myADVISE
Software Design Description

Version 1.0
December 7, 2016

1. Introduction

1.1. Purpose

The purpose of this describes the architecture and design of the myADVISE application.

1.2. Scope

This document covers module design, dependencies, program flow, and other elements of the myADVISE application. The document is to provide a framework for the myADVISE application that is useful to current developers, future developers, and end users.

1.3. Definitions, acronyms, abbreviations

UofL	University of Louisville
MVC	Model-View-Controller: A common web application design pattern
Flight Plan	UofL Graduation Plan for students

2. References

1) *Software Engineering: Modern Approaches 2nd Edition*

3. Decomposition Description

The myADVISE application is described using three models: module, use case, and state. Each of these models is discussed below.

3.1. Module Decomposition

The myADVISE application is broken into three distinct modules: templates, view and objects. The relation between these modules is described in Figure 3.1. Each of these modules is discussed below.

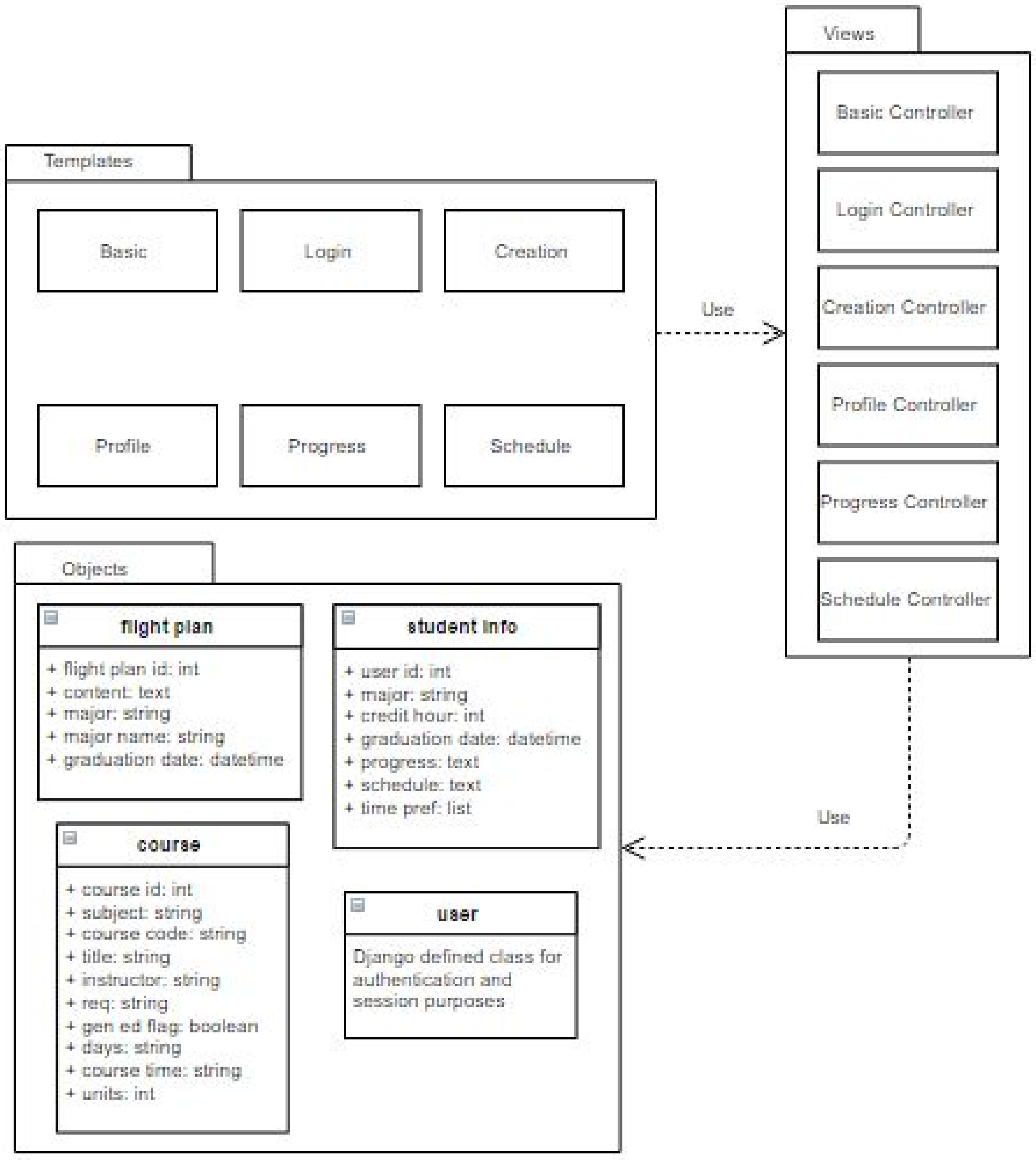


Figure 3.1: UML for modules

3.1.1. Templates

Templates act as Django's version of a traditional view layer. A template holds HTML, CSS, and element references that will be displayed as a web page. Each template is responsible for a page and holds references to each logical element needed. For example, the Login template holds the HTML, CSS, and any requests for the view to provide logic for elements rendered on the page.

3.1.2. View

The view layer can typically be considered a controller element. It acts as a go between templates and any database information they may require. The view coordinates requests made by each template and subsequently serves the necessary information. The classes defined in the object package are the views main form of serving data to the template.

3.1.3. Objects

The flight plan object consists of a flight plan id, a content string, a major string, a major name string, and a graduation date. The flight plan id is a unique identifier for the flight plan. The content string consists of the JSON for the courses in the flight plan. The major string consists of the abbreviation for the major of the specific flight plan (i.e., CECS, CE). Major name represents the full name of the flight plan's major (i.e. Computer Engineering & Computer Science). Graduation date represents the graduation date for the flight plan at hand.

The course class consists of a course id, a subject, a course code, a title string, an instructor string, a req string, a gen ed flag, a days string, a course time string, and a units integer. The course id uniquely identifies the course. Course code presents the number for the course (i.e. 505 in CECS 505). Subject represents the subject of the course (i.e. CECS in CECS 505). Title represents the full name of the course. Instructor stores the name of the professor/professors teaching the class. Req string stores prerequisites and corequisites for the course. The gen ed flag indicates if the course meets a general education requirement. The days string indicates which days of the week the course meets. The course time string indicates what time of the day the course meets. The units integer stores the number of credit hours for the course.

The student info class stores a user id, a major string, a credit hour integer, a list of time preferences, a graduation date, a progress string, and a schedule string.

The user id represents the specific user tied to the student class. The major string stores the major of the user. The credit hour field stores the desired number the student desires to take in a semester. The time preferences list stores a list of desirable course times for the user. The graduation date field stores the user’s graduation date. The progress string stores a JSON string that stores the user’s current progress. The schedule string stores a JSON string that represents a generated schedule.

The user class is a class provided by the Django framework that is used for authentication, session purposes and data storage.

3.2. Concurrent Process Decomposition

The myADVISE application features no concurrent processes.

3.3. Data Decomposition

The data structures used to communicate between modules are defined by the StudentInfo, Flight Plan, Courses and User classes. Each of these is described in section 3.1.3.

3.4. State Model Decomposition

The state model for the myADVISE application is included in Figure 3.2. Each state is based on a webpage that users can access.

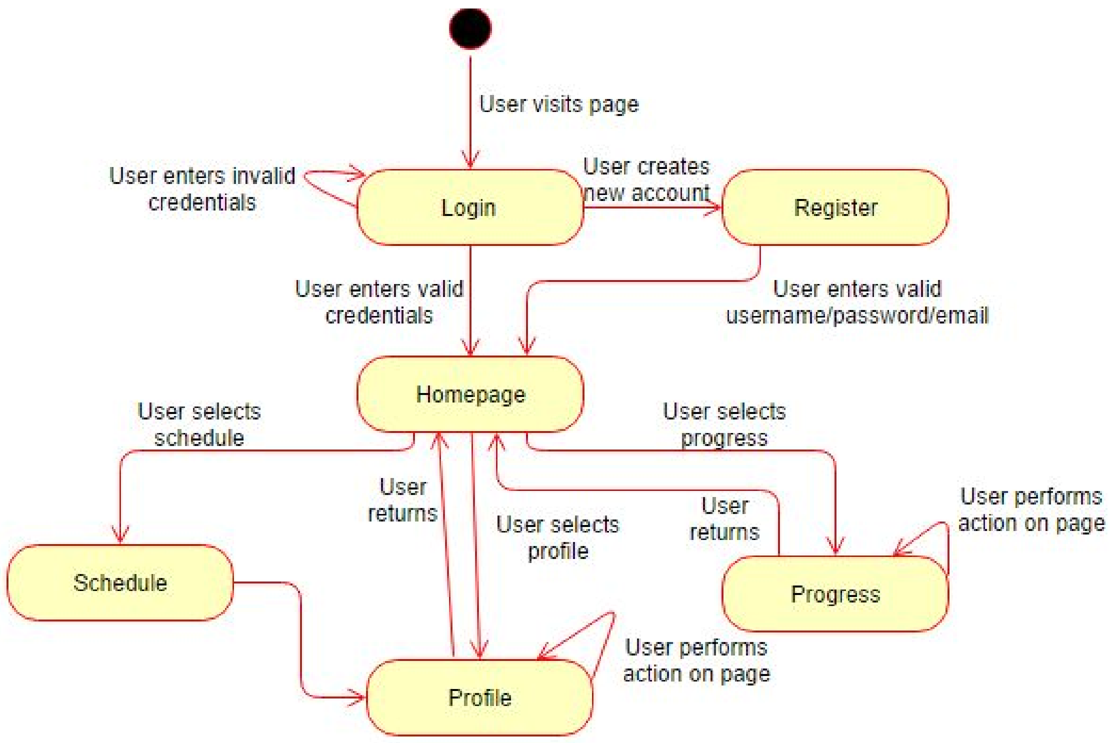


Figure 3.2: state diagram for myADVISE

3.5. Use Case Decomposition

The myADVISE application has 6 use cases: login, create account, generate schedule, edit/view profile, edit/view progress, and logout. A use case diagram is feature in figure 3.3.

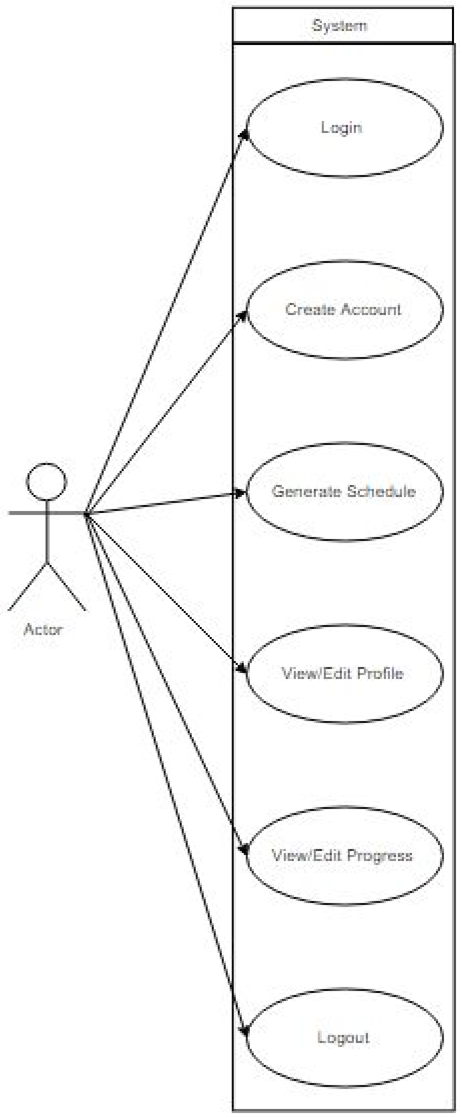


Figure 3.3: Use case diagram for myADVISE

3.5.1. Login Use Case

1) System displays user login box (username/password)
2) User enters credentials into login box
3) System verifies user credentials. If found to be valid, user is redirected to home page or destination page. If invalid, system displays error message and allows retry.

3.5.2. Create Account Use Case

1) System displays user login box (username/password)
2) User selects ‘Create new account’

3) System displays account creation page to user; account creation consists of choosing username, password, email and major
4) The system updates user database with newly acquired account information; the system redirects the user to the home page.

3.5.3. Edit/View Profile Use Case

1) The user selects the view profile option from the home page.
2) The server presents the profile page to the user, which includes the user's basic information, course preferences, and most recently generated schedule.
3) If the user chooses to edit their profile, the system updates the user's profile with the newly entered information.

3.5.4. Edit/View Progress Use Case

1) User selects progress page option from home page
2) System serves progress page to the user, displaying the user's current flight plan with any courses that have been previously taken
3) User is able to add/remove classes from their course history by checking/unchecking boxes beside corresponding courses. The user submits these changes by selecting the submit button at the bottom of the page. The system records these changes and updates the page.

3.5.5. Generate Schedule Use Case

1) User selects generate schedule page
2) System generates schedule for user based on user profile and flight plan progress
3) System redirects user to profile page where newly generated schedule is displayed.

3.5.6. Logout Use Case

1) User selects to logout.
2) System ends session and returns to home page.

4. Dependency description

This section describes dependencies for the decompositions introduced in Section 3.

4.1. Intermodule Dependencies

Packages are tightly coupled as each serves a distinct purpose in every transaction that occurs that relies on the other elements to function as intended. This separation of responsibilities is common in many web applications, and is intentionally done in such a way as to promote flexibility. User actions begin as a request made by the template layer, as it translates HTML, CSS, and client logic to a legible web page. Each request is sent to the template's corresponding view. The view layer handles these requests by performing the needed operations. These operations can vary greatly and are dependent on the request being made. The view layer has access to the object layer, and is therefore capable of storing/retrieving information from the database. The view layer is equally capable of interpreting and performing business logic based on requests. The object layer holds class information, and in this instance is primarily used as a data source to be accessed by the view layer. The dependencies are are follows: The template layer is dependent on the view layer for direction and request handling, just as the view layer is dependent on the object layer for data. These dependencies can be observed in Figure 3.1.

4.2. Interprocess Dependencies

The myADVISE application has no interprocess dependencies. Every action happens sequentially; no concurrent processes exist.

4.3. Data Dependencies

The data structures used to communicate between modules are defined by the StudentInfo, Flight Plan, Courses and User classes. These classes are essentially data stores for database relations, thus the data stored in these objects is dependent on the data in the database. No inter-object dependency exists.

4.4. State Dependencies

Each state is related to the states that transition from it. The state model is depicted in Figure 3.3

4.5. Use Case Dependencies

Each use-case aside from login is dependent on the login use case. Aside from these dependencies, all use cases are independent.

5. Interface Description

The Django web framework handles the vast majority of the interfacing that takes place in between modules as well as any communication that occurs with the database. Once provided with basic schemas and the names of each desired template, Django auto-generates and interfaces related elements in each layer of the application. Once classes are established in the object layer, Django creates corresponding queries and tables on the database to properly handle said classes.This allows interfacing with data in the database.

6. Detailed Design

This section details the non-trivial design aspects of the modules discussed in Section 3.

6.1. Template Detailed Design

6.1.1. Login Template

The login template is a login portal for the myADVISE application. The template features a single dialog box in the center of the main window consisting of two text fields and two buttons. CSS is derived from the Twitter Bootstrap API. The text fields are for the user to supply their username and password if applicable. The buttons are so that the user is either able to log in or if the user does not already have an account, can navigate to the registration/account creation page. This template request the login panel information from the login controller found on the view layer. Once the user submits their username/password or requests to be redirected, this information is sent to the login controller. A mockup of the login page is featured in the SRS.

6.1.2. Creation Template

The creation template again features a single dialog box, which is an element generated at the request of the template from the creation controller. Within this dialog box there exists three text fields, a drop down menu, and a button. CSS elements are dictated by Bootstrap. The text fields prompt the user for a username, email, and password respectively. The drop down box is used to specify the user’s desire major. The button will submit the form to the creation controller. A mockup of the creation page is featured in SRS.

6.1.3. Basic Template

The basic template acts as a homepage for the myADVISE application. It features three clickable regions, each linking the user to a different page. Requests corresponding to the region clicked. The homepage makes extensive use of bootstrap elements in it's CSS. A mockup of the homepage is featured in the SRS.

6.1.4. Profile Template

The profile template is made up of two windows, each window is made up of information received from the profile controller and wrapped in a bootstrap object. The leftmost element allows the user to update certain profile settings as well as specify optional preferences for schedule generation. The center element displays the user's current schedule as well as a progress bar displaying user's flight plan progress. Both elements on this page make heavy use of information sent from the view. A mockup of the profile template is featured in the SRS.

6.1.5. Progress Template

The Progress template is made up a single Bootstrap panel consisting of flight plan data received from the progress controller on page load. Each row of the table contains a single checkbox that allows the user to specify whether or not that course has been completed. At the bottom of the panel a button is displayed allowing users to save their updated course history. When pressed, the button sends a request to the progress controller to account for these changes. A mockup of the progress template is shown in the SRS.

6.1.6. Schedule Template

The schedule template consists of a single request sent to the schedule controller to begin the schedule generation process.

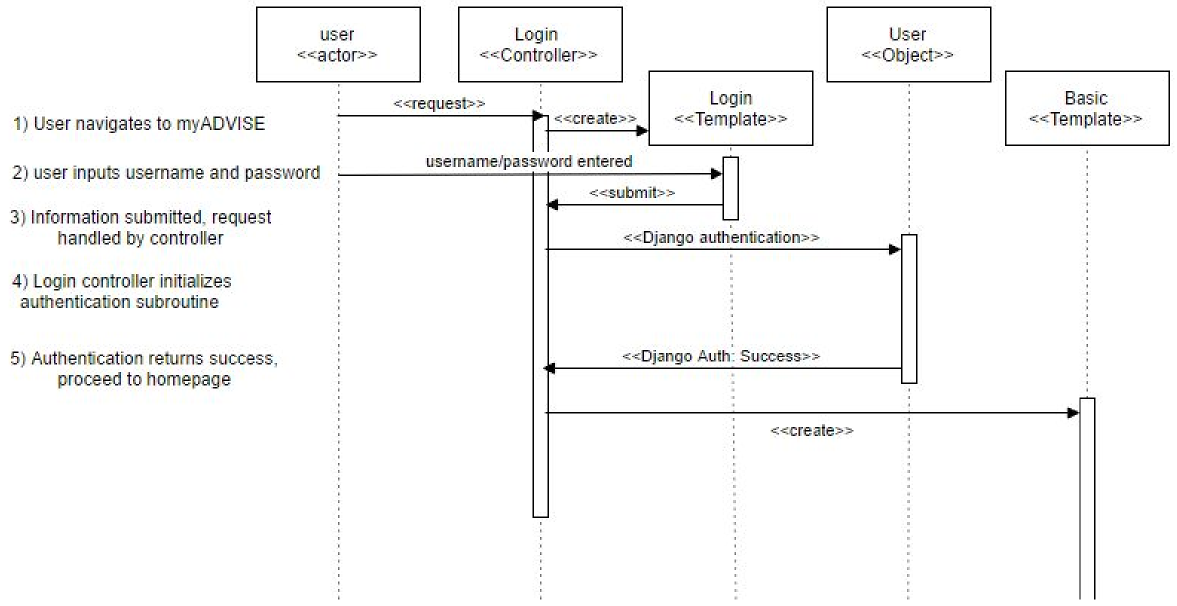
6.2. View Detailed Design

For the most part, Django handles the more difficult aspect of the website architecture. Requests to the views are very simple to implement. Aside from the scheduling algorithm, the architecture of the view controllers are fairly simple. The scheduling algorithm is described in detail using a program flowchart.

6.2.1. Login Controller

The login controller is responsible for coordinating user data and requests between the login template and user class in the object layer. If the user attempts to login and sends a login request, several interactions are required for this process to be completed. The authentication action is processed entirely by built-in Django functions. If this process returns from the object layer successfully, the controller will dispatch a redirect to the template layer to advance the user to the homepage. If this process fails to complete, or Django returns informing the controller that authentication fails, the controller reloads the login page for the user to try again.

If the user sends a registration request, the controller will direct the user to the creation template. A sequence diagram detailing the authentication process may be observed below.


user <<actor>>
Login <<Controller>>
Login <<Template>>
User <<Object>>
Basic <<Template>>
1) User navigates to myADVISE
2) user inputs username and password
3) Information submitted, request handled by controller
4) Login controller initializes authentication subroutine
5) Authentication returns success, proceed to homepage
<<request>>
<<create>>
username/password entered
<<submit>>
<<Django authentication>>
<<Django Auth: Success>>
<<create>>

6.2.2. Creation Controller

The creation controller handles requests and user data sent to and received from the creation template and object layer. The controller utilizes Django validation methods to ensure the user input is acceptable for each field it corresponds to.

6.2.3. Basic Controller

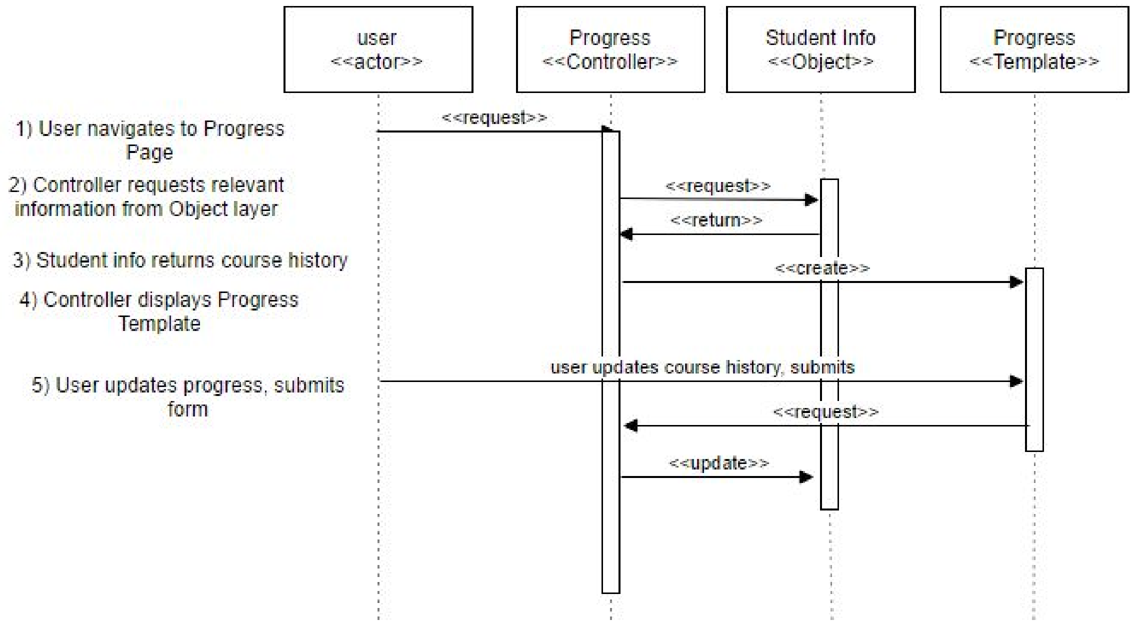
The basic controller handles requests made by the user from the basic template. Based on the request, the controller checks to make sure the user is logged in and then furnishes the request.

6.2.4. Profile Controller

The profile controller is responsible for handling requests between the profile template and the object layer. When the user attempts to navigate to the profile page, The controller requests user information including the user's major, the student name, the student's current schedule and flight plan progress. The user is able to submit changes and schedule preferences, which are forwarded to the profile controller and then sent to the object layer for submission.

6.2.5. Progress Controller

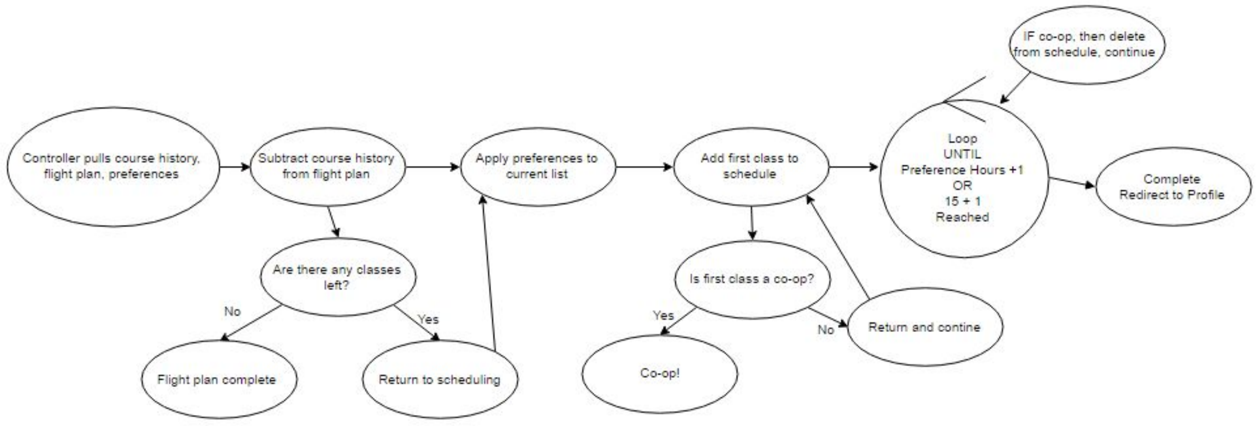
The progress controller handles the requests sent from the template, as well as the retrieving, parsing and displaying the user's flight plan information. When a request is sent to the progress controller to navigate to the progress page, the controller queries the object layer for all course information on the current user. Using the JSON object received from the object, it splits each course into rows and sends this information to be to the progress template for generation. If the user updates their course history and submits these changes via the button element located at the bottom of the rendered table. Upon clicking this button, a list is sent to the progress controller, which generates a JSON object and sends the newly updated course history to the object layer so that the user's information can be applied to their profile.



user <<actor>>
Progress <<Controller>>
Student Info <<Object>>
Progress <<Template>>
1) User navigates to Progress Page
2) Controller requests relevant information from Object layer
3) Student info returns course history
4) Controller displays Progress Template
5) User updates progress, submits form
<<request>>
<<request>>
<<return>>
<<create>>
user updates course history, submits
<<request>>
<<update>>

6.2.6. Schedule Controller

The schedule controller is responsible for generating a user's upcoming schedule. Upon request, the controller queries relevant classes for needed information in the form of JSON objects. After this data is aggregated, the controller initiates the generation algorithm which is described by the flow diagram below:



Controller pulls course history, flight plan, preferences
Subtract course history from flight plan
Apply preferences to current list
Add first class to schedule
Loop UNTIL Preference Hours +1 OR 15 + 1 Reached
IF co-op, then delete from schedule, continue
Complete Redirect to Profile
Are there any classes left?
No
Yes
Flight plan complete
Return to scheduling
Is first class a co-op?
Yes
No
Co-op!
Return and contine

After the schedule object has been completed, the controller creates an instance of the user's profile page and redirects the user there to display the new schedule as a bootstrap table element.

6.3. Objects Detailed Design

The Object module is sufficiently described in the 3.1.3. The classes found in the Object module are used only for data store purposes.

7. Table of Contents