Team 2- ChalkBox

Learning Management System
Software Design Document (SDD)

Team members:

Yidi Kou

Prathima Devanath

Yu Zhao

Yigang Yao

Karthik Kembhavi

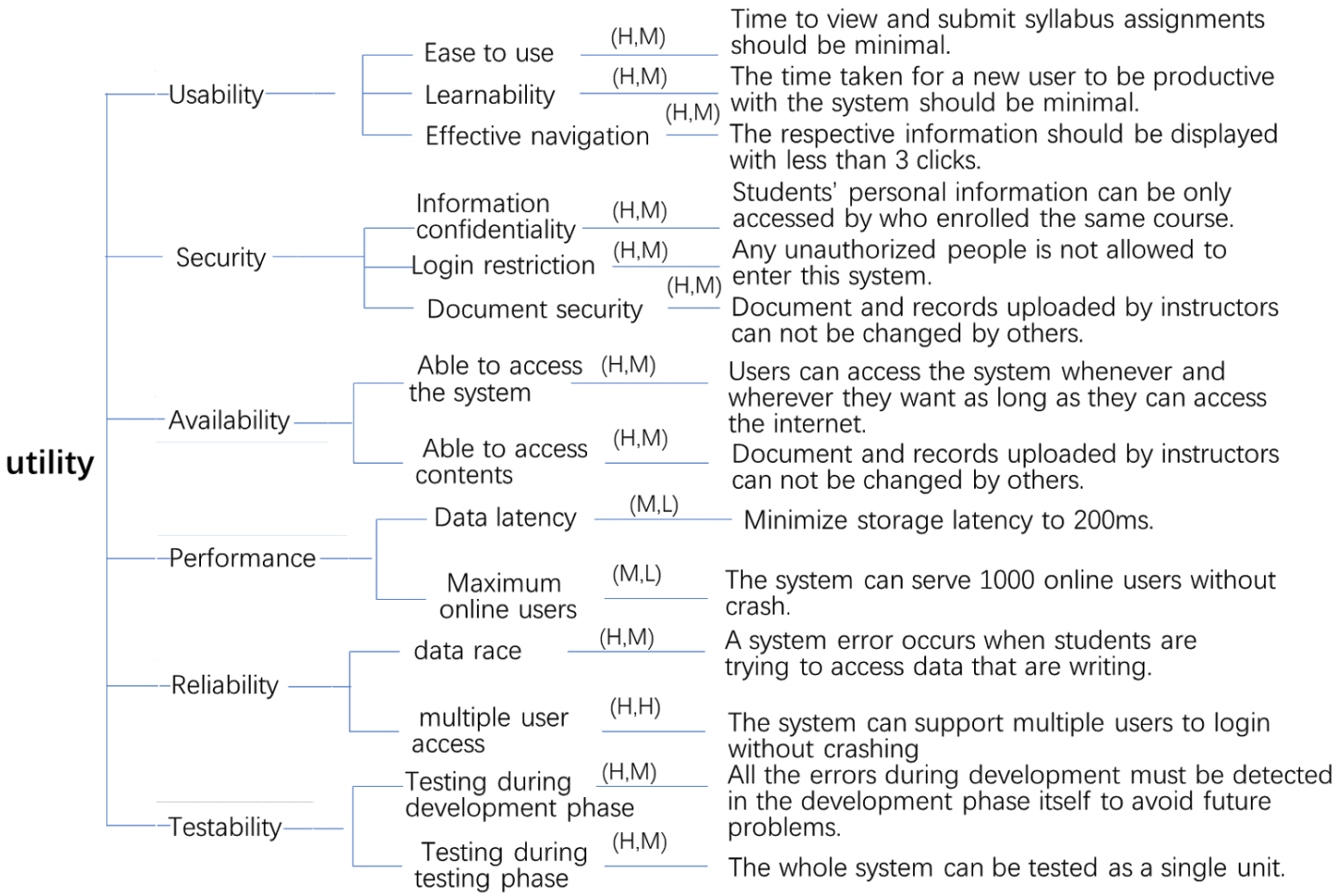
1. Product Overview

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



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

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

Quality attribute	Usability

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

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

Quality attribute	Availability
Scenario name	Able to access the system
Source	Users(Student/Instructor)

Stimulus	Users want to use the system and tries to access it.
Artifact	website
Environment	Run time
Response	The system should be accessed easily and conveniently
Response measure	Whether users can access the system or not

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

Quality attribute	Performance
Scenario	Data latency
Source	User(student/Instructor)

Stimulus	Users want to store/get data from this application.
Artifact	The System
Environment	At Runtime
Response	Latency time of the system to respond to a Get/Post request.
Response Measure	The system should answer the request of the users and store/get the data from database without obvious delay.

Quality attribute	Performance
Scenario	Maximum Online Users
Source	User(student/Instructor)
Stimulus	Lots of user are online in the same time
Artifact	The System
Environment	At Runtime
Response	The system should answer each user's request normally without crash
Response Measure	Can hold 3000 online users in the same time without crash

Quality attribute	Reliability
Scenario name	Multiple user access
Source	Users
Stimulus	Multiple users try to access the same piece of data at the same time
Artifact	website

Environment	Run time
Response	The system should respond the correct information to every user correctly.
Response measure	The response time and the correctness of the return information.

Quality attribute	Testability
Scenario	Testing in various phases of development.
Source	Developer, Tester, User.
Stimulus	Developer or the Tester want to test the system or a part of it
Artifact	The whole system or just a part of it, code.
Environment	During development, testing phase.
Response	Comparing the expected outcomes with the actual output.
Response measure	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

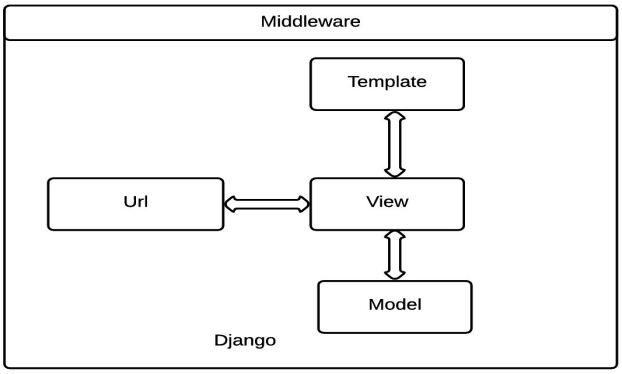
- Accessible for a range of devices
- Improved interoperability
- Easier installation and maintenance
- Adaptable to increased workload
- Increased Security
- Flexible technologies

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.

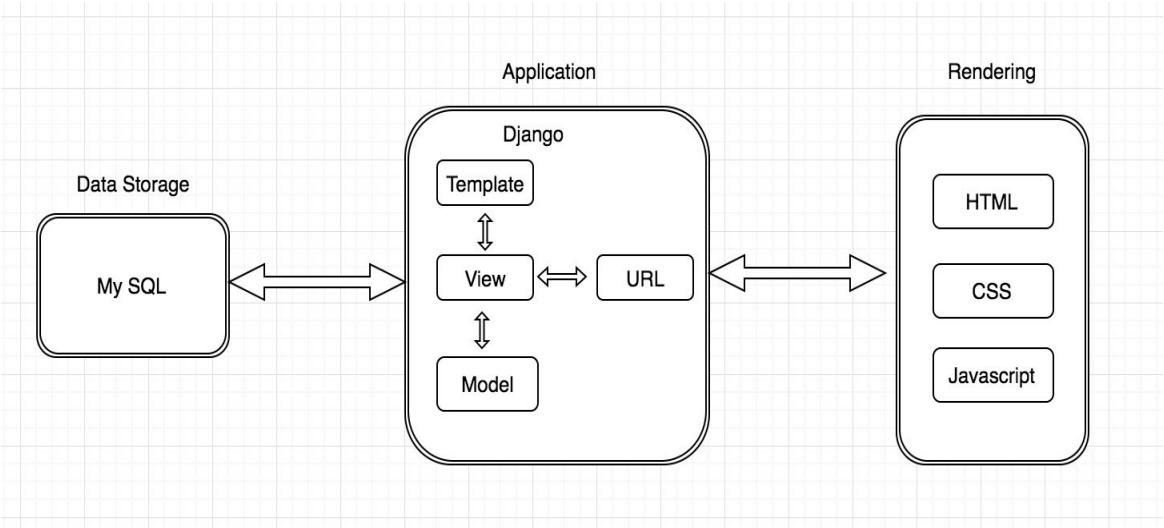
Django includes a lightweight web server which can be used in debugging and testing. However, this server only works under local environment. It is not an appropriate choice on an application using by many users.

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.

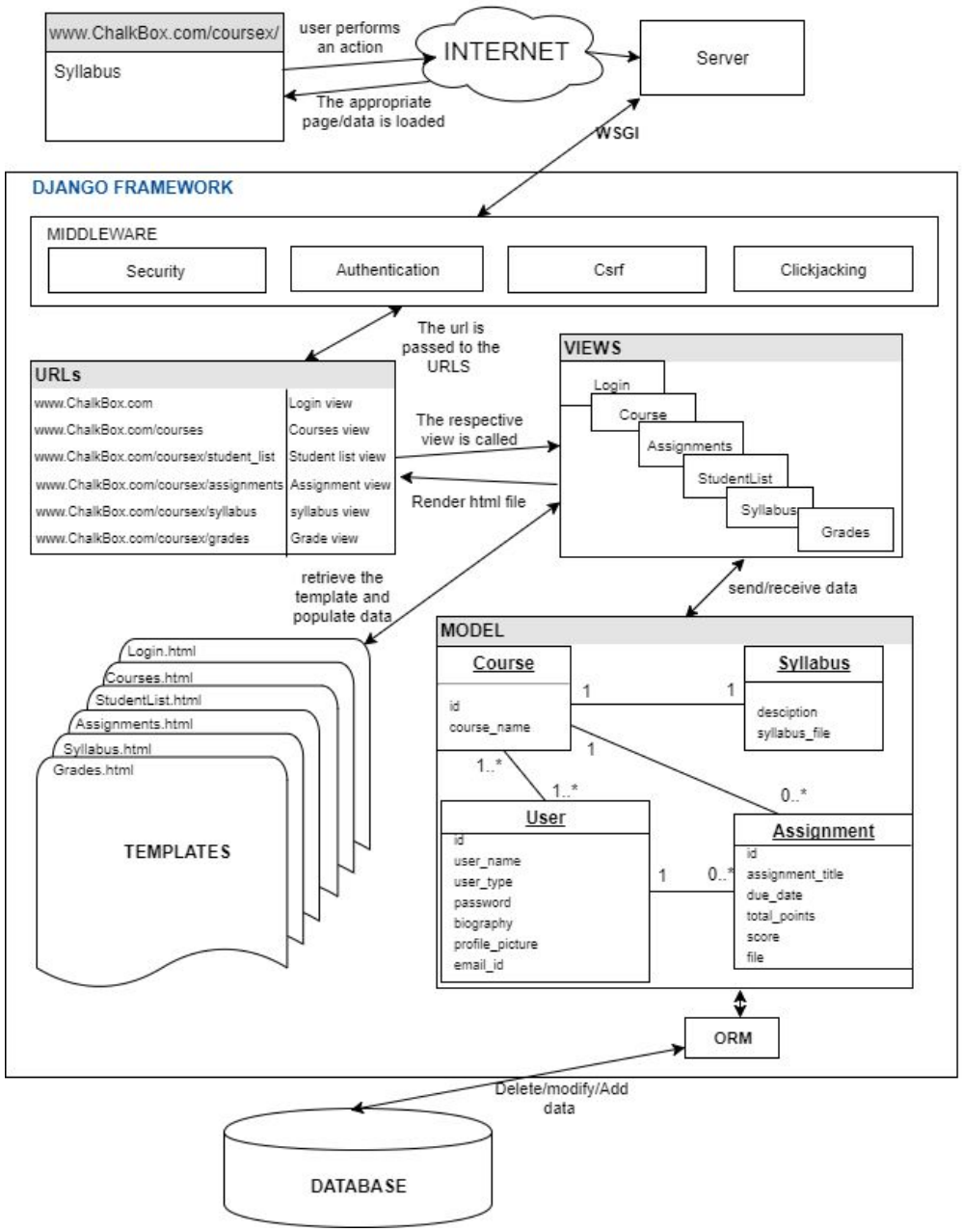
We'll then set up Apache in front of our application so that it can handle client requests directly before passing requests that require application logic to the Django app. We will do this using the mod_wsgi that can communicate with Django over the WSGI interface specification. Mod_wsgi is an Apache 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.

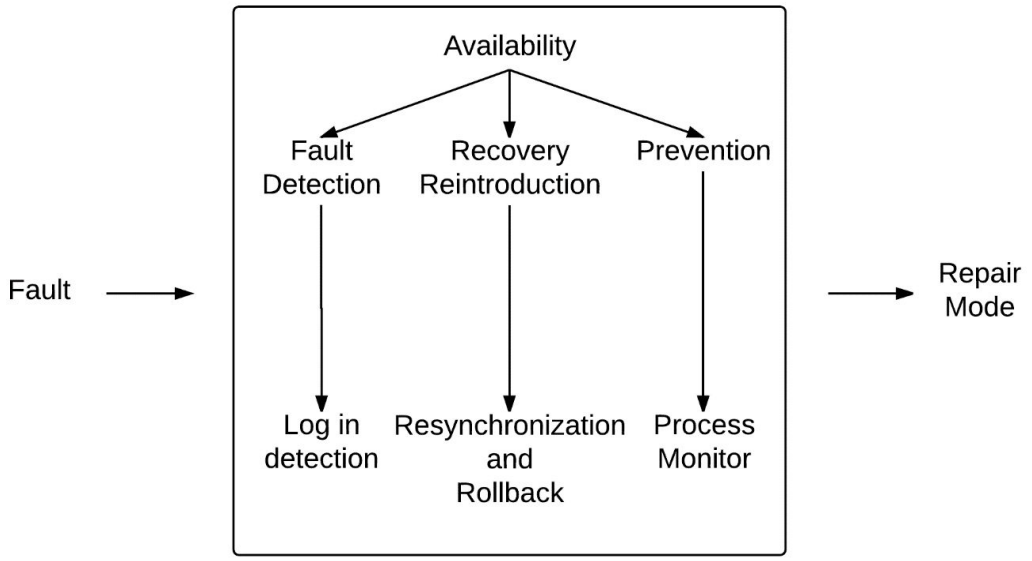
WSGI: Is an interface between web servers and web applications or frameworks for the Python programming language.

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

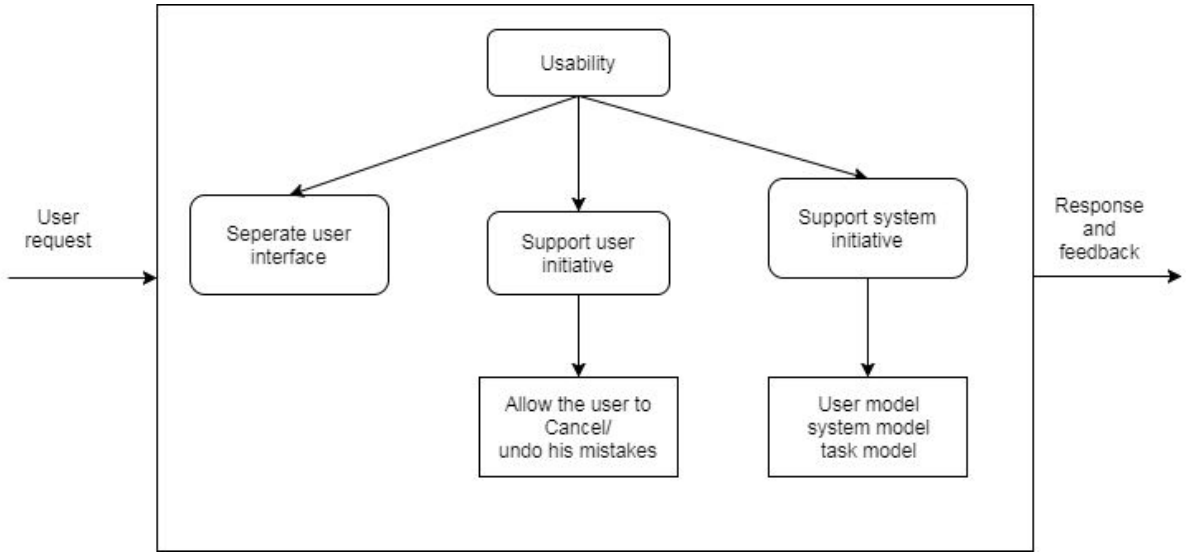
- **Tactics for availability**



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**

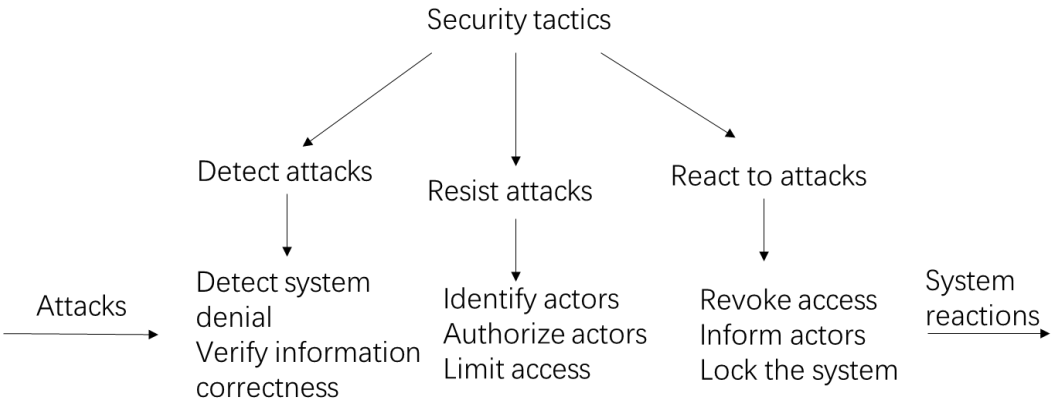


User satisfaction for the LMS can be achieved using the tactics mentioned above.

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**

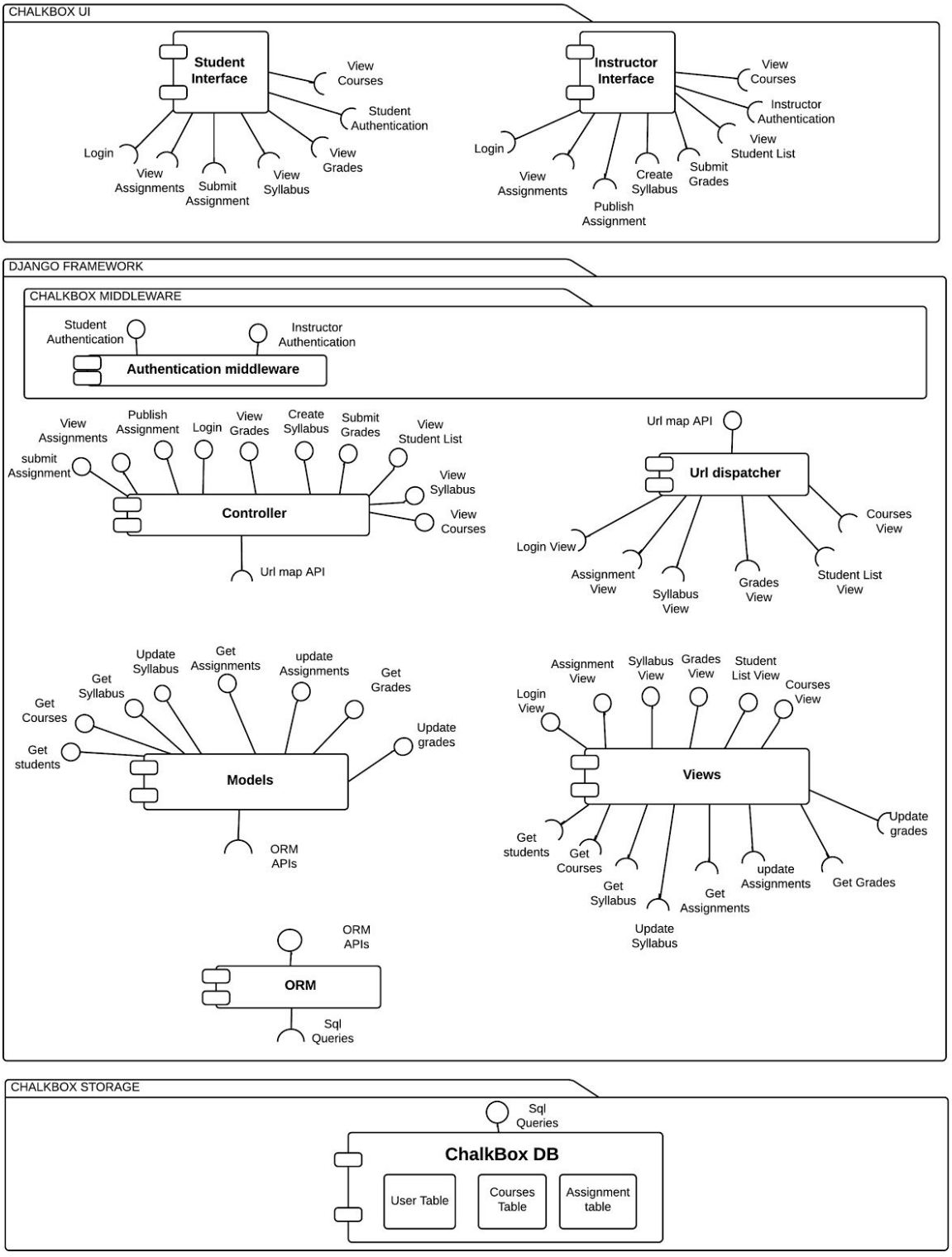


Detect Attacks: In our system, we have the middleware component in Django, which works for both user authentication and authorization. We save the username and password data in database. Once there is an attempt to login, the system can check if it matches the records in database. Then it will returns 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

Interface Student Authentication	
Operation signature	**CheckStudentExists (UserName: String):Boolean**
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	**CheckStudentIsAuthenticated (UserName: String):Boolean**
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
Interface Instructor Authentication	
Operation signature	**CheckInstructorExists (UserName: String):Boolean**
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	**CheckInstructorIsAuthenticated (UserName: String):Boolean**
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

Interface Url map API	
Operation signature	**SearchUrl (Url: String): Function view**
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	**CallView (view: function)**
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

Interface Login View	
Operation signature	**ReadUserName (UserName: String)**
Operation description	Read the username
Operation preconditions	Username not null
Operation postconditions	Read the username entered by the user
Operation signature	**ReadPassword (password: hiddenString)**
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	**HidePassword (password: hiddenString)**
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 "*******"
Interface Course View	
Operation signature	**GetCourses (UserName: String):List of Courses**
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	**DisplayCourse(Courses: List)**
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.
Interface Assignment View	
Operation signature	**GetAssignments (UserName: String, CourseName:String):List of Assignments**
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	**DisplayAssignments (Assignments: List)**
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	**SubmitAssignments (UserName: String, CourseName:String, Assignment:Object):Boolean**
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	**PublishAssignment (CourseName: String, Assignment: Object):Boolean**
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.
Interface Syllabus View	

Operation signature	**GetSyllabus (CourseName:String):Syllabus object**
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	**DisplaySyllabus (syllabus: object)**
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	**PublishSyllabus (Coursename: String, syllabus: object)**
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.
Interface Grades View	
Operation signature	**GetGrades (UserName: String, CourseName:String):List of Grades**
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	**DisplayGrades (Grades: List)**
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	**SubmitGrade (UserName: String, CourseName:String, Assignment:Object, Grade: float):Boolean**
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.
Interface Student List View	
Operation signature	**GetStudents (CourseName:String):List of Students**
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

Interface Get Students	
Operation signature	**GetStudentsOfCourse(CourseId: integer): list**
Operation description	Get the student list of a certain course.
Operation preconditions	couseId is valid
Operation postconditions	The student list is returned.
Operation signature	**GetStudent(CourseId: integer, username: string): student object**
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
Interface Get Courses	
Operation signature	**GetCourse(courseId: integer): course object**
Operation description	Get a certain course
Operation preconditions	courseId is valid
Operation postconditions	The course is returned.
Operation signature	**GetCourseListOfStudent(studentId: integer) list**
Operation description	Get the list of courses of that student
Operation preconditions	studentId is valid.
Operation postconditions	The course list is returned.
Interface Get Syllabus	
Operation signature	**GetSyllabus(couseId: integer): syllabus object**
Operation description	Get the syllabus of a certain course
Operation preconditions	courseId is valid
Operation postconditions	The syllabus is returned.
Interface Update Syllabus	
Operation signature	**UpdateSyllabus(courseId: integer): syllabus object**
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.

Interface Get Assignments	
Operation signature	**GetAssignment(courseId: integer, assignmentId: integer) assignment object**
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	**GetAssignmentListOfCourse(courseId: integer): list**
Operation description	Get the list of assignment of that course.
Operation preconditions	The course if valid and the student is enrolled in that course.
Operation postconditions	The list of assignments is read by the student.
Interface Update Assignments	
Operation signature	**UpdateAssignment(courseId: integer, assignmentId: integer) assignment object**
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.
Interface Get Grades	
Operation signature	**GetGradeListOfAssignment(assignmentId: integer, courseId: integer): list**
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	**GetGradeListOfStudent(studentId: integer): list**
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.
Interface Update Grades	
Operation signature	**UpdateGrade(studentId: integer, assignmentId: integer, courseId: integer): grade object**
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 **Submit Assignment**	
Operation signature	**GetSubmitRequest (UserName: String):Boolean**
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	**MapRequestToAssignmentView (UserName: String):Boolean**
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 **View Student Assignment**	
Operation signature	**GetViewAssignmentRequest (AssignmentName: String):Boolean**
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	**MapRequestToAssignmentView (AssignmentName: String):Boolean**
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 **Publish Assignment**	
Operation signature	**GetPublishRequest (AssignmentName: String):Boolean**
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	**MapRequestToAssignmentView (AssignmentName: String):Boolean**
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 **Login**	
Operation signature	**GetLoginRequest (UserName: String):Boolean**
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	**MapRequestToLoginView (UserName: String):Boolean**
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
Interface **View Grades**	
Operation signature	**GetViewGradesRequest (UserName: String):Boolean**
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	**MapRequestToGradesView (UserName: String):Boolean**
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
Interface **Create Syllabus**	
Operation signature	**GetCreateSyllabusRequest (UserName: String):Boolean**
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	**MapRequestToCreateSyllabusView (UserName: String):Boolean**
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 **Submit Grades**	
Operation signature	**GetSubmitGradesRequest (UserName: String):Boolean**
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	**MapRequestToGradeView (UserName: String):Boolean**
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 **View Student List**	
Operation signature	**GetViewStudentListRequest (UserName: String):Boolean**
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	**MapToStudentListView (UserName: String):Boolean**

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
Interface **View Syllabus**	
Operation signature	**GetViewSyllabusRequest (UserName: String):Boolean**
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	**MapRequestToSyllabusView (UserName: String):Boolean**
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
Interface **View Courses**	
Operation signature	**GetViewCoursesRequest (UserName: String):Boolean**
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	**MapRequestToViewCoursesView (UserName: String):Boolean**
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

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

Operation Signature	**ViewSyllabus (UserName: String):Boolean**
Operation description	An interface to view syllabus
Operation preconditions	Student must be logged in
Operation postconditions	Student can view syllabus
Operation Signature	**ViewGrades (UserName: String):Boolean**
Operation description	An interface to view grades
Operation preconditions	Student must be logged in
Operation postconditions	Student can view grades
Operation Signature	**StudentAuthentication (UserName: String):Boolean**
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	**ViewCourses (UserName: String):Boolean**
Operation description	An interface to view courses
Operation preconditions	Student must be logged in
Operation postconditions	Student can view courses
Interface "Instructor Interface"	
Operation Signature	**InstructorLogin (UserName: String):Boolean**
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	**ViewStudentsAssignment (UserName: String):Boolean**
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	**SubmitGrades (UserName: String):Boolean**
Operation description	An interface to submit grades
Operation preconditions	Instructor must be logged in
Operation postconditions	Instructor can submit grades
Operation Signature	**ViewCourses (UserName: String):Boolean**
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 **SQL Queries**	
Operation Signature	**Query (SQL request: Select): Table**

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	**Read (SQL request: Select): Table**
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	**Write (SQL request: Insert): Boolean**
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 **ORM API**	
Operation Signature	**SearchObject (Object: Class): Boolean**
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

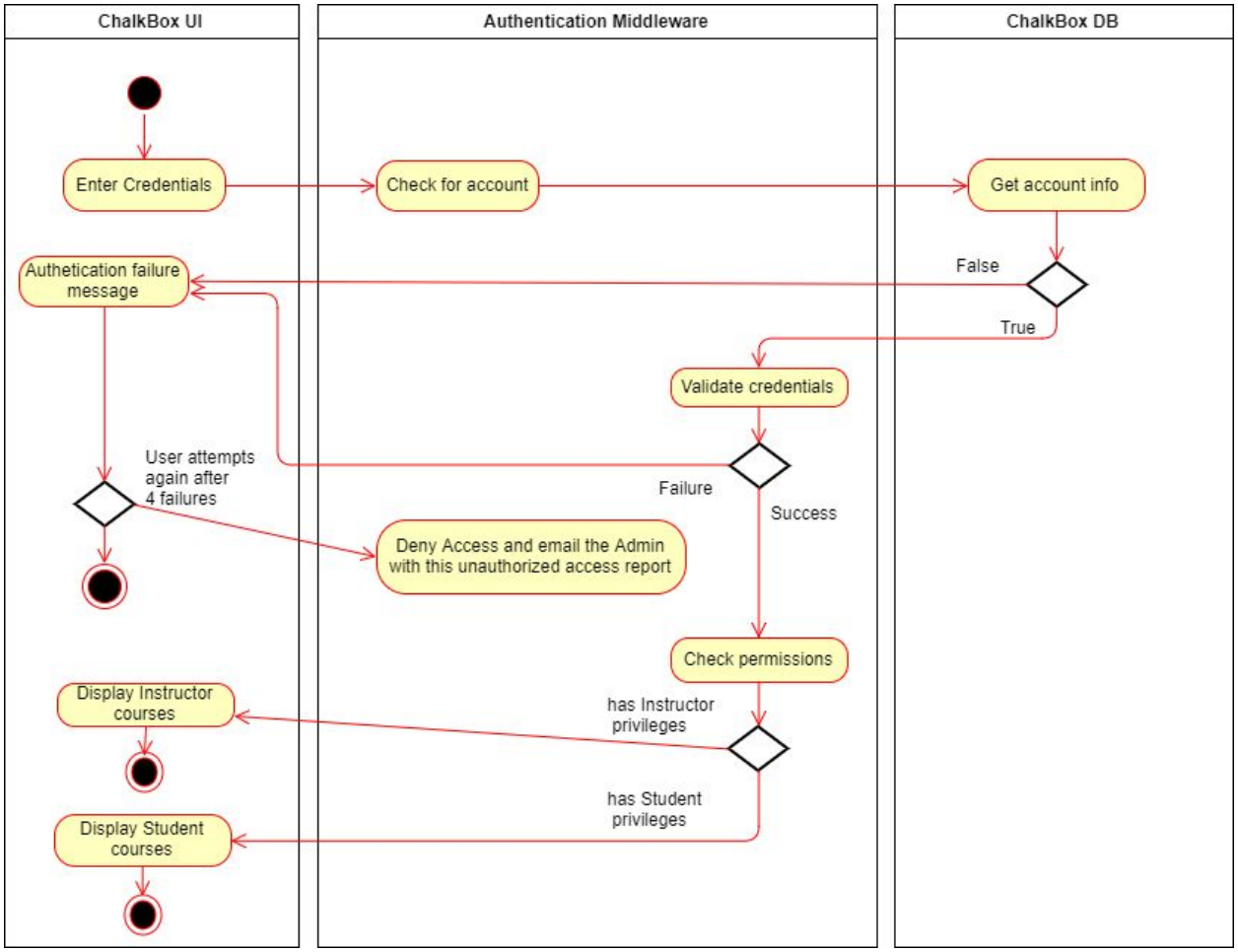
Operation Signature	**ExtractObject (Object: Class): Class**
Operation description	Extract corresponding object from the table
Operation preconditions	The mapping for the object exists
Operation postconditions	The required object get extracted

5. Architecture Behavior

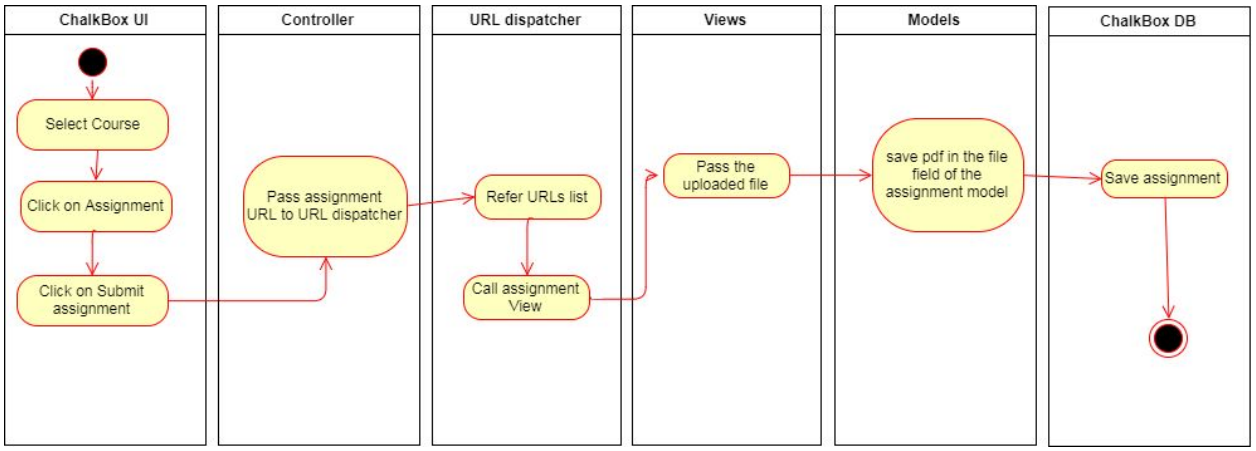
(5.1) LOGIN: Security Scenario

The ChalkBox-Learning Management System handles security as follows:

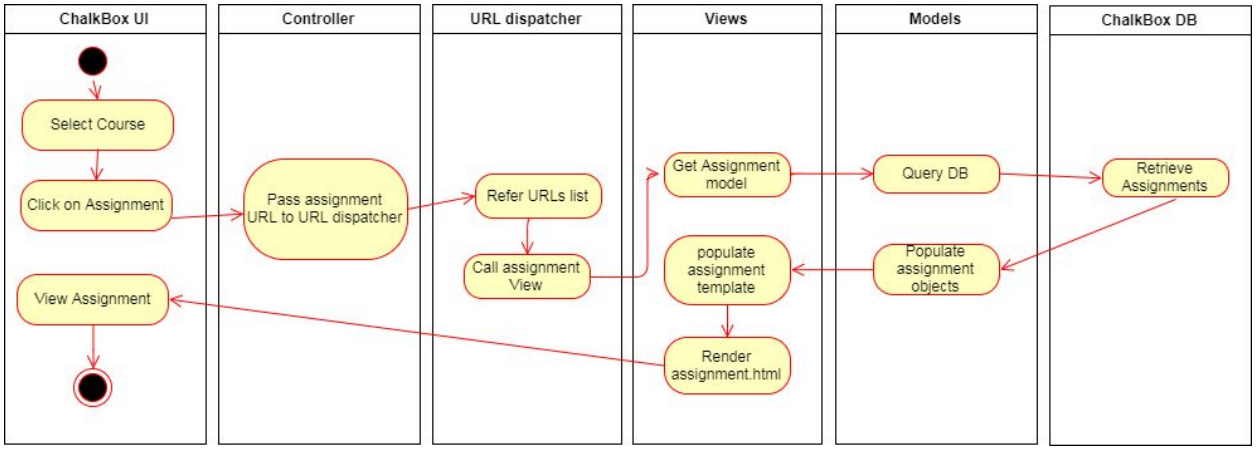
1. When a user tries to login with his/her credentials it checks if the user exists in the system by querying the ChalkBox DB. If not, the ChalkBox UI displays the error message “username and password does not exist”.
2. Checks if the username and password match. If not, the ChalkBox UI displays the 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.



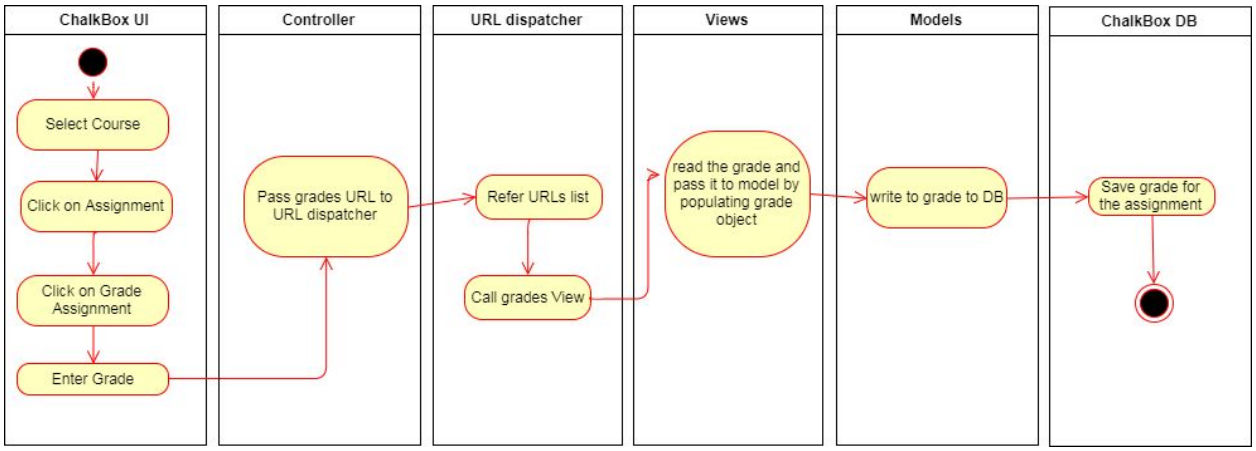
(5.2) SUBMIT ASSIGNMENT SCENARIO



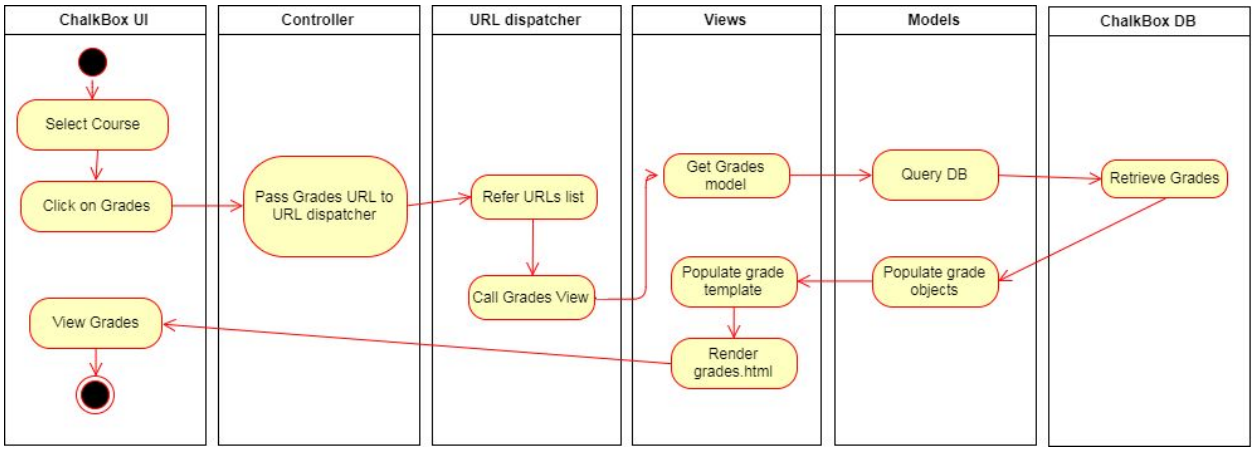
(5.3) VIEW ASSIGNMENTS SCENARIO



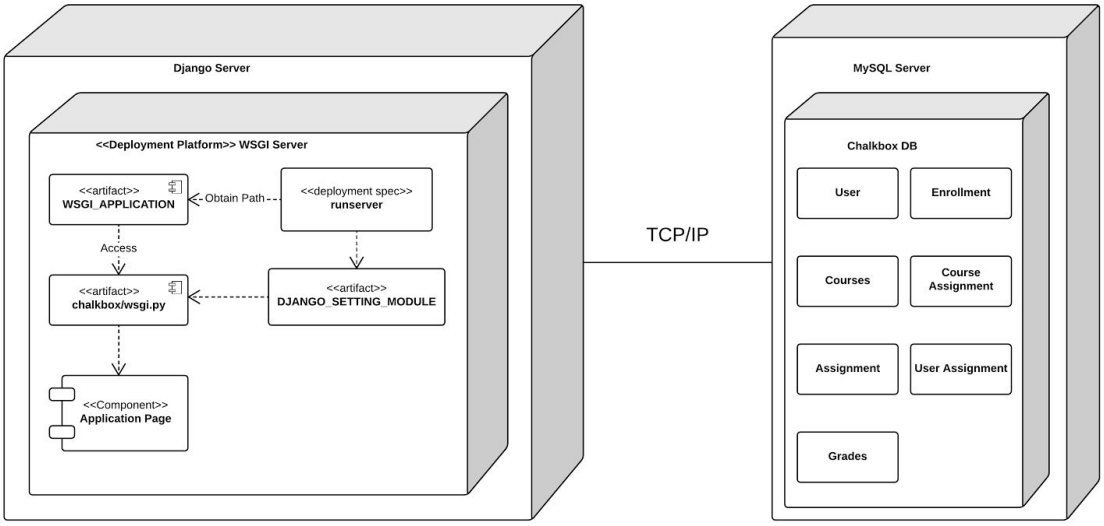
(5.4) SUBMIT GRADES SCENARIO



(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**.

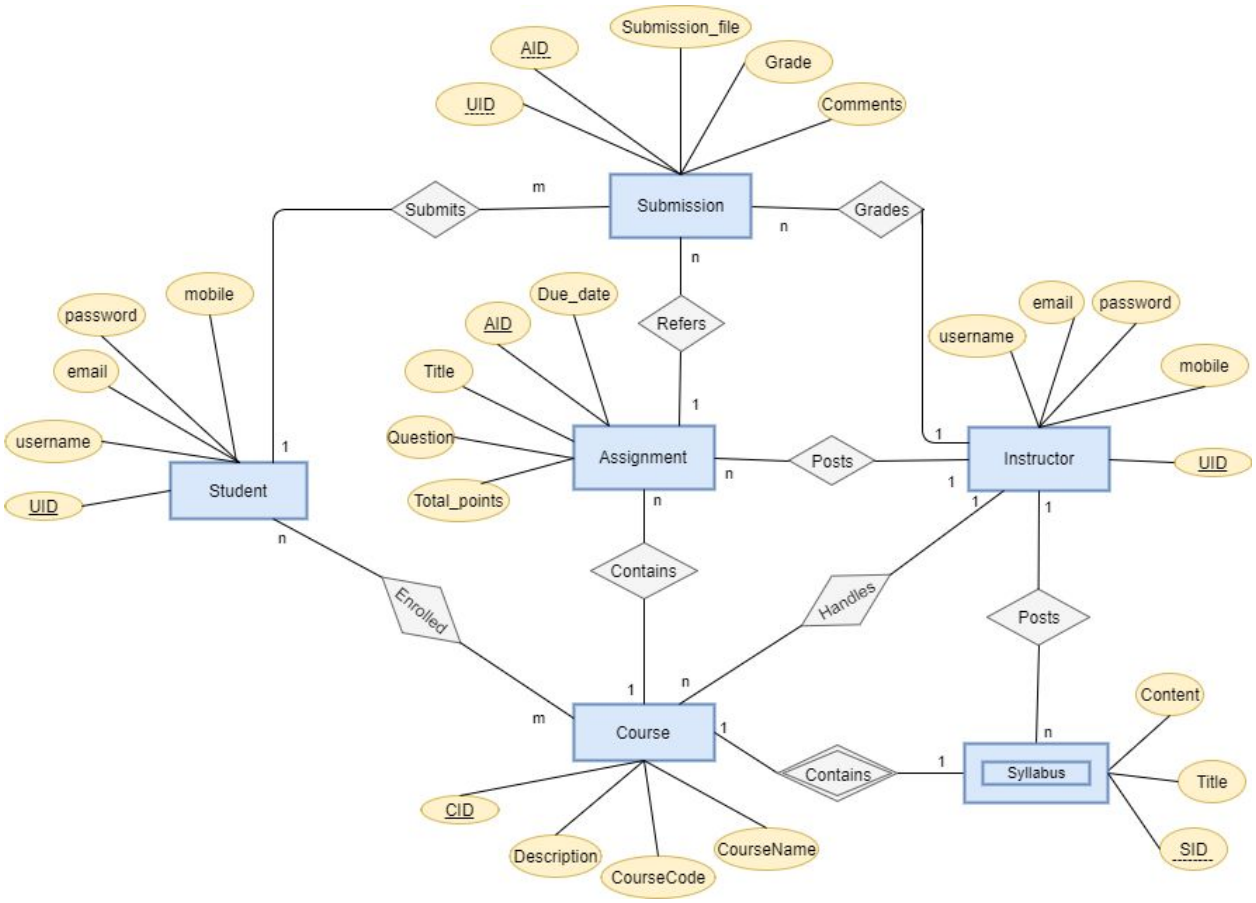
After the WSGI server loads the application, Django needs to import the settings module. The setting module control the entire application.
Django uses the **DJANGO_SETTING_MODULE** environment variable to locate the appropriate settings module. We can change a different value for development and production.

- **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
Views		Testing	Development		
Url Dispatcher	Development		Testing		
ORM	Testing			Development	
Database	Development				Testing

10. Detailed Design

Common Key Decisions and Design Patterns applicable to all the components:

1. Single Responsibility Principle:

The Project is divided into modules as:

- Accounts: handles all the User attributes and user functionality like displaying the user profile when the “profile” link is clicked.
- Assignments: handles all the assignment attributes and functionality such as assignment creation, deletion and modification.
- Courses: handles all the course attributes and functionality such as displaying course list, each course description. .
- Submissions: handles the grades for each assignment submission, also used to generate the grade book.

It can be observed that the responsibility is split between different modules called “Apps” in the ChalkBox project. The static and behavior model of the project has more details about this incorporated principle.

2. Adapter Pattern:

The ORM between models and the database acts as an adapter in ChalkBox. The object oriented language used in models is not understood by the database. And the queries of the database is not understood by the models. The ORM acts as a middle layer to facilitate the communication.

3. Template Pattern:

Templates in django as the name suggests follows the Template pattern. A base template is created with the base html. In our case the base.html has the “navbar” which is repeated in all the pages throughout the site. Hence all other pages can extend this page without Rewriting the code in every html page. The example below explains this pattern.

base.html:

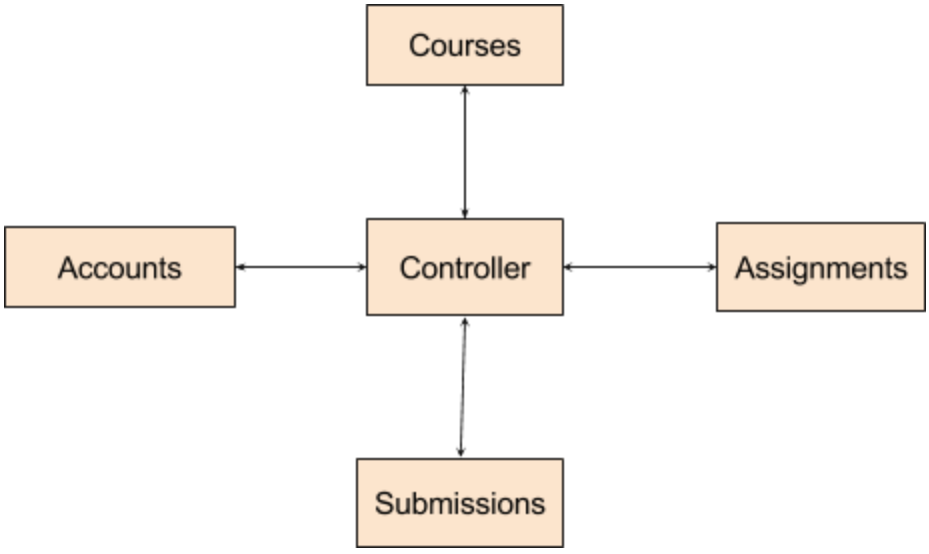
```
{% load staticfiles %}

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="icon" href="../../favicon.ico">

    <title>{% block head_title %}ChalkBox{% endblock %}</title>

    {% include 'main_css.html' %}
    <style>
      {% block style %}{% endblock %}
    </style>
    </head>
    <body>
      {% include 'navbar.html' %}
      <div class="container">

      {% block content %}

      {% endblock %}

      </div> <!-- /container -->

      {% include "main_js.html" %}
    </body>
</html>
```

assignment.html:

```
{% extends "base.html" %}

{% block title %}{{ assignment.title }}{% endblock %}

{% block content %}
    <h1>Assignments</h1>
        <table class="table">
        <thead>
            <tr>
              <th scope="col">Assignment title</th>
              <th scope="col">Due date</th>
              <th scope="col">total points</th>
            </tr>
        </thead>
        <tbody>
            {% for assignment in assignment_list %}
                <tr>
                  <td><a href="{% url 'courses:submissions:submission_list' cid=id aid=obj.id %}">{{ obj.title }}</a></td>
                  <td>{{ obj.due_date }}</td>
                  <td>{{ obj.total_points }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

4. Mediator pattern:

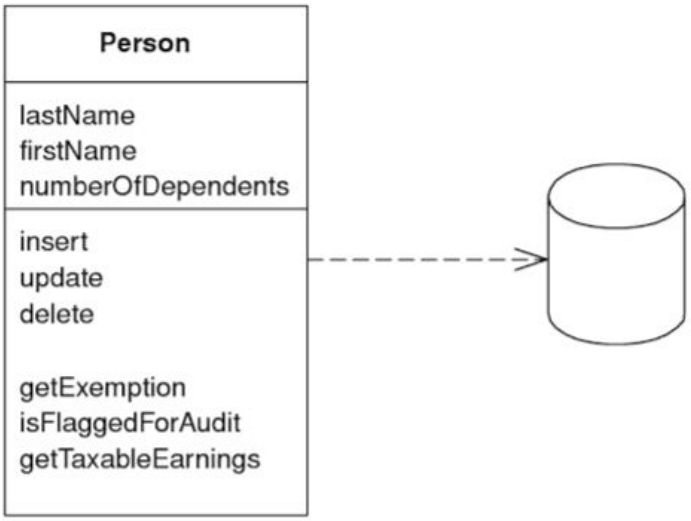
The controller in ChalkBox works as a mediator pattern. Each component has a model, view and template part associated with it. The Controller in django framework acts as a mediator for communication between different components.



5. Fowler Patterns: Martin Fowler has described patterns for the development of enterprise systems known as the “Fowler Patterns” in his book “Patterns of Enterprise Application Architecture”. Our project has incorporated few Fowler patterns.

a. Active Record Pattern(An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data):

The Models in our project encapsulates the database access and adds domain logic on that data.



***Reference:** Patterns of Enterprise Application Architecture By Martin Fowler*

```python
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponse
from .models import Course
from accounts.models import User

# Create your views here.
def course_home(request):
    user = User(id=request.user.id)
    queryset = user.courses.all()
    context = {
        "object_list": queryset,
        "title": "Courses",
    }
    return render(request, "course_home.html", context)
```

b. Identity Field(Saves a database ID field in an object to maintain identity between an in-memory object and a database row): The id field saves the database ID in an object to maintain identity.

```python
from django.db import models
from datetime import datetime
from django.utils import timezone

# Create your models here.
class Assignment(models.Model):
    title = models.CharField(max_length=120)
    due_date = models.DateTimeField()
    total_points    = models.IntegerField()
    assigment_description = models.TextField(blank=True)
    updated     = models.DateTimeField(auto_now=True)
    timestamp   = models.DateTimeField(auto_now_add=True)
    question    = models.FileField(null=True, blank=True)

    def __str__(self):
        return str(self.id)

    def is_past_due(self):
        return timezone.now() > self.due_date
```

The id field is added for all the model objects by default.

10.1. User Accounts Component

Accounts component is responsible for all the User related functionality.

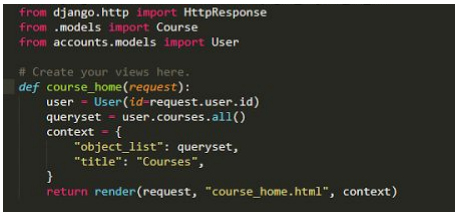
The Accounts component contains:

- Models.py: contains the User class
- Views.py: contains the functions like student_list() and user_details()
- Urls.py: responsible for routing and calling the right view based on user request.
- Admin.py: used for displaying the user tables in the admin view
- Tests.py: used for writing test cases

- __init__.py: initialization of the app
- App.py: app config

10.1.1 Key Decisions and Rationale

Key Decisions	Rationale	Design Pattern
User Login and Authentication	ChalkBox uses Django's Authentication middleware to do the authentication and handle **cookie-based user sessions**. It also consists a configurable **password hashing** system. Django authentication system handles both authentication and authorization. **Authentication** verifies a user is who they claim to be. **Authorization** determines what an authenticated user is allowed to do. Once user authentication is successful the http request is routed to the course home page	Command Pattern: HttpRequest object encapsulates GET or POST command. When a page is requested, Application creates an HttpRequest object that contains metadata about the request. Then it loads the appropriate view, passing the HttpRequest as the first argument to the view function. Detailed explanation of this pattern is presented below this table.
After Login the courses of only that student should be displayed in the home page	Linking only the user enrolled courses with the respective User using ManytoMany Field in django. 	
Creating custom users	Adding custom attributes and methods to django builtin User and creating this user using Factory Pattern.	The Factory pattern allows you to call a non constructor method to create a custom user. We can have any kind of customization to user class and create this type of user object. detailed explanation is below this table.
Uniquely identifying Users	Users are uniquely identified by their email address. However the for easier access the User class also contains an auto generated ID field which represents the database ID	Identity Field Fowler Pattern (Martin Fowler has described patterns for the development of enterprise systems known as the "Fowler Patterns" in his book "Patterns of Enterprise Application Architecture"https://books.google.com/books/about/Patterns_of_Enterprise_Application_Archi.html)

Easy Access of User Objects from the database	Each row in the User table in database represents a User Object in the code. 	Active Record Pattern(An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data) and Adapter Pattern(The ORM between models and the database acts as an adapter in ChalkBox. The object oriented language used in models is not understood by the database. And the queries of the database is not understood by the models. The ORM acts as a middle layer to facilitate the communication.)

```
from django.http import HttpResponse
from .models import Course
from accounts.models import User

# Create your views here.
def course_home(request):
    user = User(id=request.user.id)
    queryset = user.courses.all()
    context = {
        "object_list": queryset,
        "title": "Courses",
    }
    return render(request, "course_home.html", context)
```

Command Pattern:

The command pattern is a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time.

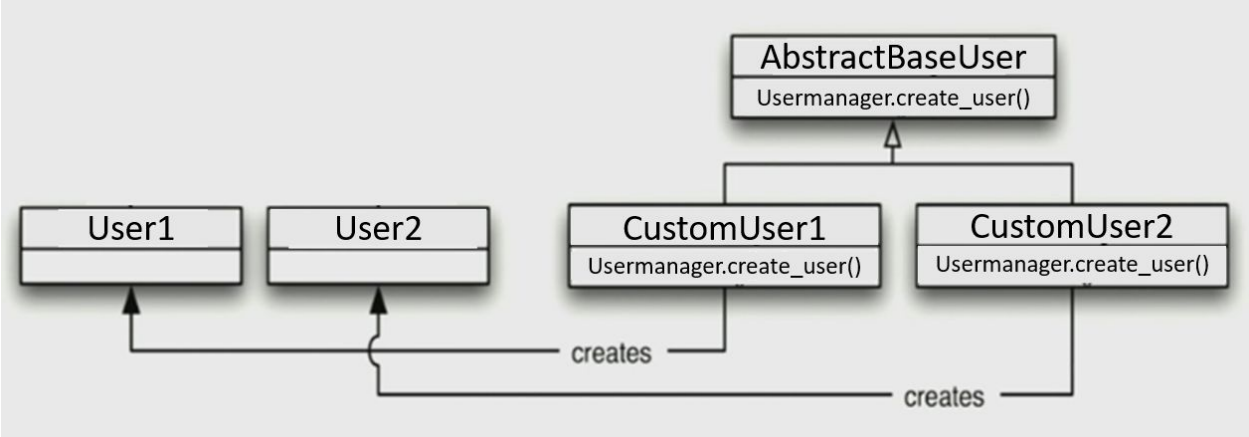
The HttpRequest object encapsulates GET or POST command. The second diagram explains how we have incorporated the command pattern in ChalkBox.

```
[docs]class HttpRequest(object):
    """A basic HTTP request."""

    # The encoding used in GET/POST dicts. None means use default setting.
    _encoding = None
    _upload_handlers = []

    def __init__(self):
        # WARNING: The `WSGIRequest` subclass doesn't call `super`.
        # Any variable assignment made here should also happen in
        # `WSGIRequest.__init__()`.

        self.GET = QueryDict(mutable=True)
        self.POST = QueryDict(mutable=True)
        self.COOKIES = {}
        self.META = {}
        self.FILES = MultiValueDict()

        self.path = ''
        self.path_info = ''
        self.method = None
        self.resolver_match = None
        self._post_parse_error = False
        self.content_type = None
        self.content_params = None

    def __repr__(self):
        if self.method is None or not self.get_full_path():
            return force_str('<%s>' % self.__class__.__name__)
        return force_str(
            '<%s: %s %r>' % (self.__class__.__name__, self.method,
force_str(self.get_full_path()))
        )
```

```python
from django.shortcuts import render
from .models import Submission

# Create your views here.
def submission_list(request, cid=None, aid=None):
    if request.method == "POST":
        instance = Submission.objects.get(id=request.POST.get('sid'))
        instance.points = request.POST.get('points')
        instance.save()

    submissions = Submission.objects.all().filter(course__id=cid, assignment__id=aid)
    context = {
        "object_list": submissions,
        "id": cid,
    }
    return render(request, "submissions.html", context)
```

Factory Pattern:

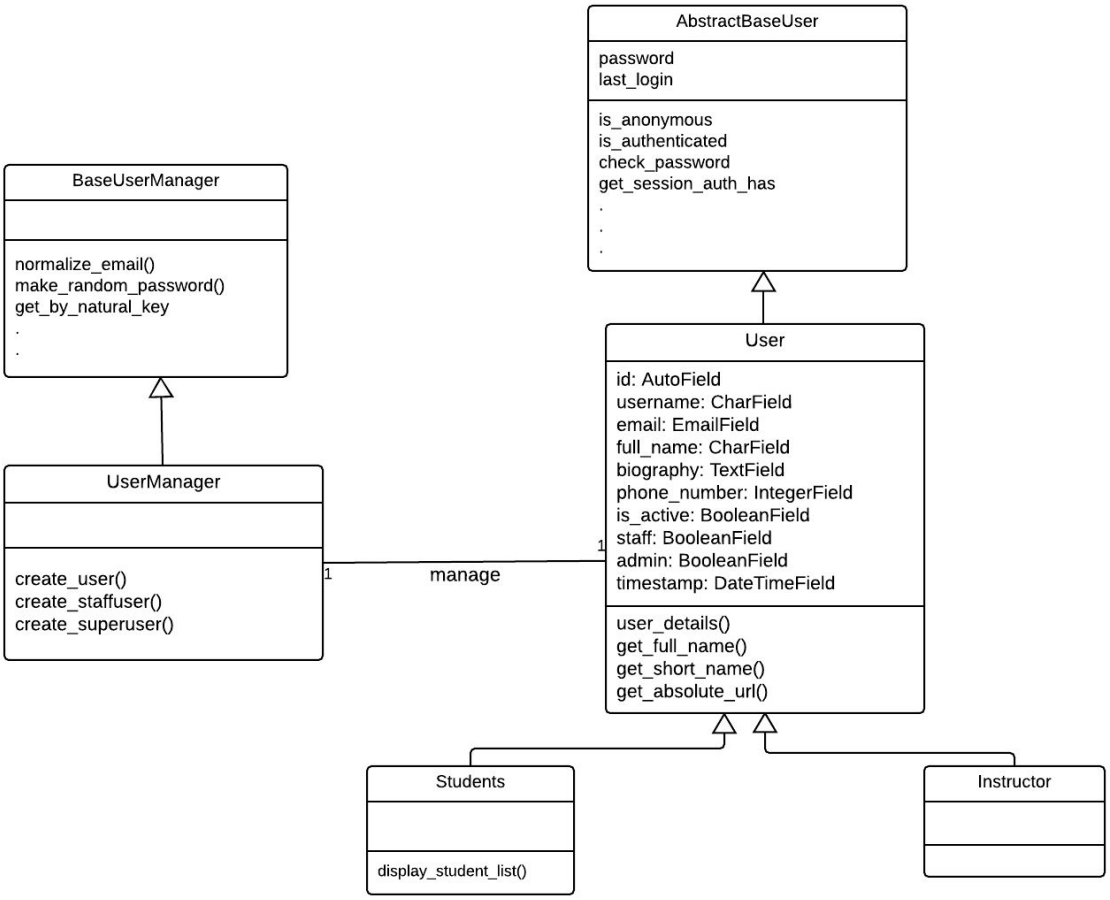
Factory Pattern is observed in the user creation for our custom user.



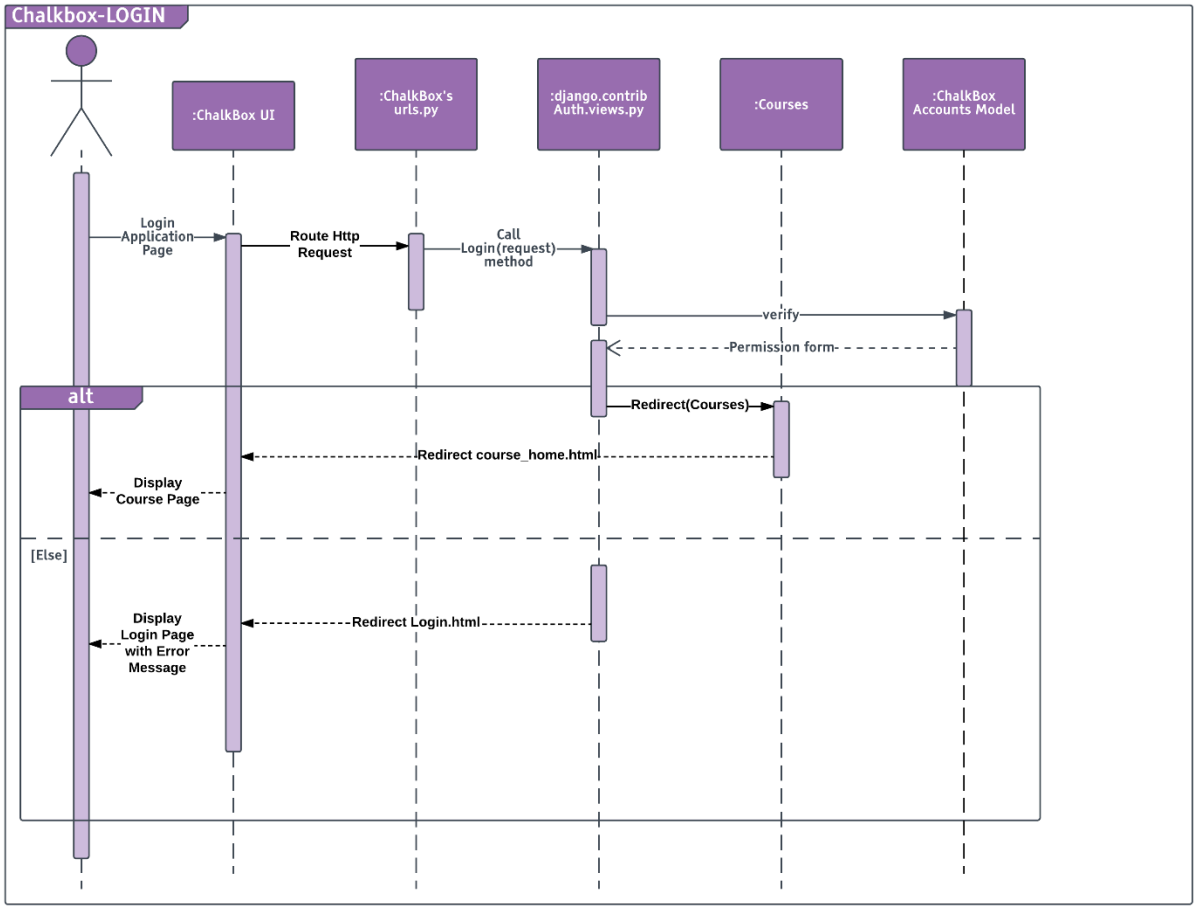
The Factory pattern allows you to call a non constructor method to create a custom user. We can have any kind of customization to user class and create this type of user object.

```python
class UserManager(BaseUserManager):
    def create_user(self, email, full_name=None, password=None, is_active=True, is_staff=False, is_
        if not password:
            raise ValueError("Users must have a password")
        if not email:
            raise ValueError("Users must have an email address")
        user_obj = self.model(
            email = self.normalize_email(email),
            full_name=full_name
        )
        user_obj.set_password(password) # change user password
        user_obj.staff = is_staff
        user_obj.admin = is_admin
        user_obj.is_active = is_active
        user_obj.save(using=self._db)
        return user_obj

    def create_staffuser(self, email, full_name=None, password=None):
        user = self.create_user(
                email,
                full_name=full_name,
                password=password,
                is_staff=True
        )
        return user

    def create_superuser(self, email, full_name=None, password=None):
        user = self.create_user(
                email,
                full_name=full_name,
                password=password,
                is_staff=True,
                is_admin=True
        )
        return user
```

10.1.2 Detailed Design Structure of Accounts Component



10.1.3 Detailed Design Behavior for Accounts component



10.1.4 Data Structures:

10.1.5 Algorithms:

Django provides a flexible password storage system and uses PBKDF2 by default.

PBKDF2 (**Password-Based Key Derivation Function 2**) is a key derivation function with a sliding computational cost, aimed to reduce the vulnerability of encrypted keys to brute force attacks.

DK = PBKDF2 (PRF, Password, Salt, c, dkLen)

Where:

- PRF is a pseudorandom function of two parameters with output length hLen (e.g. a keyed HMAC)
- Password is the master password from which a derived key is generated

- Salt is a sequence of bits, known as a cryptographic salt
- c is the number of iterations desired
- dkLen is the desired length of the derived key
- DK is the generated derived key

The password attribute of a User object is a string in this format:

```
<algorithm>$<iterations>$<salt>$<hash>
```

Those are the components used for storing a User's password, separated by the dollar-sign character and consist of: the hashing algorithm, the number of algorithm iterations (work factor), the random salt, and the resulting password hash. The algorithm is one of a number of one-way hashing or password storage algorithms Django can use; see below. Iterations describe the number of times the algorithm is run over the hash. Salt is the random seed used and the hash is the result of the one-way function.

By default, Django uses the PBKDF2 algorithm with a SHA256 hash, a password stretching mechanism recommended by NIST. This should be sufficient for most users: it's quite secure, requiring massive amounts of computing time to break.

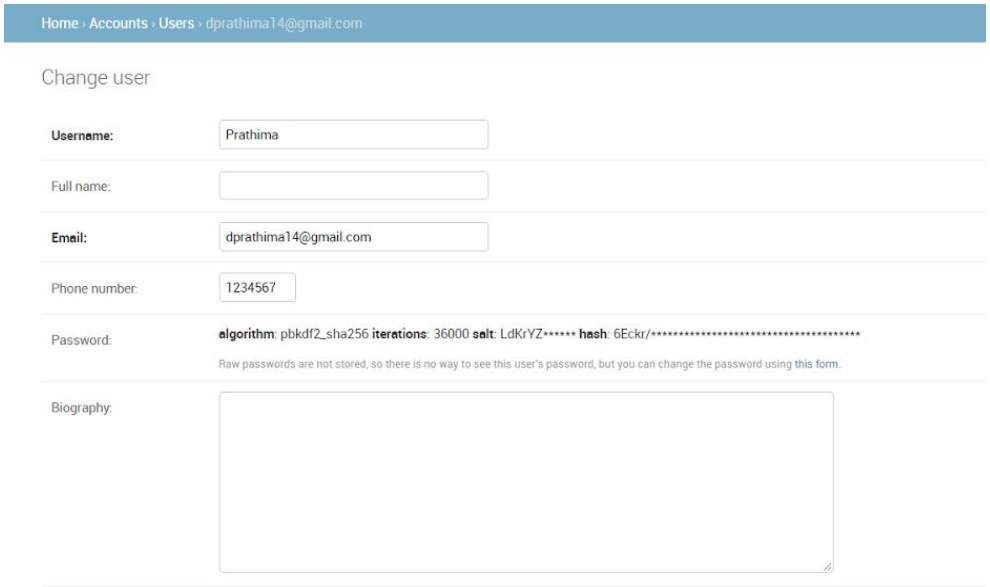
Django chooses the algorithm to use by consulting the PASSWORD_HASHERS setting. This is a list of hashing algorithm classes that this Django installation supports. The first entry in this list (that is, settings.PASSWORD_HASHERS[0]) will be used to store passwords, and all the other entries are valid hashers that can be used to check existing passwords. This means that if you want to use a different algorithm, you'll need to modify PASSWORD_HASHERS to list your preferred algorithm first in the list.

The default for PASSWORD_HASHERS is:

```
PASSWORD_HASHERS = [
  'django.contrib.auth.hashers.PBKDF2PasswordHasher',
  'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
  'django.contrib.auth.hashers.Argon2PasswordHasher',
  'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
  'django.contrib.auth.hashers.BCryptPasswordHasher',
]
```

This means that Django will use PBKDF2 to store all passwords but will support checking passwords stored with PBKDF2SHA1, argon2, and bcrypt.

Reference: ***https://en.wikipedia.org/wiki/PBKDF2***
https://docs.djangoproject.com/en/1.11/topics/auth/passwords/



10.1.6 Architecture to Detailed Design Tracing

Architecture Components/ Detailed Design components	Accounts
Chalkbox UI	x
Authentication Middleware	x
Controller	x
URL dispatcher	x
Views	x
Models	x
Chalkbox DB	x
ORM	x

10.2. Courses Component:

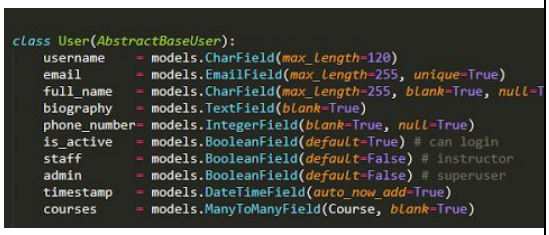
Course component is responsible for displaying all the courses as per the user enrollment. When a course is clicked it is responsible for displaying the course details such as course title, code, description.

The course component contains:

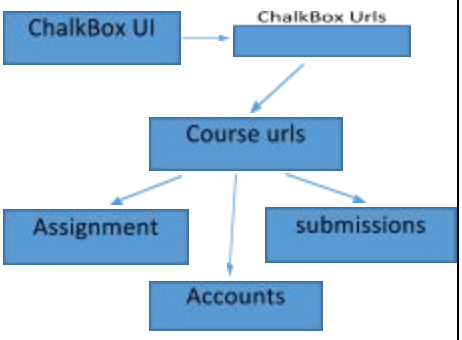
- Models.py: contains the Course class
- Views.py: contains the functions like course_list(), course_details(), view_syllabus(), uplaod_syllabus()
- Forms.py: Used to accept input from your users. Here, to add syllabus to course model.
- Urls.py: responsible for routing and calling the right view based on user request.
- Admin.py: used for displaying the course tables in the admin view
- Tests.py: used for writing test cases
- __init__.py: initialization of the app.
- App.py: app config

10.2.1 Key Decisions and Rationale:

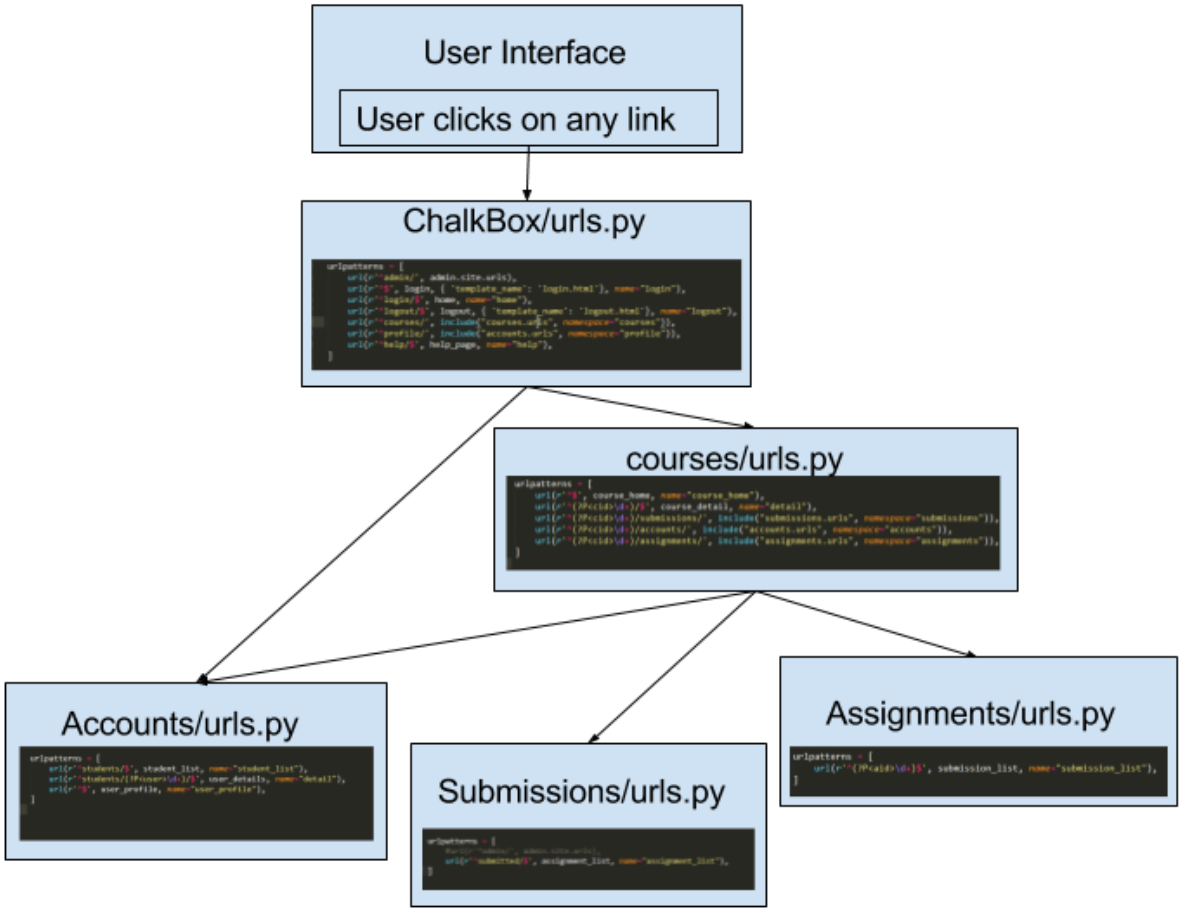
Key Decisions	Rationale	Design Pattern

Only the courses that the user is registered to is displayed	This is achieved by having a manytomany field in the user with the course class as below:
	 
	`class User(AbstractBaseUser):`
	`username = models.CharField(max_length=120)`
	`email = models.EmailField(max_length=255, unique=True)`
	`full_name = models.CharField(max_length=255, blank=True, null=T`
	`biography = models.TextField(blank=True)`
	`phone_number= models.IntegerField(blank=True, null=True)`
	`is_active = models.BooleanField(default=True) # can login`
	`staff = models.BooleanField(default=False) # instructor`
	`admin = models.BooleanField(default=False) # superuser`
	`timestamp = models.DateTimeField(auto_now_add=True)`
	`courses = models.ManyToManyField(Course, blank=True)`	

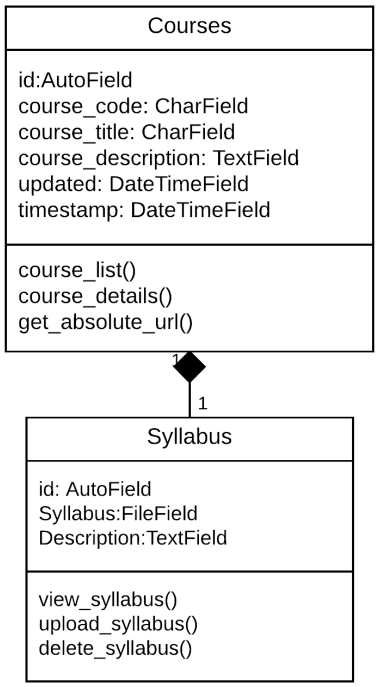
Once a course is clicked all the links inside that course view(syllabus, assignments, grades) should be with respect to the course selected.	This is achieved by using the routing functionality provided by the urls.py in the course component. For each link(syllabus/assignment/grades) selected the Component has to interact with another component. Hence these interactions need to be organized	Facade Pattern. Instead of having all the urls in the same file. The urls in the courses act as a facade for the rest of the complicated links to other apps. This is explained below:

		
Uniquely identifying Courses	Courses are uniquely identified by their course code. However for the easier access the Course class also contains an auto generated ID field which represents the database ID	Identity Field Fowler Pattern (Martin Fowler has described patterns for the development of enterprise systems known as the "Fowler Patterns" in his book "Patterns of Enterprise Application Architecture"https://books.google.com/books/about/Patterns_of_Enterprise_Application_Archi.html)
Easy Access of Course Objects from the database	Each row in the Course table in database represents a Course Object in the code.	Active Record Pattern(An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data)

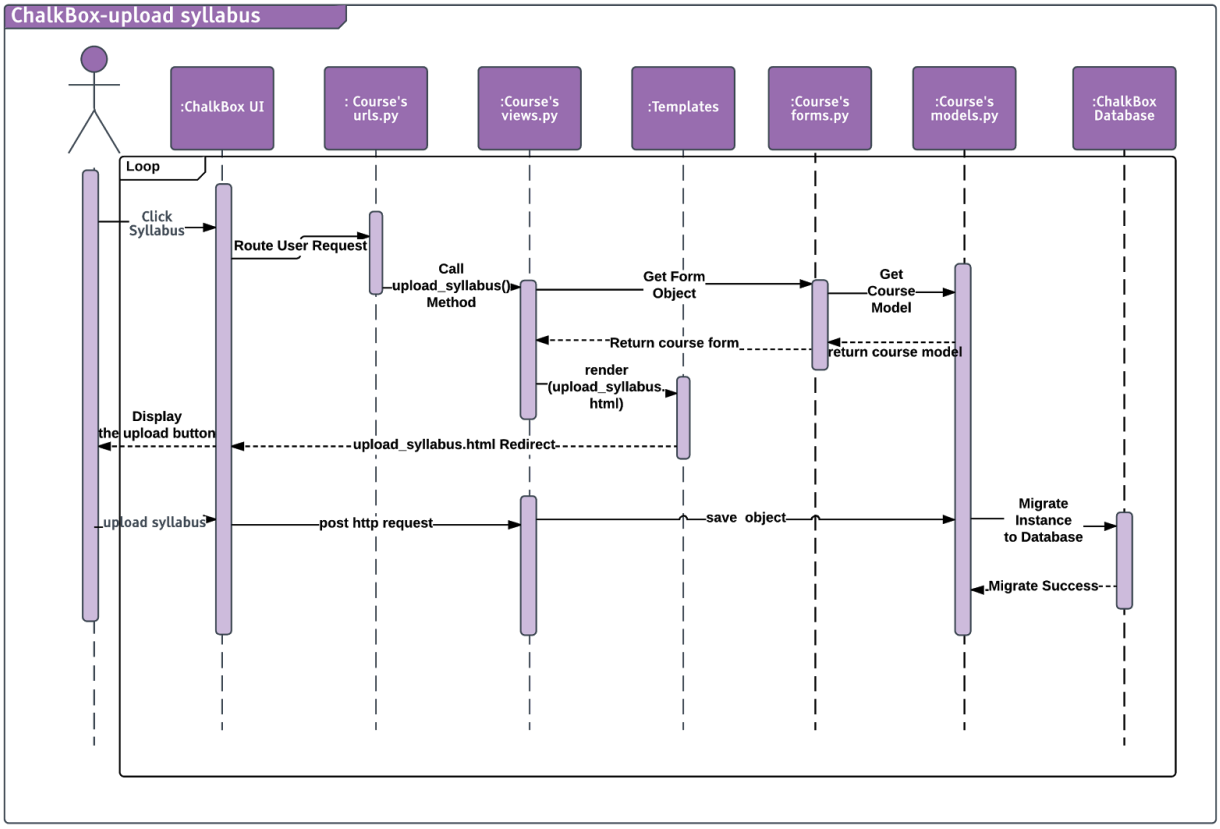
b. Facade Pattern: eases interaction between a client and a sub-system of suppliers by providing a simpler interface to the sub-system
In ChalkBox the urls.py file acts as a facade.

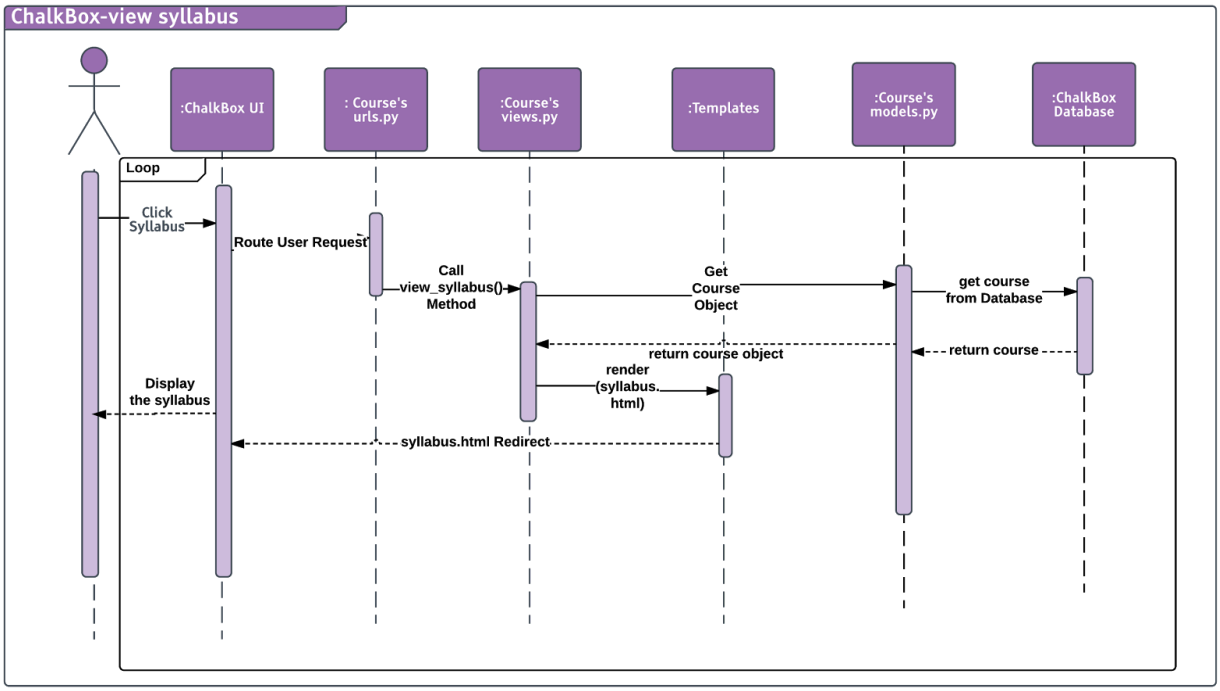


10.2.2 Detailed Design Structure of Courses Component:



10.2.3 Detailed Design Behavior for Courses component:





10.2.4 Data Structures:

A dictionary is an associative array. It contains of a {key,value} pair.

Dictionaries belong to the built-in mapping type. In Django, dictionaries are considered as unordered sets.
The dictionary elements are sent from Course “Views” to Course “Templates” in the form of a {key,value} pair. Then, these elements are used in the template.

```
# Create your views here.
def course_home(request):
    user = User(id=request.user.id)
    queryset = user.courses.all()
    context = {
        "object_list": queryset,
        "title": "Courses",
    }
    return render(request, "course_home.html", context)
```

10.2.5 Algorithms:

No algorithm used in this component.

10.2.6 Architecture to Detailed Design Tracing

Architecture Components/ Detailed Design components	Assignments	Course
Chalkbox UI	x	x
Authentication Middleware		
Controller	x	x
URL dispatcher	x	x
Views	x	x
Models	x	x
Chalkbox DB	x	x
ORM	x	x

10.3 Assignment Management Component

The Assignment component contains:

- Models.py: contains the Assignment class
- Views.py: contains the functions like assignment_home() and assignment_detail()
- Urls.py: responsible for routing and calling the right view based on user request.
- Admin.py: used for displaying the user tables in the admin view
- Tests.py: used for writing test cases
- __init__.py: initialization of the app.
- App.py: app config

10.3.1 Key Decision and Rationale

The assignment management is used to manage all the assignments corresponding to courses. There are several key decisions here and rationales, as well as the design patterns:

Key decisions	Rationale	Use of design patterns
View all assignments for the certain course	When a student clicks the assignment button in the course	**Template Pattern:**

	page, he/she can view all assignments list for that course	Templates in django as the name suggests follows the Template pattern. A base template is created with the base html. In our case the base.html has the “navbar” which is repeated in all the pages throughout the site. Hence all other pages can extend this page without Rewriting the code in every html page. Detailed explanation can be found below
Create, delete and edit assignment details and requirements.	Instructors have the authority to create a new assignment and delete or edit existing assignments.	**Factory Pattern:** The form.modelforms in django can be used in our custom form class such as AssignmentForm. It can be created by passing the model object of each type and the required attribute for each form.

Template Pattern:

Templates in django as the name suggests follows the Template pattern. A base template is created with the base html. In our case the base.html has the “navbar” which is repeated in all the pages throughout the site. Hence all other pages can extend this page without Rewriting the code in every html page. The example below explains this pattern.

base.html:

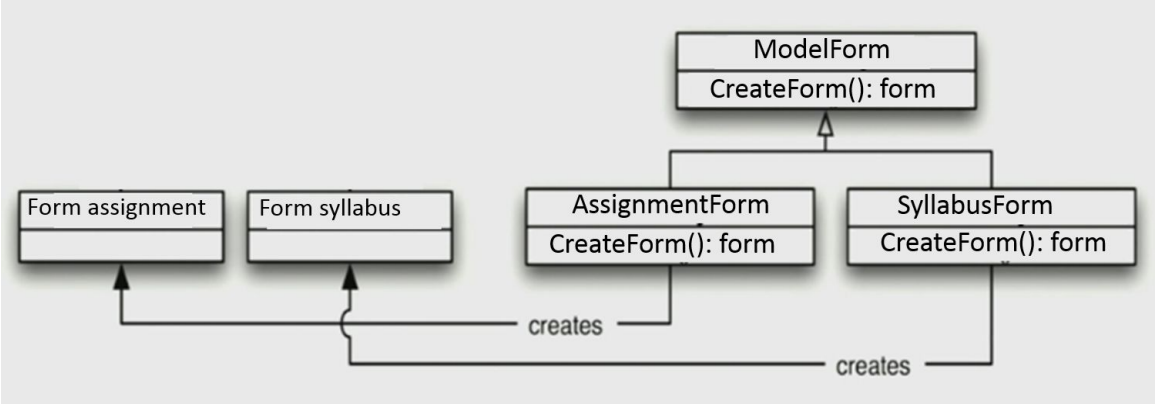
```
{% load staticfiles %}

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="icon" href="../../favicon.ico">

    <title>{% block head_title %}ChalkBox{% endblock %}</title>

    {% include 'main_css.html' %}
    <style>
      {% block style %}{% endblock %}
    </style>
    </head>
    <body>
      {% include 'navbar.html' %}
      <div class="container">

      {% block content %}

      {% endblock %}

      </div> <!-- /container -->

      {% include "main_js.html" %}
    </body>
</html>
```

assignment.html:

```
{% extends "base.html" %}

{% block title %}{{ assignment.title }}{% endblock %}

{% block content %}
    <h1>Assignments</h1>
        <table class="table">
        <thead>
            <tr>
              <th scope="col">Assignment title</th>
              <th scope="col">Due date</th>
              <th scope="col">total points</th>
            </tr>
        </thead>
        <tbody>
            {% for assignment in assignment_list %}
                <tr>
                  <td><a href="{% url 'courses:submissions:submission_list' cid=id aid=obj.id %}">{{ obj.title }}</a></td>
                  <td>{{ obj.due_date }}</td>
                  <td>{{ obj.total_points }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

Factory Pattern:

Factory Pattern is observed in the way we have implemented forms.



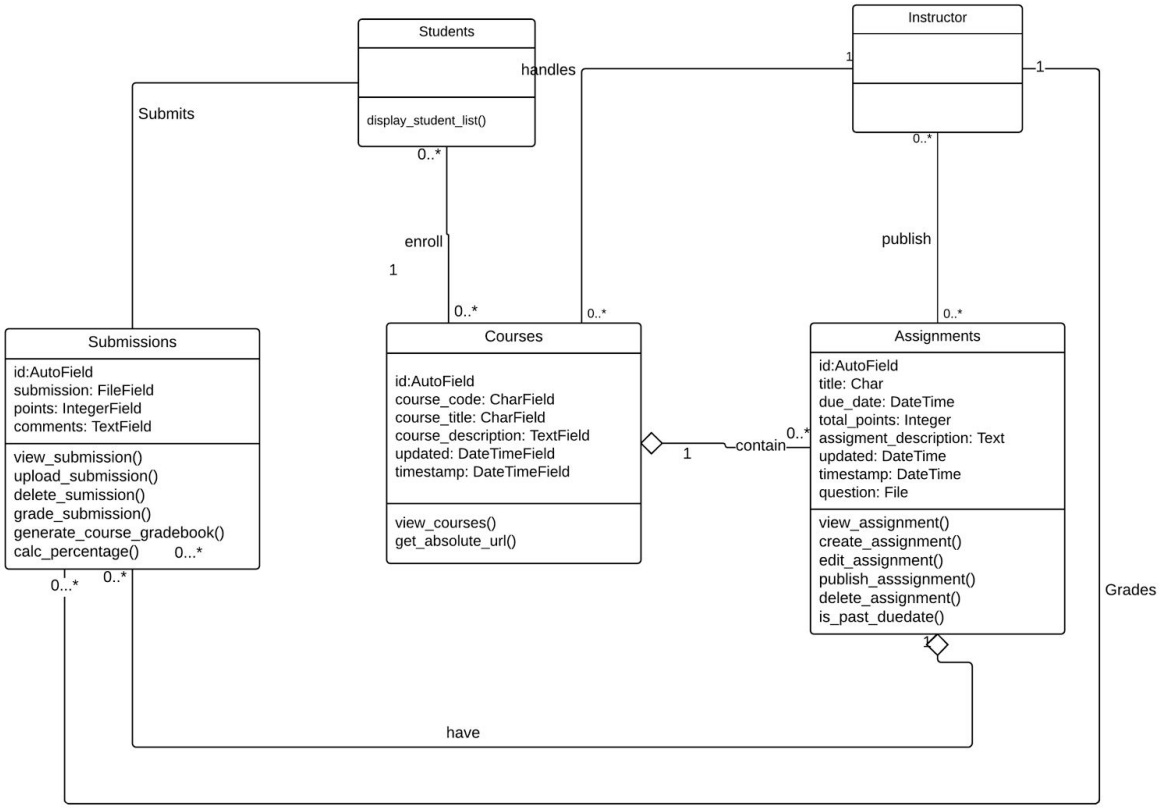
The form.modelforms in django can be used in our custom form class such as AssignmentForm or SyllabusForm and different types of forms can be created by passing the model object of each type and the required attribute for each form.

For example let us consider forms.py for Assignment component:

```
from django import forms
from .models import Assignment

class AssignmentForm(forms.ModelForm):
    class Meta:
        model = Assignment
        fields = [
            'title',
            'due_date',
            ‘total_points',
            'assignment_description',
            'question',
        ]
```

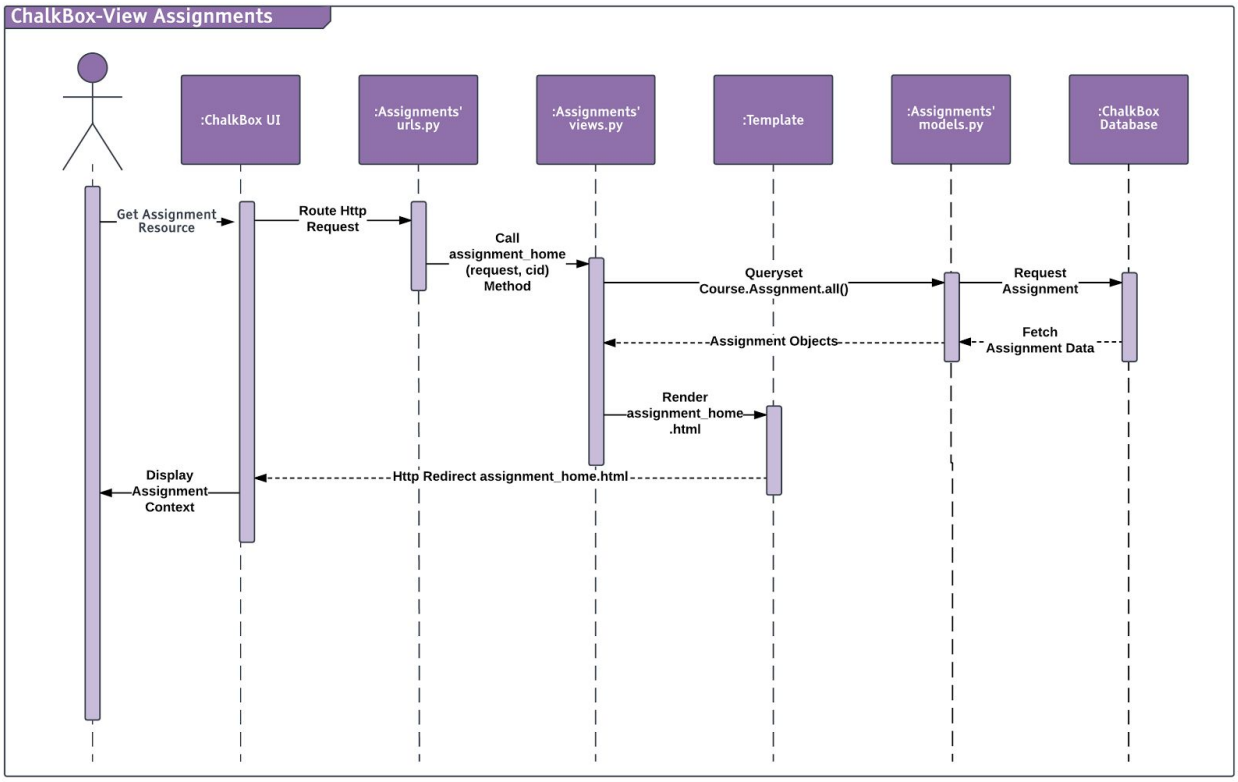
This would create a form object which can be used in our html file to take the user input.

10.3.2 Detailed Design Structure

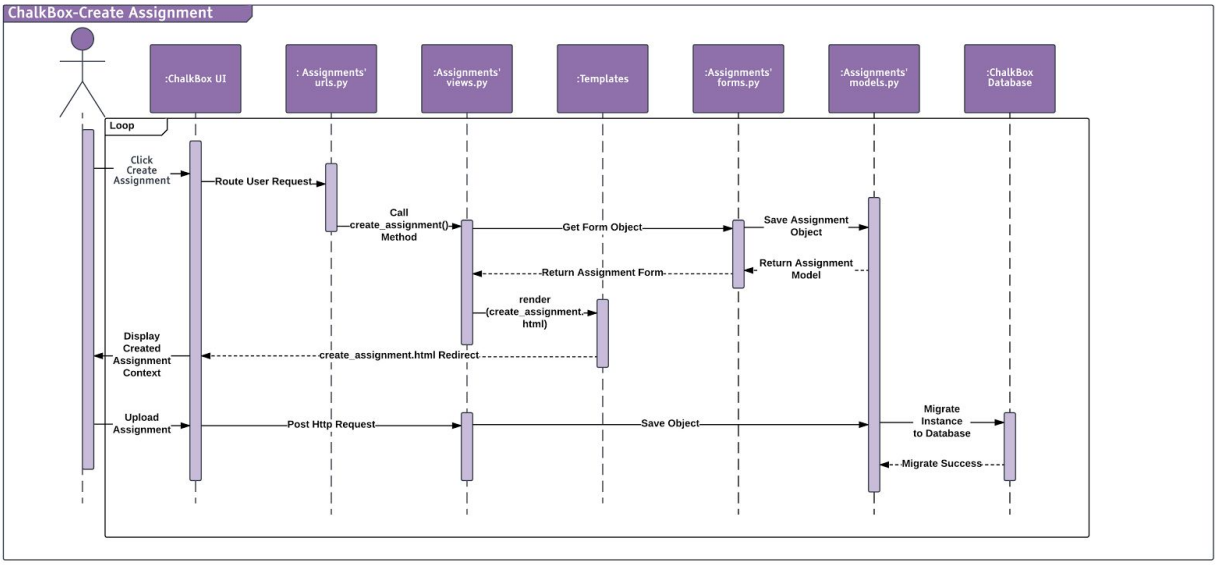


10.3.3 Detailed Design Behavior

View assignment list



Create assignment



10.3.4 Data Structures

Set is used to store assignments in the assignment management component.

```
def assignment_home(request, cid=None):
    course = Course(id=cid)
    all_assignment = course.assignments.all().order_by('due_date')
    upcoming_assignment = set()
    overdue_assignment = set()
    for c in all_assignment:
        if c.is_past_due() is False:
            upcoming_assignment.add(c)
        else:
            overdue_assignment.add(c)
```

10.3.5 Algorithms

No algorithm is used in this part.

10.3.6 Architecture to Detailed Design Tracing

Architecture Components/ Detailed Design components	Assignments	Submission
Chalkbox UI	x	x
Authentication Middleware		
Controller	x	x
URL dispatcher	x	x
Views	x	x
Models	x	x
Chalkbox DB	x	x
ORM	x	x

10.4. Submission Component

The Submission component is responsible for displaying all submissions, and allowing instructor grades the submissions. The gradebook is also maintained.

The Submission component contains:

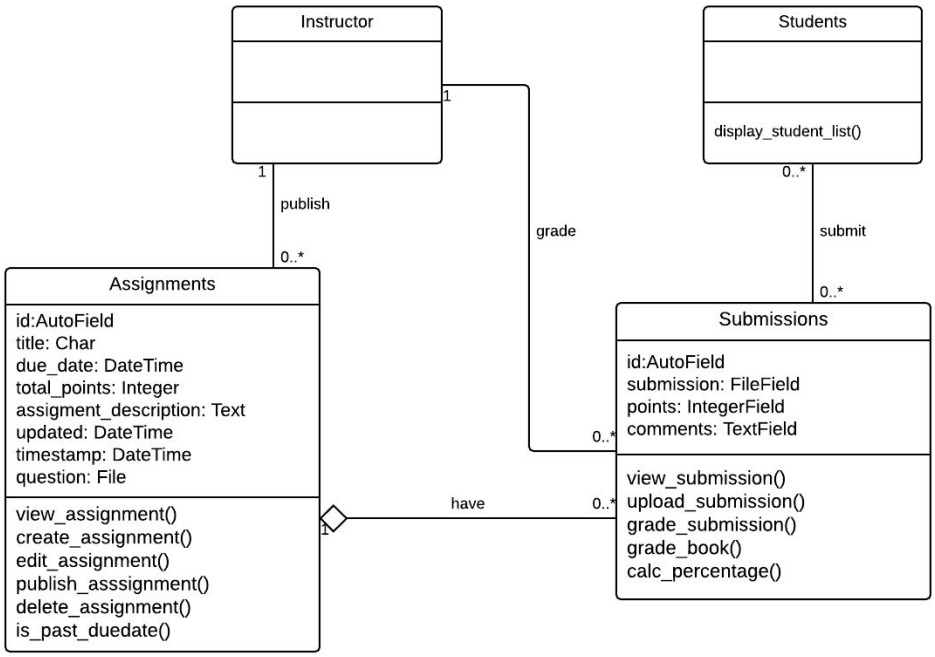
- Models.py: contains the Submission class
- Views.py: contains the functions like submission_list() and grade_book()
- Urls.py: responsible for routing and calling the right view based on user request.
- Admin.py: used for displaying the user tables in the admin view
- Tests.py: used for writing test cases
- __init__.py: initialization of the app
- App.py: app config

10.4.1 Key Decisions and Rationale

Key Decisions	Rationale	Design Pattern
Display all student's submissions for the course	Once instructor clicks 'submissions', she should be able to see all submissions from the whole class.	Template Pattern. Similar to Assignment component
Each submission is should refer an Assignment, User and Course	Django foreign key field is used to link to User, Course and Assignment Object 	
Uniquely identifying Submissions	Submissions are uniquely identified by their user id, assignment id and course id.. However for the easier access the submission class also contains an auto generated ID field which represents the database ID	Identity Field Fowler Pattern (Martin Fowler has described patterns for the development of enterprise systems known as the "Fowler Patterns" in his book "Patterns of Enterprise Application Architecture"https://books.google.com/books/about/Patterns_of_Enterprise_Application_Archi.html)

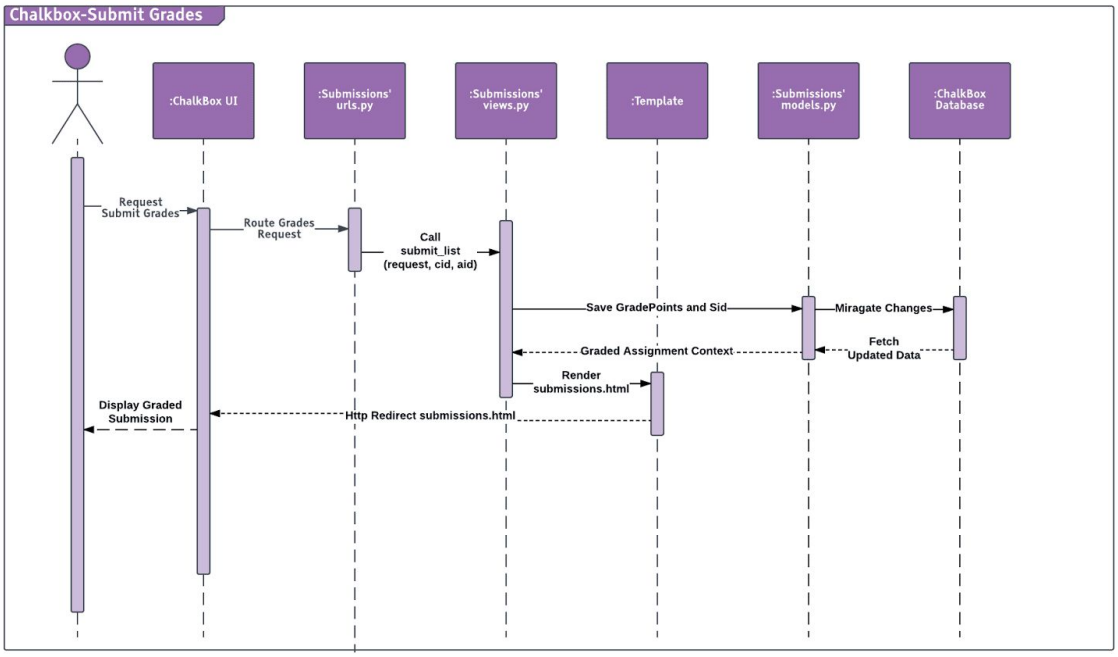
```
class Submission(models.Model):
    course = models.ForeignKey(Course)
    user = models.ForeignKey(User)
    assignment = models.ForeignKey(Assignment)
    submission  = models.FileField(null=True, blank=True)
    points  = models.IntegerField(null=True, blank=True)
    total_points    = models.IntegerField(null=True, blank=True)
    comments = models.TextField(blank=True)

    def __str__(self):
        return str(self.id)
```

10.4.2 Detailed Design Structure

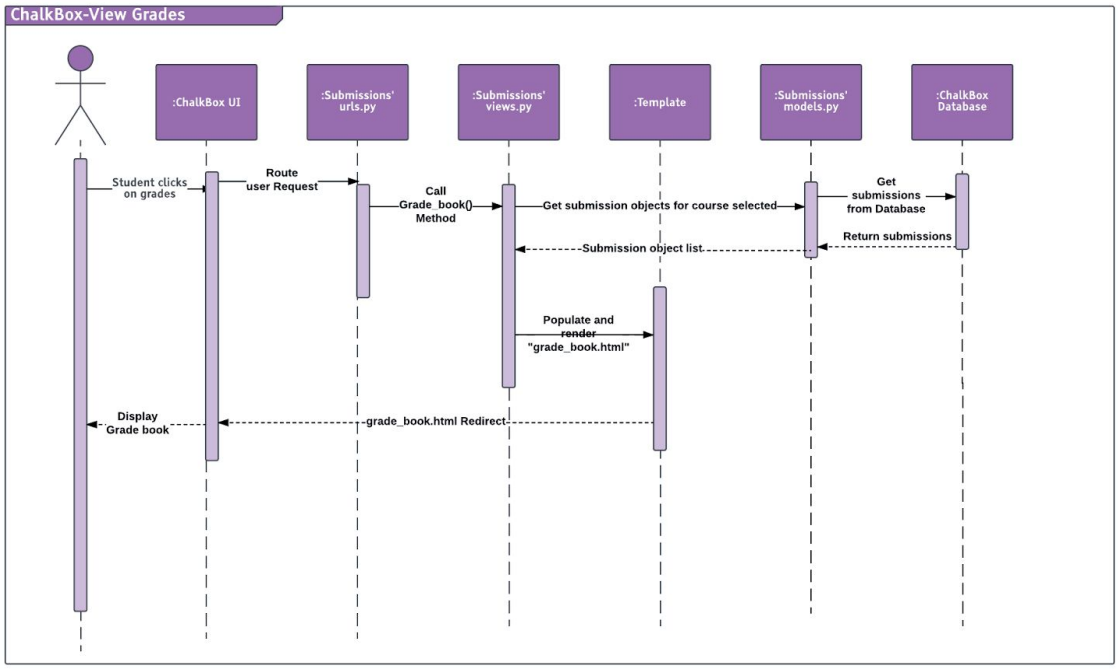


10.4.3 Detailed Design Behavior

Submit grades


Chalkbox-Submit Grades
:ChalkBox UI
:Submissions' urls.py
:Submissions' views.py
:Template
:Submissions' models.py
:ChalkBox Database
Request Submit Grades
Route Grades Request
Call submit_list (request, cid, aid)
Save GradePoints and Sid
Miragate Changes
Graded Assignment Context
Fetch Updated Data
Render submissions.html
Display Graded Submission
Http Redirect submissions.html

View grades


ChalkBox-View Grades
:ChalkBox UI
:Submissions' urls.py
:Submissions' views.py
:Template
:Submissions' models.py
:ChalkBox Database
Student clicks on grades
Route user Request
Call Grade_book() Method
Get submission objects for course selected
Get submissions from Database
Return submissions
Submission object list
Populate and render "grade_book.html"
Display Grade book
grade_book.html Redirect

10.4.4 Data Structures

The dictionary elements are sent from “Views” to “Templates” in the form of a {key,value} pair. Then, these elements are used in the template.

```
submissions = Submission.objects.all().filter(course__id=cid, assignment__id=aid)
context = {
    "object_list": submissions,
    "id": cid,
}
return render(request, "submissions.html", context)
```

In the above example, the values of “object_list” and “id” are sent from the view Submissions.py to the submission.html.

10.4.5 Algorithms

No algorithm is used in this part.

10.4.6 Architecture to Detailed Design Tracing

Architecture Components/ Detailed Design components	Submission
Chalkbox UI	X
Authentication Middleware	
Controller	X
URL dispatcher	X
Views	X
Models	X
Chalkbox DB	X
ORM	X