.
Description	The structure holding all the data needed for suggestion creation. Contains many different data types and data from user preferences to professor data.
Requirements	Requirement 13
Elements	1.2.2.5.1 UserPreferences: A structure containing values representing the user's preferences.
	1.2.1.3 SemesterPlan: A class holding all the data regarding a semester of classes, including course codes and more.
	currentPlan: A collection of previous SemesterPlans that the user has already completed.
	classData: A collection containing all the class data received from Storage.
	mapData: A collection containing all the data regarding the campus map received from Storage.
	professorData: A collection containing all the professor data received from RateMyProfessor.
Referenced By	1.2.2 Controller
Viewpoint	Class Diagram

1.2.2.5.1 User Preferences



Name	**1.2.2.5.1 User Preferences**
Purpose	Holds values related to the user's preferences in an easy to access way.
Description	A structure containing values representing the user's preferences.
Requirements	Requirement 13
Elements	classType: An enumeration representing the three types of class types.
	TOD: An enumeration representing the 3 Times Of the Day (TOD).
	walkTime: The users prefered time gap to be able to walk between classes.
	professorScore: The user's prefered score for a professor to have.
	breakTime: The user's prefered time gap between the end of a class and the start of the walk to another. (breakTime = timeBetweenClasses - walkTime).
	1.2.2.5.1 Major: A struct holding the user's major and courses required.
Referenced By	1.2.2.5 Controller
Viewpoint	UML Class Diagram

1.2.2.5.1.1 Major



Name	**1.2.2.5.1.1 Major**
Purpose	Holds the data related to a user's major.
Description	A structure that contains the users major as well as the courses required for that major.
Requirements	Requirement 6
Elements	major: A string representing the user's major.
	coursesRequired: A list of strings representing the course codes a major requires.
Referenced By	1.2.2.5.1 User Preferences
	1.2.1.1.2 createSemesterPlans
Viewpoint	Class Diagram

1.2.2.6 newUserProcess

```
newUserProcess(userPreferences, courseList)

   processSchema {
      "$schema": "http://json-schema.org/draft-07/schema#",
      "type": "object",
      "properties": {
        "userPreferences": {
          "type": "object",
          "properties": {
            "classType": { "type": "string" },
            "walkTime": { "type": "string" },
            "professorScore": { "type": "integer" },
            "TOD": { "type": "string" },
            "breakTime": { "type": "string" },
            "major": { "type": "string" }
          },
          "required": ["classType", "walkTime", "professorScore", "TOD",
"breakTime", "major"]
        },
        "courseList": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "courseName": { "type": "string" },
              "courseCode": { "type": "string" },
              "completed": { "type": "string", "enum": ["pass", "fail"] }
            },
            "required": ["courseName", "courseCode", "completed"]
           }
         }
       }
     }

   return federator.manageQueries(processSchema)
```

Name	1.2.2.6 New User Process Pseudocode
Purpose	Used to transmit relevant information to the federator and then returns a responding indicating success or failure.
Description	Sends the userPreferences and courseList in JSON to the federator and then returns whether the request was successful or failed.

Name	**1.2.2.6 New User Process Pseudocode**
Requirements	Requirements 2, 4
Elements	1.3.1: federator: Component that facilitates data transfer between Logic and Storage.
	1.3.1.1 manageQueries: Method of federator that allows data to be requested from storage.
	userPreferences: preferences for scoring plans based on selected user criteria
	courseList: list of the courses the user has taken which includes the Code, Name, and Completion
	processSchema: The JSON Schema that holds the userPreferences and courseList to be sent to the federator.
Referenced By	1.2.2 Controller Component
Viewpoint	Pseudocode

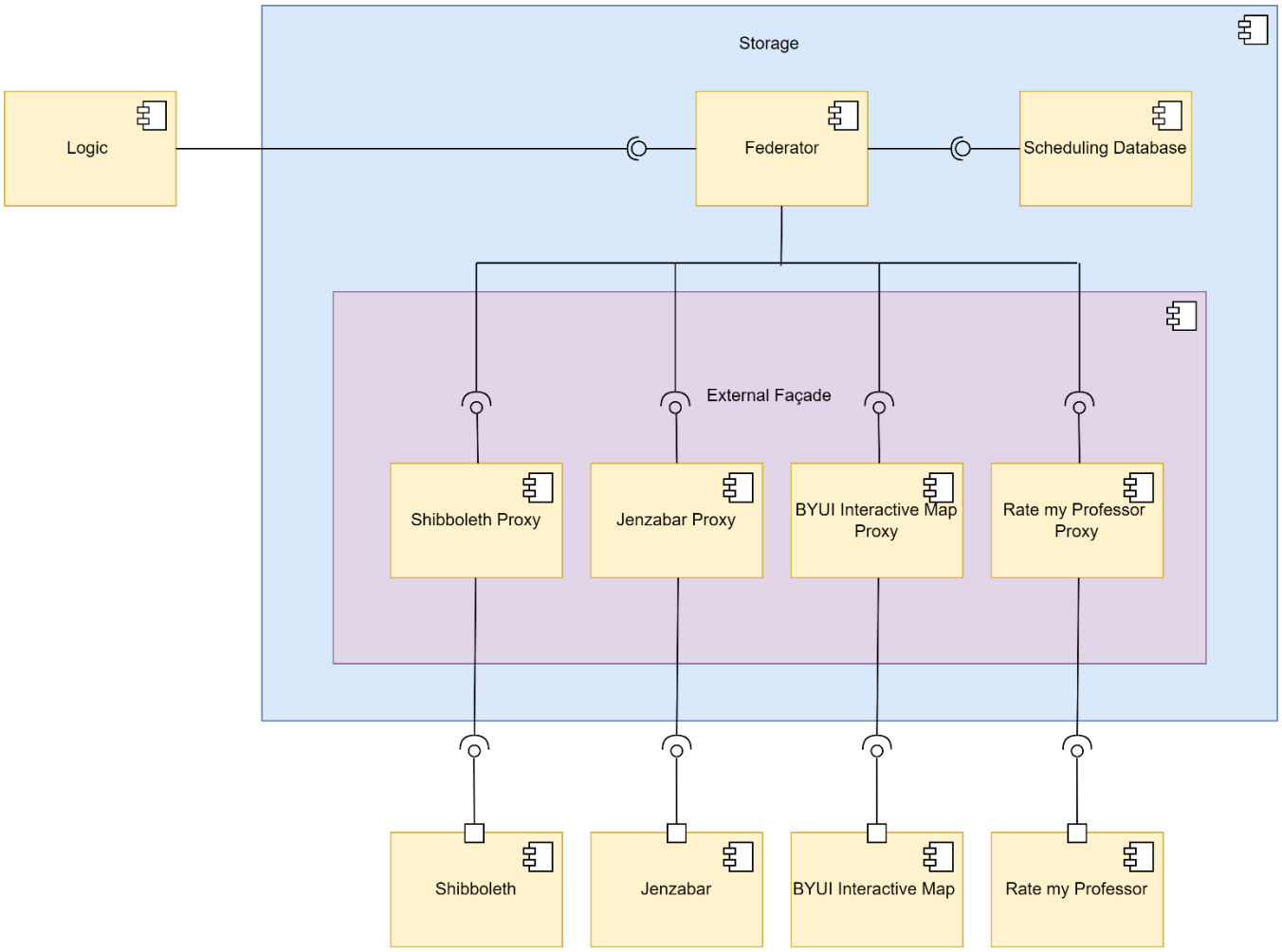
1.2.2.7 initiateRegistration

```
initiateRegistration(userSelections):
    response = federator.manageQueries(userSelections)
    return response
```

Name	**1.2.2.7 initiateRegistration**
Purpose	initiateRegistration is given the purpose of sending the userSelections it receives to federator for the classes to be registered for.
Description	The function receives a series of selected courses the user chose before sending them to federator. After it gets the return value from federator it returns that value to whoever called initiateRegistration.
Requirements	num. 5
Elements	usersSelection: The user's selections that are necessary to initiate a registration.
	response: the value received from the federator class function manageQueries
Referenced by	1.2.2 controller
Viewpoint	Pseudocode

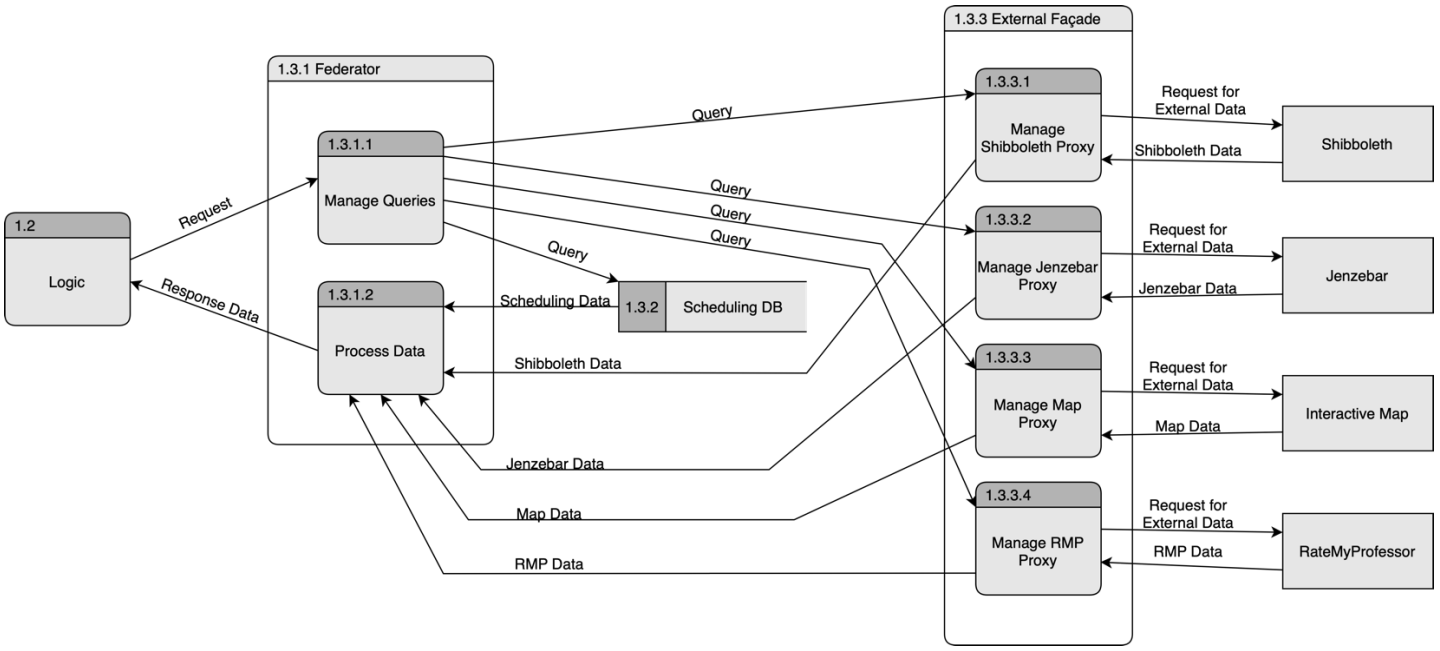
Storage

1.3A Storage



Name	1.3A Storage
Purpose	Illustrate the data flow between the various databases and how they interact with the logic component.
Description	Requests come from the logic component and go to the Federator, which then communicates with the external and scheduling databases, returning a response to the logic component.
Requirements	3-7
Elements	1.3.1 Federator
	1.3.2 Scheduling Database
	1.3.3 External Facade
	1.3.3.1 Shibboleth Proxy
	1.3.3.2 Jenzabar Proxy
	1.3.3.3 Interactive Map Proxy
	1.3.3.4 RateMyProfessor Proxy
	Shibboleth: The external service we go to for authentication.
	Jenzabar: The Universities Registration system.
	Interactive Map: A map of the school and the classes that the student can interact and use.
	RateMyProfessor: Tells the student about the professors and how well liked they are based off old students.
Referenced by	1.0 EnrollEase
Viewpoint	Component Diagram

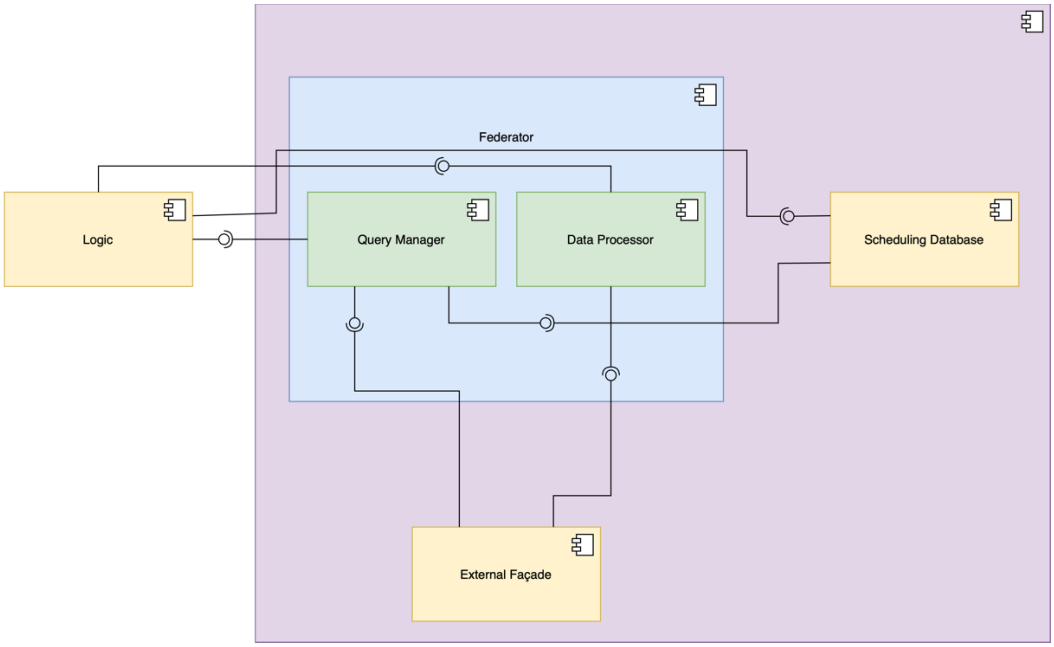
1.3B Storage



Name	**1.3B Storage Data Flow View**
Purpose	Illustrate the data flow between the various databases and how they interact with the logic component
Description	Requests come from the logic component and go to the federator, which then retrieves data from the scheduling database and external façade. A response is then sent back to Logic.
Requirements	Requirements 3-7
Elements	1.2 Logic
	1.3.1 Federator
	1.3.1.1 Manage Queries
	1.3.1.2 Process Data
	1.3.2 Scheduling Database
	1.3.3 External Façade
	1.3.3.1 Manage Shibboleth Proxy
	1.3.3.2 Manage Jenzabar Proxy

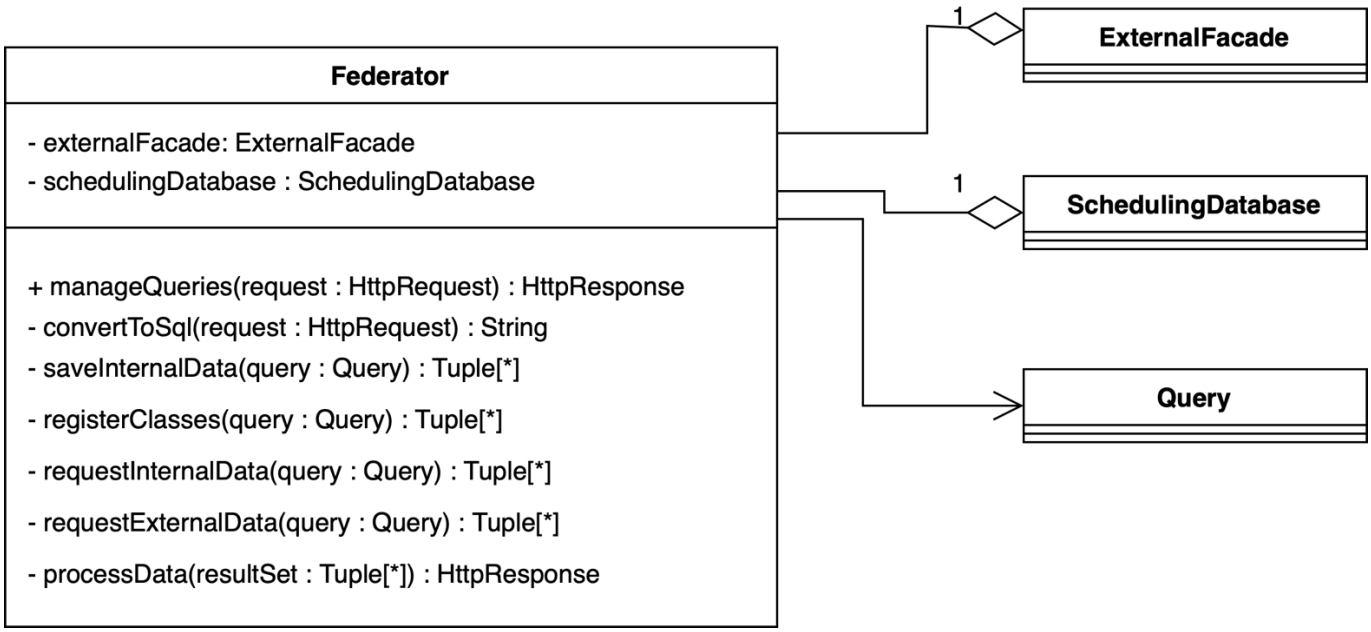
Name	**1.3B Storage Data Flow View**
	1.3.3.3 Manage Map Proxy
	1.3.3.4 Manage RMP Proxy
	1.3.1.1.6 Query
	Shibboleth: see Glossary
	Jenzabar: see Glossary
	Interactive Map: see Glossary
	RateMyProfessor: see Glossary
	Request**:** The initial request from the logic component
	Scheduling Data**:** Data from the scheduling database
	Request for External Data**:** A request for data from an external dependency
	Shibboleth Data: Data from Shibboleth
	Jenzabar Data: Data from Jenzabar
	Map Data: Data from the interactive map
	RMP Data: Data from RateMyProfessor
	Response Data: The final bit of data being given to the logic component by the federator
Referenced by	1.0 EnrollEase
Viewpoint	Data Flow Diagram

1.3.1A Federator



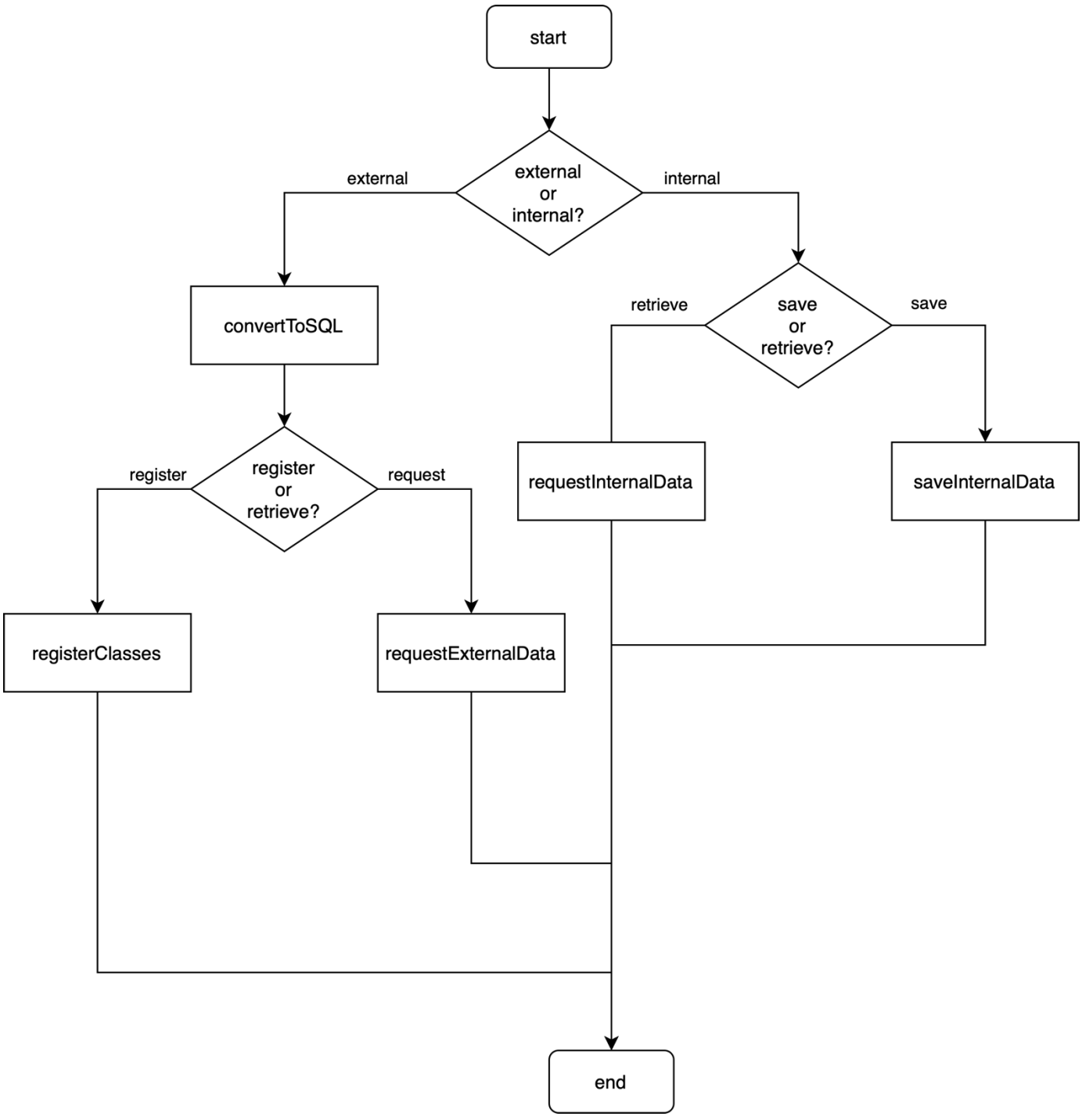
Name	1.3.1 Federator
Purpose	Displays the internal components of the Federator and how it communicates information to other parts of the program.
Description	The Federator receives queries from Logic on specific information needed, it requests information from the external databases as well as the internal scheduling database, formats the information, and returns it to Logic. It also receives schedules from Logic and stores those in the Scheduling Database.
Requirements	3, 6, 8, 14, 16
Elements	1.3.1.1 Query Manager
	1.3.1.2 Process Data
	1.3.2 Scheduling Database
	1.3.3 External Facade
Referenced by	1.3 Storage
Viewpoint	Component Diagram

1.3.1B Federator



Name	**1.3.1B Federator Class Diagram**
Purpose	Gives the basic blueprint for how the federator will communicate with our own database and the external façade.
Description	ManageQueries is going to be the driver for this class, and it will call everything else. Depending on the query, we will either save internal data, register for a class, or get some data from any of our databases. We're also going off the assumption that the result sets from the databases will take the form of a list of tuples, which is then passed into ProcessData, which converts that list into an HttpResponse, which is then returned to the logic component.
Requirements	2-5
Elements	1.3.1 Federator
	1.3.1.1 Manage Queries
	1.3.1.2 Process Data
	1.3.1.1.1 Convert to SQL
	1.3.1.1.4 Save Internal Data
	1.3.1.1.5 Register Classes
	1.3.1.1.3 Request Internal Data
	1.3.1.1.2 Request External Data
	1.3.1.1.6 Query
	1.3.2 Scheduling Database
	1.3.3 External Facade
Referenced by	1.0 EnrollEase, 1.3A Storage, 1.3B Storage, 1.3.1A Federator, 1.3.1.3 Logic Request, 1.3.2A Scheduling Database, 1.3.3A External Facade, 1.3.3.1 Shibboleth Query Request, 1.3.3.2 Jenzebar Query Request, 1.3.3.3 Interactive Map Data Request, 1.3.3.4 RateMyProfessor Query Request, 1.3.3.5 Federator to External Façade
Viewpoint	Class Diagram

1.3.1.1 Manage Queries



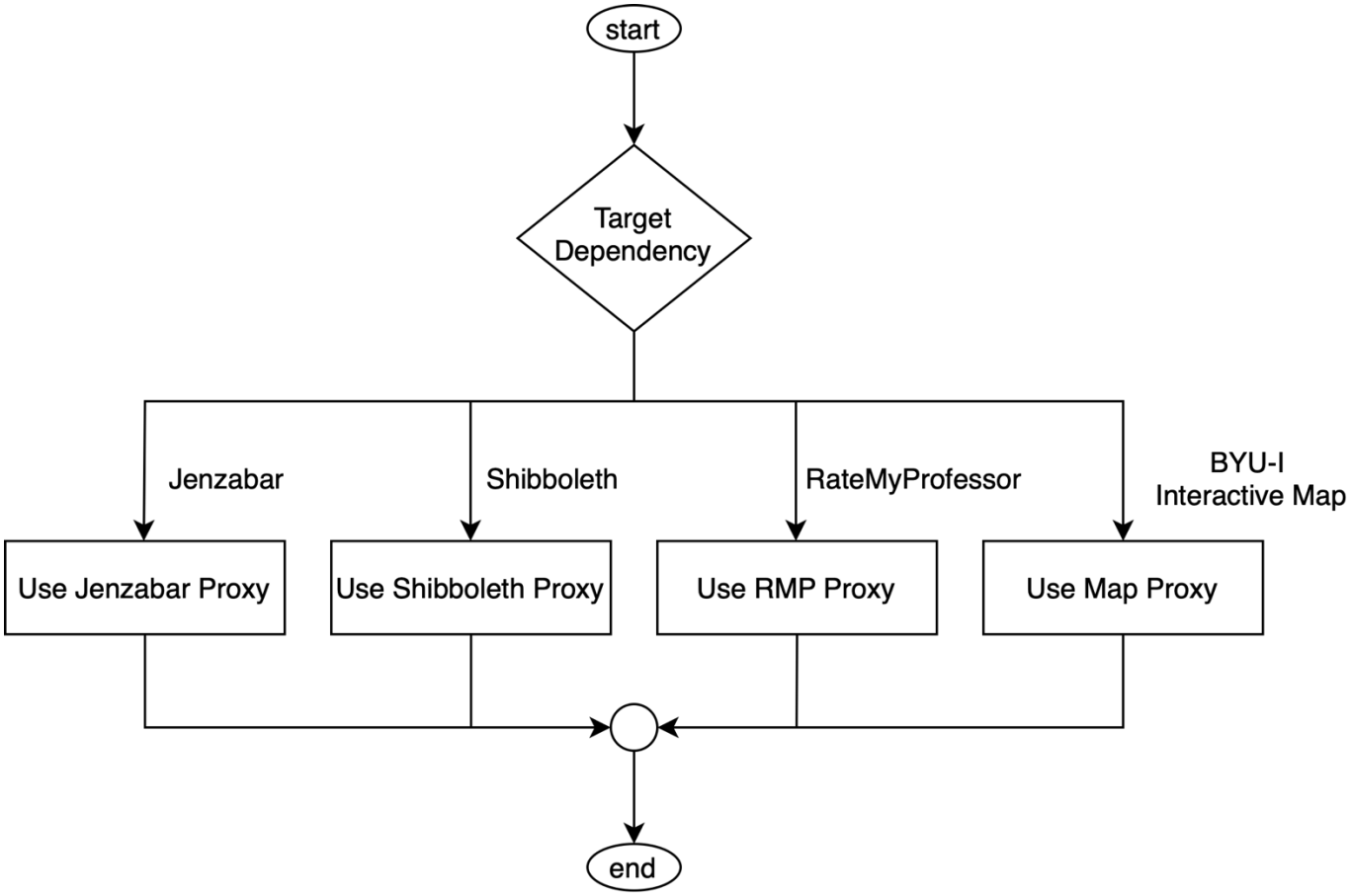
Name	**1.3.1.1 Manage Queries**
Purpose	Display how queries data requests from Logic are received and processed.
Description	The request is sent from logic, and it is checked if it is a request for external data, or internal data. If it's external, then the request is converted to SQL then sent to the external Database. If it's internal, it's determined whether the request is to retrieve data, or save data. The data is sent to be saved or sent to be retrieved from the scheduling database.
Requirements	4-6
Elements	1.3.1.1.1 convertToSQL
	1.3.1.1.2 requestExternalData
	1.3.1.1.3 requestInternalData
	1.3.1.1.4 saveInternalData
	1.3.1.1.5 registerClasses
Referenced by	1.3.1 Federator
Viewpoint	Flowchart

1.3.1.1.1 convertToSQL

```
IF query.getAction() = "insert":
    sqlQuery <- "INSERT INTO " + query.getTarget()
    sqlQuery += query.getData().keys
    sqlQuery += " VALUES "
    sqlQuery += query.getData().values
ELSE:
    sqlQuery <- "SELECT * FROM " + query.getTarget()
query += ";"
return sqlQuery
```

Name	**1.3.1.1.1 Convert to SQL**
Purpose	This is where the request made from the front end is converted into a language the database understands: SQL.
Description	The schedule to be inserted is encoded in the Query object. The table to be pulled from and the data to be pulled are also encoded in the Query object. This portion of code will convert that into SQL.
Requirements	4, 6, 13
Elements	query: see #1.3.1.1.6
	sqlQuery: the query generated from the request. Returned in string format to 1.3.1.1 Manage Queries.
Referenced by	1.3.1.1 Manage Queries
Viewpoint	Pseudocode

1.3.1.1.2 requestExternalData



Name	**1.3.1.1.2 RequestExternalData**
Purpose	Determine which external dependency needs to be called.
Description	At this point in the program, a request from Logic has been received and already converted to SQL. It is within the SQL that the target dependency will be determined.
Requirements	7-10
Elements	Target Dependency: Which external dependency we are trying to query from.
	Jenzabar: See Glossary.
	Shibboleth: See Glossary.
	RateMyProfessor: See Glossary.
	BYU-I Interactive Map: See Glossary.
	Use Shibboleth Proxy: see #1.3.3.1
	Use Jenzabar Proxy: see #1.3.3.2
	Use Map Proxy: see #1.3.3.3
	Use RMP Proxy: see #1.3.3.4
	Start: RequestExternalData function is called.
	End: Return requested data.
Referenced by	1.3.1B Federator Class Diagram
Viewpoint	Flowchart

1.3.1.1.3 requestInternalData

```
requestInternalData(query):

    action <- query.getAction()
    target <- query.getTarget()
    data <- query.getData()

    IF target == "schedules" or "preferences"
              IF action == "get":
                     result <- schedulingDatabaseManager.get(data)
                     RETURN (result.status, result.data)
```

Name	**1.3.1.1.3 Request Internal Data**
Purpose	Requests data from the scheduling database.
Description	Acts as a mediator between the Federator and the SchedulingDatabaseManager. It interprets a query and sends the request to the right database manager.
Requirements	3, 12
Elements	query: see#1.3.1.1.6
	schedulingDatabaseManager: see #1.3.2
Referenced by	1.3.1B Federator Class
	1.3.1.1 Manage Queries
Viewpoint	Pseudocode

1.3.1.1.4 saveInternalData

```
saveInternalData(query): tuple

    action <- query.getAction()
    target <- query.getTarget()
    data <- query.getData()

    if element == "schedules" or "preferences"
        result <- schedulingDatabaseManager.save(action, data)
        RETURN (result.status, result.data)
```

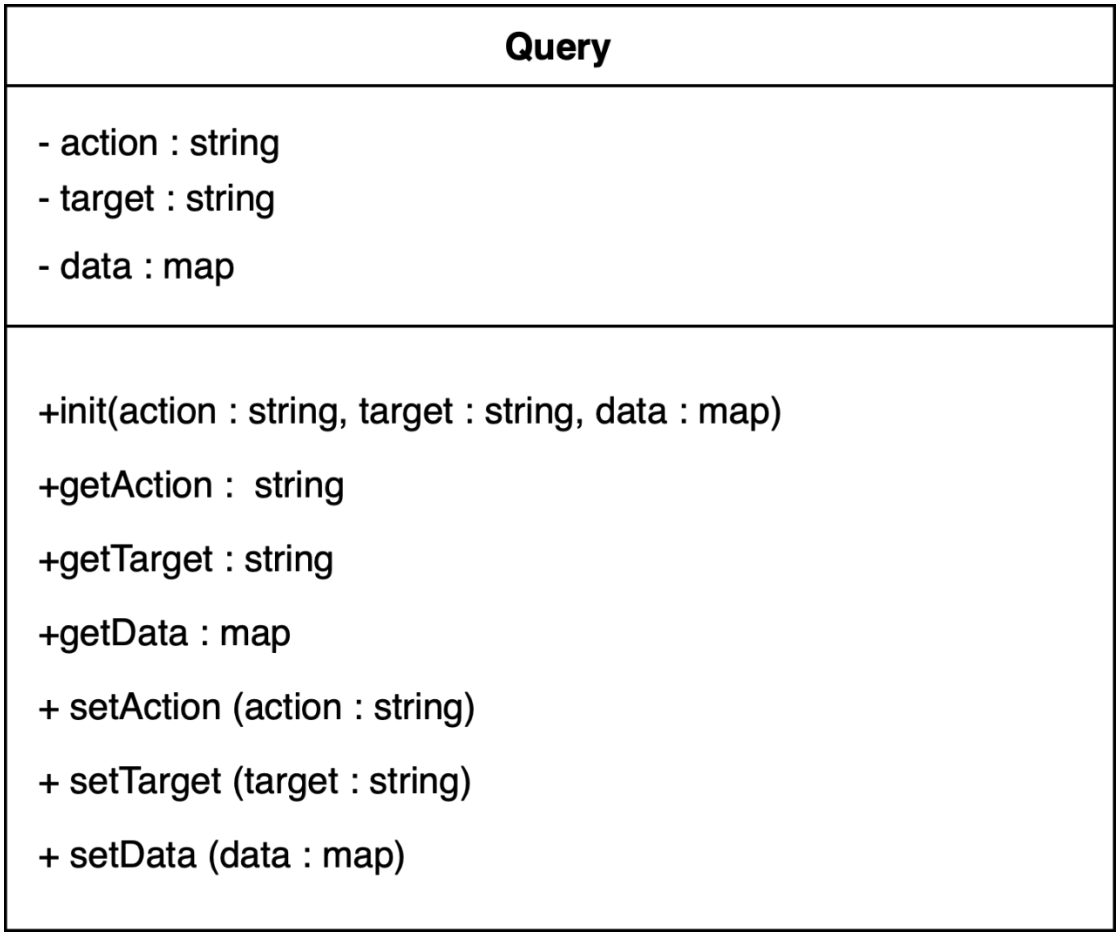
Name	**1.3.1.1.4 Save Internal Data**
Purpose	Saves data to the scheduling database.
Description	Acts as a mediator between the Federator and the SchedulingDatabaseManager. It interprets a query and sends the request to the right database manager.
Requirements	3, 12
Elements	query: the order with details about what and how to save into the database.
	schedulingDatabaseManager: see #1.3.2
Referenced by	1.3.1B Federator Class
	1.3.1.1 Manage Queries
Viewpoint	Pseudocode

1.3.1.1.5 registerClasses

```
registerClasses(query, token) : tuple

  action <- query.getAction()
  element <- query.getTarget()
  data <- query.getData()

  IF element == "classes"
       response <- jenzabarProxy.manageRequest(action, data, token)
      RETURN (response.status, response.data)
```

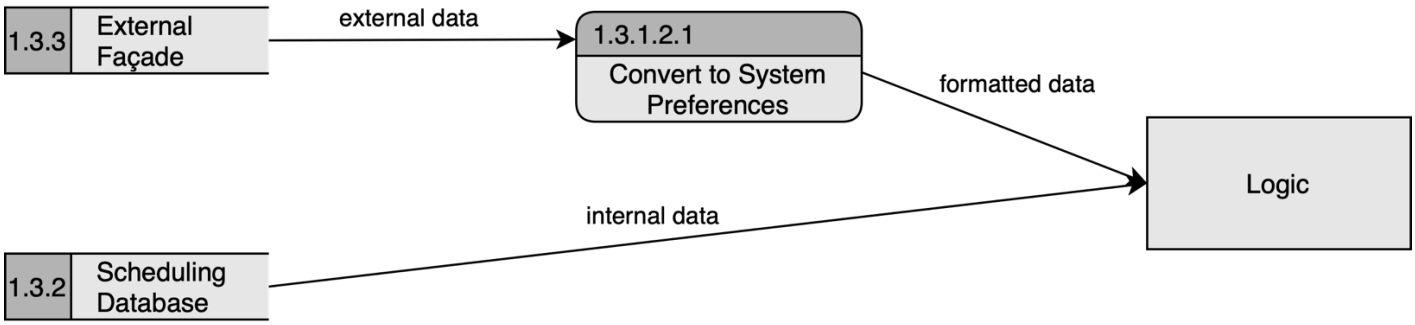
Name	**1.3.1.1.5 registerClasses**
Purpose	Posts data to Jenzabar for the student to register for classes.
Description	Extracts order from the query and sends an HTTP request to the Jenzabar external service.
Requirements	5
Elements	Query: order with details about how and what external Jenzabar APIs.
	jenzabarProxy: #see 1.3.3.2
	Token: key to send legitimate requests.
Referenced by	1.3.1B Federator
	1.3.1.1 Manage Queries
Viewpoint	Pseudocode

1.3.1.1.6 Query


Query
- action : string
- target : string
- data : map
+init(action : string, target : string, data : map)
+getAction : string
+getTarget : string
+getData : map
+ setAction (action : string)
+ setTarget (target : string)
+ setData (data : map)

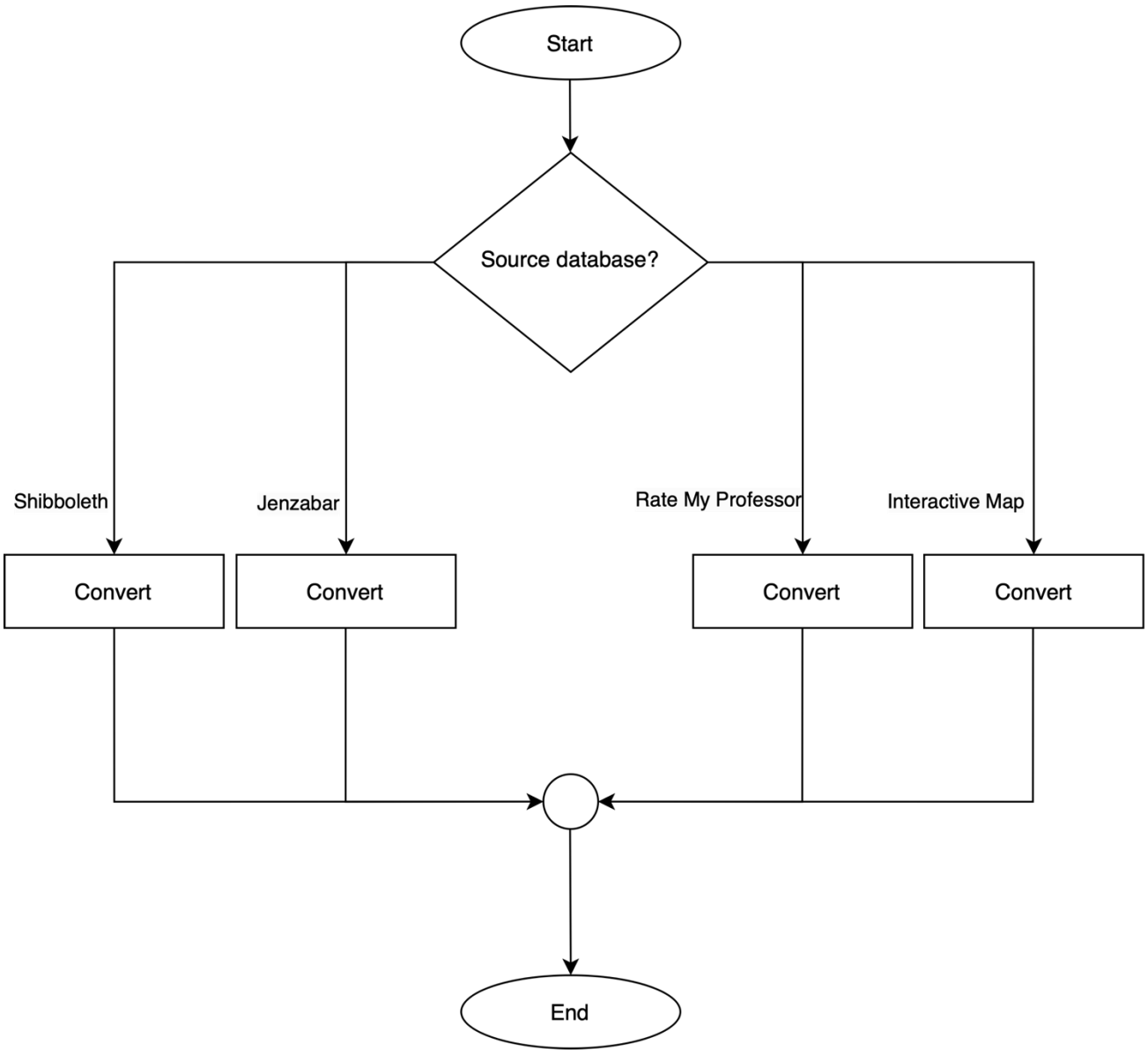
Name	**1.3.1.1.6 Query Class Diagram**
Purpose	Gives the basic blueprint for how a query can be used in the system.
Description	This is a utility class to structure the queries that will be passed between the federator and the databases.
Requirements	2-5
Elements	Action**:** The action that the query will perform, such as SELECT or INSERT.
	Target**:** The collection that the query will interact with.
	Data**:** The collection of data to be inserted into a database.
	Init**:** Represents the constructor of the class. This will simply call the setters.
	GetAction**:** The getter for the action.
	GetTarget**:** The getter for the target.
	GetData**:** The getter for the data.
	SetAction**:** The setter for the action.
	SetTarget**:** The setter for the action.
	SetData**:** The setter for the action.
Referenced by	1.3B Storage, 1.3.1B Federator, 1.3.1.1 Manage Queries, 1.3.1.1.1 Convert to SQL, 1.3.1.1.2 RequestExternalData, 1.3.1.1.4 SaveInternalData, 1.3.1.1.5 registerClasses, 1.3.1.3 Logic Request, 1.3.3.4 RateMyProfessor Query Request
Viewpoint	Class Diagram

1.3.1.2 Process Data



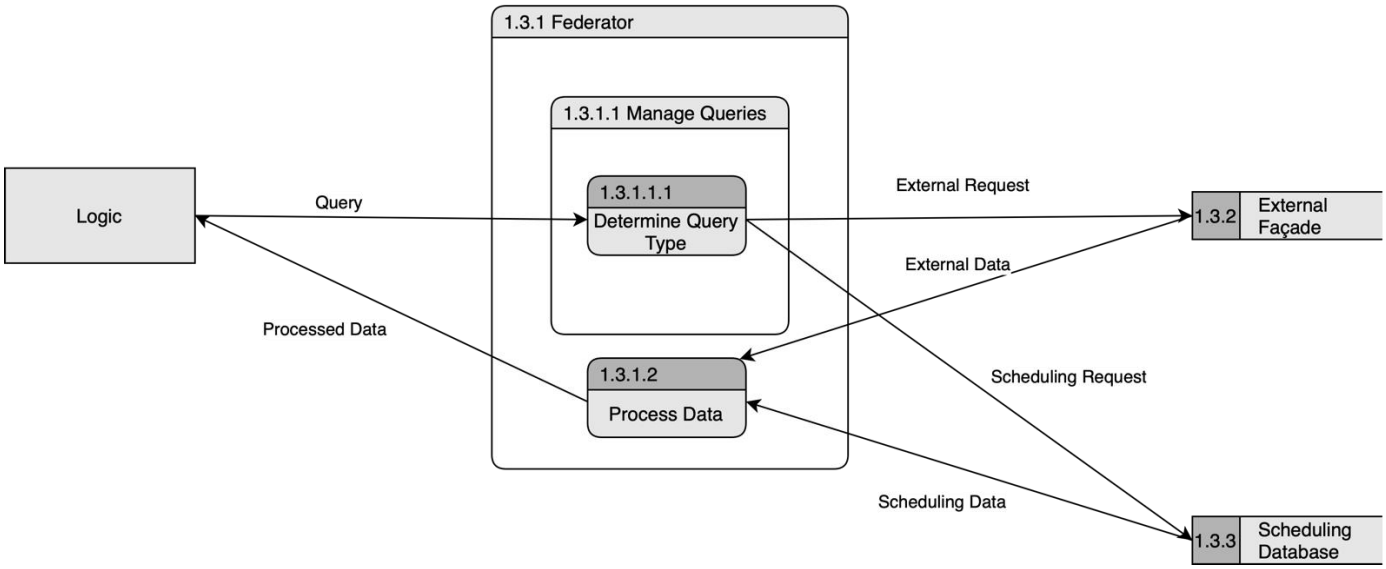
Name	**1.3.1.2 Process Data**
Purpose	Displays how data retrieved from the external database and internal database are processed so that logic can utilize it.
Description	The data is received from the external database and/or scheduling database. If it's from the external database it is converted into a format that our system wants, and then returned to logic. Data from our scheduling database is simply returned without the need for conversion.
Requirements	8-10
Elements	external data. Data retrieved from the external database
	internal data. Data retrieved from the scheduling database.
	1.3.2 Scheduling Database
	1.3.3 External Facade
	1.3.1.2.1 Convert to System Preferences
	formatted data. Data that has been packaged and processed to be utilized by Logic.
	1.2 Logic
Referenced by	1.3.1 Federator
Viewpoint	Data Flow Diagram

1.3.1.2.1 Convert to System Preferences



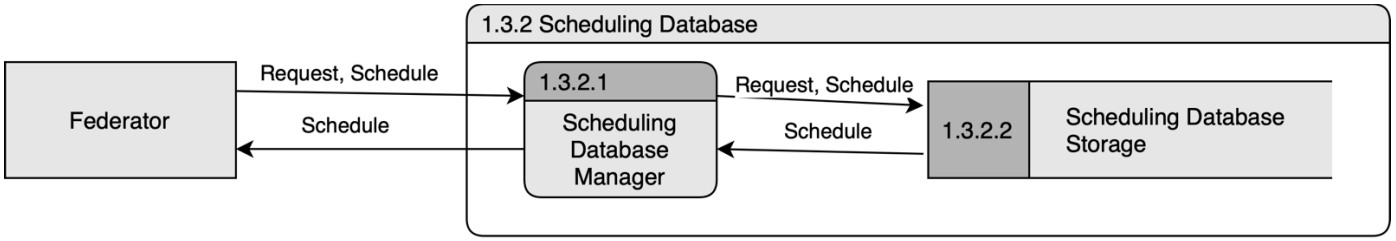
Name	**1.3.1.2.1 Convert to System Preferences**
Purpose	Format data for use in Logic.
Description	Based on which database the data is received from, format it for use in Logic.
Requirements	7-10
Elements	Start - Receive raw data
	Source database - Switch case of the 4 database options.
	Convert - Convert to TBD format for use in 1.2 Logic.
	End - Return formatted data to 1.2 Logic.
	Jenzabar: see 1.3.3.2
	Shibboleth: see 1.3.3.1
	Rate My Professor: see 1.3.3.4
	Interactive Map: see 1.3.3.3
Referenced by	1.3.1.2 Process Data
Viewpoint	Flow Chart

1.3.1.3 Logic Request



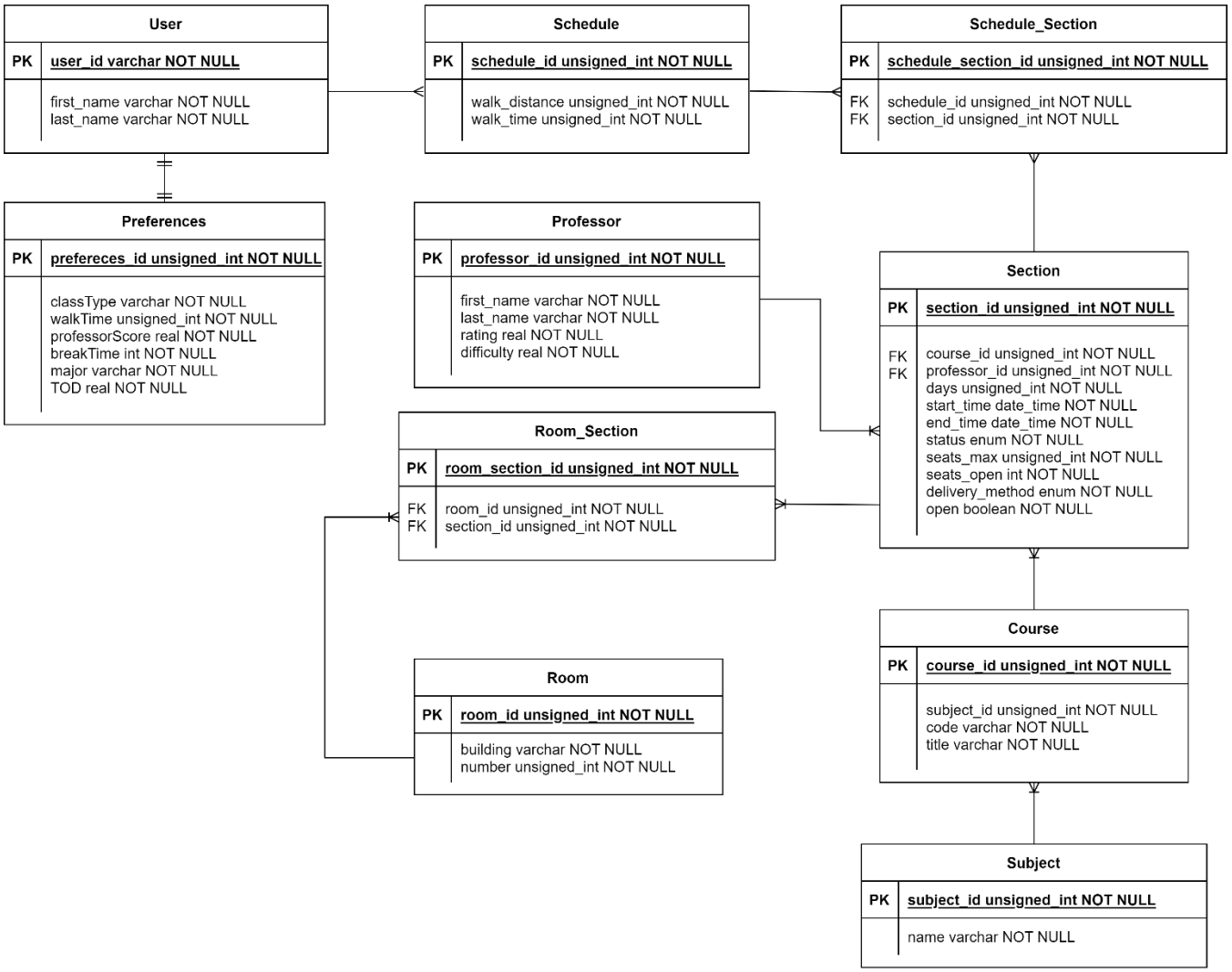
Name	**1.3.1.3 Logic Request**
Purpose	Display how queries are processed within the federator and passed along to the appropriate database.
Description	Logic sends a request to the federator, which processes the query to determine if the request is for the external or scheduling database. The request is then sent to the appropriate database, and the data retrieved processed in the federator and returned to Logic.
Requirements	3, 6, 8
Elements	1.2 Logic
	Query - request from Logic for information.
	1.3.1 Federator
	1.3.1.1 Manage Queries
	External Request. A request for information from the external database.
	1.3.3 External Facade
	External Data. Information retrieved from the external database.
	Scheduling Request. A request for data from the scheduling database.
	1.3.2 Scheduling Database
	Scheduling Data. Information retrieved from the scheduling database.
	1.3.1.2 Process Data
	Processed Data. Information retrieved from the database(s) and formatted to be used by Logic.
Referenced by	None
Viewpoint	Data Flow Diagram

1.3.2A Scheduling Database



Name	**1.3.2A Scheduling Database Data Request**
Purpose	Pass schedules to and from the scheduling database
Description	All data goes through the scheduling database manager. The manager passes requested data back to the federator, and stores scheduling data passed in by the federator.
Requirements	3-7
Elements	1.3 Storage, 1.3.1 Federator, 1.3.2 Scheduling Database
	1.3.1 Federator
	1.3.2 Scheduling Database
	1.3.2.1 Scheduling Database Manager
	1.3.2.2 Scheduling Database Storage
	TBD Request
	TBD Schedule
Referenced by	1.3 Storage
Viewpoint	Data Flow Diagram

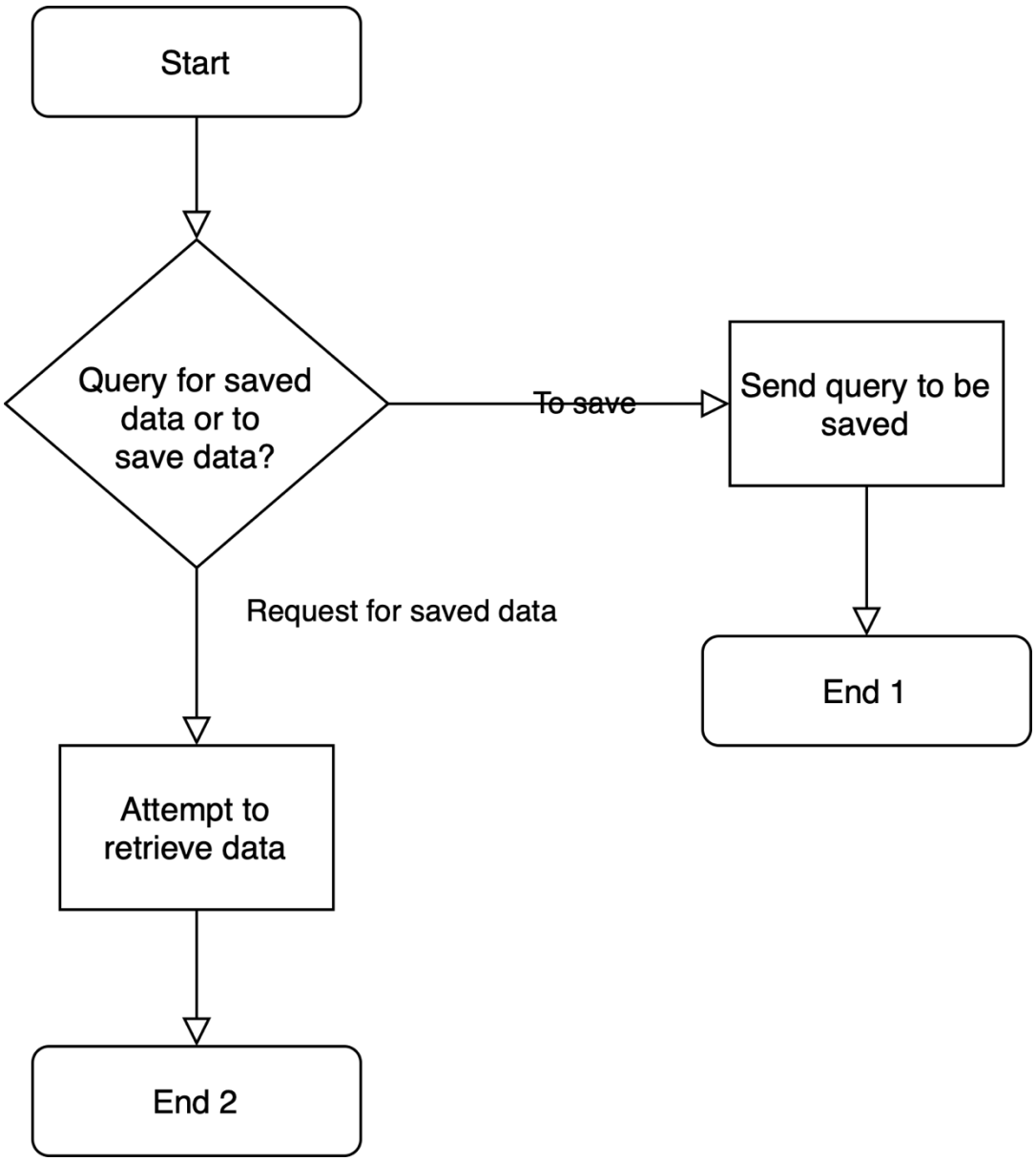
1.3.2B Scheduling Database



Name	**1.3.2B Scheduling Database**
Purpose	Store the schedules generated by EnrollEase.

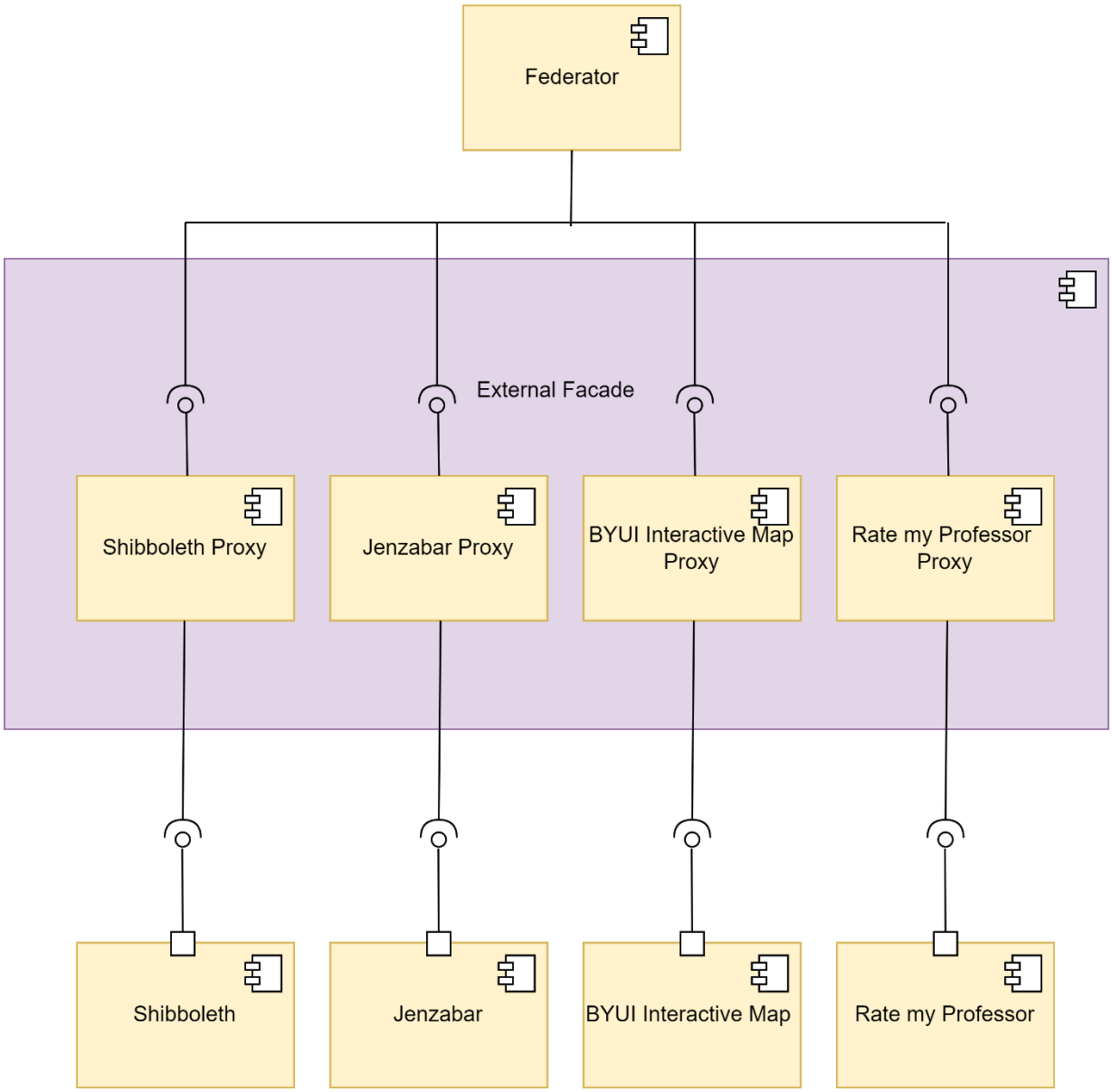
Name	**1.3.2B Scheduling Database**
Description	Each table holds the data necessary to represent a single entity within a generated schedule.
Requirements	7, 8
Elements	User. A user of EnrollEase.
	Schedule. A Schedule generated by EnrollEase.
	Schedule_Section. A join table to create a many-to-many relationship between Schedule and Section.
	Section. A particular section of a course.
	Room_Section. A join table to create a many-to-many relationship between Room and Section.
	Room. The room a section is held in.
	Course. A course offered by the university.
	Subject. The subject a course falls under.
	Professor. The professor teaching a section.
	Preferences. Student's preferences that act as filters when generating schedules.
Referenced by	1.3 Storage
Viewpoint	Entity Relationship Diagram

1.3.2.1 Scheduling Database Manager



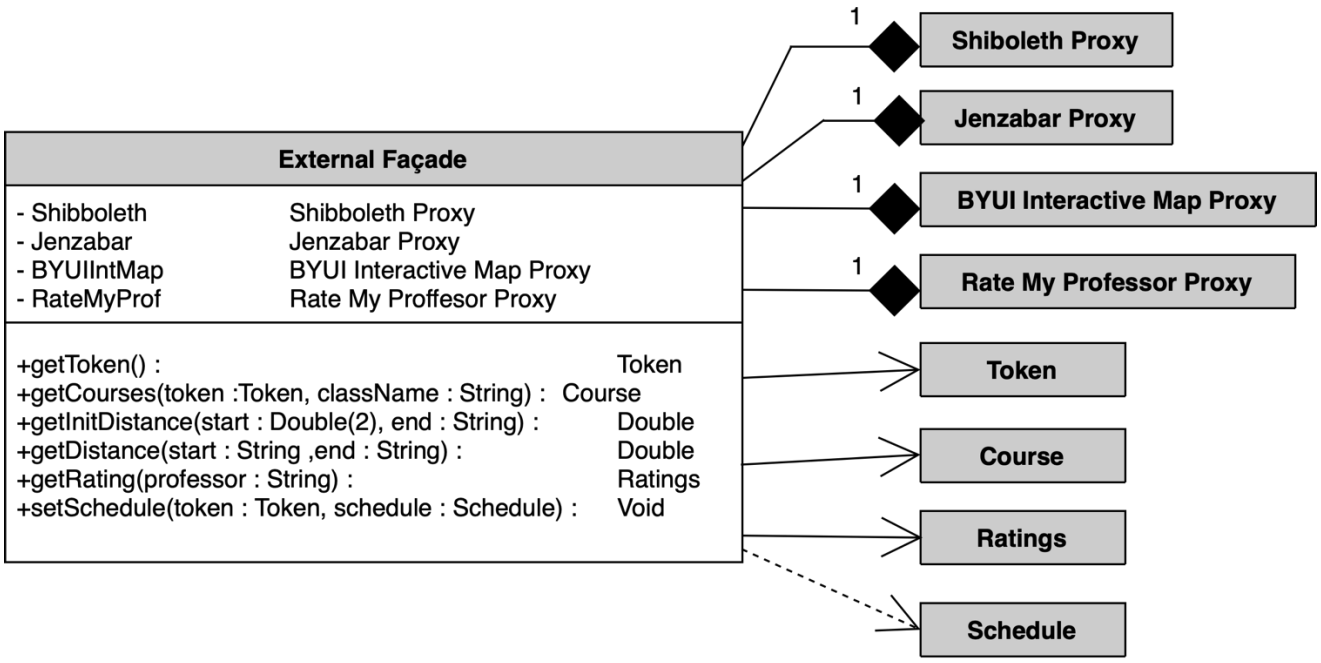
Name	**1.3.2.1 Scheduling Database Manager**
Purpose	To manage the requests between the federator and the scheduling database storage.
Description	The database manager will either save or retrieve the data based on the query received. For a data request it will attempt to retrieve the data from storage; on a successful attempt it will return the data, otherwise it will return an error. For a save the data will be sent to storage to be saved.
Requirements	6, 11
Elements	Query: see 1.3.1.1.6
	end 1: nothing to return
	send query to be saved: Sends the query to storage to be saved.
	attempt to retrieve data: Try’s to retrieve data from storage, if it isn't contained in storage returns a failure code, otherwise will get requested data.
	end 2: returns either a failure code, or data requested
Referenced by	1.3.2 Scheduling Database
Viewpoint	Flowchart

1.3.3A External Facade



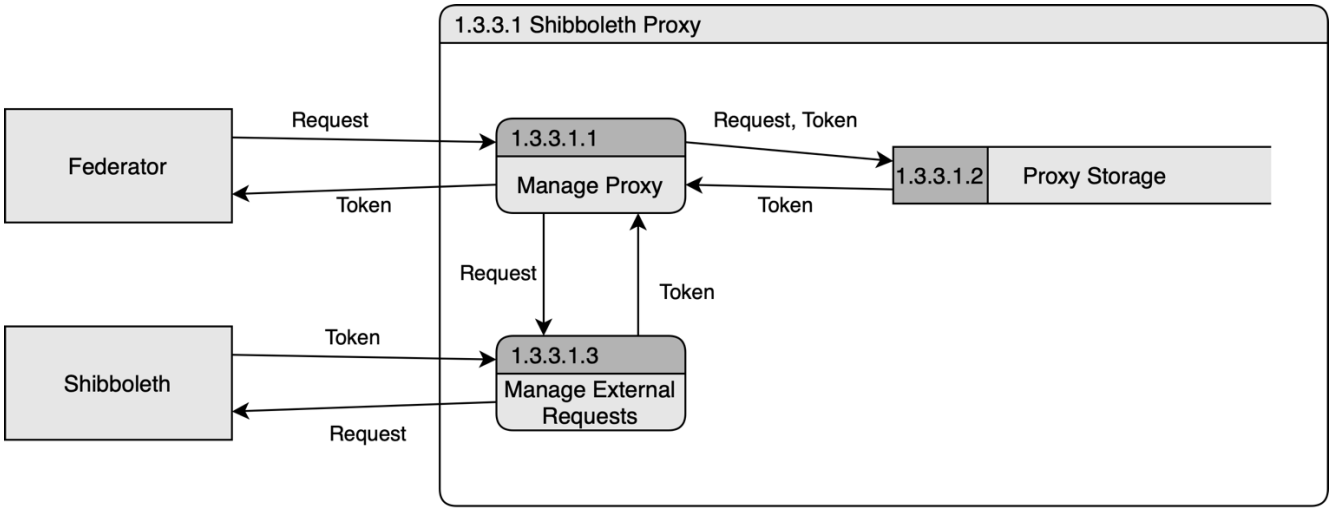
Name	**1.3.3A External Façade**
Purpose	Illustrate the interactions between the external dependencies of the system and the External storage component of the Storage.
Description	The federator requests data from the external façade that knows how to request the data from the various external dependencies of EnrollEase and stores the relevant data.
Requirements	3-8
Elements	1.2 Logic
	Jenzabar – see Glossary
	Shibboleth – see Glossary
	BYUI Interactive Map – see Glossary
	Rate My Professor – see Glossary
	1.3.1 Federator
	1.3.3.1 Shibboleth Proxy
	1.3.3.2 Jenzabar Proxy
	1.3.3.3 BYUI Interactive Map Proxy
	1.3.3.4 Rate My Professor Proxy
Referenced by	1.3
Viewpoint	Component Diagram

1.3.3B External Facade



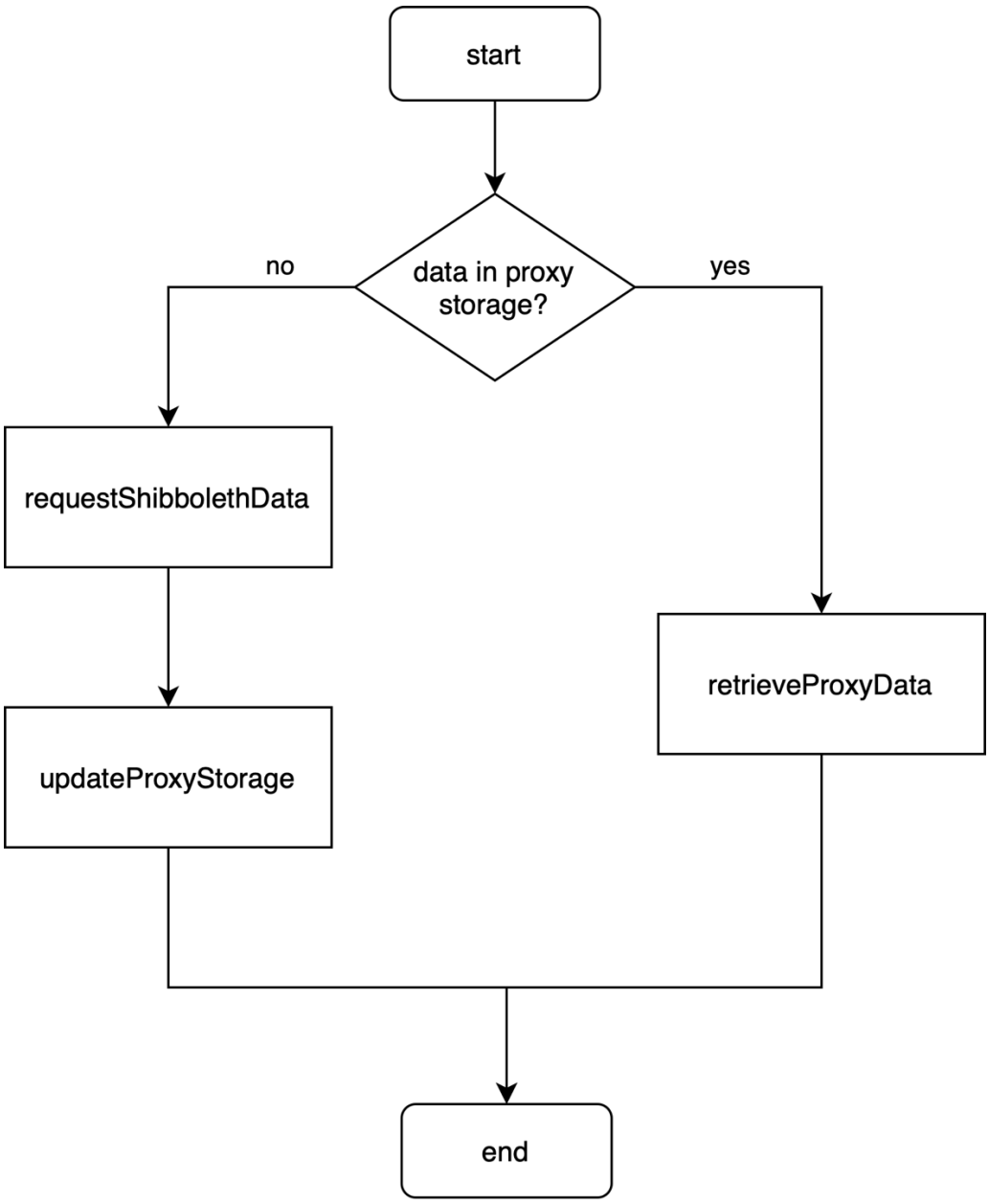
Name	**1.3.3B External Facade**
Purpose	Illustrate the interactions between the external dependencies of the system and the External storage component of the Storage.
Description	The federator requests data from the external facade that knows how to request the data from the various external dependencies of EnrollEase and stores the relevant data.
Requirements	5, 7, 8, 15
Elements	1.2 Logic
	1.3.1 Federator
	1.3.2 Scheduling Database
	1.3.3.1 Shibboleth Proxy
	1.3.3.2 Jenzabar Proxy
	1.3.3.3 Interactive Map Data Request
	1.3.3.4 Rate my Professor Proxy
	Shibboleth – see Glossary
	Jenzabar – see Glossary
	BYUI Interactive Map – see Glossary
	Rate my Professor – see Glossary
Referenced by	1.3 Storage
Viewpoint	Component Diagram

1.3.3.1 Shibboleth Query Request



Name	**1.3.3.1 Shibboleth Query Request**
Purpose	Display how information (an access token) is retrieved from Shibboleth, stored in proxy storage, and then delivered to the federator when requested.
Description	A request from logic is received by the proxy, processed by the proxy manager, and sent to the external request manager. The External Request Manager then requests the information from Shibboleth, and the retrieved access token is stored in Proxy Storage and set back to the federator.
Requirements	15
Elements	1.3.1 Federator
	Request. Query from the federator to retrieve an access token from the Shibboleth Proxy.
	Token. Access Token retrieved from Shibboleth, stored in Proxy Storage, and returned to the Federator.
	1.3.3.1 Shibboleth Proxy
	1.3.3.1.1 Manage Shibboleth Proxy
	1.3.3.1.2 Shibboleth Proxy Storage
	1.3.3.1.3 Manage Shibboleth Requests
	Shibboleth – see Glossary
Referenced by	1.3.3 External Facade
Viewpoint	Data Flow Diagram

1.3.3.1.1 Manage Shibboleth Proxy



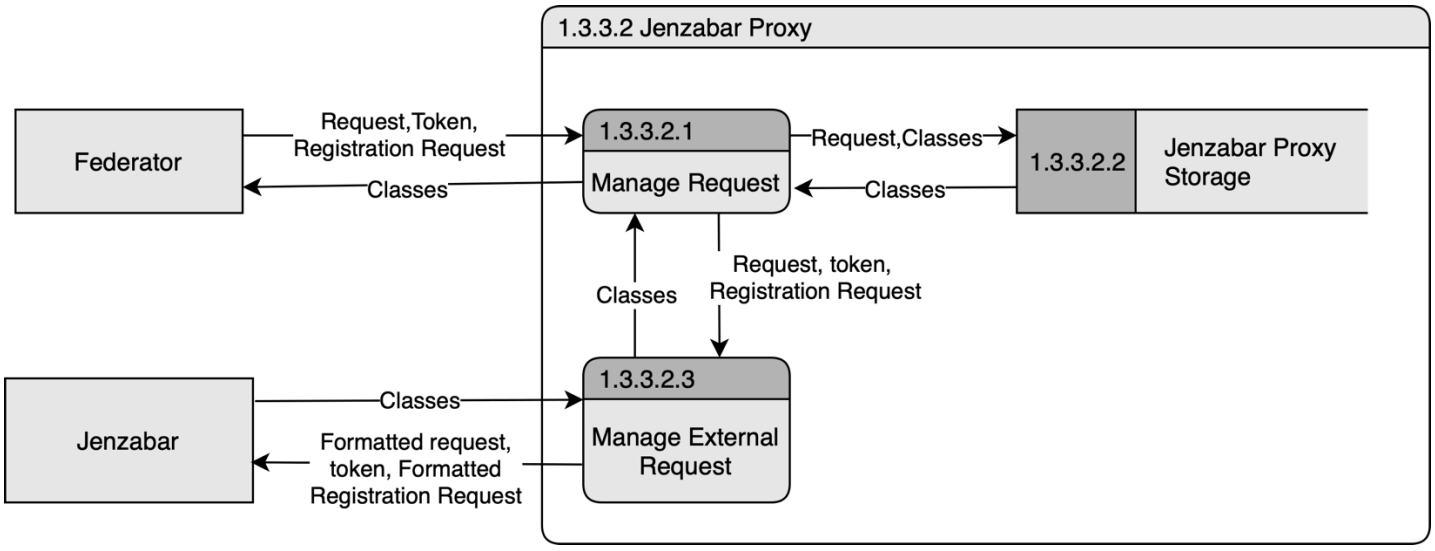
Name	**1.3.3.1.1 Manage Shibboleth Proxy**
Purpose	To Show how requests are processed are processed and data is retrieved from Shibboleth or the Proxy Storage
Description	Manage Proxy receives a request from the system, and checks if the data that is requested is within the proxy storage. If it is, it's retrieved and sent back. If not, a request is made to Shibboleth for the data. The retrieved data is stored within the proxy storage, and then sent back.
Requirements	3-7
Elements	start. a request is received from the federator for a shibboleth token.
	requestShibbolethData. This function will be to initiate a request to the external resource Shibboleth for the token.
	updateProxyStorage. Once the token is retrieved from Shibboleth, it's temporarily saved in our proxy storage for future use.
	retriveProxyData. Retrieves the desired token from the proxy storage.
	end. The retrieved token is returned to the federator.
Referenced by	1.3.3.1 Shibboleth Proxy
Viewpoint	Flowchart

1.3.3.1.3 Manage External Requests

```
IMPORT ShibbolethAPI

function manage_shibboleth_requests(query)
        DEFINE token = Shibboleth.get(query)
        RETURN token
```

Name	1.3.3.1.3 Manage Shibboleth Requests
Purpose	To show how data is requested from Shibboleth.
Description	The function receives a query for an access token from shibboleth. It performs a GET call to the Shibboleth API and returns the result.
Requirements	3-7
Elements	Query: see Glossary
	Token: see Glossary
	Shibboleth: see Glossary
Referenced by	1.3.3.1 Shibboleth Proxy
Viewpoint	Pseudocode

1.3.3.2 Jenzabar Query Request



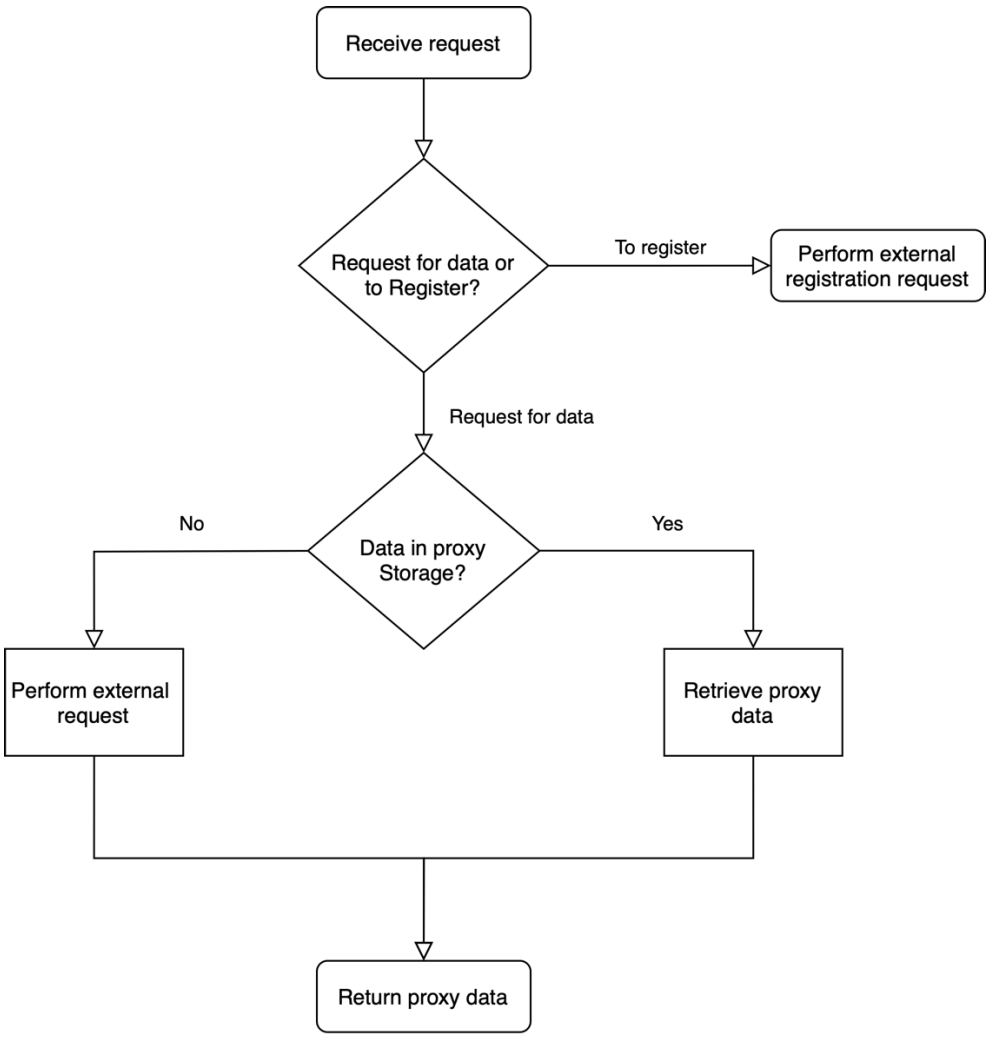
Name	**1.3.3.2 Jenzabar Query Request**
Purpose	Illustrate the dataflow between the Federator and the external dependency, Jenzabar.
Description	The federator requests data from the external database, specifically from the Jenzabar part of the database. The Jenzabar proxy then verifies the request, formats it for the API, and then sends it to Jenzabar. Jenzabar will then send the requested data back.
Requirements	5
Elements	1.3.1 Federator
	1.3.3.2.1 Manage Jenzabar Proxy
	1.3.3.2.2 Jenzabar Proxy Storage
	1.3.3.2.3 Manage Jenzabar Requests
Referenced by	1.3.3 External Facade
Viewpoint	Data Flow Diagram

1.3.3.2.1 Manage Jenzabar Proxy

```
IMPORT JenzabarEasyAPI

function manage_jenzabar_request(Token, Request)
   DEFINE classes → JenzabarEasyAPI.get(Token, Request)
   RETURN classes

function manage_jenzabar_request(Token, RegisterRequest)
   JenzabarEasyAPI.put(Token, RegisterRequest)
```

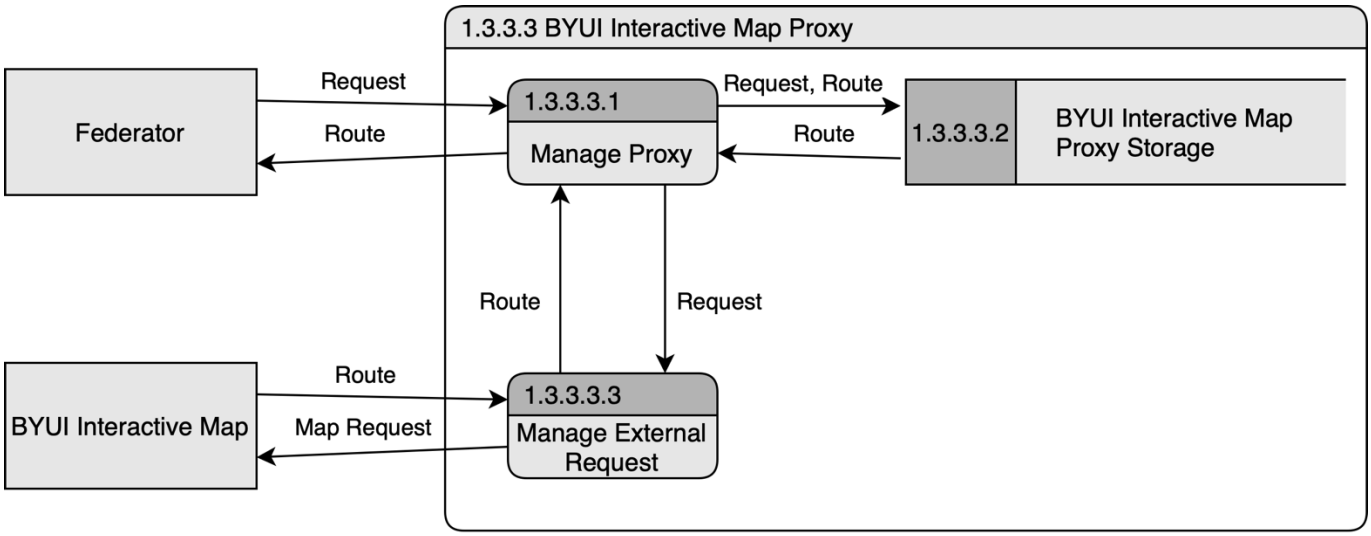
Name	**1.3.3.2.1 Manage Jenzabar Proxy**
Purpose	To illustrate the relevant details and methods for the management of requests between Manage Proxy and the External Dependency.
Description	Manage Jenzabar Proxy determines the request it was given and then sends the student's schedule to be registered through Jenzabar, retrieves the data from the proxy database if it has it, or send it to be requested from Jenzabar if it doesn't have it.
Requirements	3-7
Elements	Request: see Glossary
	Start: Starts when the proxy is called and receives a request
	End 1: No return
	End 2: Returns the requested data
Referenced by	1.3.3.5 Federator to External Database, 1.3.3.2 Jenzabar Query Request
Viewpoint	Pseudocode

1.3.3.2.3 Manage Jenzabar Requests



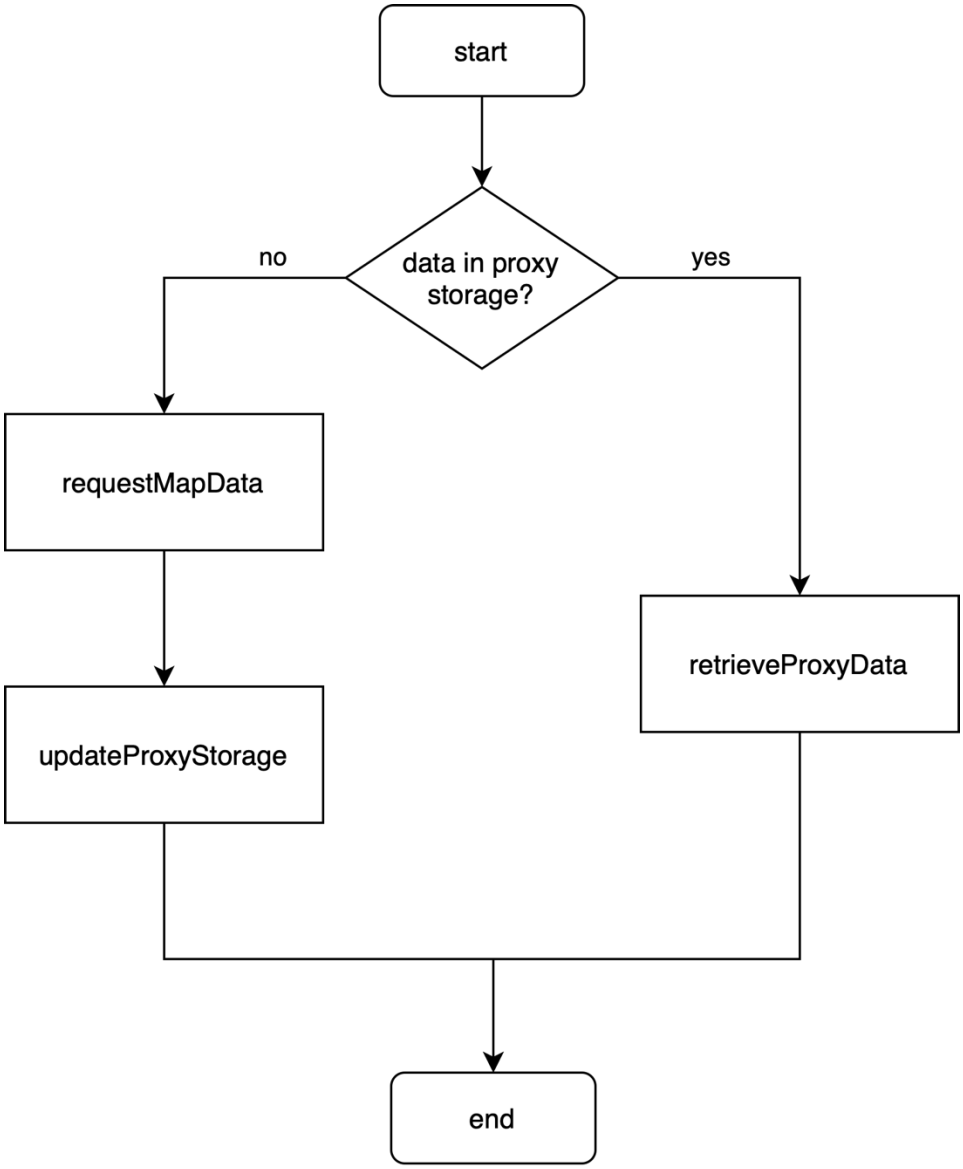
Name	**1.3.3.2.3 Manage Jenzabar Requests**
Purpose	To illustrate the relevant details and methods for the management of requests by Manage Jenzabar Requests.
Description	Manage Jenzabar Requests the data from Jenzabar if it is a request call or puts the data into Jenzabar if it is a registration request.
Requirements	3-7
Elements	Request: see Glossary
	Token: see Glossary
	Jenzabar: see Glossary
Referenced by	1.3.3.5 Federator to External Database, 1.3.3.2 Jenzabar Query Request, 1.3.3.2.3 Manage Jenzabar Requests
Viewpoint	Flow chart

1.3.3.3 Interactive Map Data Request



Name	**1.3.3.3 BYUI Interactive Map Query Request**
Purpose	Pass requested map data to the federator
Description	Data already stored in the proxy will simply be passed back to the federator. Otherwise, data will be requested from BYUI Interactive Map, stored in the proxy, and passed back to the federator.
Requirements	Requirement 10
Elements	1.3.1 Federator
	1.3.3.3 BYUI Interactive Map Proxy
	1.3.3.3.1 Manage Proxy
	BYUI Interactive Map Proxy Storage
	1.3.3.3.3 Manage External Request
	TBD Request
	TBD Route
	TBD Map Request
Referenced by	1.3.3 External Facade
Viewpoint	Data Flow Diagram

1.3.3.3.1 Manage Proxy



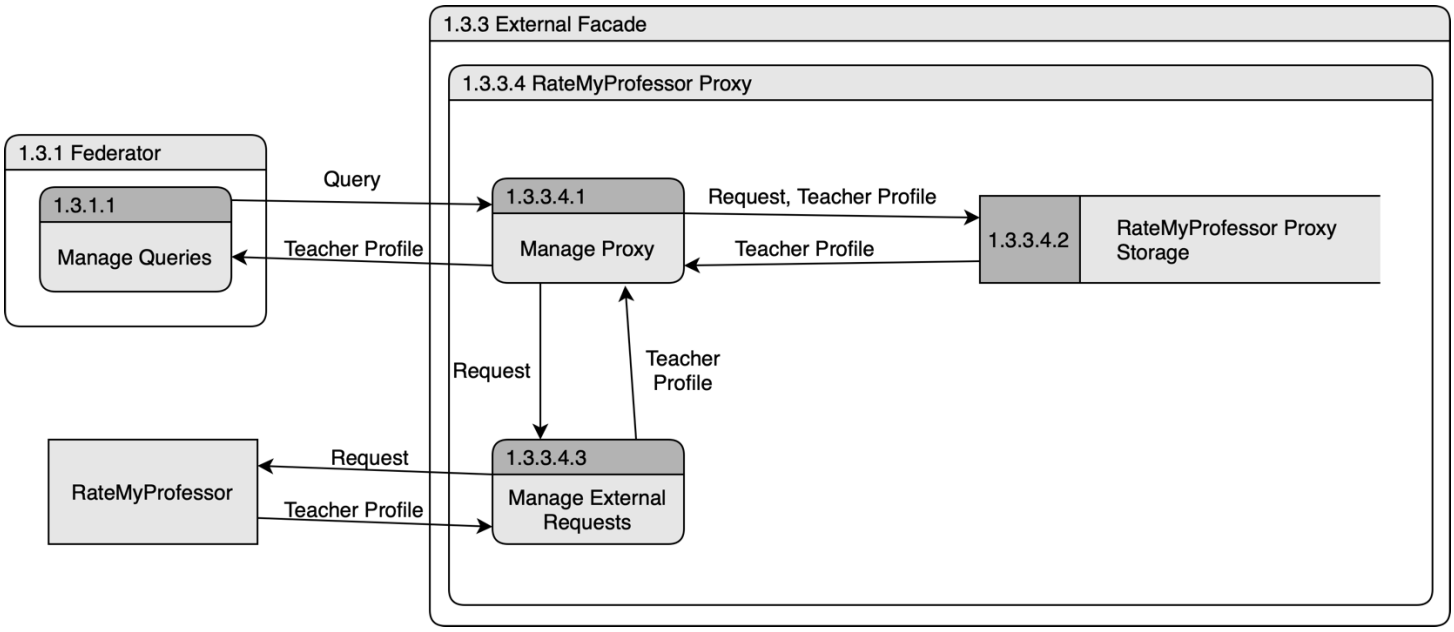
Name	**1.3.3.3.1 Interactive Map Manage Proxy**
Purpose	To Show how requests are processed are processed and data is retrieved from Shibboleth or the Proxy Storage
Description	Manage Proxy receives a request from the system, and checks if the data that is requested is within the proxy storage. If it is, it's retrieved and sent back. If not, a request is made to the Interactive Map for the data. The retrieved data is stored within the proxy storage, and then sent back.
Requirements	3-7
Elements	start. A request for a map route is received from the federator.
	requestMapData. This function will be to initiate a request to the external resource Interactive Map for the desired route.
	updateProxyStorage. Once the route is retrieved from Interactive Map, it's temporarily saved in our proxy storage for future use.
	retriveProxyData. Retrieves the desired route from the proxy storage.
	end. The map route is returned to the federator.
Referenced by	1.3.3.3 Interactive Map Proxy
Viewpoint	Flowchart

1.3.3.3.3 Manage External Request

```
IMPORT ByuiMapAPI

function manage_byui_map_request(string room_num_start, string room_num_end)
    DEFINE distance = ByuiMap.get_distance(room_num_start, room_num_end)
    DEFINE time = ByuiMap.get_time(room_num_start, room_num_end)
    RETURN distance, time
```

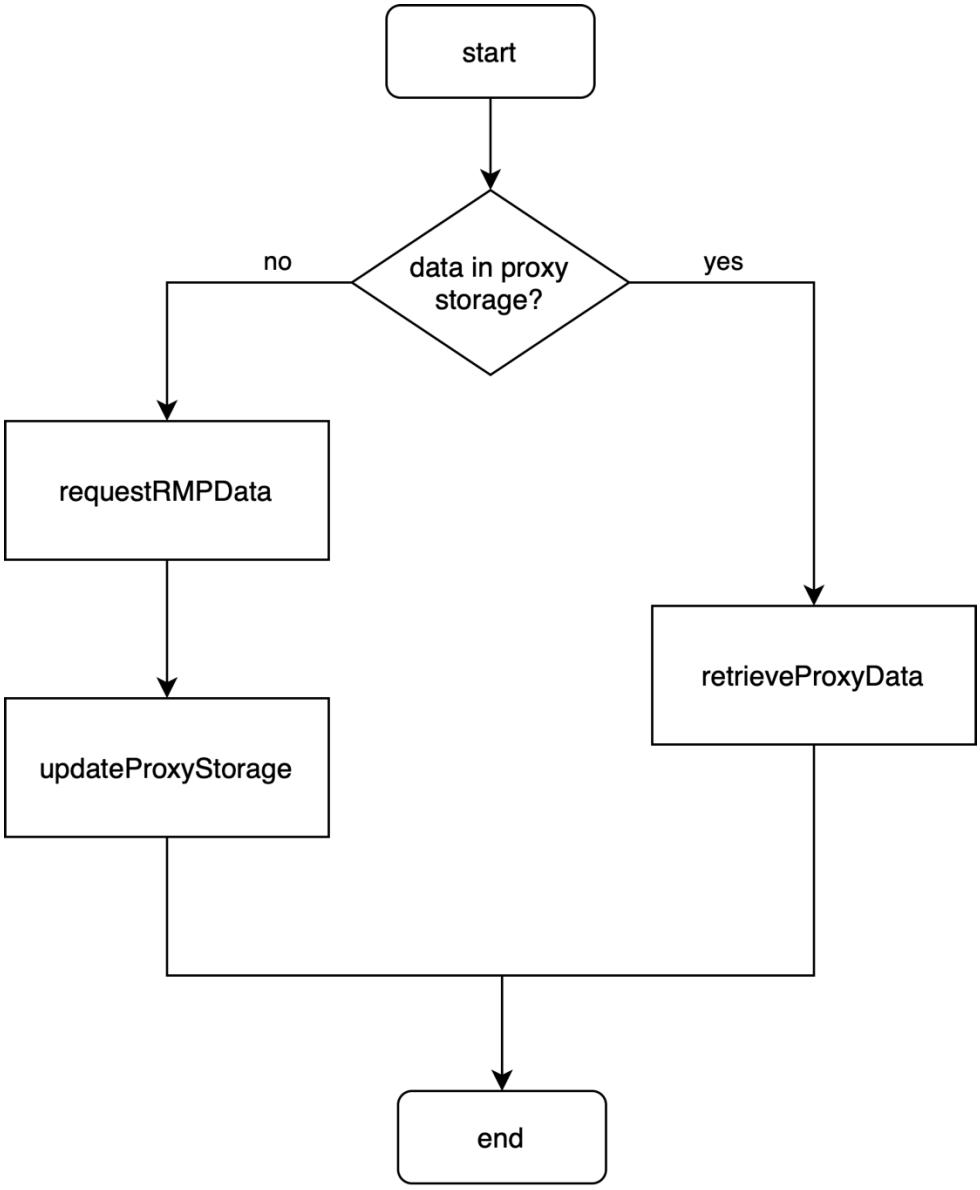
Name	**1.3.3.3.3 Manage BYUI Map Requests**
Purpose	To show how data is requested from the BYUI Interactive Map.
Description	This function receives two room numbers. It returns the distance between them and the walking time between them.
Requirements	3-7
Elements	room_num_start: A string representing a room number used as the starting position, e.g. "STC375".
	room_num_end: A string representing a room number used as the ending position.
	distance: The distance in miles between two classrooms.
	time: The walking time in minutes between two classrooms.
Referenced by	1.3.3.3 BYUI Interactive Map Proxy
Viewpoint	Pseudocode

1.3.3.4 RateMyProfessor Query Request



Name	**1.3.3.4 RateMyProfessor Data Flow View**
Purpose	Illustrate the data flow between RateMyProfessor and our own database
Description	The federator sends a request to the proxy manager. The proxy manager then determines where to get the data from, which could either be our own proxy database or from RateMyProfessor itself. The data is then either brought back to the federator or our proxy database.
Requirements	Requirement 11
Elements	1.3.1.1 Manage Queries
	1.3.3.4 RateMyProfessor Proxy
	1.3.3.4.1 Manage Proxy
	1.3.3.4.2 RateMyProfessor Proxy Storage
	1.3.3.4.3 Manage External Requests
	1.3.7 RateMyProfessor
	Query**:** The SQL query coming from the federator
	Request**:** A request for certain data
	Teacher Profile**:** Data on requested teacher(s)
Referenced by	1.3 Storage, 1.3.3 External Façade
Viewpoint	Data Flow Diagram

1.3.3.4.1 RateMyProfessor Manage Proxy



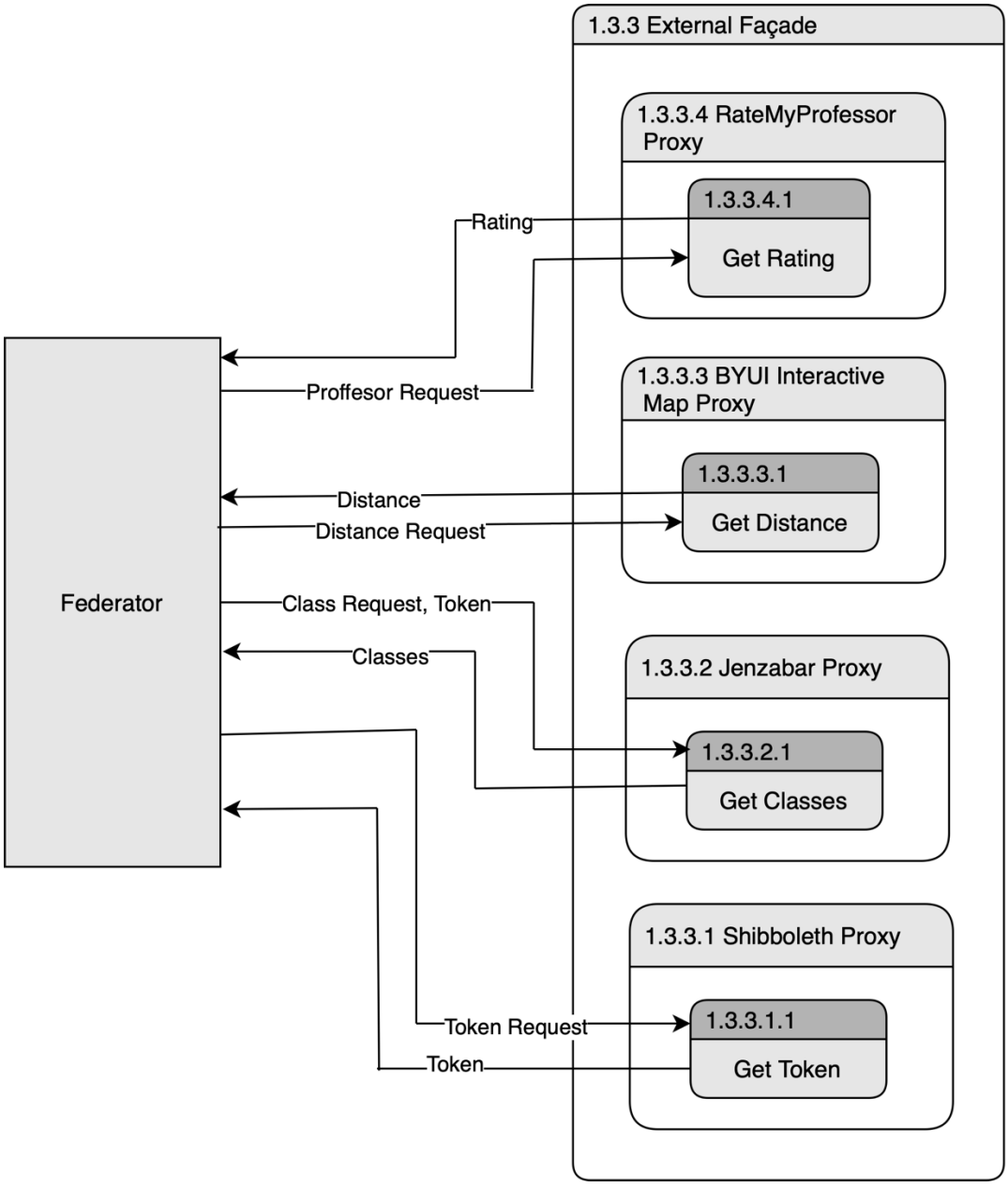
Name	**1.3.3.4.1 RateMyProfessor Manage Proxy**
Purpose	To Show how requests are processed are processed and data is retrieved from Shibboleth or the Proxy Storage
Description	Manage Proxy receives a request from the system, and checks if the data that is requested is within the proxy storage. If it is, it's retrieved and sent back. If not, a request is made to RateMyProfessor for the data. The retrieved data is stored within the proxy storage, and then sent back.
Requirements	3-7
Elements	start. A request for a professor RateMyProfessor profile is received from the federator.
	requestRMPData. This function will be to initiate a request to the external resource RateMyProfessor for the desired professor profile.
	updateProxyStorage. Once the professor profile is retrieved from RateMyProfessor, it's temporarily saved in our proxy storage for future use.
	retriveProxyData. Retrieves the desired professor profile from the proxy storage.
	end. The professor profile is returned to the federator.
Referenced by	1.3.3.4 RateMyProfessor Proxy
Viewpoint	Flowchart

1.3.3.4.3 Manage External Requests

```
IMPORT RateMyProfApi
IMPORT SearchProfessor

manage_external_request(string first_name, string last_name)
   byui_rmp_data <-- RateMyProfApi(1754)
   details <-- byui_rmp_data.SearchProfessor("{first_name} {last_name}")
   RETURN details
```

Name	**1.3.3.4.3 RateMyProfessor Manage External Requests**
Purpose	Contacts the API for the external RateMyProfessor database and returns the professor's profile from the site.
Description	The API only needs a first and last name (we are hard-coding in BYU-I's ID number from RateMyProfessor). By using the API, it will return a JSON object with the necessary data on the professor.
Requirements	8
Elements	RateMyProfApi: The API we need to use to get the professor's data. When an ID number is passed in, it creates an object for the specified school, and that object holds all the data on its professors.
	SearchProfessor**:** This method returns a JSON object on the specified professor. It just needs a string with the professor's first and last name.
	first_name**:** The first name of the professor.
	last_name: The last name of the professor.
	byui_rmp_data: The API object created for this function. The number "1754" is the school's ID on RateMyProfessor.
	Details**:** The JSON object with the professor's details on RateMyProfessor. The format we will get is found in this documentation: https://github.com/tisuela/ratemyprof-api
Referenced by	1.3.3.4
Viewpoint	Pseudocode

1.3.3.5 Federator to Eternal Facade



Name	**1.3.3.5 Federator to External Facade View**
Purpose	Illustrate the flow of data for the data request made by the federator to the external façade component of the Storage.
Description	The federator requests data from the external façade to one of its four proxies. The proxies then format the request to suit the APIs of the external dependencies before sending the request to them. The external dependencies then return the requested data.
Requirements	5, 7, 8, 15
Elements	1.3.3 External Facade
	1.3.1 Federator
	1.3.3.1 Shibboleth Proxy
	1.3.3.2 Jenzabar Proxy
	1.3.3.3 BYUI Interactive Map Proxy
	1.3.3.4 Rate my Professor Proxy
	1.3.3.5.1 Token Request
	1.3.3.5.2 Classes Request
	1.3.3.5.3 Distance Request
	1.3.3.5.4 Ratings Request
	1.3.3.5.5 Token
	1.3.3.5.6 Classes
	1.3.3.5.7 Distance
	1.3.3.5.8 Ratings
Referenced by	1.3 Storage view
Viewpoint	Data Flow Diagram

Appendix

Software Design Textbook	Helfrich, James. Software Design. Available from: VitalSource Bookshelf, Kendall Hunt Publishing, 2023.
Web Interface Traceability Matrix	TraceabilityMatrixA.xlsx
Logic Traceability Matric	TraceabilityMatrixB.xlsx
Storage Traceability Matrix	TraceabilityMatrixC.xlsx
EnrollEase SRS Document	EnrollEase.docx

Glossary

Access Token	Token received as proof of successful login, stored for future authorization attempts for a set amount of time
Additions	Any additions the user makes to the schedule
Authentication	Checking the user's credentials
Authentication Request	Carries the Credentials to the Federator in the right format
Authentication Response	Response from the Federator saying if validating the Credentials was a success or failure.
breakTime	The time between two classes.
breakTimeScore	A score representing how closely a SemesterPlan's breaks line up with the users break preference.
Class Data	Details on a specific Class at BYUI
classCodes	The codes representing a course. EX: CSE430.
Classes	Class Sections available for the student.
Course.	A course offered by the university.
coursesFromSchedule	Courses from an optimized schedule.
coursesTaken	The courses that a user has already taken.
Credentials	Login Credentials
Current Plan	Currently Selected Graduation Plan
Current Semester Schedule	The schedule for the current semester the student has selected
Deletion	any deletions the user makes to the schedule
Domain - String	Everything right of the @ of the email which determines the organization.

Retired Numbers

1.1.1.1.1 (simple email check)
1.1.1.1.2 (simple string manipulation)
1.1.1.1.3 (simple database request)
1.1.1.1.4 (redirect w/callback)
1.1.1.1.5 (simple display text)
1.1.1.1.6 (simple token storage)

1.1.1.2.1 (trivial)
1.1.1.2.2 (trivial)

1.1.1.3.1.1
1.1.1.3.2 (simple middleman)
1.1.1.3.3 (assembles HTML)
1.1.1.3.4 (makes small update)
1.1.1.3.5 (simple flag setter)

1.1.1.4.1 (simply extracts data)
1.1.1.4.2 (assembles HTML)

1.1.1.5 (simple implementation)
1.1.2.2
1.1.2.3
1.1.2.4
1.1.2.5

1.2.2.2.2
1.2.1.2.3
1.2.1.2.9
1.2.2.2.3
1.3.2.2
1.3.1.1.2
1.3.3.1.2
1..3.3.2.2
1.3.3.3.2
1.3.3.2.4
1.3.3.4.2
1.3.4
1.3.5
1.3.6
1.3.7