**<CollegeMeet>**

**Software Design**

**CSCI-P465/565 (Software Engineering I)**

**Project Team**

**<Shweta Bhartia>**

**<Scott Mathews>**

**<** **Yuanzhi Bao>**

**<** **Karanpaul Multani>**

**1. Introduction**

The architecture is MVC (Model-View-Controller).

View is the web Interface, through which the user will be able to use this product.

Model is the data store. And, Controller transfers data between View and Model. This section introduces the design approach to the software system.

It is a 3-tiered architecture: It has three layers View layer, Control layer and Data layer. There is also an intermediator data control layer between the data layer and the control layer  
  
**1.1 System Description**

CollegeMeet is a social media platform aimed at facilitating dating between members of colleges and universities. Other dating platforms are based primarily on looks, or a survey taken once. CollegeMeet will provide a platform through which topics and interests can be explored through discussion, thus giving more insight into the character of the people you’re dating.

**1.2 Design Evolution**

This section is intended to document the rationale behind the selected design solution.

**1.2.1 Design Issues**

Software Requirements

* Operating Systems: Windows 10, Linux redhat, MacOS
* Environment: Django framework, PostgreSQL, JavaScript libraries
* Web Browser: To access the website

Hardware Requirements

* Processor: 2 Ghz
* RAM: 1 GB
* Hard Disk: 10 GB

**1.2.2 Candidate Design Solutions**

Initially, our team decided to use Node.js framework for backend but later we switched to Django framework. We are using third party API for implementing chat functionality using socket programming. We are using Docker to build the application and deploy it. We decided to deploy the application on Heroku rather than on Amazon Web Services.

**1.2.3 Design Solution Rationale**

Django has a more structured implementation for building common endpoints, and common application functionality on the backend. Since Docker is the containerizing application, deploying the application using Docker is advantageous. Also, it helps in deploying any operating system without having to rebuild the application specifically. Heroku provides better functionality for hosting our application as it is a “Platform as a Service”.

**1.3 Design Approach**

**1.3.1 Methods**

We are using Model–view–controller (MVC) design patter for our project development. MVC is a [software design pattern](https://en.wikipedia.org/wiki/Software_design_pattern) for implementing [user interfaces](https://en.wikipedia.org/wiki/User_interface) on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and parallel development.

**1.3.2 Standards**

Our project follows strict security guidelines. We are doing password encryption using PBKDF2 algorithm with a SHA256. Since this is the standard security protocol for passwords and we can’t ever access the plain text passwords.

Django framework adhere to best practices. The core component in Django is called the 'app', which is a logical collection of code which completes a specific purpose. In Django, every component has its own single Django app. This ensure that all the components are logically separated and each app contains its own models, view, controllers and tests for logic isolation.

We will use Heroku for deployment which is a PAAS and it will follow the standards of using production server.

Design standards are followed using front-end framework React.js and Bootstrap. Front end code will conform to create-react-app eslint profiles. These coding standards can be enforced through automated checking of the system.

**1.3.3 Tools**

Create-React-App: framework upon which front-end design works. Provides functionality for code generation of React components through the command line.

Web Server: Django for API, Node for serving web pages.

Database: PostgreSQL

Containerizing Application: Docker

Deploying Application: Heroku

**2. System Architecture**

**2.1 System Design**

We have two different workflows in our system -user and page. We will be building a web based application that accomplishes all these tasks. The UI for this website will be built with React.js, and bootstrap. The backend for this is done using Django framework. The database is PostgreSQL.

Graphic Demonstrating Relationship between API separated layers of application.

Adjacent layers communicate with each other.

**2.2 External Interfaces**

User interface:

The web page interacts with the user through familiar components, such as Forms, Buttons, Checkboxes, and Selectors. All interactions trigger actions in the Redux layer, which update the client state, and if necessary make API calls to the Server.

**3. Component Design**

The system involves cooperation between a Client and a Server, in which the Client Display, Client State, Server API, and Database all communicate through well-defined protocols with each other. The protocols are further described in the component manifesto below.

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Change Description** |
| Initial Draft V 1.0 | 2/19/17 | First conception of System design structure |
| Modified Version V 1.1 | 3/05/17 | Added standards for our System |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Page Author/Creator:** [Adeel Bhutta](http://homes.soic.indiana.edu/aabhutta/) **Last Modified:** 8/23/2016