<ENPM613>

<*GELMS*>  
Software Architecture and Detailed Design Document (SADD)

Team members:

Daniel Sokol

Manu Venugopal

Vaibhav Sharda

Sagar Anvekar

Nishanth Kanala

# Contents

[1. Contents 2](#_Toc531083685)

[2. Product Overview 4](#_Toc531083686)

[3. Architecture Drivers 5](#_Toc531083687)

[Quality Attribute Drivers: 5](#_Toc531083688)

[Software Architecture Drivers: 5](#_Toc531083689)

[Utility Tree: 6](#_Toc531083690)

[4. Architecture Key Decisions and Rationale 7](#_Toc531083691)

[Key Architecture Diagram: 7](#_Toc531083692)

[Architecture Rationale: 8](#_Toc531083693)

[Technology Trade Off Analysis: 8](#_Toc531083694)

[Trade Off Decisions: 8](#_Toc531083695)

[5. Architecture Logical View 9](#_Toc531083696)

[Package Diagram: 9](#_Toc531083697)

[Package Diagram Description: 10](#_Toc531083698)

[Component Diagram: 11](#_Toc531083699)

[Component Diagram Description: 11](#_Toc531083700)

[6. Architecture Behavior View 19](#_Toc531083701)

[Admin Adds New Course: 19](#_Toc531083702)

[Teacher Turns on Reader Mode: 20](#_Toc531083703)

[Teacher Adds Announcement: 21](#_Toc531083704)

[Student Logs In: 22](#_Toc531083705)

[Student Views Tools: 23](#_Toc531083706)

[7. Architecture Deployment View 24](#_Toc531083707)

[Deployment Diagram: 24](#_Toc531083708)

[8. Information View 25](#_Toc531083709)

[Entity Relationship Diagram: 25](#_Toc531083710)

[Entity Relationship Diagram Description**:** 25](#_Toc531083711)

[9. Requirements Allocation 26](#_Toc531083712)

[10. Architecture Work Allocation 27](#_Toc531083713)

[11. Detailed Design Key Decisions and Rationale 28](#_Toc531083714)

[12. Detailed Design Structure 29](#_Toc531083715)

[View Detail Structure Diagram: 29](#_Toc531083716)

[View Detail Structure Diagram Description: 29](#_Toc531083717)

[Controller Detail Structure Diagram: 30](#_Toc531083718)

[Accounts App Diagram: 30](#_Toc531083719)

[Accounts App Description: 30](#_Toc531083720)

[Dashboard App Diagram: 31](#_Toc531083721)

[Dashboard App Description: 31](#_Toc531083722)

[Course App Diagram: 32](#_Toc531083723)

[Course App Description: 32](#_Toc531083724)

[Settings App Diagram: 33](#_Toc531083725)

[Settings App Description: 33](#_Toc531083726)

[Announcement App Diagram: 34](#_Toc531083727)

[Announcements App Description: 34](#_Toc531083728)

[Syllabus App Diagram: 36](#_Toc531083729)

[Syllabus App Description: 36](#_Toc531083730)

[People App Diagram: 37](#_Toc531083731)

[Tools App Diagram: 38](#_Toc531083732)

[Tools App Description: 38](#_Toc531083733)

[13. Detailed Design Behavior 39](#_Toc531083734)

[Admin Adds Teacher to Course: 39](#_Toc531083735)

[Grader Views People: 40](#_Toc531083736)

[Student Views Announcement: 41](#_Toc531083737)

[Teacher Adds Syllabus: 42](#_Toc531083738)

[Teacher Locks Tools: 43](#_Toc531083739)

[14. Physical Data Model 44](#_Toc531083740)

[15. Algorithms 46](#_Toc531083741)

[16. Architecture to Detailed Design Tracing 47](#_Toc531083742)

# Product Overview

GELMS (Generic Electronic Learning Management System) is an online learning management system that is designed to be both streamlined to the specific user and adaptive to the different educational domains. In addition to the required features, GELMS will include a “tools” feature that will provide users with access to tools that are relevant to their course (i.e. a calculator for an engineering student, an instrument tuner for a music student). Different courses will also display aesthetics unique to their educational domain. Next, GELMS will include a “reader mode” option for users with an impaired sight disability. Finally, GELMS will also provide multi-factor authentication as an option for users with elevated security concerns. The different GELMS features are highlighted in the list below and are divided up by the different users for clarity:

# Architecture Drivers

## Quality Attribute Drivers:

Below is the list of the quality attribute drivers that influenced our software architecture, detailed design, and, ultimately, the final product.

Admin features:

* Admins can add and remove courses
* Admins can add and remove students to a specific course
* Admins can assign a teacher and/or grader to a specific course

Teacher features:

* Teachers can post, edit, and remove announcements to specific course
* Teachers can post, edit, and remove a syllabus to a specific course
* Teachers can view a list of all people (students, teachers, graders) in a specific course
* Teachers can lock and unlock tools for a specific course
* Teachers can turn reader mode on and off
* Teachers can turn multi-factor authentication on and off

Student features:

* Students can view announcements for a specific course
* Students can view a syllabus for a specific course
* Students can view tools for a specific course
* Students can turn reader mode on and off
* Students can turn multi-factor authentication on and off

## Software Architecture Drivers:

We considered several software best practices throughout our architecture, detailed design, and implementation process.

Modularity: We sought to separate the many different functionalities of our product into distinct modules. For example, the software for the “announcements” responsibilities and the software for the “syllabus” responsibilities each occupy their own module in our source code. This way, writing and editing either module does not affect the other. The modular structure of our code is highlighted in the various sections below.

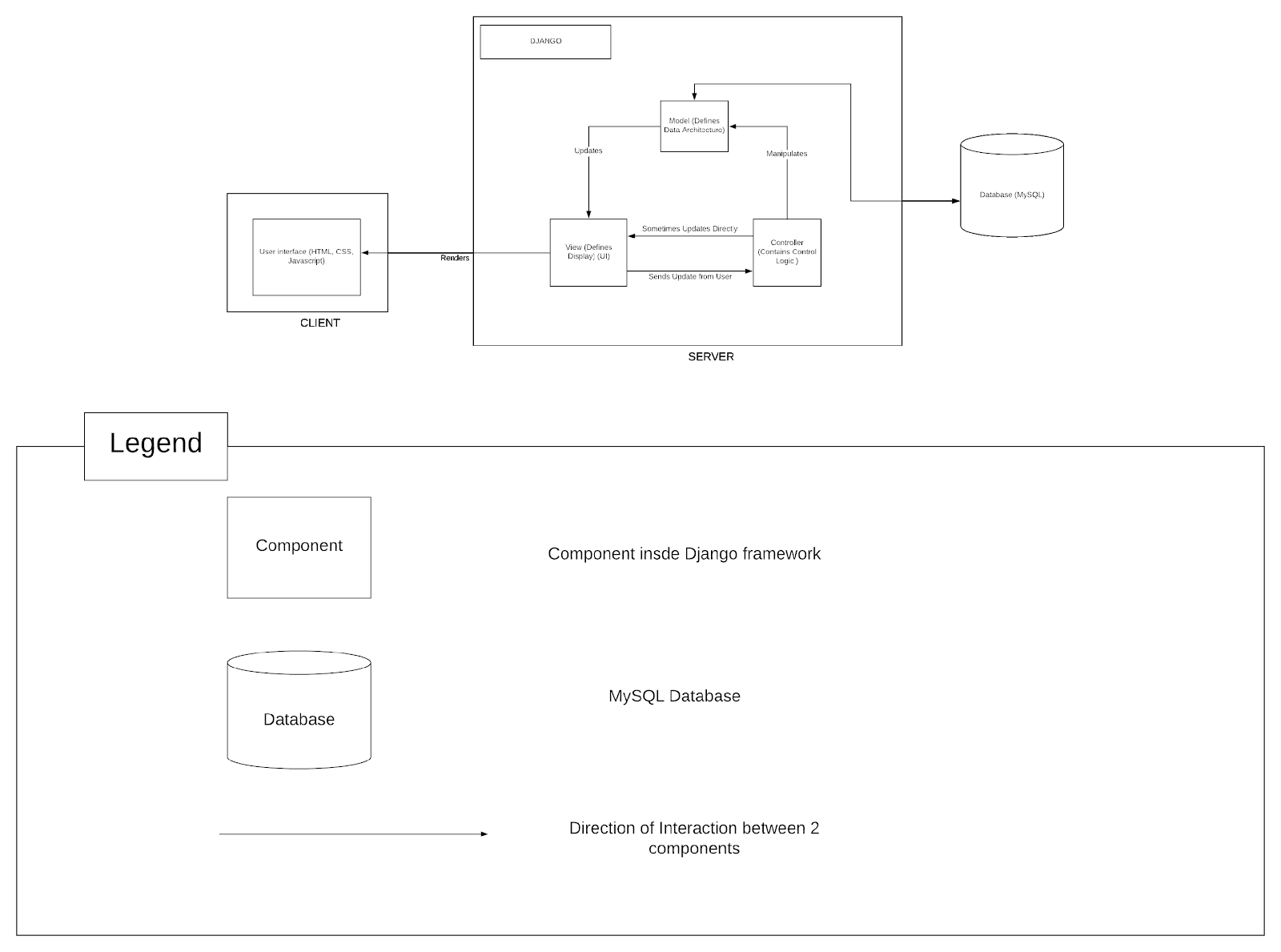
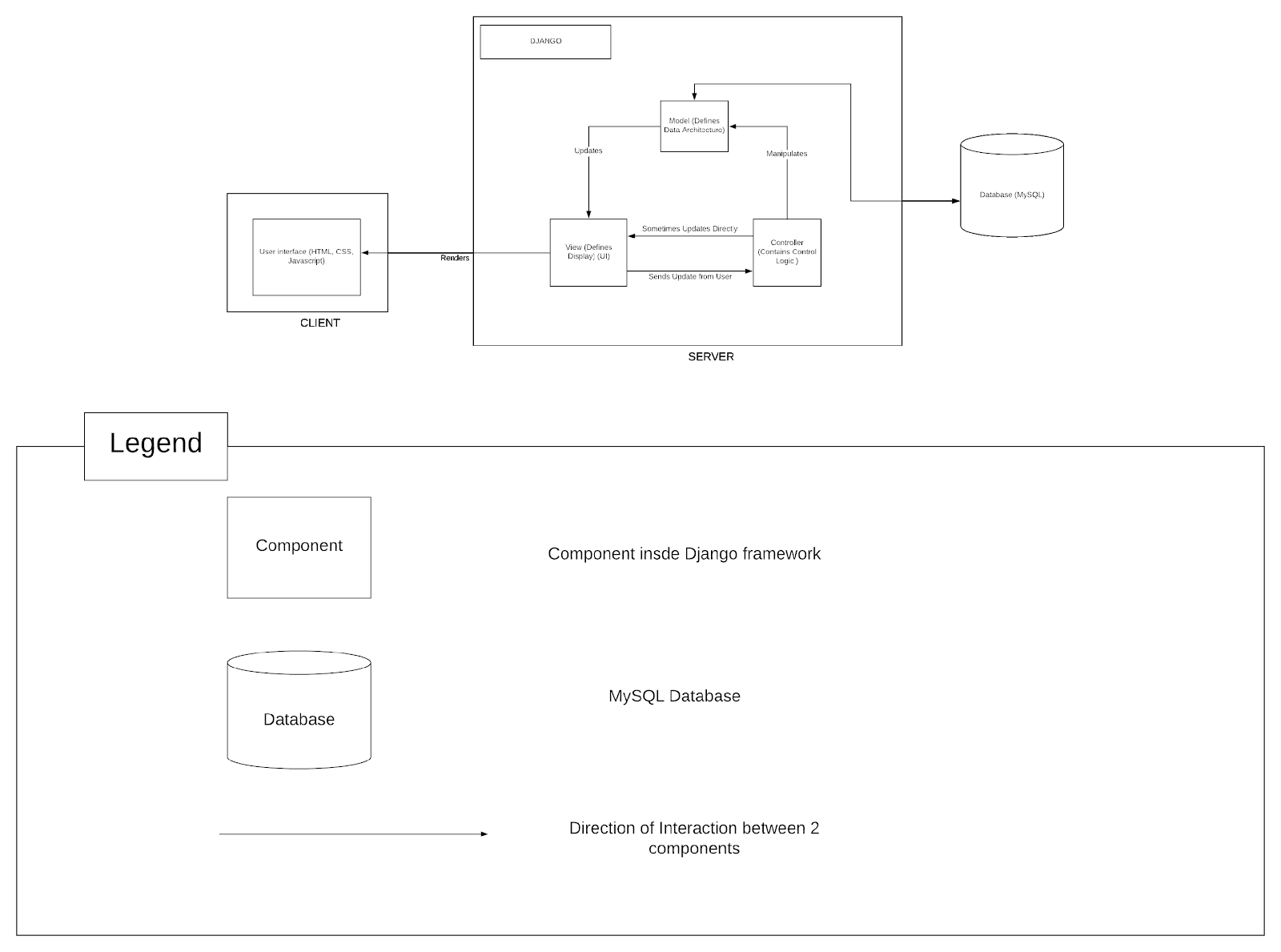
Reuse: Reuse was one of our biggest drivers. Reuse influenced us to use a technology that already included many of the features we needed for our product including user authentication, client server exchanges, and a modifiable MVC framework.

Coupling and Cohesion: We managed to implement loosely coupled code by considering modularity in our design. As mentioned before, different functionalities can be managed independently of one another without affecting each other. This was a major advantage as multiple team members could work on different portions of the project simultaneously. Furthermore, in some cases we fortified our specific framework’s best practices in favor of code that was more cohesive. Here we rearranged the file structure of our project so that code with similar functions are in close proximity to one another. This made it easier to manage the different portions of our project as locating a particular functionality in the code was intuitive and straightforward.

## https://lh6.googleusercontent.com/FvFazbSJRyoBkI4mnzMpIdoMjkGULWi2zYpA47yTyWmT9vVdGoO___Lxzf8jO0P8Gqpnd2Jmzla-RBh58pcgi3yuzXTrNd0WMwWe5F1JRQUKTxeO926kUi6tVg_cNdxPfFfuUqamUtility Tree:

# Architecture Key Decisions and Rationale

## Key Architecture Diagram:



SQLite3

User Interface (HTML, CSS)

SQLite3

## Architecture Rationale:

We are using a mix of Architecture styles, for making the entire application we are using the MVC architecture but after we complete the application we are going to upload all the files and related to application on a Production server and everyone that are accessing this website becomes our client like students, teachers etc. they all want to access the learning management system to fulfill their needs and requirements. So, this becomes a Client – Server model.

## Technology Trade Off Analysis:

## Trade Off Decisions:

User Side: User of this system can enter the URL of the system in the system in the browser and through which a HTTP get request will be generated and forwarded to our URL and front-end of the website would be rendered.

Frontend: In the front user or user side, we choose technologies like HTML, CSS, JavaScript and Bootstrap to design our front end of the website. We also included Bootstrap, an extension of CSS, in our application.

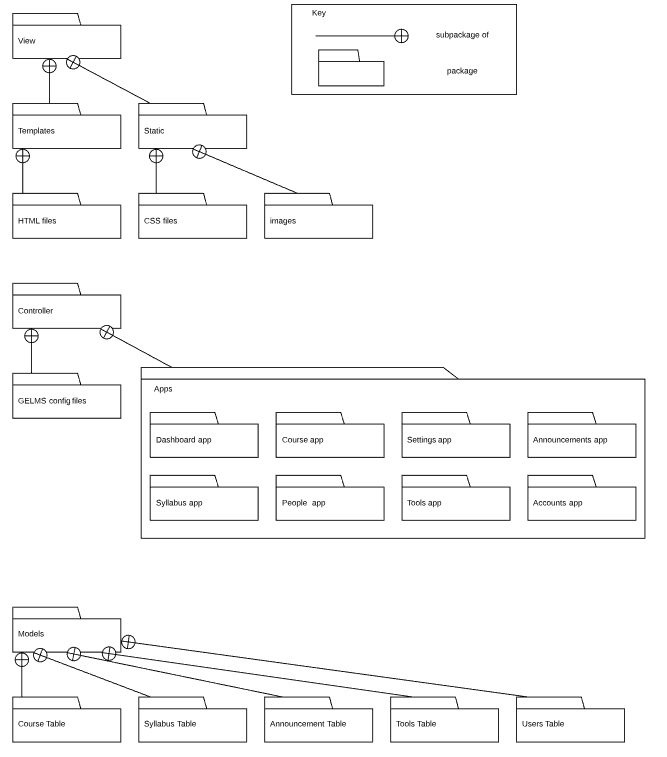
Backend: In the backend of our website we are using Python as the main language which has various inbuilt frameworks such as Django, Flask, web2py etc. We decided to use Django over other frameworks as it included various inbuilt functions that we could directly inherit. Also, with its free and open source, which makes the web development process very easy and the developer can fully focus on the designing process and boost performance.

Webserver: We have decided to use PythonAnywhere which includes all the packages for webhosting and client server interactions. PythonAnywhere is easy to use and provides a lot of services including the management of webserver, automatic load balancer, among other features.

Database: We have decided to use the SQLite3 database because it is inbuilt in Django and serves all our needs effectively. Furthermore, Django provides an inbuilt API for interactions between our python code and the SQLite3 database.

# Architecture Logical View

## Package Diagram:

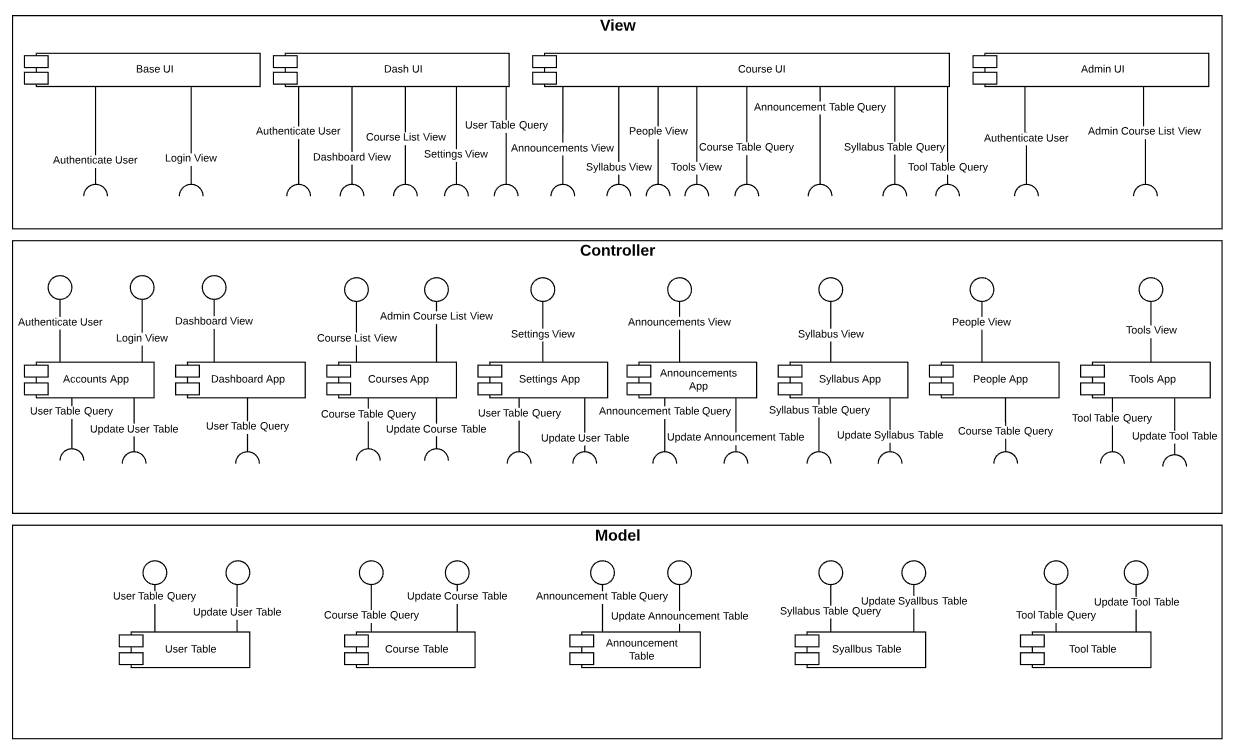


## Package Diagram Description:

The GELMS system will consist of three high-level packages based on the Model View Control (MVC) architecture and its constituent subpackages.

1. View Package
   1. The View package is responsible for the rendered pages that the user sees in the front. It contains the relevant subpackages that contains all the files necessary for the front-end. These subpackages are as follows:
      1. Templates: This contains the subpackage HTML Files that is a collection of all the HTML files that work to render the webpages.
      2. Static: This contains the subpackages CSS files which contains the CSS style file that provides the styling for the web pages and Images which contains all the static image resources used throughout the webpages.
2. Controller Package
   1. The Controller package is responsible for the programmatic logic behind the functionality of the GELMS software website. It contains the subpackages that together realize one of the two important parts of the GELMS back-end. The subpackages are as follows:
      1. GELMS config files: This subpackage contains the configuration files that Django provides.
      2. Apps: This subpackage contains the major logic behind the functionality of GELMS realized as subpackages which controls the web applications of GELMS:
         1. Dashboard App
         2. Course App
         3. Settings App
         4. Announcements App
         5. Syllabus App
         6. People App
         7. Tools App
         8. Accounts App
3. Models Package
   1. The Models package is the second import part of the GELMS back-end which contains all the relevant data involved in the GELMS software. These data as realized as database tables contained within the following subpackages:
      1. Course Table
      2. Syllabus Table
      3. Announcement Table
      4. Tools Table
      5. Users Table

## Component Diagram:



## Component Diagram Description:

Base UI

* Description: Covers the basic views for the entire website, as well as handles some authentication functionality
* Provided interfaces: None
* Required interfaces: Authenticate user, Login view

Dash UI

* Description: Covers the next layer of views for the dashboard level webpages, queries the database for user details
* Provided interfaces: None
* Required interfaces: Authenticate User, Dashboard View, Course List View, Settings View, User Table Query

Course UI

* Description: Covers the lowest level views for the course level webpages, queries the database for course details, provides forms for teachers to add/edit/remove course syllabus and announcements
* Provided interfaces: None
* Required interfaces: Announcements View, Syllabus View, People View, Tools View, Course Table Query, Announcement Table Query, Syllabus Table Query, Tool Table Query

Admin UI

* Description: Covers the views for the admin webpages, provides forms for the admin to add/edit/remove users and courses
* Provided interfaces: None
* Required interfaces: Authenticate User, Admin Course List View

Accounts App

* Description: Describes the user model, provides authentication functionality to the views, provides view logic for the Login page, queries the database for user data, captures changes to the user model via input from the Admin UI
* Provided interfaces: Authenticate User, Login View
* Required interfaces: User Table Query, Update User Table

Dashboard App

* Description: Provides view logic for the dashboard page, queries the database for user details
* Provided interfaces: Dashboard view
* Required interfaces: User Table Query

Courses App

* Description: Provides view logic for the course list page, provides view logic for the admin course list page, queries the database for course details, captures changes to the course model via input from the Admin UI
* Provided interfaces: Course List View, Admin Course List View
* Required interfaces: Course Table Query, Update Course Table

Settings App

* Description: Provides view logic for the settings page, queries the database for user details, captures changes to the user model via input from the Dash UI
* Provided interfaces: Settings View
* Required interfaces: User Table Query, Update User Table

Announcements App

* Description: Provides view logic for the announcements page, queries the database for course and announcement details, captures updates to the announcement model via input from the Course UI
* Provided interfaces: Announcements View
* Required interfaces: Query Announcements Table, Update Announcements Table

Syllabus App

* Description: Provides view logic for the syllabus page, queries the database for course and syllabus details, captures changes to the syllabus model via input from the Course UI
* Provided interfaces: Syllabus View
* Required interfaces: Syllabus Table Query, Update Syllabus Table

People App

* Description: Provides view logic for the people page, queries the database for course details
* Provided interfaces: People View
* Required interfaces: Course Table Query

Tools App

* Description: Provides view logic for the tools page, queries the database for course and tools details, captures changes to the tools model via input from the Course UI
* Provided interfaces: Tools View
* Required interfaces: Tool Table Query, Update Tool Table

User Table

* Description: Contains user model data, responds to queries from other components
* Provided interfaces: User Table Query, Update User Table
* Required interfaces: None

Course Table

* Description: Contains course model data, responds to queries from other components
* Provided interfaces: Course Table Query, Update Course Table
* Required interfaces: None

Announcement Table

* Description: Contains announcement model data, responds to queries from other components
* Provided interfaces: Announcement Table Query, Update Announcement Table
* Required interfaces: None

Syllabus Table

* Description: Contains syllabus model data, responds to queries from other components
* Provided interfaces: Syllabus Table Query, Update Syllabus Table
* Required interfaces: None

Tool Table

* Description: Contains tool model data, responds to queries from other components
* Provided interfaces: Tool Table Query, Update Tool Table
* Required interfaces: None

**Interface name:** Authenticate User

**Interface responsibilities:** Provides authentication logic to the views

Class name: User

**Operations:** Django built in operations

**Operations description:** Authenticates user, displays error message when incorrect, and encrypts stored passwords

Interface name: Login View

**Interface responsibilities:** Handles the view logic for the login page, passes input to the authenticator processes

Class name: N/A

**Operations:** Django built in operations

**Operations description:** Captures input from the Base UI and sends to the built in Django authentication processes

**Interface name:** Course List View

**Interface responsibilities:** Generates the course list page for the user

Class name: CourseView

**Operations:** user\_courses

**Operations description:** Takes in a URL as input and returns rendering logic to the Dash UI if the URL is the courses page

|  |  |
| --- | --- |
| user\_courses |  |
| Operation signature | user\_courses(URL) |
| Operation description | If URL = “courses” this function is called and the courses.html page is rendered |
| Operation preconditions | User is logged in |
| Operation postconditions | Courses page loads |

**Interface name:** Admin Course List View

**Interface responsibilities:** Generates the list of courses on the admin pages

Class name: CourseAdmin

**Operations:** admin.site.register

**Operations description:** Assigns a controller to the admin pages that allows the admin to manipulate the course table

|  |  |
| --- | --- |
| admin.site.register |  |
| Operation signature | admin.site.register(CourseAdmin) |
| Operation description | Takes in a controller class, CourseAdmin, and sends it to the Admin UI page |
| Operation preconditions | None |
| Operation postconditions | Admin can access the CourseAdmin controller class in the Admin UI |

Interface name: Settings View

**Interface responsibilities:** Generates the settings page for the user

**Class name:** SettingsView

**Operations:** user\_settings

**Operations description:** Takes in a URL as input and returns rendering logic to the Dash UI if the URL is the settings page

|  |  |
| --- | --- |
| user\_settings |  |
| Operation signature | user\_settings(URL) |
| Operation description | If URL = “settings” this function is called and the settings.html page is rendered |
| Operation preconditions | User is logged in |
| Operation postconditions | Settings page loads |

**Interface name:** Announcements View

**Interface responsibilities:** Generates the announcements page for the user

**Class name:** AnnouncementsView

**Operations:** course\_announcements

**Operations description:** Takes in a URL and a course model as input and returns rendering logic to the Course UI if the URL is the announcements page

|  |  |
| --- | --- |
| course\_announcements |  |
| Operation signature | course\_announcements(URL, course) |
| Operation description | If URL = “<course\_name>/announcements” this function is called and the announcements.html page is rendered |
| Operation preconditions | User is logged in, user is registered to course |
| Operation postconditions | Announcements page loads |

Interface name: Syllabus View

**Interface responsibilities:** Generates the syllabus page for the user

**Class name:** SyllabusView

**Operations:** course\_syllabus

**Operations description:** Takes in a URL and a course model as input and returns rendering logic to the Course UI if the URL is the syllabus page

|  |  |
| --- | --- |
| course\_syllabus |  |
| Operation signature | course\_syllabus(URL, course) |
| Operation description | If URL = “<course\_name>/syllabus” this function is called and the syllabus.html page is rendered |
| Operation preconditions | User is logged in, user is registered to course |
| Operation postconditions | Syllabus page loads |

Interface name: People View

**Interface responsibilities:** Generates the announcements page for the user

Class name: PeopleView

**Operations:** course\_people

**Operations description:** Takes in a URL and a course model as input and returns rendering logic to the Course UI if the URL is the people page

|  |  |
| --- | --- |
| course\_people |  |
| Operation signature | course\_people(URL, course) |
| Operation description | If URL = “<course\_name>/people” this function is called and the people.html page is rendered |
| Operation preconditions | User is logged in, user is registered to course |
| Operation postconditions | People page loads |

Interface name: Tools View

**Interface responsibilities:** Generates the tools page for the user

Class name: ToolsView

**Operations:** course\_tools

**Operations description:** Takes in a URL and a course model as input and returns rendering logic to the Course UI if the URL is the tools page

|  |  |
| --- | --- |
| course\_tools |  |
| Operation signature | course\_tools(URL, course) |
| Operation description | If URL = “<course\_name>/tools” this function is called and the tools.html page is rendered |
| Operation preconditions | User is logged in, user is registered to course |
| Operation postconditions | Tools page loads |

**Interface name:** User Table Query

**Interface responsibilities:** Receives query requests from the other components and returns the corresponding user object

Class name: N/A

**Operations:** user.objects.all()

**Operations description:** Returns a full list of user objects in the database, additional arguments can filter the list for specific objects

|  |  |
| --- | --- |
| user.objects.all() |  |
| Operation signature | user.objects.all(optional filter()) |
| Operation description | Returns a full list of the user objects in the database, can be filtered with optional arguments |
| Operation preconditions | None |
| Operation postconditions | Full list is returned if objects exist, otherwise an empty list is returned |

**Interface name:** Update User Table

**Interface responsibilities:** Captures changes from the controllers and updates the database accordingly

**Class name:** CustomUserAdmin

**Operations:** Django built in operations

**Operations description:** Registers a form (a controller) to a UI, the user can use the form to make changes to the user model in the database

**Interface name:** Course Table Query

**Interface responsibilities:** Receives query requests from the other components and returns the corresponding course object

Class name: N/A

**Operations:** course.objects.all()

**Operations description:** Returns a full list of course objects in the database, additional arguments can filter the list for specific objects

|  |  |
| --- | --- |
| course.objects.all() |  |
| Operation signature | course.objects.all(optional filter()) |
| Operation description | Returns a full list of the course objects in the database, can be filtered with optional arguments |
| Operation preconditions | None |
| Operation postconditions | Full list is returned if objects exist, otherwise an empty list is returned |

**Interface name:** Update Course Table

**Interface responsibilities:** Captures changes from the controllers and updates the database accordingly

Class name: CourseAdmin

**Operations:** Django built in operations

**Operations description:** Registers a form (a controller) to a UI, the user can use the form to make changes to the course model in the database

**Interface name:** Announcement Table Query

**Interface responsibilities:** Receives query requests from the other components and returns the corresponding announcement object

Class name: N/A

**Operations:** announcement.objects.all()

**Operations description:** Returns a full list of announcement objects in the database, additional arguments can filter the list for specific objects

|  |  |
| --- | --- |
| announcement.objects.all() |  |
| Operation signature | announcement.objects.all(optional filter()) |
| Operation description | Returns a full list of the announcement objects in the database, can be filtered with optional arguments |
| Operation preconditions | None |
| Operation postconditions | Full list is returned if objects exist, otherwise an empty list is returned |

**Interface name:** Update Announcement Table

**Interface responsibilities:** Captures changes from the controllers and updates the database accordingly

**Class name:** AnnouncementForm

**Operations:** Django built in operations

**Operations description:** Registers a form (a controller) to a UI, the user can use the form to make changes to the announcement model in the database

**Interface name:** Syllabus Table Query

**Interface responsibilities:** Receives query requests from the other components and returns the corresponding syllabus object

Class name: N/A

**Operations:** syllabus.objects.all()

**Operations description:** Returns a full list of syllabus objects in the database, additional arguments can filter the list for specific objects

|  |  |
| --- | --- |
| syllabus.objects.all() |  |
| Operation signature | syllabus.objects.all(optional filter()) |
| Operation description | Returns a full list of the syllabus objects in the database, can be filtered with optional arguments |
| Operation preconditions | None |
| Operation postconditions | Full list is returned if objects exist, otherwise an empty list is returned |

**Interface name:** Update Syllabus Table

**Interface responsibilities:** Captures changes from the controllers and updates the database accordingly

**Class name:** SyllabusForm

**Operations:** Django built in operations

**Operations description:** Registers a form (a controller) to a UI, the user can use the form to make changes to the syllabus model in the database

**Interface name:** Tool Table Query

**Interface responsibilities:** Receives query requests from the other components and returns the corresponding object

Class name: N/A

**Operations:** tool.objects.all()

**Operations description:** Returns a full list of tool objects in the database, additional arguments can filter the list for specific objects

|  |  |
| --- | --- |
| tool.objects.all() |  |
| Operation signature | tool.objects.all(optional filter()) |
| Operation description | Returns a full list of the tool objects in the database, can be filtered with optional arguments |
| Operation preconditions | None |
| Operation postconditions | Full list is returned if objects exist, otherwise an empty list is returned |

**Interface name:** Update Tool Table

**Interface responsibilities:** Captures changes from the controllers and updates the database accordingly

Class name: ToolForm

**Operations:** Django built in operations

**Operations description:** Registers a form (a controller) to a UI, the user can use the form to make changes to the tool model in the database

# Architecture Behavior View

## Admin Adds New Course:

1. Precondition: Admin has logged in to the application
2. Admin selects the courses tab
3. Course app queries the database
4. Course table returns all the courses and provides it to the Admin UI
5. Admin UI displays the courses
6. Admin selects add new course tab
7. The course app takes in the request submitted by the admin and loads the new course form
8. The admin then adds the course name
9. The Course app then queries the database
10. If the Course name already exists then the course table informs the Admin UI which then displays an error message and indicates to the admin to add the name again
11. If the Course name doesn’t exist then the course table provides this information to the Admin UI which then allows the Admin to add the course details and course people
12. The Course app captures all the information given by the Admin and provides this to the Course table
13. The course table then saves all this information

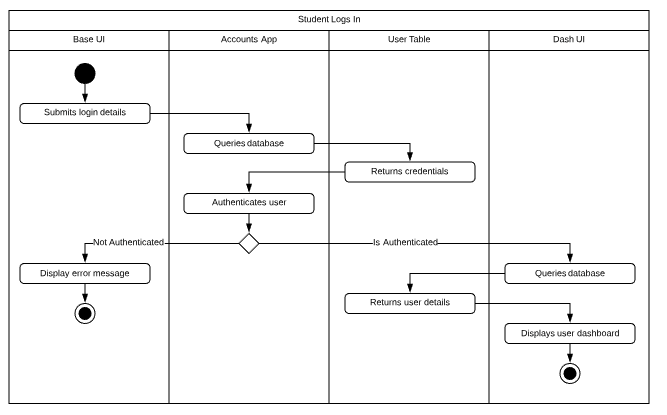
## Teacher Turns on Reader Mode:

1. Precondition: Teacher has logged in to the application
2. In the Dash UI, teacher selects the Settings tab
3. The User table returns all the User settings and provides to the Dash UI
4. The Dash UI displays the user settings
5. If the reader mode is off, the teacher selects the “reader mode on”
6. The settings app captures the change made by the teacher and provides to the User Table
7. The User saves the changes made by the teacher

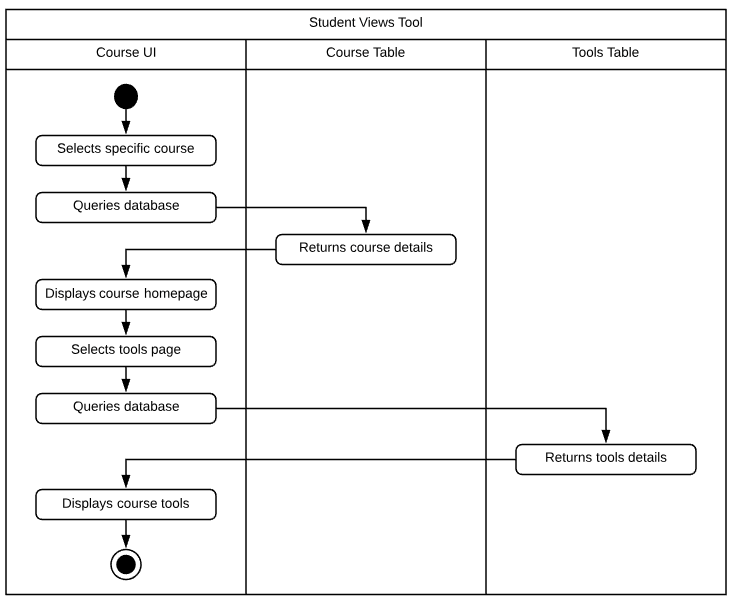
## Teacher Adds Announcement:

1. Precondition: Teacher has selected the course
2. In the selected course view, teacher selects the announcement tab
3. The announcement table returns all the announcements for the selected course and provides to Course UI
4. Course UI displays the announcements
5. Teacher selects the add announcements tab
6. The announcement app takes in the request submitted by the teacher and loads the announcement form
7. In the Course UI the teacher writes the announcement and selects the submit tab
8. The Announcement app captures the announcement and provides it to the announcement table
9. The Announcement table saves that announcement and returns the announcement to the Course UI
10. The Course UI displays the new announcement

## Student Logs In:



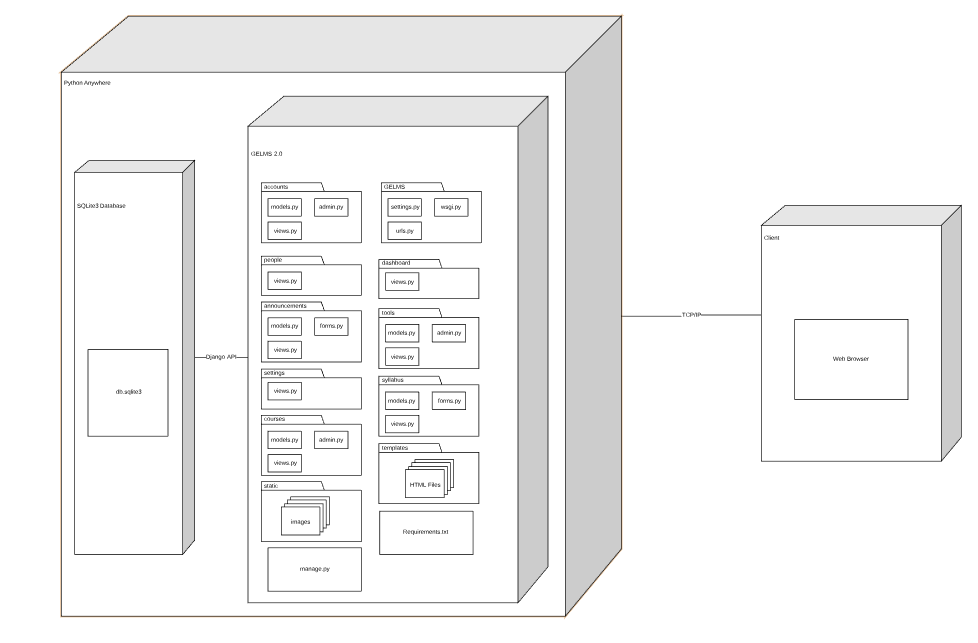
1. In the Base UI, the student submits his/her login details entered
2. The Accounts app queries the database
3. The User table returns the credentials and provides to the Account app which then tries to check if the user is authenticated or not
4. If the student is not authenticated, it provides the information to the Base UI which then displays an error message
5. If the student is authenticated, then the Dash UI queries the database
6. The User returns the User details
7. The dash UI displays the User dashboard

Student Views Tools:

1. Student has logged in to the application and is in the Course UI
2. Course UI displays all the courses taken by the student
3. Student selects the course for which she/he wishes to check the tool
4. Course Table returns the course details for the course selected by the student and provides to Course UI
5. Course UI displays the course homepage
6. Student selects the Tools view
7. Tools Table returns the tools details and provides to Course UI
8. Course UI displays the tools of that course

# Architecture Deployment View

## Deployment Diagram:



Deployment Diagram Description:

The deployment of the system is relatively simple. The goal is to have the entire system hosted by cloud services, namely PythonAnywhere. This includes the SQLite3 database which will contain the tables that hold the different information for our system. Users will access the application via the internet, changes made by users will be captured by the application server and configured in the SQLite3 database. Similarly, the application server will pull data from the database and display it as necessary to the user.

# Information View

## **https://lh6.googleusercontent.com/p6TT0h91RFIqDHDHBNucKoiSml01zzJuiwxS5hbevw_v1ljfcaurUmWrj-haUSF30VNn5ATOUs7On9b1f8p3N4EMT7_F8Vu6VeMUU9XqY2D1dAWj9iZ35AIWICsWOXkiylRiXyCVR9BnKGwQqQ**Entity Relationship Diagram:

## Entity Relationship Diagram Description**:**

The Users entity in the ERD Diagram represents the main entity which has various sub attributes and then it includes the two entities which are Students and Teachers. Students and Teachers have their attributes as listed above.

1. Starting from Teacher entity, we are assuming that one Teacher can teach many Courses, but Courses should not have more than one Teacher
2. Also, we are assuming that Teachers will only add the one Syllabus per course, but the Teacher can add and delete multiple Announcements.
3. Students can register for multiple Courses and can view Announcements and Syllabi posted by the Teacher
4. A Course can have multiple Tools that a Teacher can either enable or disable accordingly

# Requirements Allocation

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Base UI | Dash UI | Course UI | Admin UI | Accounts App | Dashboard App | Courses App | Settings App | Announcements App | Syllabus App | People App | Tools App | User Table | Course Table | Announcement Table | Syllabus Table | Tool Table |
| Admin adds/removes courses |  |  |  | X |  |  | X |  |  |  |  |  |  | X |  |  |  |
| Admin adds/removes student to course |  |  |  | X | X |  | X |  |  |  |  |  | X | X |  |  |  |
| Admin assigns teachers/grader to course |  |  |  | X | X |  | X |  |  |  |  |  | X | X |  |  |  |
| Teacher posts/edits/deletes announcements | X | X | X |  |  |  |  |  | X |  |  |  |  |  | X |  |  |
| Teacher posts/edits/deletes syllabus | X | X | X |  |  |  |  |  |  | X |  |  |  |  |  | X |  |
| Teacher views list of of people in course | X | X | X |  |  |  |  |  |  |  | X |  |  |  |  |  |  |
| Teacher locks/unlocks tools for course | X | X | X |  |  |  |  |  |  |  |  | X |  |  |  |  | X |
| Teacher turns on/off reader mode | X | X |  |  |  |  |  | X |  |  |  |  | X |  |  |  |  |
| Teacher turns on/off multi factor authentication | X | X |  |  |  |  |  | X |  |  |  |  | X |  |  |  |  |
| Student views announcements of course | X | X | X |  |  |  |  |  | X |  |  |  |  |  | X |  |  |
| Student views syllabus for course | X | X | X |  |  |  |  |  |  | X |  |  |  |  |  | X |  |
| Student view tools of course | X | X | X |  |  |  |  |  |  |  |  | X |  |  |  |  | X |
| Student turns on/off reader mode | X | X |  |  |  |  |  | X |  |  |  |  | X |  |  |  |  |
| Student turns on/off multi factor authentication | X | X |  |  |  |  |  | X |  |  |  |  | X |  |  |  |  |

# Architecture Work Allocation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Daniel | Nishanth | Manu | Vaibhav | Sagar |
| Base UI | Testing | Testing | Development | Development | Testing |
| Dash UI | Testing | Testing | Development | Development | Testing |
| Course UI | Testing | Testing | Development | Development | Testing |
| Admin UI | Testing | Testing | Development | Development | Testing |
| Accounts App | Development | Development | Testing | Testing | Testing |
| Dashboard App | Development | Testing | Testing | Testing | Testing |
| Courses App | Development | Development | Testing | Testing | Testing |
| Settings App | Development | Development | Testing | Testing | Testing |
| Announcements App | Testing | Testing | Development | Development | Testing |
| Syllabus App | Testing | Testing | Development | Development | Testing |
| People App | Development | Development | Testing | Testing | Testing |
| Tools App | Development | Development | Testing | Testing | Testing |
| User Table | Development | Testing | Testing | Testing | Development |
| Course Table | Development | Testing | Testing | Testing | Development |
| Announcements Table | Testing | Testing | Development | Development | Development |
| Syllabus Table | Testing | Testing | Development | Development | Development |
| Tools Table | Testing | Testing | Testing | Development | Development |

# Detailed Design Key Decisions and Rationale

The main influences for our detailed design decisions stem from the Django framework. When decomposing our architecture into the detailed components, we considered Django best practices. For example, we separated the different website functions into distinct “apps.” Within each app we included classes that represent the data called “Models,” classes for rendering webpages called “Views,” and classes for editing the models called “Forms” or “Admins.” A brief discussion on software drivers ensues below:

Modularity: Per Django best practices, we broke our detailed design down into unique components, namely the “apps.” Each app can be added to, edited, or managed independently of another. This allows for different team members to work on different apps simultaneously.

Reuse: We reused a lot of inbuilt Django features and, in fact, selected Django because of its high degree of reusability. The features we reused include, but are not limited to, user authentication, an admin portal, a database API protocol, and a framework for a flexible model view controller.

Cohesion: Here we went against Django best practices in favor of more cohesive code. In one case, Django best practices recommends that all URL references be kept in their respective apps. For example, the code for the URL functionality of the announcement page would be written within the announcements app. However, we decided it would be best to keep all URL related code in a single URL file within the GELMS config files. This made the code easier to maintain and edit as we add extra pages/URL functionalities to our product.

Patterns: The patterns we made use of are namely the broker patterns and, more specifically, the façade pattern. This is due largely to the framework of Django. Django provides a lot of inbuilt functionality, a lot of which we utilized while not directly viewing the source code. In order to design our application’s behavior, we created facades that offered a simplified interface between our own code and the lower level source code of Django.

# Detailed Design Structure

## View Detail Structure Diagram:

A screenshot of a cell phone

Description automatically generated

## View Detail Structure Diagram Description:

This diagram demonstrates decomposition of the view components. The base UI is the highest layer and the other UI’s inherits some of the base UI files. The dash UI is the middle layer and it is the first UI the user sees when they log in. The course UI is the lowest layer and displays course data. Finally, the admin UI is the stand-alone UI that is inbuilt in Django and displays the admin pages.

## Controller Detail Structure Diagram:

## Accounts App Diagram:

A screenshot of a cell phone

Description automatically generated

instantiates

## Accounts App Description:

1. Custom user class is the model which inherits from the inbuilt django user class and inherits its attributes. We have added some extra attributes like uid and status to populate this model. It has functionality which will gives the list of courses to different user like teacher, grader and student.
2. Custom user admin class inherits from the inbuilt django admin class and CustomUserInline class and is solely responsible for creating a profile of the user which will include all the necessary details. It realizes custom user class to perform the above functionality.

## Dashboard App Diagram:

A screenshot of a cell phone

Description automatically generated

## Dashboard App Description:

1. DashboardView class is responsible for handling the user request and rendering the appropriate html page (dashboard.html) in this case.

## Course App Diagram:

A screenshot of a social media post

Description automatically generated

displays

instantiates

uses

## Course App Description:

1. CourseAdmin:
   1. A class describing the model admin manager extension from the  Django-inbuilt model admin manager which adds custom functionality to the admin interface for adding/deleting courses and adding people to it.
   2. It provides the functionality to add a course, and assigns users(teachers,students and/or graders) to that course
   3. It realizes the Course model class and is used by the CourseView.
2. Course Model
   1. A model that describes the course object with relevant fields that are as follows:
      1. Name: Name of the course
      2. Title: Short description of the course
      3. Domain: The domain of the course(such as Engineering, Arts etc)
      4. Student: A many-to-many foreign key to the User model essentially describing all the students associated with that course.
      5. Teacher: A many-to-many foreign key to the User model essentially describing all the teachers associated with that course.
      6. Grader: A many-to-many foreign key to the User model essentially describing all the graders associated with that course.
   2. The model also has the Announcement model, Syllabus model, People model and Tools model related to it via various foreign keys.
   3. The Course model also provides the following functions:
      1. \_\_str\_\_(): A function to conveniently return important fields of the course in a readable format.
      2. get\_students(): A function that returns the list of students associated with it.
      3. get\_teachers(): A function that returns the list of teachers associated with it.
      4. get\_graders(): A function that returns the list of graders associated with it.
      5. people(): A function that returns the list of all people(users) associated with it.
3. CourseView
   1. CourseView is a class that is responsible for the rendering the course page.

## Settings App Diagram:

A screenshot of a cell phone

Description automatically generated

## Settings App Description:

1. SettingsForm
   1. A class that contains the functionality to allow users to update the CustomUser model to enable/disable reader mode and multi-factor authentication in the front end.
   2. It is used by SettingsView.
2. SettingsView
   1. A class that is responsible for rendering the form that it uses(SettingsForm) to render the Settings page to update the CustomUser Model.

## Announcement App Diagram:

A screenshot of a cell phone

Description automatically generated

instantiates

displays

## Announcements App Description:

1. Announcement Model
   1. A class that describes the announcement objects and contain all the relevant details that captures an announcement as following:
      1. Content: Content of the announcement
      2. Date: The date the announcement is published on.
      3. Course: A foreign key to relate the model to the Course Table.
   2. The functions provided by the class are as follows:
      1. \_\_str\_\_(): Returns the relevant field values of the model in a convenient readable format.
2. AnnouncementForm
   1. A class that contains the functionality to allow users to update the Announcement model to add/edit/delete an announcement.
   2. It is used by AnnouncementView.
3. Meta
   1. A class that is contains meta information about the AnnouncementForm class
4. AnnouncementView
   1. A class that is responsible for rendering the form that it uses(AnnouncementForm) to render the Announcement page to update the Announcement Model.

## Syllabus App Diagram:

A screenshot of a cell phone

Description automatically generated

displays

instantiates

## Syllabus App Description:

1. Syllabus Model
   1. A class that describes the syllabus objects and contain all the relevant details that captures an syllabus as following:
      1. Content: Content of the syllabus
      2. Date: Date associated with a syllabi item.
      3. Course: A foreign key to relate the model to the Course Table.
   2. The functions provided by the class are as follows:
      1. \_\_str\_\_(): Returns the relevant field values of the model in a convenient readable format.
2. SyllabusForm
   1. A class that contains the functionality to allow users to update the Syllabus model to add/edit/delete an announcement.
   2. It is used by SyllabusView.
3. Meta
   1. A class that is contains meta information about the SyllabusForm class
4. SyllabusView
5. A class that is responsible for rendering the form that it uses (SyllabusForm) to render the syllabus page to update the Syllabus Model.

## People App Diagram:

A screenshot of a social media post

Description automatically generated

People App Description:

1. PeopleView is the class which has the functionality for displaying the people that are registered for a particular course.

## Tools App Diagram:

A screenshot of a social media post

Description automatically generated

displays

displays

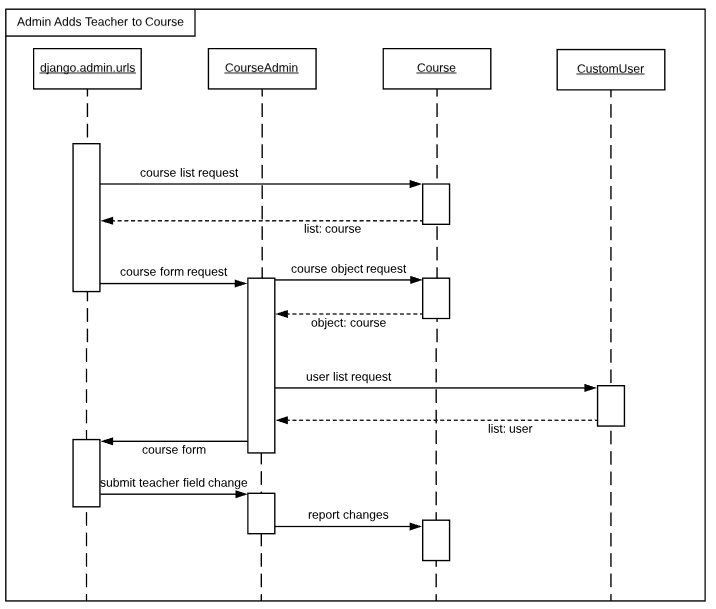
instantiates

## Tools App Description:

1. Tools\_model contains the class that describes the Tool object capturing the information about the tool such as:
   1. Link: Links to the tool
   2. Status: A boolean field describing whether the tool is locked or unlocked
   3. Course: A Foreign key to the Course that the tool belongs to
   4. Domain: Describing which domain the tool is a part of
2. ToolsForm contains a class that has the functionality to allow users(teachers) to update the status of the tool in the Tools\_model. It is used by ToolsView.
3. ToolsView contains the class that is responsible for rendering the form that it uses(ToolsForm) to render the syllabus page to update the Tools\_model.

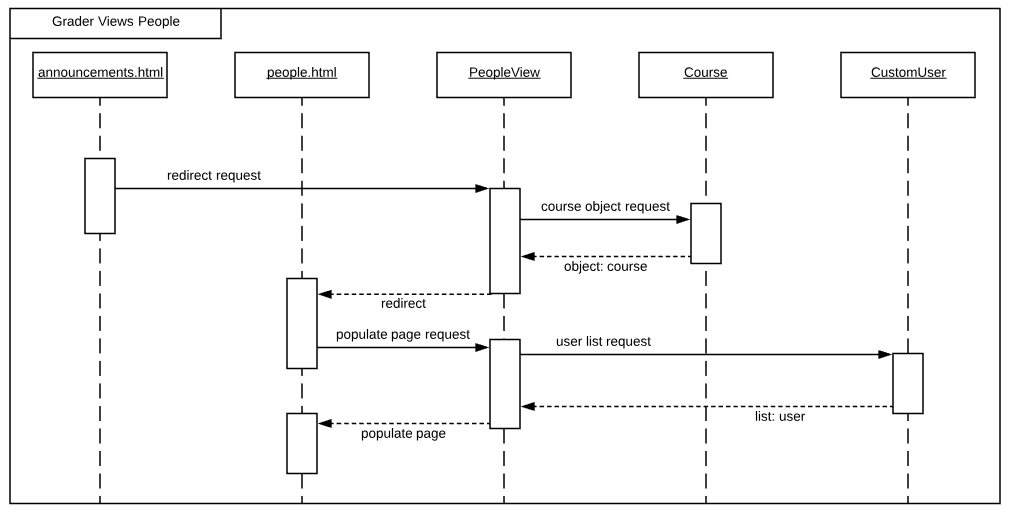
# Detailed Design Behavior

## Admin Adds Teacher to Course:



1. The Django.admin.urls queries the database
2. The course table returns the course list and provides it to the Django.admin.urls
3. In Django.admin.urls the admin selects the existing course, the CourseAdmin then prepares the course form
4. The course table returns the selected course object
5. The user table returns the available teachers as a list of users
6. The Django.admin.urls displays the available teachers, the admin then selects a teacher from the available teachers and submits the change
7. The CourseAdmin then acknowledges the change and submits it to the course table
8. The course table then saves these changes

## Grader Views People:



1. Precondition: The grader has logged in to the application and has selected the course
2. In the announcement.html the grader selects the people tab, a redirect request is sent to PeopleView
3. PeopleView queries the database for the course object
4. The course table returns the appropriate course object
5. PeopleView fulfils the redirect request from announcements.html and user is redirected to people.html
6. People.html requests PeopleView to populate the page
7. PeopleView queries the database for the list of users for the particular course
8. The user table returns the list of users for the course
9. PeopleView then sends the data to people.html for display

## Student Views Announcement:

A screenshot of a cell phone

Description automatically generated

1. In the course.html, the student selects the course. The announcement view queries the database
2. The course table returns the course object and provides it to the announcement view which then again queries the database
3. If the announcement for the course doesn’t exist then the announcement.html displays “no announcement”
4. If the announcement for the course exists then announcement table returns all the announcements and provides it to announcement.html which then displays all the announcements

## Teacher Adds Syllabus:

A screenshot of a cell phone

Description automatically generated

1. Precondition:  Teacher has selected the Syllabus tab
2. The syllabus view queries the database
3. The course table returns the course object and provides it to the syllabus view which then again queries the database
4. The syllabus table then checks if the syllabus for the course exists or not. If the syllabus exists then the syllabus table returns that course syllabus and provides to syllabus.html which then displays it
5. If the syllabus doesn’t exist, the syllabus.html displays syllabus form for the Teacher to enter the syllabus
6. Once the teacher enters the syllabus, the syllabus.html submits the entered syllabus to the syllabus form which then captures the information and provides it the syllabus table
7. The syllabus table saves the information

## Teacher Locks Tools:

A screenshot of a cell phone

Description automatically generated

1. Precondition:  Teacher has selected the Tools tab
2. The tools view queries the database
3. The course table returns the course object and provides it to the tools view which then again queries the database
4. The tools table returns the tool of the course
5. If the tool is locked the tools.html displays the locked tool and the teacher takes no further action
6. If the tool is unlocked the tools.html displays the unlocked tool and the teacher then changes the unlocked tool to locked
7. The tools.form captures the changes made by the teacher and provides this information to the tools.table
8. The tools.table saves the changes

# Physical Data Model

User Table

|  |  |
| --- | --- |
| Field | Data Type |
| id (primary key) | int |
| first\_name | charfield |
| last\_name | charfield |
| username | charfield |
| password | passwordfield |
| email | emailfield |
| uid | charfield |
| status | charfield |

Course Table

|  |  |
| --- | --- |
| Field | Data Type |
| id (primary key) | int |
| name | charfield |
| title | charfield |
| domain | charfield |
| students | list: user object |
| graders | list: user object |
| teachers | list: user object |

Announcement Table

|  |  |
| --- | --- |
| Field | Data Type |
| id (primary key) | int |
| date | datefield |
| content | textfield |
| course | course object |

Syllabus Table

|  |  |
| --- | --- |
| Field | Data Type |
| id (primary key) | int |
| date | datefield |
| content | textfield |
| course | course object |

Tool Table

|  |  |
| --- | --- |
| Field | Data Type |
| id (primary key) | int |
| link | urlfield |
| status | booleanfield |
| domain | domain |
| course | course object |

# Algorithms

Considering that our GELMS application does not require any complex logic to sift through data, in addition to the fact that the Django framework provides plenty of inbuilt functionalities, we did not have a need to develop any sophisticated algorithms other than searching for data via for loops and querying the database via API calls.

# Architecture to Detailed Design Tracing

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Base UI | Dash UI | Course UI | Admin UI | Accounts App | Dashboard App | Courses App | Settings App | Announcements App | Syllabus App | People App | Tools App | User Table | Course Table | Announcement Table | Syllabus Table | Tool Table |
| Base.html | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| login.html | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| style.css | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| announcements.html |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| syllabus.html |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| people.html |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| tools.html |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| django.admin.urls |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |
| base\_dash.html |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| dashboard.html |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| courses.html |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| settings.html |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DjangoAdmin |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |
| CustomUserInline |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |
| CustomUserAdmin |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |
| User |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |
| CustomUser |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |
| CourseAdmin |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |
| Course |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |
| CourseView |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |
| Syllabus |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |
| SyllabusForm |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |
| Meta(Syllabus) |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |
| SyllabusView |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |
| Announcement |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |
| AnnouncementForm |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |
| Meta(Announcement) |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |
| AnnouncementView |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |
| Tools |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |
| ToolsForm |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |
| ToolsView |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |
| DashboardView |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |
| SettingsForm |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |
| SettingsView |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |
| PeopleView |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |
| UserTable |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |
| CourseTable |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |
| AnnouncementTable |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |
| SyllabusTable |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |
| ToolTable |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |