# System Design for Shibainu

# Table of Contents

# System Interaction with Environment

OS Requirement:

Unix (Linux) or Windows 10

Programming Language Compilers:

JavaScript Engine - Browser dependent (V8, Spidermonkey, etc)

Relational Database Management System:

PostgreSQL - version 11 or higher

Network Configuration:

Right now, the web application is undeployed, so it runs on https://localhost:3000 on any local machine.

Tools Required:

NodeJS - version 12 or higher

NPM - version 13 or higher

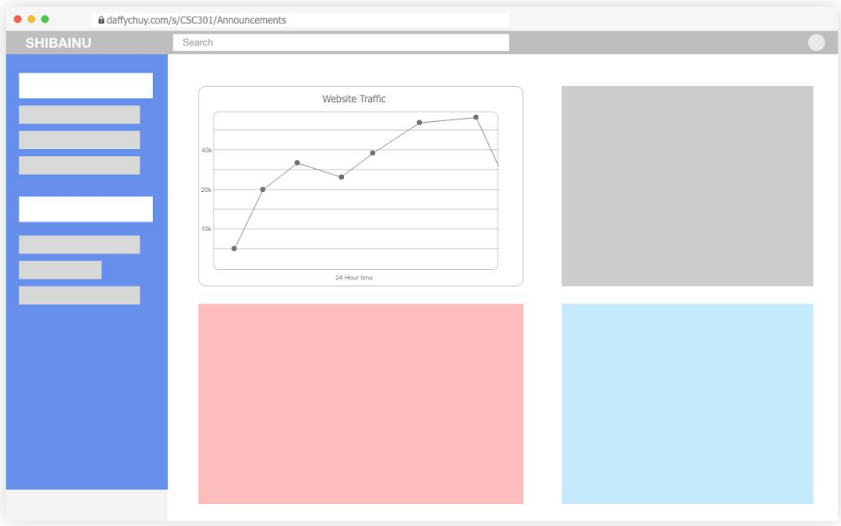
IDE/Editor:

Any of Visual Studio Code, Eclipse or IntelliJ

# UI/UX Diagram

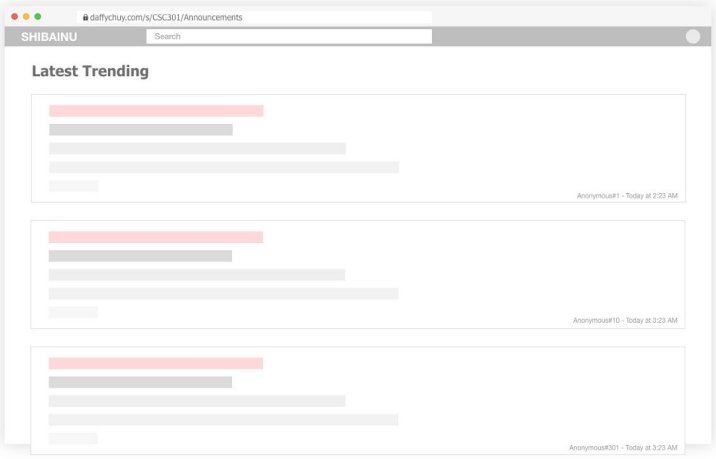
Note: We are using UI/UX diagrams as an alternative to CRC cards

**Admin page** for extensive moderating of the website, including web traffic.



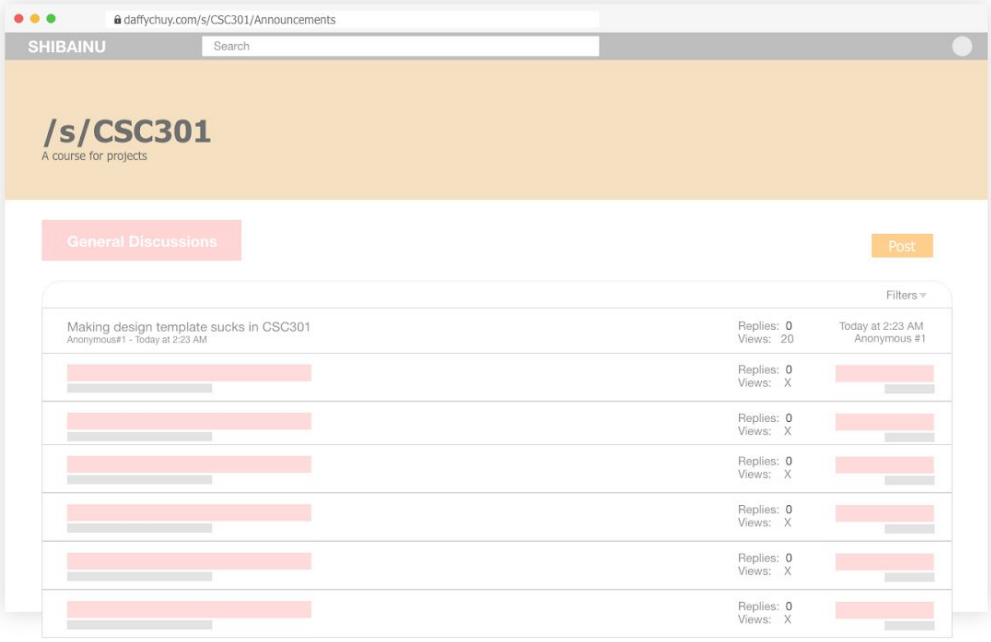

**Front page** for taking a look at what's trending along with links to the threads and its accompanying category.

Moderator can option to not have their forum to be on trending if their subpage is about their classes

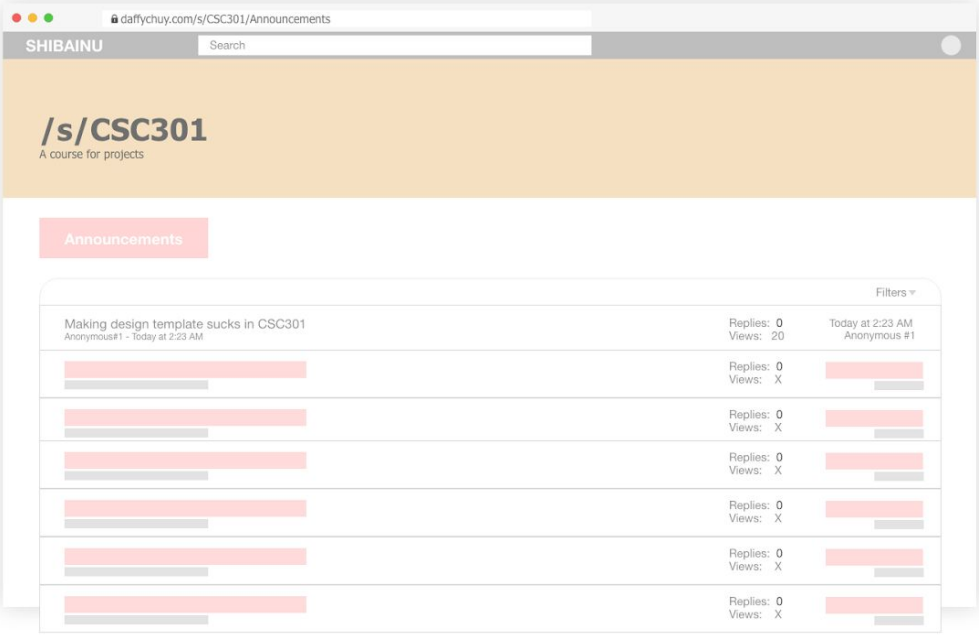


Example of a **Category page** for CSC301

Forum where they're allowed to post



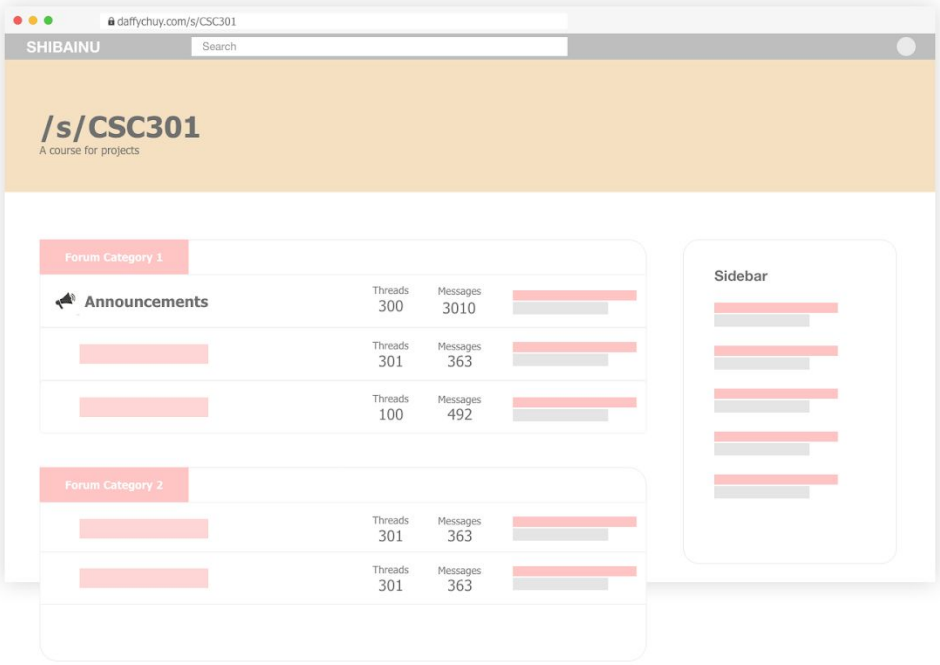

Moderator can also limit which thread can be posted by who



Example of a **Subpage** which further concentrates topics within a Category page in CSC301

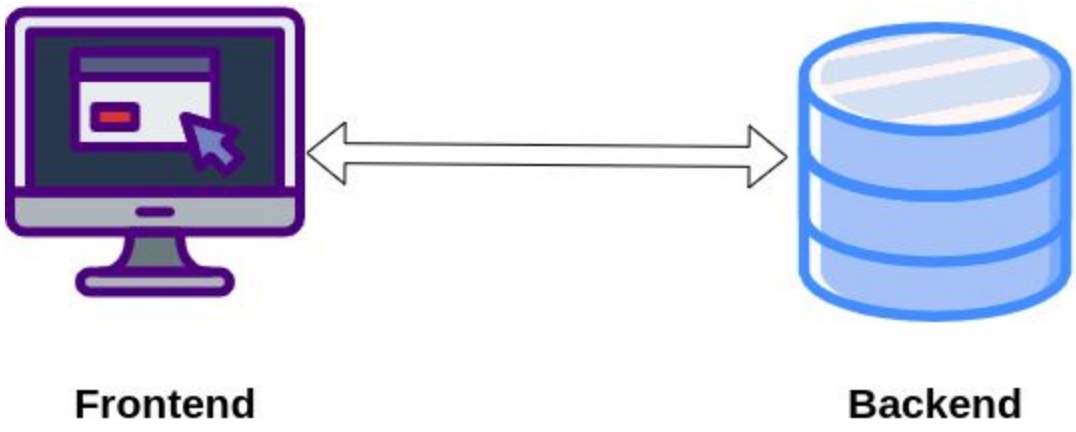
They can also customize the theme to their liking

Moderator can also set only registered anonymous can post in said forum



## Architecture of the System

The system, in an abstracted view, is divided into the frontend and backend.




The frontend is the user-facing interface that users, moderators and admins of the website interact with. Users of the website will be able to consume, provide and interact with the content in it.

When users provide new content through the form of posts, comments or showing other forms of media, it will connect to the database through a REST api. In addition, serving up web pages, updating or deleting resources is done by following the REST api.

The backend is the data access layer which contains the database. It provides long-term storage for all resources on the website and serves it up for the frontend to use when it is accessed.

# System Decomposition

## Roles of components

- Frontend
    - Tailwind CSS - styling and customizing the website
    - EJS - generates html markup with javascript
    - All files related to the frontend should be in the views folder.

- Backend
    - PostgreSQL - our database for storing resources (schema and er diagram here).
    - ExpressJS - a lightweight server framework that will handle requests and responses to the client.

- REST Api
    - We are using REST because it makes it easy and simple to create documentation, develop a public api, and to separate the client and server.
    - Our api will be exposed to all users that will want to take advantage of it.
    - Documentation can be found here or when running the web app, go to https://localhost:3000/api/api-docs.
    - The queries.js file will contain the actual code for the HTTP handlers that will service the endpoints
        - The index.js file will be where the context for the endpoints will be declared and linked to the handlers specified in queries.js

## Errors and Exceptional cases

Errors will be handled in different manners depending on what kind of error it is. In general, all errors will be shown through the UI. For example,

- A user logging in with the wrong credentials will be routed back to the login page with a popup indicating the wrong username/password.
- Accessing pages/resources that don't exist (e.g. User profiles, Categories, Subcategories, etc.) will return a custom 404 page.
- Posting content that is bannable (to be defined) will show an error to the user posting it through a popup or message.