Mentoring Platform

Group 2

Ananya Bhatia	2017020
Karan Tandon	2017058
Shaurya Bagga	2017104

Goal

The main aim of the project is to facilitate this mentoring by providing a platform for such mentors to guide other students and faculty to the right path. What we mean by mentoring is not giving online lectures about technical topics, but directing the mentees towards relevant online resources like lectures, tutorial videos, upcoming programming workshops/contests etc.

The platform would include certain disciplines (like CSE, ECE, ME etc.) where respective mentors and mentees can sign up and connect with each other.

Scope

There is a lot of scope for top tier colleges and universities to help other colleges / institutions in India. This can be broken down into two major needs:

1. Improving the learning of students and make them job ready
2. Improving the quality of teachers so they can improve their teaching, which would in turn facilitate the first need

Students and faculty in top institutions are in a very good position to fulfill these needs by mentoring students and faculty of other institutions. With the growing popularity and acceptance of electronic mediums to meet and collaborate, creating an online platform to tap this potential of the upper tier colleges would be a very viable and interesting idea to bridge this opportunity gap.

Design Details

Language

We will be using **Python** for developing our project. Python has been widely used for server-side programming, owing to its dynamic website creation capabilities. It is widely used for fast prototyping and building highly scalable web applications.

Apart from this, Python also has a ton of frameworks for web development, Django being the most widely used, which will be taken up in later slides.

Framework (front-end)

For the front-end, we will be using **Vue.js** framework for the following reasons:

- **Performant**: Vue scales nicely and has built-in optimizations to make the app more responsive
- **Simplicity**: As this was the first time any of us was using a front-end framework, we wanted to start with one which is beginner friendly. Vue has a very intuitive coding syntax, with minimal number of lines needed to get what we want.
- **Rich community support**: Vue has a vast community support. This would help us get started quickly (by utilizing the tutorials made by developers) and resolving any errors that may occur.
- **Extensive documentation**: Vue has a very detailed and clearly written documentation, making it easy to start learning and using it.

Framework (back-end)

For the back-end we will be using **Django** framework for the following reasons

- **Fully Featured**: It has an admin panel, ORM for easy database interactions, a directory structure for apps
- **Well established:** Django has been on the market for a long time and has a big community
- **Efficient:** due to its built-in template engine
- **Fully loaded:** There is no need to use third party tools or libraries
- **Secure:** Provides support to prevent SQL-injection, XSS
- **Well documented:** Django has a very detailed and clearly written documentation, making it easy to start learning and using it.

Database

We will be using **PostgreSQL** to manage our database

- **Highly Extensible**: You can define your own data types, build out custom functions, even write code from different programming languages without recompiling your database! It also has support for international characters.
- **Concurrency and Performance**: Some powerful types of basic/advanced indexing methods like B-tree, Bloom Filters, etc.
- **Security**: PostgreSQL have SSL encryption native support on traffic data, between Client and Databases, and so the data is protected from man in the middle attacks
- **Data integrity**: Postgre is fully ACID compliant

We will use Django ORM to interact with the application data in PostgreSQL

Server

We decided to go with **NGINX** as our web server. As the development server that Django or any other Python framework (e.g. Flask) provides is suitable only for development purposes and cannot handle large number of requests, we use a web server which overcomes this limitation. We chose NGINX because:

- **Extremely fast**: NGINX can easily handle upto 10,000 simultaneous users connected to a website
- **Easy to use**: One can set up an NGINX server in a very few lines of code
- **Load Balancing**: By running multiple instances of our application server in parallel and using NGINX as a load balancer, we can easily scale our website and maintain its performance even in high workloads
- **Serves static content**: NGINX can be configured to serve static content (HTML pages, images etc.) and not bother the application server for such requests

Architecture Style

We are planning to use a **3 tier architecture** for our project. For this particular project, the advantages of using a 3 tier architecture would be:

- It allows building the application in a modular fashion
- The interfaces would be well-defined, thereby leading to a neat code base and effortless debugging.
- Would make the application extensible, as any of the three components can be upgraded independently in response to changes in requirements or technology.

3 Tier Architecture

Client Tier

This tier consists of the **User Interface (UI)**. The main job of this layer is to take inputs from the users (in the form of clicks / forms) and pass it to the middle tier for for processing.

Business Logic

This tier includes **logical checks** like:

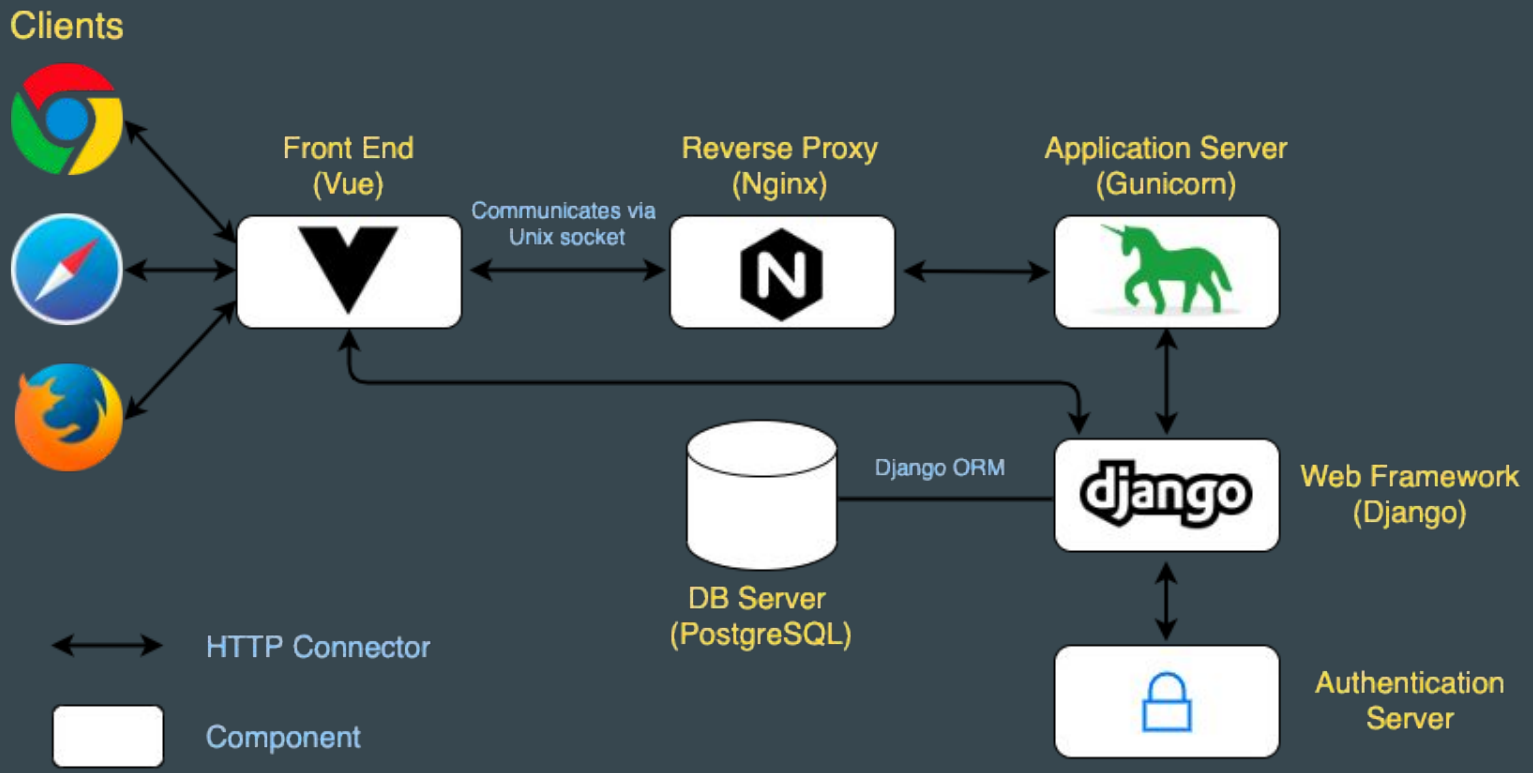
- Authentication
- Is the user logged in
- Is the user allowed to a particular task? For instance, a mentee can't rate a mentor who hasn't mentored him.

Database Tier

This tier includes the **database** that stores:

- List of users
- Details of each user
- Posts of the discussion board.
- Mentor-mentee chat history
- Files like resumes, resources etc.

Component Connector View (Diagram)



Component Connector View (Explanation)

- Multiple clients can get their requests served parallely. When a client sends a request through the UI (Vue) it is redirected to the NGINX reverse proxy, which basically decides where a request should go. For example, if the incoming request is an HTTP request Nginx redirects it to gunicorn, if it is for a static file, it serves it itself.
- From there, it is passed on to the Django backend server through the Gunicorn application server, whose job is basically:
  - communicating with multiple web servers
  - reacting to lots of web requests at once and distributing the load
  - keeping multiple processes of the web application running
- The Django backend also communicates with:
  - The PostgreSQL server for retrieving data units
  - Vue front end to display the retrieved data

Architectural Attributes

Operational Architectural Attributes

- **Availability**: The server would be running 24x7, thereby ensuring incessant availability
- **Reliability**: We would handle all the border cases to ensure smooth, reliable functioning of the application
- **Response time**: The response time is expected to be ≤ 100 ms for 90% of the time
- **Peak number of transactions**: Although NGINX would ensure proper load balancing of client requests, we were planning to limit the number of transactions (for security reasons) to 25 per second initially, which can later be changed.
- **Scalability**: We will code the application in such a manner such that it has provisions for upscaling/downscaling at later stages. NGINX would also help in this particular task, in terms of peak number of transactions.
- **Recoverability**: We are planning to use Replication for fault tolerance

Structural Architectural Attributes

- **Customizability**: Users would have privacy settings [High] and colour theme settings [Optional]
- **Portability**: We are not planning to containerize the application, so the supported platforms would be limited.
- **Internationalization**: English would be the only language available, as all the users are expected be well versed with it. Adding other languages' support can be a future augmentation.
- **Extensibility**: The source code would be in a modular fashion, thereby ensuring easy extensibility if the need arises (Eg. add more disciplines, add features etc.)

Note: *Priority mentioned as [priority-level]*

Miscellaneous Architectural Attributes

- **Authentication**: would be done through Google login, thereby ensuring the validity and authenticity of users' contact details.
- **Authorization**: The business logic will include exhaustive and extensive checks to ensure authorization.
- **Privacy**: The users will have the power to decide what information to make public and what to keep private, through the "Privacy Settings" provision.
- **Security**: Django ensures immunity towards all common cyber attacks like SQL injection, XSS attacks, MITM attacks etc.