

Team 2- ChalkBox

## **Software Architecture Document (SAD)**

Team members:

Yidi Kou

Prathima Devanath

Yu Zhao

Yigang Yao

Karthik Kembhavi

## 1. Product Overview

Learning Management System is an web application serving students and instructors with a virtual class experience. The system is focused on improving learning and teaching experience by providing user-friendly interfaces and features.

The System allows instructors to view his/her classes and students enrolled in the courses.

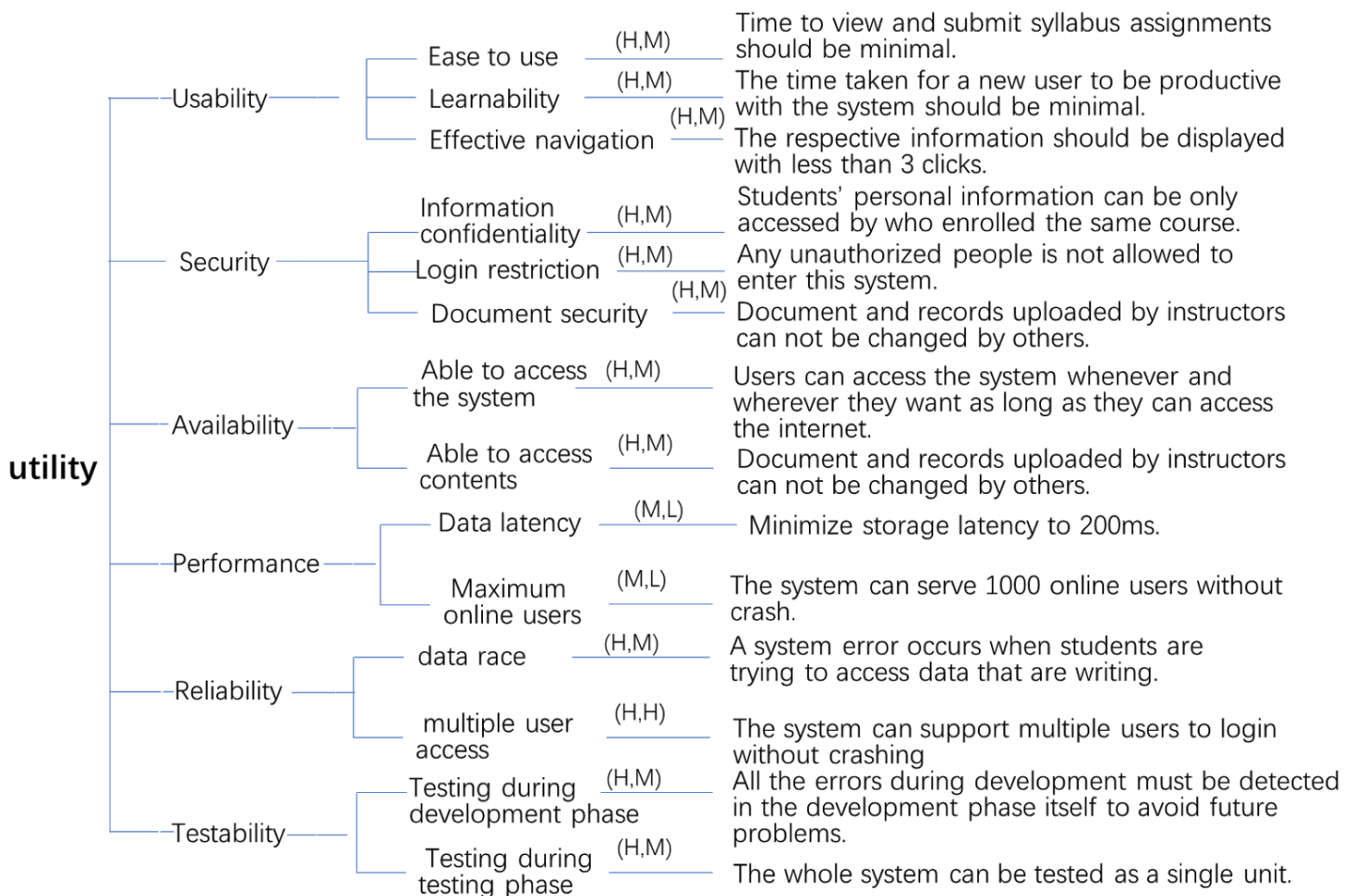
Instructors can create, edit and delete course syllabus and assignments, as well as review and grade students' submissions. Students are allowed to view syllabus and assignments. When they finish their assignments, they can submit their assignments to professor through the system.

After professor grades their assignment, they can check the grades of their submissions.

Meanwhile, the system can automatically create and maintain a grade book for each course, which facilitates users to calculate the total grade.

## 2. Architecture Drivers

### (2.1) Utility Tree



## (2.2) Scenarios

<b>Quality attribute</b>	Usability
<b>Scenario</b>	Ease of use
<b>Source</b>	User(student/Instructor)
<b>Stimulus</b>	users want to use the system
<b>Artifact</b>	The System
<b>Environment</b>	At Runtime
<b>Response</b>	Access syllabus, assignments and grades, submit assignments/grades/syllabus.
<b>Response Measure</b>	Time to view and submit syllabus, assignments and grades should be minimal (less than 3 seconds).

<b>Quality attribute</b>	Usability
<b>Scenario name</b>	Effective navigation
<b>Source</b>	A user(Student/Instructor)
<b>Stimulus</b>	Navigates the site in search of a page he/she needs
<b>Artifact</b>	Website
<b>Environment</b>	At runtime
<b>Response</b>	Displays the right page when the respective tab/link is clicked.
<b>Response measure</b>	The respective information should be displayed with less than three clicks

<b>Quality attribute</b>	Usability
<b>Scenario name</b>	Learnability
<b>Source</b>	A New User(Student/Instructor)
<b>Stimulus</b>	A New user tries to use the system
<b>Artifact</b>	Website
<b>Environment</b>	Normal operation
<b>Response</b>	A first-time user is able to login to the system and move through pages without any additional help.
<b>Response measure</b>	The time taken for new user to be productive the first time he/she uses the system should be minimal.

<b>Quality attribute</b>	Security
<b>Scenario name</b>	Login restriction
<b>Source</b>	Unauthorized people
<b>Stimulus</b>	Try to log in without out correct account
<b>Artifact</b>	The system
<b>Environment</b>	At runtime
<b>Response</b>	Displays the notice of denying invalid login activity.
<b>Response measure</b>	Identifies 99% invalid account and password.

<b>Quality attribute</b>	Availability
<b>Scenario name</b>	Able to access the system

<b>Source</b>	Users(Student/Instructor)
<b>Stimulus</b>	Users want to use the system and tries to access it.
<b>Artifact</b>	website
<b>Environment</b>	Run time
<b>Response</b>	The system should be accessed easily and conveniently
<b>Response measure</b>	Whether users can access the system or not

<b>Quality attribute</b>	Availability
<b>Scenario name</b>	Able to access the contents
<b>Source</b>	Users
<b>Stimulus</b>	Users want to access the information they need and tries to access them.
<b>Artifact</b>	website
<b>Environment</b>	Run time
<b>Response</b>	The content should be accessed easily and conveniently
<b>Response measure</b>	Whether users can access the content or not.

<b>Quality attribute</b>	Performance
<b>Scenario</b>	Data latency

<b>Source</b>	User(student/Instructor)
<b>Stimulus</b>	Users want to store/get data from this application.
<b>Artifact</b>	The System
<b>Environment</b>	At Runtime
<b>Response</b>	Latency time of the system to respond to a Get/Post request.
<b>Response Measure</b>	The system should answer the request of the users and store/get the data from database within 200 ms.

<b>Quality attribute</b>	Performance
<b>Scenario</b>	Maximum Online Users
<b>Source</b>	User(student/Instructor)
<b>Stimulus</b>	Lots of user are online in the same time
<b>Artifact</b>	The System
<b>Environment</b>	At Runtime
<b>Response</b>	The system should answer each user's request normally without crash
<b>Response Measure</b>	Maximum number of online users in the same time without crash

<b>Quality attribute</b>	Reliability
<b>Scenario name</b>	Multiple user access
<b>Source</b>	Users
<b>Stimulus</b>	Multiple users try to access the same piece of data at the same time

<b>Artifact</b>	website
<b>Environment</b>	Run time
<b>Response</b>	The system should respond the correct information to every user correctly.
<b>Response measure</b>	The response time and the correctness of the return information.

<b>Quality attribute</b>	Testability
<b>Scenario</b>	Testing in various phases of development.
<b>Source</b>	Developer, Tester, User.
<b>Stimulus</b>	Developer or the Tester want to test the system or a part of it
<b>Artifact</b>	The whole system or just a part of it, code.
<b>Environment</b>	During development, testing phase.
<b>Response</b>	Comparing the expected outcomes with the actual output.
<b>Response measure</b>	Number of bugs/errors must be reduced after consecutive tests.

### **3. Architecture Key Decisions and Rationale**

#### **(3.1) Architecture style**

We choose Django as our framework which use MVC as its architecture style. M, V, C stands for Model, View and Controller.

Model specifies the logical structure of data in a software application as well as the high-level class associated with it. It represents the data which describes the working of an application. When a model changes its state, domain notifies its associated views, so they can be refreshed.

View component is used for all the UI logic of the application and these are the components that display the application's user interface (UI). It renders the model into a form suitable for interaction. Multiple views can exist for a single model for different purposes.

Controllers act as an interface between Model and View components. It processes all the business logic and incoming requests, manipulate data using the Model component, and interact with the Views to render the final output. It receives input and initiates a response by making calls on model objects.

#### Advantages of MVC

- 1) Faster development process: MVC supports rapid and parallel development. With MVC, one programmer can work on the view while other can work on the controller to create business logic of the web application.
- 2) Ability to provide multiple views: You can create multiple views for a model. You could avoid or decrease the code duplication, because MVC separates data and business logic from the display.
- 3) Support for asynchronous technique: MVC also supports asynchronous technique, which helps developers to develop an application that loads very fast.
- 4) Modification does not affect the entire model: Modification does not affect the entire model because model part does not depend on the views part. Therefore, any changes in the Model will not affect the entire architecture.
- 5) MVC model returns the data without formatting: MVC pattern returns data without applying any formatting so the same components can be used and called for use with any interface.
- 6) SEO friendly Development platform: Using this platform, it is very easy to develop SEO-friendly URLs to generate more visits from a specific application.

#### **(3.2) Trade-off analysis for candidate solutions and technologies**

##### Solution: Web application

A web-based application is any application that uses a website as the interface or front-end. Users can easily access the application from any computers, mobiles or other devices which connected to the Internet using a standard browser.

This contrasts with traditional desktop applications, which are installed on a local computer.

Advantages of web application:

- Cost effective development
- Accessible anywhere



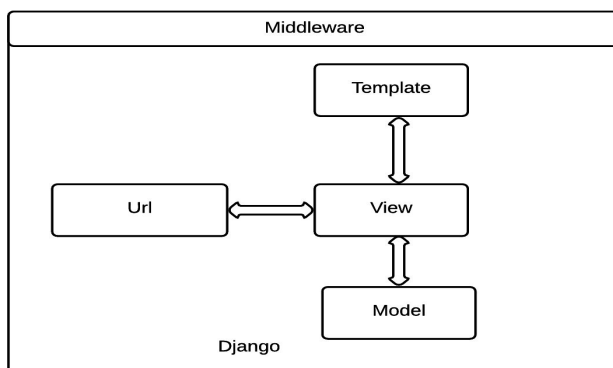
- Easily customizable
- Accessible for a range of devices
- Improved interoperability
- Easier installation and maintenance
- Adaptable to increased workload
- Increased Security
- Flexible technologies

### Framework - Django framework

Reasons for using python:

- Learning Ease
- Support Availability
- Readability and efficiency, especially when compared to other languages like Java, PHP, or C++
- Python lets you build functions with fewer lines of code.
- Extensive Support Libraries: Internet protocols, string operations, web services tools and operating system interfaces
- Strong control over indentation. Python interpreter automatically enforces correct indentation, Java compiler does not and it is the responsibility of the developer.
- Python supports around 54 frameworks such as **Django**, **Pyramid**, **Zope** and **Flask**.

Django is the most famous framework with over 4700 sites written in Django. Some famous websites developed using django include: Instagram, pinterest, disqus, mozilla.



Advantages:

- Fast
- Django supports reusability and pluggability of components
- It provides various features such as user authentication, content administration, site maps, RSS feeds.

- It handles many security concerns such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking.
- Scalable: Some of the heavy traffic sites use django for its flexibility and speed.
- Built-in admin interface is helpful for early data entry and testing
- Its clean design makes it easy to follow best web-development practices.
- Object-Relational Mapping (ORM) Support
- Multilingual Support: Django's built-in internationalization system provides support for many languages.
- A caching framework that can use any of several cache methods: Django exposes a simple, low-level cache API. You can use this API to cache the results that rarely change.
- provides an interface to Python's built-in unit test framework
- Django has a built-in lightweight and standalone web server for development and testing.

### Database

Django acts as an extension between the data model and the database engine, and backing a huge arrangement of database frameworks including MySQL, Oracle, PostgreSQL, and so forth. Django additionally supports NoSQL database through Django-nonrel fork. The main NoSQL databases upheld are MongoDB and google application engine.

Reasons for choosing RDBMS over NoSQL databases:

- it is organized and composed
- simple to join, dependability and support.
- One of the main points is ACID(Atomicity, Consistency, Isolation, and Durability) consistence.

Out of all the RDBMS options supported by django we prefer MySQL because:

- It is a well-known DataBase. An extraordinarily large number of web-applications utilize MySQL in view of its **adaptability** and **flexibility**.
- It is outstanding amongst other DB to use for a learning management system considering the downsides of postgresQL. For example, Postgresql lacks performance for simple read operations compared to MySQL, because of its drawbacks it is hard to find hosts or service providers.
- Replication feature is well implemented in MySQL
- Simple to work with.
- Secure: A lot of security features, some rather advanced, are built in MySQL.
- Versatile and effective
- Speed

- Scalable and powerful

## API

Restful (Representational state transfer): Django supports the Restful style API. Restful API is a Web software architecture style, the purpose of which is to facilitate different software / programs in the network (such as the Internet) to pass information. REST is a design style rather than a standard.

Advantages of Restful API:

- good performance
- Scalability
- Simplicity of a uniform Interface
- Modifiability Visibility of communication between components by service agents
- Portability of components
- Reliability

## Front-end

Font Awesome:

The font awesome can help us use a scalable image on our learning system regardless of the size and scaling factors. Thus, whether we are using different font sizes, the icon font remains to be the same. Moreover, the Font Awesome Icons provides various types of choices that could suit the media icon needs for many tasks. The fonts and icons can be adjusted in terms of their colors and transparency without worrying about disturbing their qualities.

HTML:

As a plain text language, it is very easy to edit Html and it's easy to create a web page by HTML. With the help of template, HTML is friendly for beginners to start building the web page.

CSS:

CSS is a language used to detail the presentation of a web page, such as colors, fonts, and layout. One of the key benefits is the way it allows the separation of document content (written in HTML) from document presentation (written in CSS).

Bootstrap:

Bootstrap is a free and open-source front-end web framework for design web application. It contains HTML and CSS based design template for typograph, forms, buttons, navigation and other interface components, as well as JS extensions. It's easy to begin the procedure with

Bootstrap and it is also adaptive. We can develop the web page using Bootstrap along with the HTML and CSS.

GIMP:

GIMP is a cross-platform image editor and is totally free to use. It can be used to design the logo of our learning system. GIMP provides a set of sophisticated tools for us to edit the picture and design the photos what we want to present in the system webpage.

JavaScript:

Being client-side, JavaScript is very fast because any code functions can be run immediately instead of having to contact the server and wait for an answer. It is relatively simple to learn and implement. JavaScript plays nicely with other languages and can be used in a huge variety of applications. JavaScript can be inserted into any web page regardless of the file extension.

### Web server

As we are going to use Django framework for our project, we need to consider which web server fits Django best.

Django includes a lightweight web server which can be used in debugging and testing. However, this server only works under local environment. It is not an appropriate choice on an application using by many users.

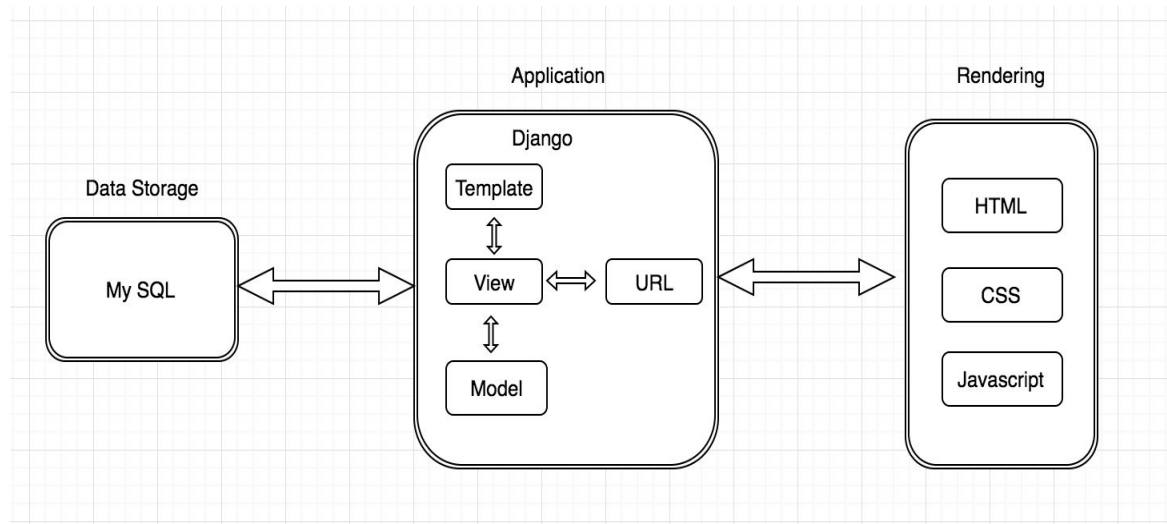
Therefore, we need to set up a production web server on Django. We find two combinations, one is Nginx with uWSGI, the other is Apache with mod\_wsgi. Compared with Nginx, Apache is more functional and stable, and we are more experienced with Apache. Therefore, we decide on Apache with mod\_wsgi as our web server.

We'll then set up Apache in front of our application so that it can handle client requests directly before passing requests that require application logic to the Django app. We will do this using the mod\_wsgi that can communicate with Django over the WSGI interface specification. Mod\_wsgi is an Apache module which can host any Python WSGI application, including Django, which can translate HTTP requests into a predictable application format and Django will work with any version of Apache which supports mod\_wsgi.

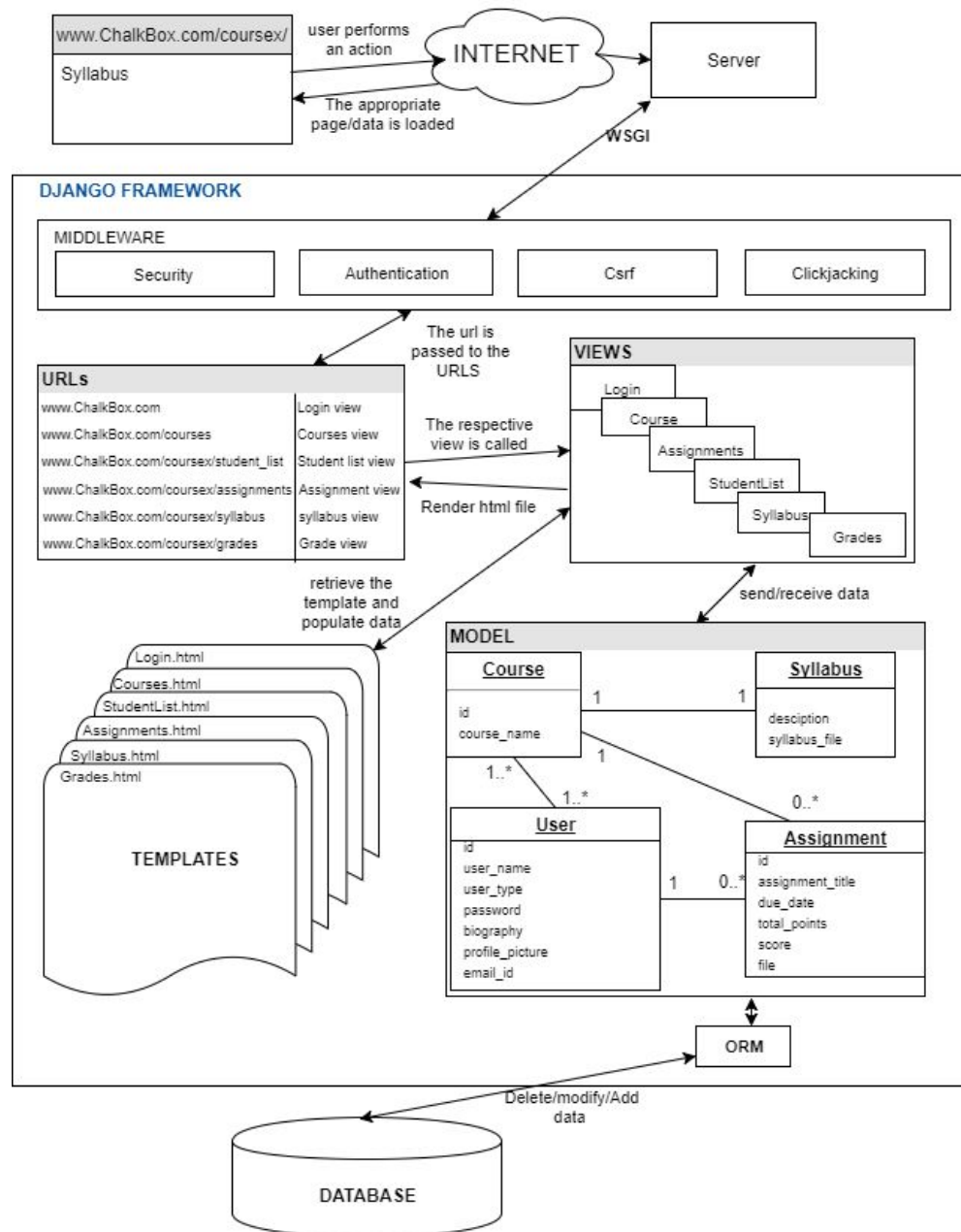
### **(3.3) Architecture elements and relations**

#### Overall Architecture Structure:

The core of our architecture is based on Django. It plays as an important role as middleware and connects the front-end and back-end together. It connected the Apache Server with application, and plays as a role as ORM and connect the application with MySQL database automatically.



Detailed Architecture Graph:



The above diagram explains the system in a more detailed way. The Django python application consists of:

**URL:** This is responsible for routing. When a user makes a request, the corresponding view is mapped by the urls.

**Views:** Handles the data to be displayed to the user.

**Templates:** Django makes it possible to separate python and HTML, the python goes in views and HTML goes in templates. To link the two, Django relies on the render function and the Django Template language. Inheritance of Templates in django also facilitate reusability.

**Model:** represents the data in the Database.

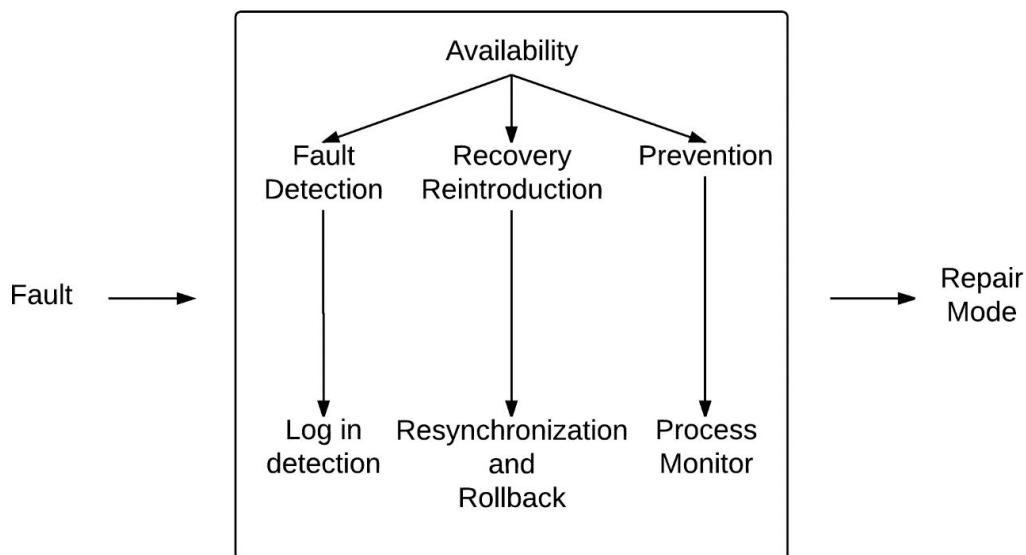
**WSGI:** Is an interface between web servers and web applications or frameworks for the Python programming language.

**Object-relational mapping (ORM):** ORM acts as an abstraction layer between applications and the data sources. In this application, Django plays as the role of ORM. When the API of data sources change, only ORM needs to change, not the applications that use ORM to avoid this kind of effort.

**Middleware** is a part of Django that hooks into Django's request/response processing. It is a plugin to modify system's input or output.

### (3.4) Tactics and mechanisms for the top priority quality scenarios

- **Tactics for availability**



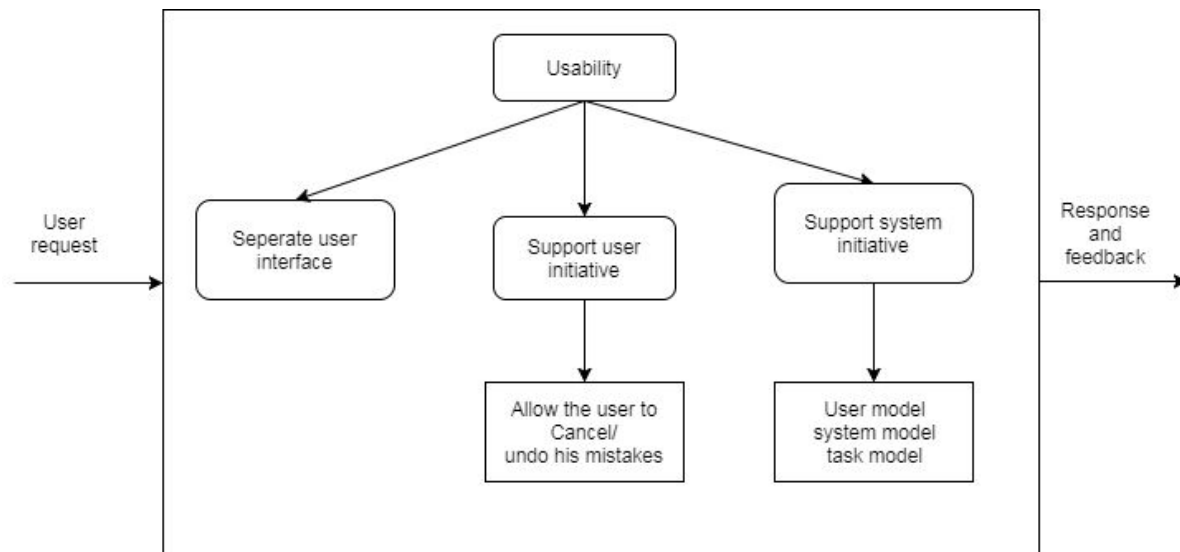
High availability software refers to the use of software to ensure that systems are available most of the time. Availability is a characteristic of a system and is defined as the percentage of time that the system is functioning. As we plan to use local web server and implement the application in Django. We address these issues successfully. The application is available whenever user need to access it.

**Log in detection** is easy to realize, the authentication middleware detects the inconsistency of the password and the username.

**Recovery and Reintroduction** depends on the database we will use. MySQL offers a variety of backup strategies from which we can choose the methods that best suit the requirements for the installation. Detailed implementation will be adopted in further development.

**Process monitor:** Django framework's admin application provides us features such as user management and recording transactions which facilitates process monitoring.

- **Tactics for usability**



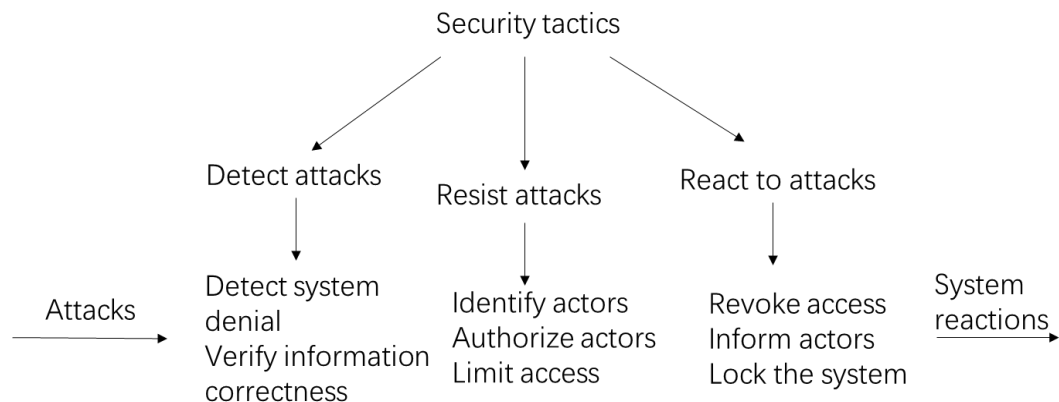
User satisfaction for the LMS can be achieved using the tactics mentioned above.

1. **Separating User interface** from the rest of the system functionality. This facilitates the frequent change in the User interface without disturbing the system functionality. The model View Controller Architecture style supports this tactic.
2. **Support user initiative** i.e., if the user takes any action the system must be ready to respond with an appropriate response. The user should be able to cancel/undo/ delete his mistakes. In LMS the user is given appropriate privileges based on the type of user. For example, the instructor has can change the grade of any student and correct himself/herself at any point by deleting/editing his/her submission.
3. The **system initiative** can be achieved using models about the user, the task by the user, or the system state itself. Each model requires various types of input to accomplish its initiative. In LMS:
  - **Task model:** The model is used to determine what the user is attempting and provide assistance. For example, if a user is lost and navigates to a 404 page, the user can be redirected to the help page.



- User model: This determines the user's knowledge and capabilities. For example: The LMS should have subtitles for hearing impairment students. It should have reasonable font size.
- System model: It determines the system behavior so that appropriate feedback can be given to the user.

- **Tactics for security**

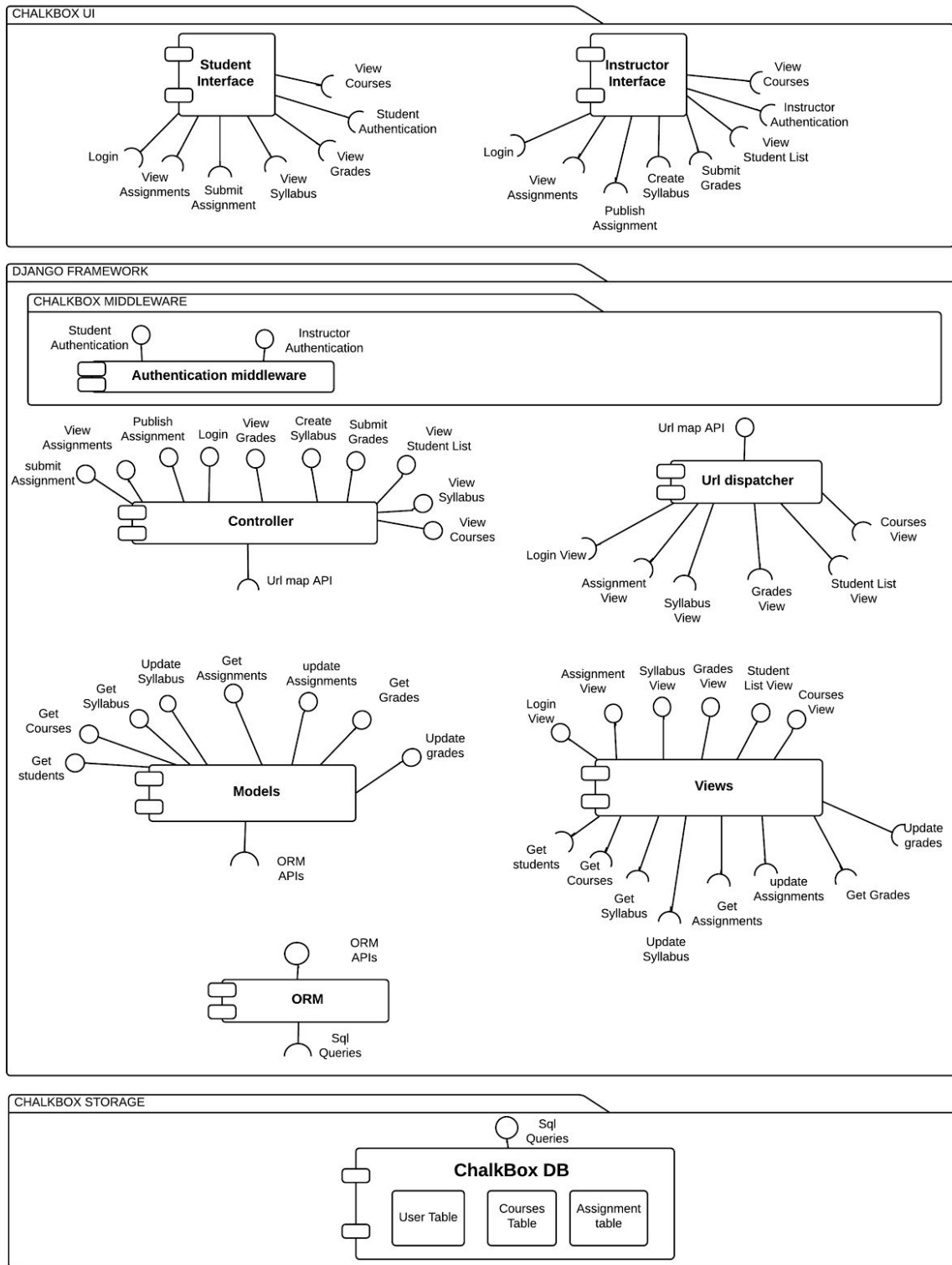


**Detect Attacks:** In our system, we have the middleware component in Django, which works for both user authentication and authorization. We save the username and password data in database. Once there is an attempt to login, the system can check if it matches the records in database. Then it will return a User object if the credentials are valid for a backend. If the credentials aren't valid for any backend or if a backend raises PermissionDenied, it returns None. The middleware component of the system will **react to attacks** by displaying a notice page saying, 'Invalid username or password' for illegal user login. Also an additional feature can be added to notify the admin when there are continuous illegal access attempts as shown in the behavioral view security scenario.

**Resist Attacks:** Security issues such as CSRF attack, SQL injection, password encryption are handled by Django form, ORM and password field. Django has built-in protection against most types of CSRF attacks, providing you have enabled and used it where appropriate. CSRF protection works by checking for a secret in each POST request. This ensures that a malicious user cannot simply "replay" a form POST to your website and have another logged in user unwittingly submit that form. The malicious user would have to know the secret, which is user specific. SQL injection is a type of attack where a malicious user is able to execute arbitrary SQL code on a database. This can result in records being deleted or data leakage. By using Django's querysets, the resulting SQL will be properly escaped by the underlying database driver. It is also a good idea to limit the accessibility of caching system and database using a firewall.

## 4. Architecture Logical View

### (4.1) Component Diagram



## (4.2) Component descriptions

### Component: Authentication middleware

- Responsibilities: Performs authentication of all the users
- Provided Interfaces: Student Authentication, Instructor Authentication
- The provided Interface Student Authentication has two operation: CheckStudentExists, CheckStudentIsAuthenticated
- The provided Interface Instructor Authentication has two operation: CheckInstructorExists, CheckInstructorIsAuthenticated

<b>Interface Student Authentication</b>	
Operation signature	<b>CheckStudentExists (UserName: String):Boolean</b>
Operation description	Checks in the storage if the student exists, depending on the success returns true or false
Operation preconditions	Username not null
Operation postconditions	If student exists true is returned else false is returned
Operation signature	<b>CheckStudentIsAuthenticated (UserName: String):Boolean</b>
Operation description	Checks if the username and password match
Operation preconditions	Username exists
Operation postconditions	Returns true if student's username & password match, else false is returned
<b>Interface Instructor Authentication</b>	
Operation signature	<b>CheckInstructorExists (UserName: String):Boolean</b>
Operation description	Checks in the storage if the instructor exists, depending on the success returns true or false
Operation preconditions	Username not null

Operation postconditions	If Instructor exists true is returned else false is returned
Operation signature	<b>CheckInstructorIsAuthenticated (UserName: String):Boolean</b>
Operation description	Checks if the instructor's username and password match
Operation preconditions	Username exists
Operation postconditions	Returns true if instructor's username and password match, else false is returned

#### Component: URL dispatcher

- Responsibilities: Performs the mapping of each url to views. It basically facilitates the navigation within the system.
- Provided Interfaces: Url map API
- The provided Interface Login view has 2 operation: SearchUrl, CallView

<b>Interface Url map API</b>	
Operation signature	<b>SearchUrl (Url: String): Function view</b>
Operation description	Read the url and search the list of urls to find the respective view function.
Operation preconditions	User has clicked on a link
Operation postconditions	The respective view function is returned
Operation signature	<b>CallView (view: function)</b>
Operation description	Call the respective view to perform the required action
Operation preconditions	The mapping for the view exists.
Operation postconditions	The required View function is called.

### Component: Views

- Responsibilities: Performs the required actions which usually involve reading and writing to the models and presenting information to the user. It basically retrieves data from the model, performs the required manipulations, populates the templates and sends it to the front end.
- Provided Interfaces: Login View, Assignment view, Syllabus view, grades view, student list view, courses view
- The provided Interface Login view has 6 operation: ReadUserName, ReadPassword, HidePassword
- The provided Interface course view has two operation: GetUserCourses, DisplayCourses
- The provided Interface assignment view has 4 operation: GetAssignments, DisplayAssignments, SubmitAssignment, PublishAssignment
- The provided Interface syllabus view has 3 operation: GetSyllabus, DisplaySyllabus, PublishSyllabus
- The provided Interface Grade view has 3 operation: GetGrades, DisplayGrades, SubmitGrade
- The provided Interface student list view has 1 operation: GetStudents

<b>Interface Login View</b>	
Operation signature	<b>ReadUserName (UserName: String)</b>
Operation description	Read the username
Operation preconditions	Username not null
Operation postconditions	Read the username entered by the user
Operation signature	<b>ReadPassword (password: hiddenString)</b>
Operation description	Read the password.
Operation preconditions	Password is not null
Operation postconditions	User password is encrypted and read by the system to authenticate the user

Operation signature	<b>HidePassword (password: hiddenString)</b>
Operation description	Hides the password when user enters it.
Operation preconditions	Password is entered by the user
Operation postconditions	User password is hidden and shown as “*****”
<b>Interface Course View</b>	
Operation signature	<b>GetCourses (UserName: String):List of Courses</b>
Operation description	Returns a list of course objects the user has enrolled to.
Operation preconditions	User has logged in successfully
Operation postconditions	If the user has logged in with right credentials, returns a list of courses the user has enrolled to.
Operation signature	<b>DisplayCourse(Courses: List)</b>
Operation description	Displays the list of course objects by populating the courses.html template file.
Operation preconditions	The list of course objects is retrieved successfully
Operation postconditions	The user enrolled courses is displayed as links to view more details like assignments, syllabus and grades.
<b>Interface Assignment View</b>	
Operation signature	<b>GetAssignments (UserName: String, CourseName:String):List of Assignments</b>
Operation description	Returns a list of assignment objects for a course the user has enrolled to.
Operation preconditions	User has logged in successfully. The user clicks on the assignments link in the course home page.

Operation postconditions	If the user has logged in with right credentials, once he clicks on the assignments link in the course page, returns a list of assignments the user has enrolled to.
Operation signature	<b>DisplayAssignments (Assignments: List)</b>
Operation description	Displays the list of assignment objects by populating the assignment.html template file for a course the user has enrolled to.
Operation preconditions	The list of assignment objects are retrieved successfully
Operation postconditions	The course's assignments is displayed as links to view more details like question and submission.
Operation signature	<b>SubmitAssignments (UserName: String, CourseName:String, Assignment:Object):Boolean</b>
Operation description	Adds the assignment object to the list of assignment objects for a course the user has enrolled to.
Operation preconditions	Student can view the assignment question
Operation postconditions	The assignment is successfully submitted
Operation signature	<b>PublishAssignment (CourseName: String, Assignment: Object):Boolean</b>
Operation description	Add the assignment object with question, due date and the file under assignment objects for the course so that students to view it.
Operation preconditions	Instructor has logged in successfully.
Operation postconditions	Instructor successfully posted the assignment question and the information reflects in the student's dashboard.
<b>Interface Syllabus View</b>	

Operation signature	<b>GetSyllabus (CourseName:String):Syllabus object</b>
Operation description	Returns the syllabus object for a course the user has enrolled to
Operation preconditions	User has logged in successfully and clicked on the syllabus tab inside a course.
Operation postconditions	Returns the syllabus object for the course.
Operation signature	<b>DisplaySyllabus (syllabus: object)</b>
Operation description	Displays the syllabus object by populating the syllabus.html template file for a course the user has enrolled to.
Operation preconditions	The syllabus object is retrieved successfully
Operation postconditions	The course's syllabus is displayed with link to download the file..
Operation signature	<b>PublishSyllabus (CourseName: String, syllabus: object)</b>
Operation description	Adds the populated syllabus object for a course the instructor handles.
Operation preconditions	The instructor has successfully logged in.
Operation postconditions	The course's syllabus gets published and all the students enrolled to the course can view it.
<b>Interface Grades View</b>	
Operation signature	<b>GetGrades (UserName: String, CourseName:String):List of Grades</b>
Operation description	Returns a list of grade objects for a course the user has enrolled to.
Operation preconditions	User has logged in successfully. The student clicks on the grades link in the course page. The instructor clicks on a particular student in the student list and clicks on the gradebook



	for that student.
Operation postconditions	Returns a list of grades for the course the student is enrolled to.
Operation signature	<b>DisplayGrades (Grades: List)</b>
Operation description	Displays the list of grade objects by populating the gradebook.html template file for a course the student has enrolled to.
Operation preconditions	The list of grade objects are retrieved successfully
Operation postconditions	The course's gradebook for a particular student is displayed
Operation signature	<b>SubmitGrade (UserName: String, CourseName:String, Assignment:Object, Grade: float):Boolean</b>
Operation description	Add the grade to the grade object for a course the student is enrolled to.
Operation preconditions	Instructor clicks on the assignment of a particular student and clicks on the grade button
Operation postconditions	The grade is added to the "UserName" student's assignment successfully.
<b>Interface Student List View</b>	
Operation signature	<b>GetStudents (CourseName:String):List of Students</b>
Operation description	Returns a list of students objects for the course the instructor selected.
Operation preconditions	Instructor has logged in successfully. The Instructor clicks on the Students link in the selected course home page.
Operation postconditions	Returns a list of students enrolled for the course.

## Component: Models

- Responsibilities: It comprise the code that is to deal with the database (can be a simple Java bean). It is built without appearance concerns at the time of presenting of the student. To perform its functionality, it implements the public functionality. A model is the representation of data as python classes. These classes are directly mapped to the tables in the database.
- Provided Interfaces: get students, get courses, get syllabus, update syllabus, get assignments, update assignment, get grades, update grade.
- The provided Interfaces get students has 2 operations: GetStudentsOfCourse, GetStudent
- The provided Interfaces get courses has 2 operations: GetCourse, GetCourseListOfStudent
- The provided Interfaces get syllabus has 1 operation: GetSyllabus
- The provided Interfaces update syllabus has 1 operation: UpdateSyllabus
- The provided Interfaces get assignments has 2 operations: GetAssignment, GetAssignmentListOfCourse
- The provided Interfaces update assignments has 1 operation: UpdateAssignment
- The provided Interfaces get grades has 2 operations: GetGradeListOfAssignment, GetGradeListOfStudent
- The provided Interfaces update grades has 1 operation: UpdateGrade

<b>Interface Get Students</b>	
Operation signature	<b>GetStudentsOfCourse(CourseId: integer): list</b>
Operation description	Get the student list of a certain course.
Operation preconditions	couseId is valid
Operation postconditions	The student list is returned.
Operation signature	<b>GetStudent(CourseId: integer, username: string): student object</b>
Operation description	Get the student info. Of a certain student(password, name)
Operation preconditions	The password and name exists in the database and they are matched.

Operation postconditions	The student is returned
<b>Interface Get Courses</b>	
Operation signature	<b>GetCourse(courseId: integer): course object</b>
Operation description	Get a certain course
Operation preconditions	courseId is valid
Operation postconditions	The course is returned.
Operation signature	<b>GetCourseListOfStudent(studentId: integer) list</b>
Operation description	Get the list of courses of that student
Operation preconditions	studentId is valid.
Operation postconditions	The course list is returned.
<b>Interface Get Syllabus</b>	
Operation signature	<b>GetSyllabus(couseId: integer): syllabus object</b>
Operation description	Get the syllabus of a certain course
Operation preconditions	courseId is valid
Operation postconditions	The syllabus is returned.
<b>Interface Update Syllabus</b>	
Operation signature	<b>UpdateSyllabus(courseId: integer): syllabus object</b>
Operation description	Update the syllabus of a certain course.
Operation preconditions	courseId is valid and this course has syllabus
Operation postconditions	The new syllabus is returned.

<b>Interface Get Assignments</b>	
Operation signature	<b>GetAssignment(courseId: integer, assignmentId: integer) assignment object</b>
Operation description	Get the assignment of a certain course
Operation preconditions	The assignment is in the list of assignments of that course, and the student is enrolled in that course.
Operation postconditions	The information of that assignment is read by the student.
Operation signature	<b>GetAssignmentListOfCourse(courseId: integer): list</b>
Operation description	Get the list of assignment of that course.
Operation preconditions	The course is valid and the student is enrolled in that course.
Operation postconditions	The list of assignments is read by the student.
<b>Interface Update Assignments</b>	
Operation signature	<b>UpdateAssignment(courseId: integer, assignmentId: integer) assignment object</b>
Operation description	Update the assignment of a certain course
Operation preconditions	The assignment is in the list of assignments of that course, and the student is enrolled in that course.
Operation postconditions	The information of that assignment is read by the student.
<b>Interface Get Grades</b>	
Operation signature	<b>GetGradeListOfAssignment(assignmentId: integer, courseId: integer): list</b>
Operation description	Get the grade list of certain assignment in a certain course.

Operation preconditions	The course has this assignmentId, and the courseId is valid
Operation postconditions	The grade list is returned.
Operation signature	<b>GetGradeListOfStudent(studentId: integer): list</b>
Operation description	Get the grade list of certain student
Operation preconditions	The studentId is valid, and this student has grade
Operation postconditions	The grade list is returned.
<b>Interface Update Grades</b>	
Operation signature	<b>UpdateGrade(studentId: integer, assignmentId: integer, courseId: integer): grade object</b>
Operation description	Update the grade of a student's assignment in a certain course.
Operation preconditions	The course has this assignmentId, and this studentId is enrolled in this course.
Operation postconditions	The new grade is returned.

#### Component: Controller

- Responsibilities: Takes the user requests from the user interface component and maps the request to their respective views through the URL dispatcher component.
- Provided Interfaces: Submit Assignment, View Student Assignment, Publish Assignment, Login, View Grades, Create Syllabus, Submit Grades, View Student List, View Syllabus, View Courses.
- The provided interface submit assignment has two operations: GetSubmitRequest, MapRequestToAssignmentView
- The provided interface View Student Assignment has two operations: GetViewAssignmentRequest, MapRequestToAssignmentView
- The provided interface Publish Assignment has two two operations: GetPublishRequest, MapRequestToAssignmentView

- The provided interface login GetLoginRequest has two operations: GetLoginRequest, MapRequestToLoginView
- The provided interface View Grades has two operations: GetViewGradesRequest, MapRequestToGradesView
- The provided interface Create Syllabus has two operations: GetCreateSyllabusRequest, MapRequestToCreateSyllabusView
- The provided interface Submit Grades has two operations: GetSubmitGradesRequest, MapRequestToGradeView
- The provided interface View Student List has two operations: GetViewStudentListRequest, MapToStudentListView
- The provided interface View Syllabus has two operations: GetViewSyllabusRequest, MapRequestToSyllabusView
- The provided interface View Courses has two operations: GetViewCoursesRequest, MapRequestToViewCoursesView

Interface <b>Submit Assignment</b>	
Operation signature	<b>GetSubmitRequest (UserName: String):Boolean</b>
Operation description	Get the submit assignment request from user
Operation preconditions	User is logged in and ready for submission
Operation postconditions	If submit request got true is returned else null
Operation signature	<b>MapRequestToAssignmentView (UserName: String):Boolean</b>
Operation description	Map the submit assignment request to assignment view by URL dispatcher
Operation preconditions	Submit Request is taken from user and saved
Operation postconditions	Assignment view is shown to user
Interface <b>View Student Assignment</b>	
Operation signature	<b>GetViewAssignmentRequest (AssignmentName: String):Boolean</b>
Operation description	Get view assignment request from users

Operation preconditions	User is logged in and ready for submission
Operation postconditions	If view request got true is returned else null
Operation signature	<b>MapRequestToAssignmentView (AssignmentName: String):Boolean</b>
Operation description	Map the view assignment request to assignment view by URL dispatcher
Operation preconditions	View Assignment Request is taken from user and saved
Operation postconditions	Assignment view is shown to user
Interface <b>Publish Assignment</b>	
Operation signature	<b>GetPublishRequest (AssignmentName: String):Boolean</b>
Operation description	Get the publish assignment request from user
Operation preconditions	User is logged in and ready for publish assignment
Operation postconditions	If publish request got true is returned else null
Operation signature	<b>MapRequestToAssignmentView (AssignmentName: String):Boolean</b>
Operation description	Map the publish assignment request to assignment view by URL dispatcher
Operation preconditions	Publish Request is taken from user and saved
Operation postconditions	Assignment view is shown to user
Interface <b>Login</b>	
Operation signature	<b>GetLoginRequest (UserName: String):Boolean</b>
Operation description	Get the login assignment request from user
Operation preconditions	User is ready to login with correct account and password

Operation postconditions	If login request got true is returned else null
Operation signature	<b>MapRequestToLoginView (UserName: String):Boolean</b>
Operation description	Map the login assignment request to login view by URL dispatcher
Operation preconditions	Login Request is taken from user and verified
Operation postconditions	Login assignment view is shown to user
<b>Interface View Grades</b>	
Operation signature	<b>GetViewGradesRequest (UserName: String):Boolean</b>
Operation description	Get the view grades assignment request from user
Operation preconditions	User is logged in and ready for view grades
Operation postconditions	If view grades request received, return true , else null
Operation signature	<b>MapRequestToGradesView (UserName: String):Boolean</b>
Operation description	Map the view grades request to grades view by URL dispatcher
Operation preconditions	View grades request is taken from user and saved
Operation postconditions	View grades view is shown to user
<b>Interface Create Syllabus</b>	
Operation signature	<b>GetCreateSyllabusRequest (UserName: String):Boolean</b>
Operation description	Get the create syllabus request from user
Operation preconditions	User is logged in and ready for syllabus creation
Operation postconditions	If create syllabus request got true is returned else null



Operation signature	<b>MapRequestToCreateSyllabusView (UserName: String):Boolean</b>
Operation description	Map the create syllabus request to syllabus creation view by URL dispatcher
Operation preconditions	Create syllabus request is taken from user and saved
Operation postconditions	Syllabus view is shown to user
Interface <b>Submit Grades</b>	
Operation signature	<b>GetSubmitGradesRequest (UserName: String):Boolean</b>
Operation description	Get the submit grades request from user
Operation preconditions	User is logged in and need to submit grade
Operation postconditions	If grades submit request got true is returned else null
Operation signature	<b>MapRequestToGradeView (UserName: String):Boolean</b>
Operation description	Map the grades submit request to grade view by URL dispatcher
Operation preconditions	Submit grades request is taken from user and saved
Operation postconditions	Grade view is shown to user
Interface <b>View Student List</b>	
Operation signature	<b>GetViewStudentListRequest (UserName: String):Boolean</b>
Operation description	Get the view student list request from user
Operation preconditions	User is logged in and needs to view student list
Operation postconditions	If view student list request got true is returned else null
Operation signature	<b>MapToStudentListView (UserName: String):Boolean</b>

Operation description	Map the view student list request to student list view by URL dispatcher
Operation preconditions	View student list request is taken from user and saved
Operation postconditions	Student list view is shown to user
<b>Interface View Syllabus</b>	
Operation signature	<b>GetViewSyllabusRequest (UserName: String):Boolean</b>
Operation description	Get the view syllabus request from user
Operation preconditions	User is logged in and ready for view syllabus
Operation postconditions	If view syllabus request got true is returned else null
Operation signature	<b>MapRequestToSyllabusView (UserName: String):Boolean</b>
Operation description	Map the request of syllabus request to syllabus view by URL dispatcher
Operation preconditions	View syllabus request is taken from user and saved
Operation postconditions	Syllabus view is displayed to user
<b>Interface View Courses</b>	
Operation signature	<b>GetViewCoursesRequest (UserName: String):Boolean</b>
Operation description	Get the view courses request from user
Operation preconditions	User is logged in and wants to view courses
Operation postconditions	If view courses request got true is returned else null
Operation signature	<b>MapRequestToViewCoursesView (UserName: String):Boolean</b>
Operation description	Map the view courses request to course view by URL dispatcher

Operation preconditions	View courses request is taken from user and saved
Operation postconditions	Courses view is displayed to user

#### Component: User Interface

- Responsibilities: Communication channel between the user and the system
- Provided Interfaces: Student Interface, Instructor Interface
- The provided Interface “Student Interface” has seven operations: Login, ViewAssignment, SubmitAssignment, ViewSyllabus, ViewGrades, StudentAuthentication, ViewCourses
- The provided Interface “Instructor Interface” has eight operations: Login, ViewStudentsAssignment, SubmitGrades, PublishAssignment, CreateSyllabus, ViewSyllabus, ViewStudentList, InstructorAuthentication, ViewCourses

<b>Interface “Student Interface”</b>	
Operation Signature	<b>StudentLogin (UserName: String):Boolean</b>
Operation description	Login method for students
Operation preconditions	Student must have an account
Operation postconditions	Student will be able to login
Operation Signature	<b>ViewAssignment (UserName: String):Boolean</b>
Operation description	An interface to view assignments
Operation preconditions	Student must be logged in
Operation postconditions	Student can view assignments
Operation Signature	<b>SubmitAssignment (UserName: String):Boolean</b>
Operation description	An interface to submit assignments
Operation preconditions	Student must be logged in
Operation postconditions	Student can submit assignments

Operation Signature	<b>ViewSyllabus (UserName: String):Boolean</b>
Operation description	An interface to view syllabus
Operation preconditions	Student must be logged in
Operation postconditions	Student can view syllabus
Operation Signature	<b>ViewGrades (UserName: String):Boolean</b>
Operation description	An interface to view grades
Operation preconditions	Student must be logged in
Operation postconditions	Student can view grades
Operation Signature	<b>StudentAuthentication (UserName: String):Boolean</b>
Operation description	Validates the user details with the data base
Operation preconditions	Student must have an account
Operation postconditions	Student is authenticated if he/she has an account
Operation Signature	<b>ViewCourses (UserName: String):Boolean</b>
Operation description	An interface to view courses
Operation preconditions	Student must be logged in
Operation postconditions	Student can view courses
<b>Interface “Instructor Interface”</b>	
Operation Signature	<b>InstructorLogin (UserName: String):Boolean</b>
Operation description	An interface for instructor to login

Operation preconditions	Instructor must have an account
Operation postconditions	Instructor will be able to login
Operation Signature	<b>ViewStudentsAssignment (UserName: String):Boolean</b>
Operation description	An interface to view student's assignment
Operation preconditions	Student must have submitted the assignment
Operation postconditions	Instructor can view the assignment
Operation Signature	<b>SubmitGrades (UserName: String):Boolean</b>
Operation description	An interface to submit grades
Operation preconditions	Instructor must be logged in
Operation postconditions	Instructor can submit grades
Operation Signature	<b>ViewCourses (UserName: String):Boolean</b>
Operation description	An interface to view courses
Operation preconditions	Instructor must be logged in
Operation postconditions	Instructor can view courses

#### Component: ChalkBox DB

- Responsibilities: Store application data and response to the request from the view component.
- Provided Interface: SQL Queries
- Provided Interface has three Operation: Query, Read, Write

Interface <b>SQL Queries</b>	
Operation Signature	<b>Query (SQL request: Select): Table</b>

Operation description	Get the SQL query request from the front end
Operation preconditions	The query is grammatically correct
Operation postconditions	If the query content exist, return it
Operation Signature	<b>Read (SQL request: Select): Table</b>
Operation description	Read the table content in the front end
Operation preconditions	The table exists and have corresponding element
Operation postconditions	The content get display in the front end
Operation Signature	<b>Write (SQL request: Insert): Boolean</b>
Operation description	Write content into database table
Operation preconditions	Database connection is ready and has enough space for writing
Operation postconditions	New data successfully written into table

#### Component: Object-Relational Mapping

- Responsibilities: Perform the mapping relationship between every object. It basically facilitates the relationship within the database.
- Provided Interface: ORM API
- Provided Interface has two Operation: SearchObject, ExtractObject

Interface <b>ORM API</b>	
Operation Signature	<b>SearchObject (Object: Class): Boolean</b>
Operation description	Read the object and search the tables to find the respective object
Operation preconditions	Database receive a search request
Operation postconditions	The respective object is returned

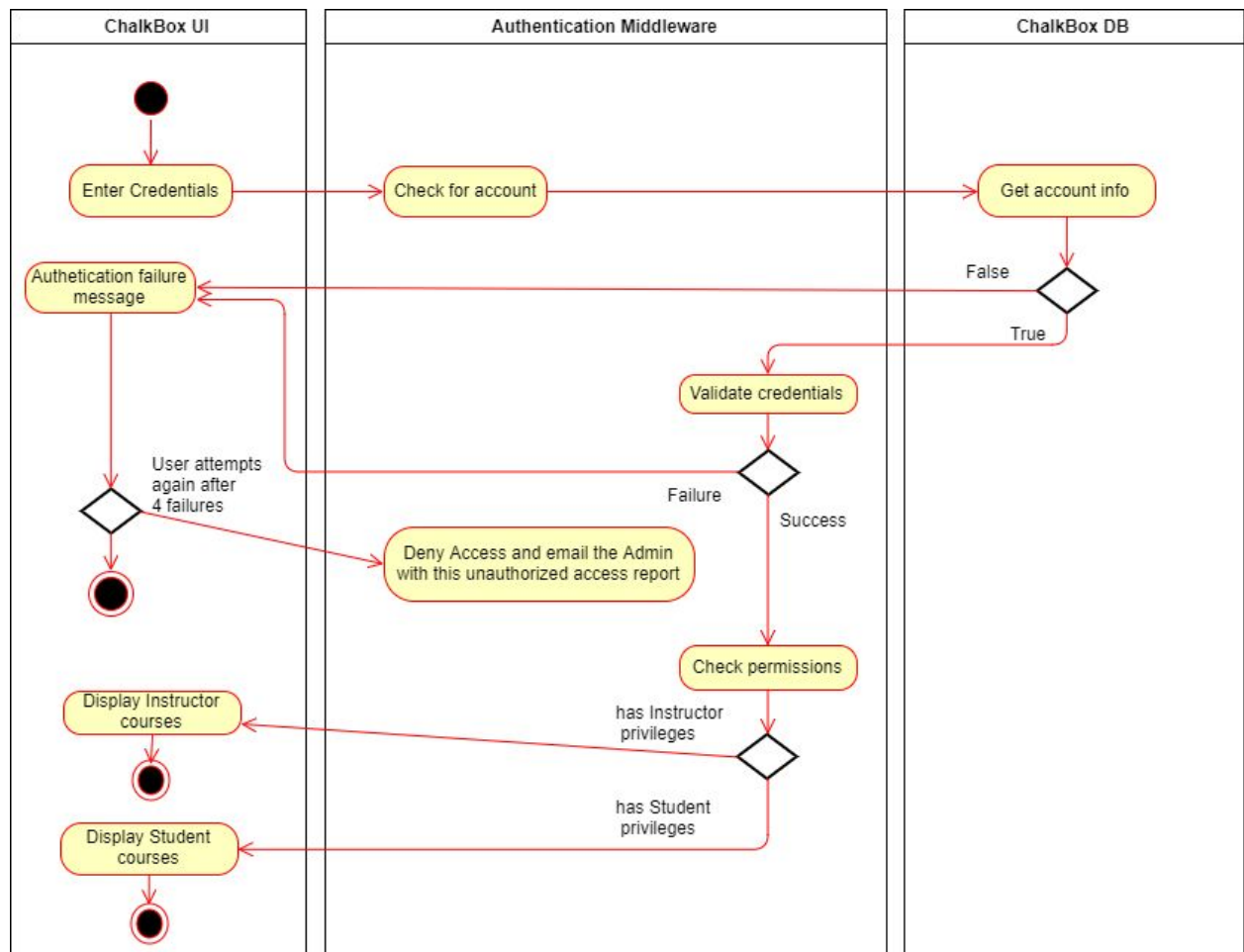
Operation Signature	<b>ExtractObject (Object: Class): Class</b>
Operation description	Extract corresponding object from the table
Operation preconditions	The mapping for the object exists
Operation postconditions	The required object get extracted

## 5. Architecture Behavior

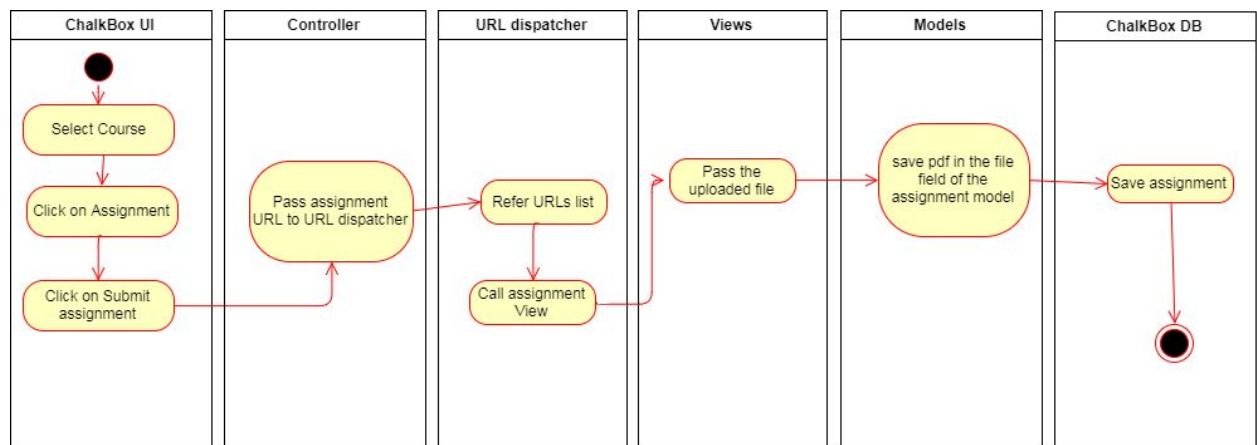
### (5.1) LOGIN: Security Scenario

The ChalkBox-Learning Management System handles security as follows:

1. When a user tries to login with his/her credentials it checks if the user exists in the system by querying the ChalkBox DB. If not, the ChalkBox UI displays the error message “username and password does not exist”.
2. Checks if the username and password match. If not, the ChalkBox UI displays the error message “Invalid password”.
3. If the username and password exist and match, it checks the privileges of the user and allows access to content based on the privilege.
4. If there are multiple failed login attempts the admin is notified immediately to prevent brute force attack.

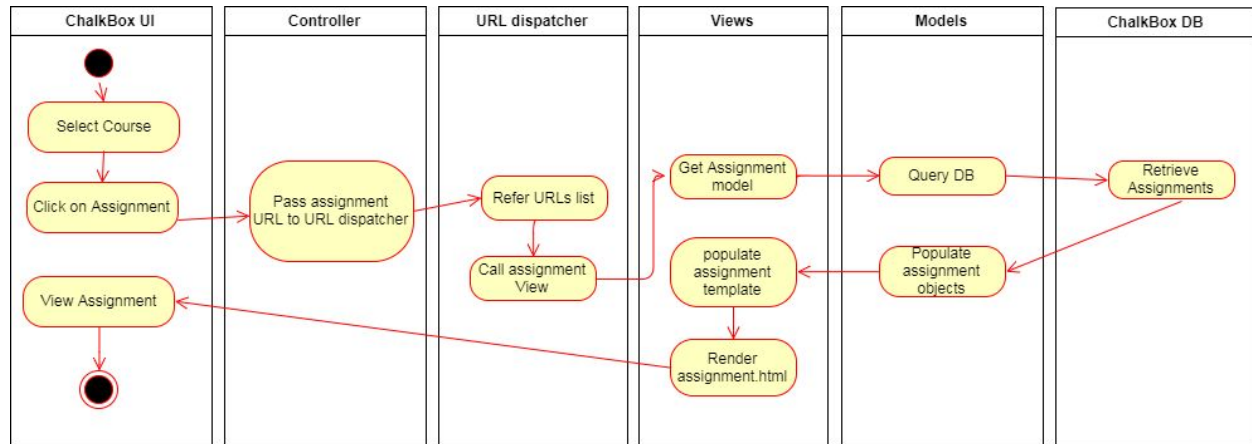


## (5.2) SUBMIT ASSIGNMENT SCENARIO

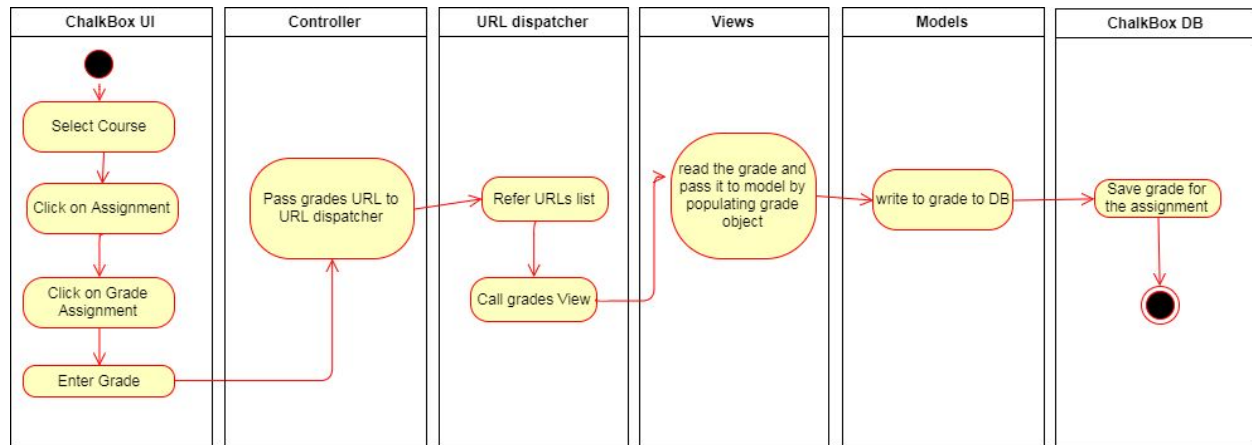




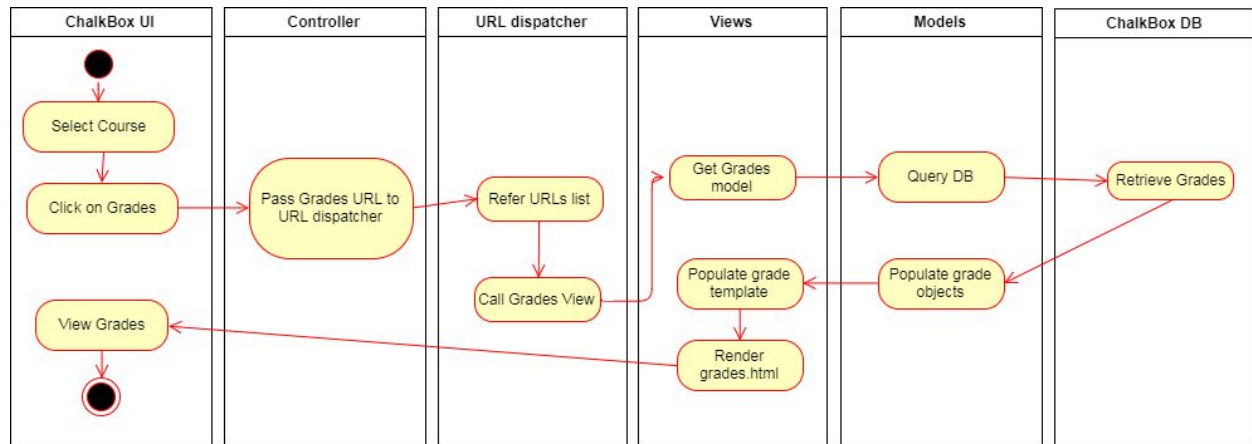
### (5.3) VIEW ASSIGNMENTS SCENARIO



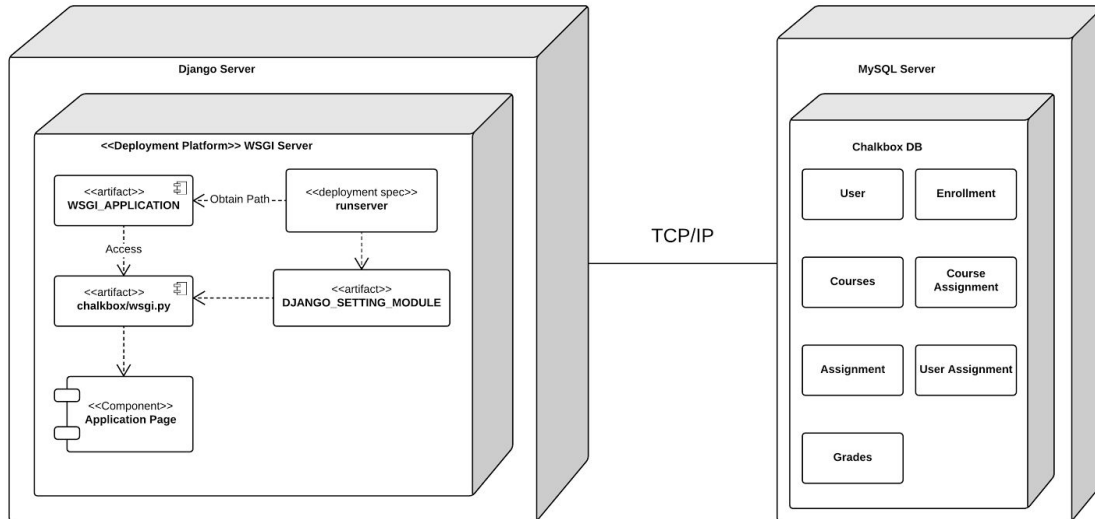
### (5.4) SUBMIT GRADES SCENARIO



### (5.5) VIEW GRADES SCENARIO



## 6. Architecture Deployment View



Django's primary deployment platform is WSGI. It is the Python standard for web servers and applications. We decide to use WSGI to set up deployment.

- **WSGI.**

The key concept of deploying with WSGI is the **application** callable which the application server uses to communicate with the code. It's commonly provided as an object named `application` in a Python module accessible to the server. In our project, we will name it **Chalkbox/wsgi.py**. To start the deployment. We first need to call **startproject** command to create a file **Chalkbox/wsgi.py** that contains such an **application** callable.

- **Server.**

Django has its own built-in server, namely the **runserver** command. By calling this command in terminal, we can start running Django built-in server.

WSGI servers obtain the path to the **application** callable from their configuration. Django built-in server reads the path from the **WSGI\_APPLICATION** setting. By default, the application path is set to **Chalkbox.wsgi.application**, which points to the **application** callable in **Chalkbox.wsgi.py**.

- **Configuration.**

After the WSGI server loads the application, Django needs to import the settings module. The setting module control the entire application.

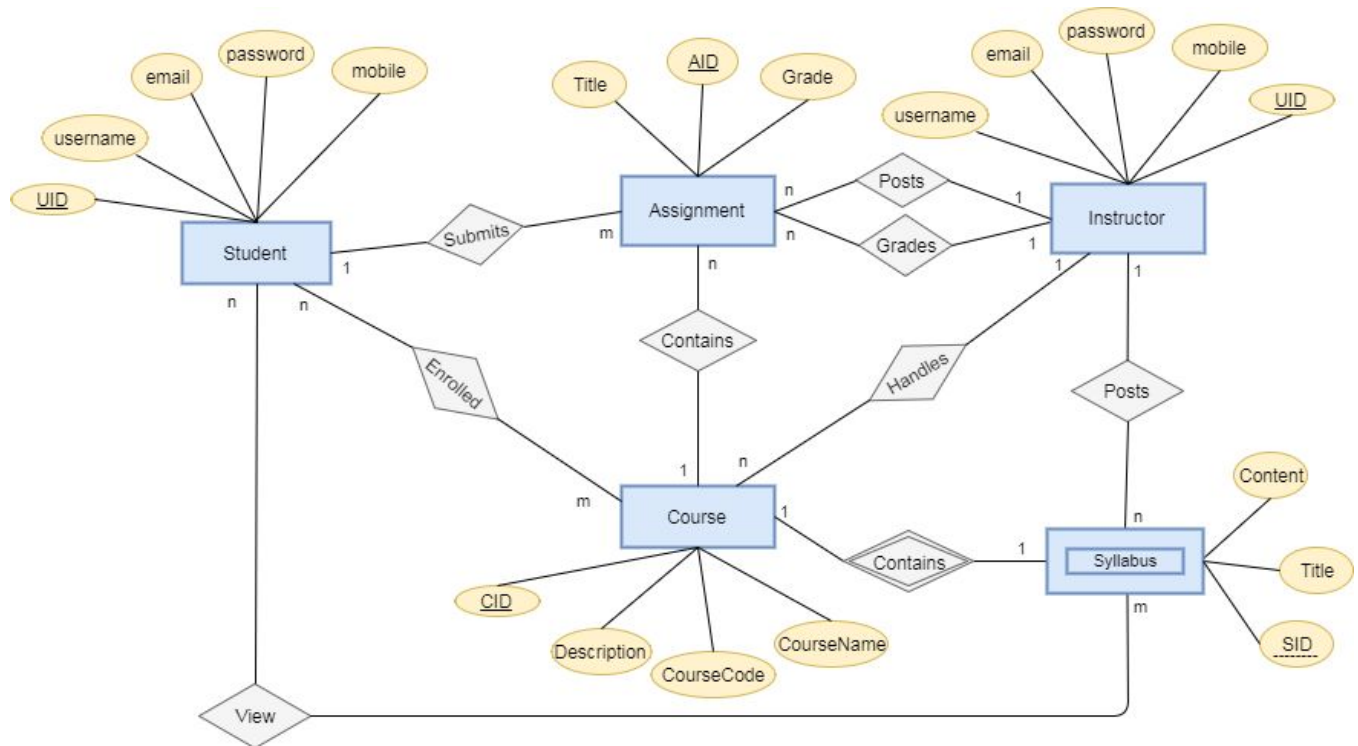
Django uses the **DJANGO\_SETTING\_MODULE** environment variable to locate the appropriate settings module. We can change a different value for development and production.

- **Database**

To connect database, we first need to install MySQL server and database connector. Create a new database named **Chalkbox DB** as our database. The Django server connects and communicates with **Chalkbox DB** by TCP as shown in the diagram. After logging in the MySQL server in terminal, we could create the initial database that will hold the data for the system. Finally, we will be adding the database connection credentials to our Django application. Navigate to the **settings.py** and replace current **DATABASES** to MySQL. We configure the database dictionary so that it knows to use MySQL as our database backend and from what file to read the database connection credentials.

## 7. Information View

### (7.1) ER Diagram



## 8. Requirements Allocation

Use Cases/ Architecture Components	Chalkbox UI	Controller	URL dispatcher	Views	Models	Chalkbox DB	ORM
See the list of students	X	X	X	X	X	X	X
Modify syllabus	X	X	X	X	X	X	X
Modify assignments	X	X	X	X	X	X	X
See submissions	X	X	X	X	X	X	X
Grade submissions	X	X	X	X	X	X	X
View syllabus	X	X	X	X	X	X	X
View assignments	X	X	X	X	X	X	X
Submit assignment	X	X	X	X	X	X	X
View grades	X	X	X	X	X	X	X
Maintain grade book		X			X	X	X

## 9. Architecture Work Allocation

Architecture component name/ Team member	Yidi Kou	Prathima Devanath	Yu Zhao	Yigang Yao	Karthik Kembhavi
Student Interface		Development			Testing
Instructor Interface			Testing		Development
Authentication Middleware		Development		Testing	
Controller		Testing		Development	
Models	Testing				Development

<b>Views</b>		Testing	Development		
<b>Url Dispatcher</b>	Development		Testing		
<b>ORM</b>	Testing			Development	
<b>Database</b>	Development				Testing