“Milestone”

Software Architecture Document



**Jordan Mower**

**Jordan Nienaber**

**Jonathan Hua**

**Nick Luckey**

10/13/2019

Table of Contents

[**Introduction**](#_snx11c7s39z3) **2**

[**Architecture**](#_huki7ejdrkzx) **2**

[**2. Server Design**](#_k2e354846oml) **3**

[2.1 Server Module](#_t689pcbfwqhw) 3

[2.2 Communications Module](#_hh35sv42aho) 4

[2.3 User Module](#_hqmfqd9532j2) 5

[2.3.1 Authentication/Passwords Module](#_3cpe3jpdqhio) 6

[2.3.2 Course Module](#_ld82x7t5u2l7) 7

[2.3.3 Task Module](#_xz2rcut2hze2) 9

[2.3.4 User Information Module](#_rfszfggs1vr5) 10

[2.3.5 Subscriptions](#_5e4zu21ttwjt) 11

[**3. Client**](#_z1r0gql0uj8u) **13**

[3.1 Client Controller Module](#_ccm6jzobvxm3) 13

[3.1.1 User Information Controller](#_cqzsho7f9z2m) 13

[3.1.2 Course Controller](#_h9atgktzttkv) 15

[3.1.3 Subscription Controller](#_z0b6vgq6bs82) 18

[3.1.4 Task Controller](#_p58bzvri1h5l) 19

[3.2 Communications Module](#_sv36dws6zawc) 22

[3.3 Client View Module](#_4mbqiwz8vhv7) 22

[3.3.1 Task View](#_vgp4htkimmrk) 23

[3.3.2 Course View](#_zbsanbvyt2cp) 24

[3.3.3 Home Page View](#_uje5zo00or2) 25

[3.3.4 Search View](#_lwu9bxysficx) 25

[3.3.5 Log In View](#_ce80abnkhyfz) 25

[3.3.6 Sign Up View](#_wykszy4pfknv) 26

# 

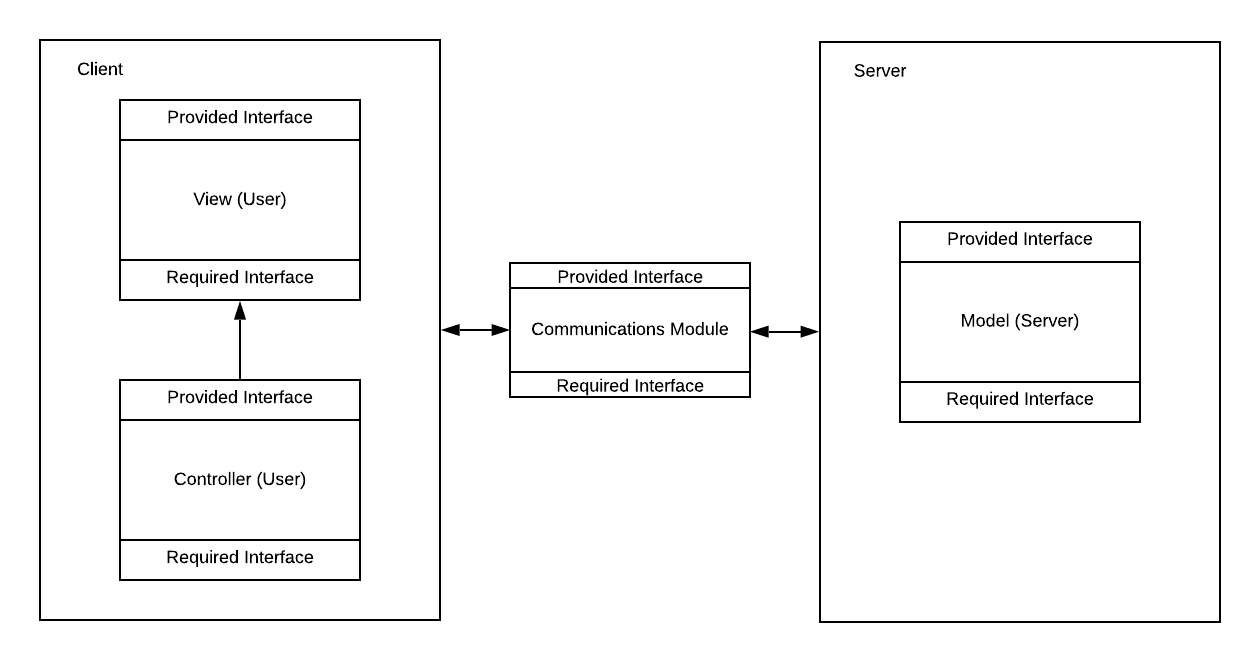
# 

# Introduction

The “Milestone” application consists of users who would like to keep track of various course assignments. Users can have many courses that are stores for the tasks that they create. With this in mind, it is important to have a data store that houses all the necessary information that users generate and also require. For this reason, it is important to separate the data store, controlling logic, and views from each other so that the system is easily extensible, maintainable, and readable.

# Architecture

**Figure 1-1: High-Level Architecture Diagram**



The “Milestone” application will make use of Model-View-Controller architecture. This suits our application well because of the three elements that are consistent with our needs. Our clients will be interacting with their view on the mobile devices which in turn will update the model or be updated by the model. The controller will provide the necessary ability to update the model and view based on the interactions with data components. The model provides us with the capabilities to store, access, and manipulate the necessary data for display and control.

**Server (AWS Amplify Framework)**

Please see chapter 2 for a detailed description of the server component.

**Client (Android Mobile Device)**

Please see chapter 3 for a detailed description of the client component.

# 2. Server Design

## 2.1 Server Module

**Purpose**

The purpose of this module is to provide a data store where information created by users and the system can be stored, manipulated, searched, and retrieved.

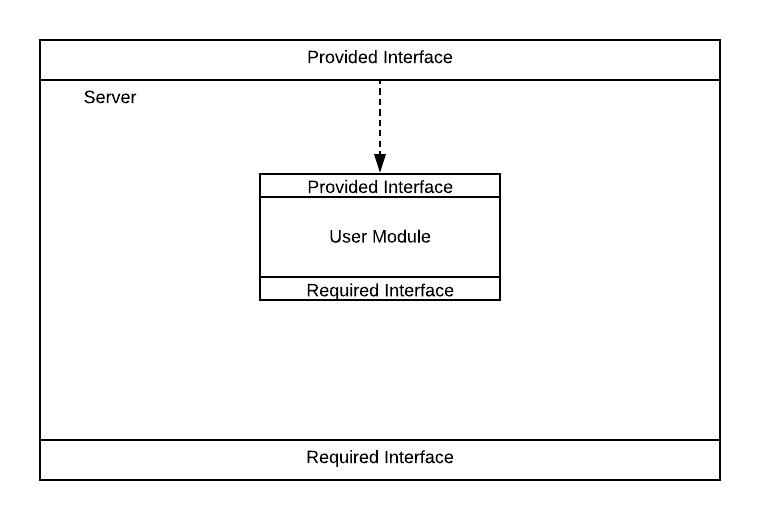
**Rationale**

This module is created because it provides a central location for data while also providing encapsulation. Data stored on the server would include user details, account authentication, courses, tasks, and subscribed courses.

**High-Level Server Design**

The server module consist of the user module which handles functions in regards to user requests for data.

**Figure 2-1: High-Level Server Design**



**Required Interface**

There is no required interface

**Provided Interface**

The provided interface is the communications interface which is described in section 2.2

## 2.2 Communications Module

**Purpose**

The purpose of the communications module is to provide services for exchange of information between the users of the system and the server. This module represents the various aspects of the AWS ecosystem that provide this functionality to the application.

**Rationale**

This module encapsulates and centralizes communications between clients and the server as well as providing abstraction from AWS services.

**Required Interface**

This module has no required interfaces.

**Provided Interface**

The provided interface is the interface provided by the Server Module and the Client Module. This module merely facilitates the communication between client and server.

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 2.3 User Module

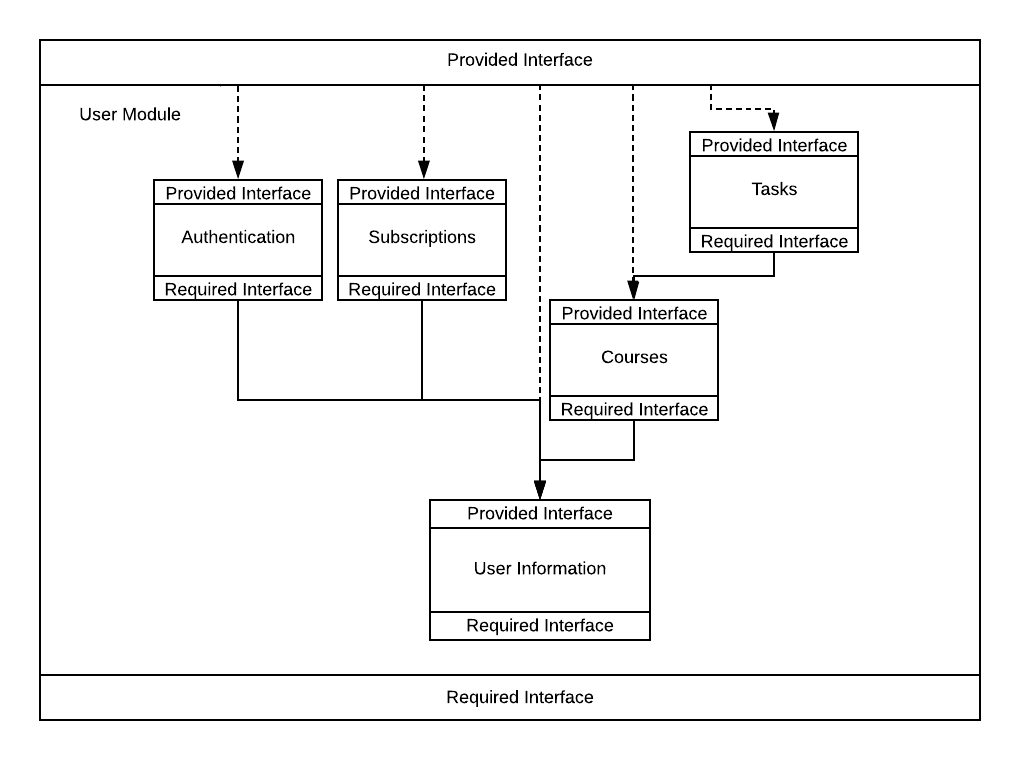
**Purpose**

The purpose of this module is to authenticate users and to provide storage and management functions for users personal information, courses, tasks, subscriptions and user information.

**Rationale**

This module is created to centralize the management of all the services that users will require and to facilitate their storage, access, and manipulation.

**High-Level Module Design**

**Figure 2-2: High-Level User Module Design**

**Required Interface**

This module has no required interface.

**Provided Interface**

The provided interface of this module is the union of the submodules:

* Tasks
* Authentication
* Subscriptions
* Courses
* User Information

These submodules are described in detail below.

## 

### 2.3.1 Authentication/Passwords Module

**Purpose**

The purpose of this module is to authenticate users username and password combinations. This module will also allow users to change their passwords and provide them privileges.

**Rationale**

This module is created to centralize authentication and password management.

**Required Interface**

Boolean userExists(String uName);

**Provided Interface**

Void setPassword(String uName, String pword);

throws NoSuchUserException;

**Description:**

Stores a users password and username combination

**Parameters:**

**uName**: Username of the user whose setting a password

**pWord**: Password of the user

**Exceptions:**

NoSuchUserException: If the user does not exist.

Boolean logOn(String uName, String pword) throws NoSuchUserException,InvalidPasswordUsernameCombination;

**Description:**

Validates a username and password combination.

**Parameters:**

**uName**: The users username whose signing in.

**Pword**: The users password of the user signing in.

**Returns:**

True if credentials are correct, false otherwise.

**Exceptions:**

**NoSuchUserException**: Given user does not exist.

**InvalidPasswordUsernameCombo**: Incorrect login credentials.

Void logOff(String uName) throws UserNotLoggedOnException;

**Description:**

Logs a user out from the system.

**Parameters:**

**uName**: The username of the user who is logging off.

**Exceptions:**

**UserNotLoggedOnException**: If the given user is not logged into the system.

**Module ADTs**

Username: See User Information section

String Password;

### 2.3.2 Course Module

**Purpose**

The purpose of this module is to provide access to and management of persistent data store for course related information.

**Rationale**

This module is created to centralize and encapsulate storage of course information and services.

**Required Interface**

Boolean userExists(Username uName);

**Provided Interface**

Void addCourse(String uName, String courseName, String instructorName, Enum meetingDays, Color color) throws CourseAlreadyExistsException, InvalidCharactersException;

**Description:**

Adds a course to the logged on users courses in the data store.

**Parameters:**

**uName**: Username of the logged in user.

**courseName**: user generated course name. (Alphanumeric, 1-20 characters)

**instructorName**: user generated instructor name. (Alphabetical, 1-25 characters)

**meetingDays**: The days the user selected in which the class is held. (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday)

**Color**: The color of the course.

**Exceptions:**

**CourseAlreadyExistsException**: If the user attempts to add a course that already exists within their set of courses.

**InvalidCharactersException**: If the fields contain invalid characters.

Void deleteCourse(String uName,String courseName) throws NoSuchCourseException, NoOwnershipException;

**Description:**

Deletes a course from the logged on users courses in the data store.

**Parameters:**

**uName**: Username of the logged in user.

**courseName**: The course name of the course to be deleted.

**Exceptions:**

**NoSuchCourseException**: If the entered course name does not exist.

**NoOwnershipException**: If the entered course name was not created by logged in user.

Void updateCourse(String uName, String courseName) throws NoSuchCourseException, NoOwnershipException;

**Description:**

Allows a user to edit an existing course in the data store.

**Parameters:**

**uName**: Username of the logged in user.

**courseName**: The name of the course the user wants to edit.

**Exceptions:**

**NoSuchCourseException:** If the entered course name does not exist.

**NoOwnershipException**: If the entered course name was not created by the logged in user.

Course[] searchCourses(String courseName);

**Description:**

Allows a user to search the data store for courses.

**Parameters:**

**courseName:** The name of the course they are searching for.

**Returns:**

A list of courses matching the course name.

Boolean courseExists(String uName, String courseName)

**Description:**

Verifies whether or not a course exists for that user in the data store.

**Parameters:**

**uName:** The logged in users username.

**courseName:** The course name that is being verified.

**Returns:**

True if the course exists for that user, false otherwise.

**Module ADTs**

uName: See User Information Module 2.3.4

Tasks:See Tasks Module 2.3.3

Course {

String courseName;

String instructorName

Enum meetingDays {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

Color color;

Boolean isOwner;

}

### 2.3.3 Task Module

**Purpose**

The purpose of this module is to allow users access to task information and a persistent store of task information. All tasks must belong to a course.

**Rationale**

This module is created to centralize and encapsulate task management services.

**Required Interface**

Course: See Course Module 2.3.2

Boolean courseExists(String uName, String courseName)

**Provided Interface**

Void addTask(Course courseName, String taskTitle, Date dueDate, int priority, double percentage, String comments)throws NoSuchCourseException, NoCourseOwnershipException;

**Description:**

Adds a task to the selected course in the data store. User must own the selected course to add tasks.

**Parameters:**

**courseName:** Name of the course that the task is being added to.

**taskTitle:** Title of the task to be added. (8-20 alphanumeric)

**dueDate:** The date that the task is due to be completed. (date must be from present to future)

**priority:** The priority level of the task. (0-3)

**percentage:** The percentage value of the task. (0-100)

**comments:** Additional comments relating to the task. (0-255 alphanumeric)

**Exceptions:**

**NoSuchCourseException:** If the course does not exist.

**NoCourseOwnershipException:** If the user does not own the course.

Void updateTask(String taskTitle) throws NoSuchCourseException, NoCourseOwnershipException;

**Description:**

Allows the user to post changes to a selected task in the data store if they own it.

**Parameters:**

**taskTitle:** The name of the task

**Exceptions:**

**NoSuchCourseException:** If the course does not exist.

**NoCourseOwnershipException:** If the user does not own the course.

Void deleteTask(String taskTitle) throws NoSuchCourseException, NoCourseOwnershipException;

**Description:**

Allows the logged in user to delete a task from a course in the data store so long as they own it.

**Parameters:**

**taskTitle:** The title of the task to be deleted by the user.

**Exceptions:**

**NoSuchCourseException:** If the course does not exist.

**NoCourseOwnershipException:** If the user does not own the course.

Void completeTask(String taskTitle) throws TaskAlreadyCompletedException, TaskPastDueDateException;

**Description:**

Allows the user to confirm the completion of a task in the data store.

**Parameters:**

**taskTitle:** The title of the task to be marked as completed.

**Exceptions:**

**TaskAlreadyCompletedException:** If the task is already completed.

**TaskPastDueDateException:** If the tasks due date is in the past.

**Module ADTs**

Task {

String taskTitle;

Date dueDate;

int priority;

double percentage;

String comments;

}

## 

### 2.3.4 User Information Module

**Purpose**

The purpose of this module is to provide access to and a persistent data store for user information module.

**Rationale**

This module is created to centralize user information and provide management functions.

**Required Interface**

This module has no required interface.

**Provided Interface**

Void addUserInfo(UserInformation u) throws UserAlreadyExistsException;

**Description:**

Stores a users information in the data store if no user with that username exists yet.

**Parameters:**

**u:** User information to store in the data store.

**Exceptions:**

**UserAlreadyExistsException:** If the username attempting to be used already exists.

UserInformation getUserInformation(String uName) throws NoSuchUserException;

**Description:**

Retrieves a users information from the data store.

**Parameters:**

**uName:** The user’s username.

**Exceptions:**

**NoSuchUserException:** The username used does not exist.

boolean userExists(String uName);

**Description:**

Determines if a user exists within the data store.

**Parameters:**

**uName:** The user’s username to validate against the data store.

**Returns:**

True if the user exists within the data store, false otherwise.

Void deleteUserInformation(String uName) throws NoSuchUserException;

**Description:**

Removes a user’s information from the data store.

**Parameters:**

**uName:** The username to be removed from the data store.

**Exceptions:**

**NoSuchUserException:** If the user does not exist within the data store.

**Module ADTs**

Course: See module 2.3.2

UserInformation {

String username;

String schoolName;

Int yearOrGrade;

Date dateOfBirth;

}

### 2.3.5 Subscriptions

**Purpose**

The purpose of the subscriptions module is to provide access to and manipulation of subscriptions for users in a persistent data store.

**Rationale**

This module is created to encapsulate and centralize data storage and retrieval related to subscriptions.

**Required Interface**

Boolean userExists(String uName);

**Provided Interface**

Void addSubscription (String uName, String courseName);

**Description:**

Adds a subscription to a course in the logged in users data store.

**Parameters:**

**uName:** The creators username.

**courseName:** The name of the course in which the user wishes to subscribe.

Void removeSubscription (String uName, String courseName);

**Description:**

Removes a subscription from a logged in users subscriptions in the data store.

**Parameters:**

**uName:** The name of the logged in user who is removing the subscription.

**courseName:** The name of the course to be removed.

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

# 3. Client

## 3.1 Client Controller Module

**Purpose**

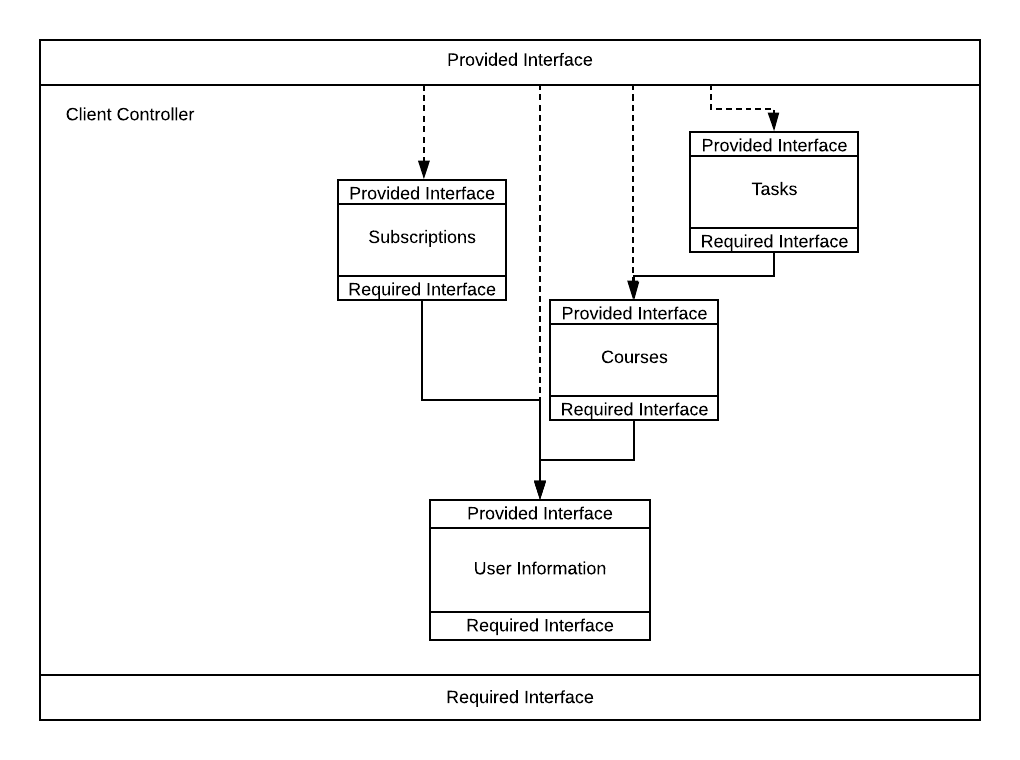
The purpose of this module is to provide the user client control functions that update the model and the view. This is the software that affects the view and also changes information in the model. Its purpose is to define the way the view reacts to user input and maintain view-model consistency.

**Rationale**

This module is created to relieve the view of having to process both data manipulation and presentation.

**High-Level Module Design**

**Figure 3-1: High-Level Client Controller Module (This image may change)**

****

**Required Interface**

Communication Module: See section 3.2

**Provided Interface**

The provided interface is the union of the submodules:

* Subscription Controller
* Task Controller
* Course Controller
* User Information Controller

### 3.1.1 User Information Controller

**Purpose**

The purpose of this module is to provide controlling functions on the client side for user information data.

**Rationale**

This module is created to centralize controlling duties for user information.

**Required Interface**

This module has no required interface.

**Provided Interface**

Void addUserInformation(String username, String password, Date birthday, String school, int gradeOrYear);

**Description:**

Adds a user to the model.

**Parameters:**

**Username:** The users username.

**Password:** The users password.

**Birthday:** Date of birth.

**School:** School attending if any.

**gradeOrYear:** Year in college, or grade in school.

UserInformation getUserInformation(String username) throws NoSuchUserException;

**Description:**

Gets a users information from the model.

**Parameters:**

**Username:** The users name.

**Exceptions:**

**NoSuchUserException:** If the user does not exist in the model.

**Returns:**

The users information.

Void editUserInformation(String username)

**Description:**

Allows a user to edit their information.

**Parameters:**

**Username:** The logged in users username.

Void validateUserInformation(String username, String password, Date birthday, String school, int gradeOrYear);

**Description:**

Validates user input before being passed to the model.

**Parameters:**

**Username:** The users username.

**Password:** The users password.

**Birthday:** Date of birth.

**School:** School attending if any.

**gradeOrYear:** Year in college, or grade in school.

**Returns:**

True if input is valid, false otherwise.

Boolean validateLogOnInfo(String username, password)

**Description:**

Validates login information.

**Parameters:**

**Username:** The users username.

**Password:** The users password.

**Returns:**

True if login details are valid, false otherwise.

Boolean userLoggedOn();

**Description:**

Determines if a user is logged on or not.

**Returns:**

True if the user is logged on, false otherwise.

**Module ADTs**

UserInformation {

String userName;

String password;

Date birthday;

String school;

Int gradeOrYear;

}

### 3.1.2 Course Controller

**Purpose**

The purpose of this module is to provide controlling functions on the client side for course information.

**Rationale**

This module is created to centralize controlling duties for course information.

**Required Interface**

This module has no required interface.

**Provided Interface**

Course[] searchCourses(String searchCriteria);

**Description:**

Searches all the courses in the model matching a specified criteria.

**Parameters:**

**searchCriteria:** The string the user is searching.

**Returns:**

A list of courses matching the search criteria.

Void addCourse(String courseName, String instructorName, int meetingDays, Color color);

**Description:**

Adds a course to the model.

**Parameters:**

**courseName:** The course name.

**instructorName:** The instructors name.

**meetingDays:** The meeting days of the course (Enum days of the week)

**Color:** The selected course color.

Course[] getAllCoursesFromModel(String userName) throws NoSuchUserException;

**Description:**

Gets all courses for a selected user.

**Parameters:**

**userName:** The logged in users username.

**Exceptions:**

**NoSuchUserException:** If the specified user does not exist.

**Returns:**

A list of all the courses for that user.

Course getCourseFromModel(String courseName) throws NoSuchCourseException;

**Description:**

Gets a single course from the model based on the course name.

**Parameters:**

**courseName:** The course name.

**Exceptions:**

**NoSuchCourseException:** If the specified course does not exist.

**Returns:**

The course that was requested.

Course getCourseFromView();

**Description:**

Fetches course details from the course view. (Response from user input)

**Returns:**

The course information from the view.

Void editCourse(String courseName) throws NoSuchCourseException;

**Description:**

Allows the client to edit a course by course name.

**Parameters:**

**courseName:** The specified course name.

**Exceptions:**

**NoSuchCourseException:** If the specified course does not exist.

Void deleteCourse(String courseName);

**Description:**

Deletes a course from the model with the specified course name.

**Parameters:**

**courseName:** The name of the course to be deleted.

**Exceptions:**

**NoSuchCourseException:** If the specified course does not exist.

Boolean validateCourseInfo(String courseName, String instructorName, int meetingDays) throws InvalidCourseCharactersException;

**Description:**

Validates course information.

**Parameters:**

**courseName:** The course name.

**instructorName:** The instructors name.

**meetingDays:** The meeting days of the course (Enum days of the week)

**Color:** The selected course color.

**Returns:**

Returns true of course information is valid, false otherwise.

**Module ADTs**

Course {

Task[] tasks;

String courseName;

String instructorName;

Enum meetingDays {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}

Color color;

}

### 3.1.3 Subscription Controller

**Purpose**

The purpose of this module is to provide controlling functions on the client side for subscription information.

**Rationale**

This module is created to centralize controlling duties for subscription information.

**Required Interface**

This module has no required interface.

**Provided Interface**

Void addSubscription(String courseName, String creatorName)

**Description:**

Adds a subscription to the logged in users subscriptions.

**Parameters:**

**courseName:** The specified course name.

Void updateSubscriptions(String userName) throws NoSuchSubscriptionException;

**Description:**

Updates the logged in user’s subscriptions from the model.

**Parameters:**

**userName:** The logged in user’s username.

**Exceptions:**

**NoSuchSubscriptionException:** If the subscription is not registered to the user.

Subscription[] getSubscriptionsFromModel() throws NoSuchSubscriptionException;

**Description:**

Gets all the subscriptions of the logged in user.

**Exceptions:**

**NoSuchSubscriptionException:** If the user has no subscriptions.

**Returns:**

A list of all the logged in users subscriptions.

Void removeSubscription(String courseName)throws NoSuchSubscriptionException;;

**Description:**

Removes a subscription from the logged in users subscriptions.

**Parameters:**

**courseName:** The name of the course to be removed from subscriptions.

**Exceptions:**

**NoSuchSubscriptionException:** If the specified subscription does not exist

**Module ADTs**

Course[] subscribedCourses;

Subscriptions {

String courseCreator;

String courseName;

}

### 3.1.4 Task Controller

**Purpose**

The purpose of this module is to provide controlling functions on the client side for task information.

**Rationale**

This module is created to centralize and coordinate functions related to task information.

**Required Interface**

Course Controller: See section 3.1.2

**Provided Interface**

Void addTask(String courseName, String taskTitle, Date dueDate, int priorityLevel, double taskPercentage, String comments);

**Description:**

Adds a task to the users data store. (Responds to user input)

**Parameters:**

**courseName:** The course name.

**taskTitle:** That tasks title.

**dueDate:** The tasks due date.

**priorityLevel:** The task priority level.

**taskPercentage:** The task percentage.

**comments:** Additional notes from the user.

Task[] getAllTasksByDayFromModel(String courseName, Date day) throws NoTasksException;

**Description:**

Retrieves all tasks from the model for specified date for the logged in user.

**Parameters:**

**courseName:** Specified course name.

**Day:** Date of the requested tasks.

**Returns:**

A list of tasks for that day, empty if no tasks exist.

Task getTaskFromModel(String courseName, String taskTitle) throws NoSuchTaskExistsException, NoSuchCourseExistsException;

**Description:**

Gets a specific task from the model for the logged in user.

**Parameters:**

**courseName:** The name of the course.

**taskTitle:** The name of the task.

**Exceptions:**

**NoSuchTaskExistsException:** If the specified task does not exist.

**NoSuchCourseExistsException:** If the specified course does not exist.

**Returns:**

The specified task.

Task getTaskFromView();

**Description:**

Gets task information from the task view. (from user input).

**Returns:**

Task information

Task editTask(String courseName, String taskTitle) throws NoSuchTaskExistsException, NoSuchCourseExistsException;

**Description:**

Allows the user to edit a task from the model.

**Parameters:**

**courseName:** Name of the course.

**taskTitle:** Name of the task.

**Exceptions:**

**NoSuchTaskExistsException:** If the specified task does not exist.

**NoSuchCourseExistsException:** If the specified course does not exist.

**Returns:**

The task to be edited.

Void completeTask(String courseName, String taskTitle);

**Description:**

Marks a task as complete.

**Parameters:**

**courseName:** The name of the course.

**taskTitle:** The name of the task.

Void deleteTask(String courseName, String taskTitle) throws NoSuchTaskExistsException, NoSuchCourseExistsException;

**Description:**

Deletes a task from the user’s data store.

**Parameters:**

**courseName:** The course name.

**taskTitle:** The task title.

**Exceptions:**

**NoSuchTaskExistsException:** If the specified task does not exist.

**NoSuchCourseExistsException:** If the specified course does not exist.

Void expireTask(String courseName, String taskTitle) throws NoSuchTaskExistsException, NoSuchCourseExistsException;

**Description:**

Expires a task that is past due.

**Parameters:**

**courseName:** The course name.

**taskTitle:** The task title.

**Exceptions:**

**NoSuchCourseExistsException:** If the course does not exist.

**NoSuchTaskExistsException:** If the task does not exist.

Boolean validateTaskInfo(String courseName, String taskTitle, Date dueDate, int priorityLevel, double taskPercentage, String comments);

**Description:**

Validates that all task info is correct.

**Parameters:**

**courseName:** The course name.

**taskTitle:** That tasks title.

**dueDate:** The tasks due date.

**priorityLevel:** The task priority level.

**taskPercentage:** The task percentage.

**comments:** Additional notes from the user.

**Returns:**

Returns true if task info does not violate validity check, false otherwise.

**Module ADTs**

Task {

String taskTitle;

Date dueDate;

Int priorityLevel;

Double taskPercentage;

String comments;

}

## 

## 3.2 Communications Module

**Purpose**

The purpose of the communications module is to provide services for exchange of information between the users of the system and the server. This module represents the various aspects of the AWS ecosystem that provide this functionality to the application.

**Rationale**

This module encapsulates and centralizes communications between clients and the server as well as providing abstraction from AWS services.

**Required Interface**

This module has no required interfaces.

**Provided Interface**

The provided interface is the interface provided by the Server Module and the Client Module. This module merely facilitates the communication between client and server.

## 3.3 Client View Module

**Purpose**

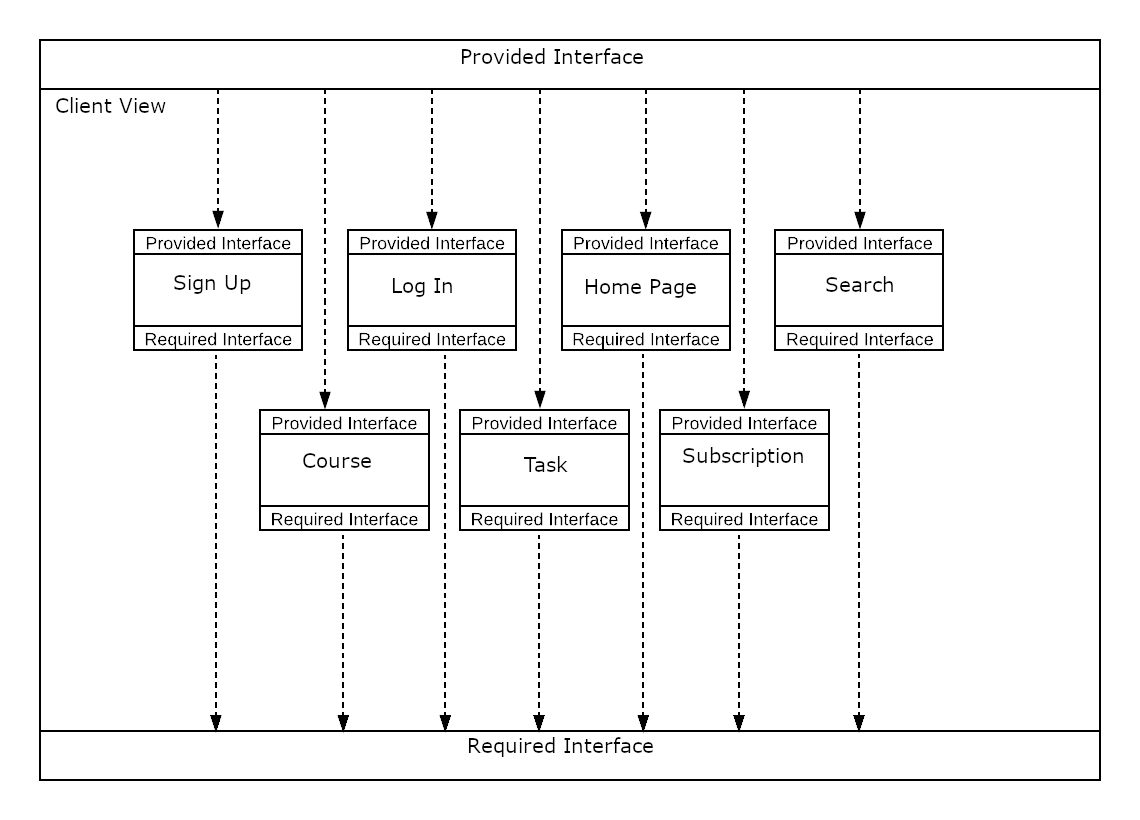
The purpose of this module is to handle all graphical displays that are presented to the user. It may be updated indirectly by the model or be asked to redraw itself by the controller.

**Rationale**

This module is created so that the presentation is independent of the business logic and the handling of user-input events.

**High-Level Module Design**

**Figure 3-2: High Level View Module**

****

**Required Interface**

Controller Module: See section 3.1

Communications Module: See section 3.2

**Provided Interface**

The provided interface for this module is the union of the submodules:

* Task View
* Course View
* Home Page View
* Search View
* Log In View
* Sign Up View
* Subscriptions View

### 3.3.1 Task View

**Purpose**

The purpose of this module is to provide graphical display of tasks.

**Rationale**

This module is created to centralize the graphical display of tasks to the user.

**Required Interface**

Controller Module: See section 3.1

Communications Module: See section 3.2

**Provided Interface**

Void taskView(Task[] tasks);

**Description:**

Receives tasks from the model or controller and displays them.

**Parameters:**

**Tasks:** A list of tasks to display.

Void editTaskView(Task task);

**Description:**

Creates a display to receive a task to be edited from the controller or the model.

**Parameters:**

**task:** The task to be edited.

Task addTaskView()

**Description:**

Creates the display for the user to add tasks.

### 3.3.2 Course View

**Purpose**

The purpose of this module is to provide graphical display of courses.

**Rationale**

This module is created to centralize the presentation of courses to the user.

**Required Interface**

Controller Module: See section 3.1

**Provided Interface**

void courseView();

**Description:**

Creates a graphical interface to present courses.

Void addCourseView();

**Description:**

Creates a graphical interface that allows users to add courses.

Void editCourseView(Course course);

**Description:**

Creates a graphical interface that allows users to edit courses. Receives a course to be edited from the model or controller.

**Parameters:**

**course:** The course to be edited.

### 3.3.3 Home Page View

**Purpose**

The purpose of this module is to provide presentation details of the home page to the user.

**Rationale**

This module is created to centralize the presentation of home page details to the user.

**Required Interface**

Controller Module: See section 3.1

**Provided Interface**

homePageView(Course[] courses);

**Description:**

Creates a graphical interface that displays course and task information in a calendar format.

**Parameters:**

**courses:** All the courses that need to be displayed in the calendar.

### 3.3.4 Search View

**Purpose**

The purpose of this module is to provide a search view to the user.

**Rationale**

This module is created to centralize the presentation of the search view.

**Required Interface**

**Provided Interface**

searchView();

**Description:**

Creates a graphical interface that allows a user to search for courses to subscribe to.

### 3.3.5 Log In View

**Purpose**

The purpose of this module is to provide a login view to the user.

**Rationale**

This module is created to centralize presentation details of the login view.

**Required Interface**

This module has no required interface.

**Provided Interface**

logOnView();

**Description:**

Creates a graphical interface that allows the user to log in to the application

### 

### 

### 

### 3.3.6 Sign Up View

**Purpose**

The purpose of this module is to provide a sign-up view to the user.

**Rationale**

This module is created to centralize presentation details of the sign-up view.

**Required Interface**

This module has no required interface.

**Provided Interface**

signUpView();

**Description:**

Creates a graphical interface that allows the user to sign up for the application.