System Design

CSCC01 Summer 2022

Final Project

Team: Amorr Yes

Lingfei Cai, Zhenyuan Xiang, Changhao Wang,

Zhongyu Liu, Pei Zhang

Contents

CRC Cards	3
System Architecture	10
System Decomposition	12

CRC Cards

NavBar	
Parent Class: App Subclass: None	
Responsibilities: Enables users to navigate to one of the 3 primary pages of the website, or to log out. Navigation button availability depends on the login state of the user. It always stays at the top of the webpage and does not interfere with the main body.	Collaborators:

Body	
Parent Class: App Subclass: None	
Responsibilities: Serves as a container for the main body of each webpage. Allows redirection to different pages without actually reloading the entire page.	Collaborators: ● SignUp ● Login ● ClientProfile ● ClientIDUpload ● ProviderIDUpload ● ProviderCertificateUpload ● ProviderPosts

	• DetailedPost • MainPage

App	
Parent Class: None Subclass: None	
Responsibilities: Runs the frontend application. Simply combines and displays NavBar and Body.	Collaborators: • NavBar • Body

SignUp	
Parent Class: Body Subclass: None	
Responsibilities: Displays the signup page. Allows users to enter signup information and submit it to the server for account creation.	Collaborators:

Login	
Parent Class: Body Subclass: None	
Responsibilities: Displays the login page. Allows users to enter login information and submit it to the server for verification.	Collaborators:

ClientProfile	
Parent Class: Body Subclass: None	
Responsibilities: Displays the client profile page. Allows users to view and edit their profile.	Collaborators:

ClientIDUpload	
Parent Class: Body Subclass: None	
Responsibilities: Allows clients to upload and reupload their photo ID to the server for verification purposes. Display the ID if already uploaded.	Collaborators:

ProviderIDUpload	
Parent Class: Body Subclass: None	
Responsibilities: Allows providers to upload and reupload their photo ID to the server for verification purposes. Display the ID if already uploaded.	Collaborators:

ProviderCertificateUpload	
Parent Class: Body Subclass: None	
Responsibilities: Allows providers to upload and reupload their certificate to the server for verification purposes. Display the certificate if already uploaded.	Collaborators:

ProviderPosts	
Parent Class: Body Subclass: None	
Responsibilities: Displays all posts of the service provider in a list. Providers can click on each post	Collaborators:

to see and edit information about the post.	

DetailedPost	
Parent Class: Body Subclass: None	
Responsibilities: Displays detailed information of a post. Providers can also edit any information they want.	Collaborators:

MainPage	
Parent Class: Body Subclass: None	
Responsibilities: Serves as the home page. Customers can search for certain posts using the search bar and the filter on this page. Customers may search by keywords in the titles of the posts and filter the results by distance (e.g. within 5 km) to avoid unwanted wait time or transportation fee.	Collaborators:

Database table schemas

User	
id	integer
first_name	varchar(60)
last_name	varchar(60)
email	varchar(100)
password	varchar(100)
about	text
photoid	varchar(200)
phone	varchar(100)
user_type	varchar(200)
certificate	varchar(200)
categories	text

Post	
id	integer
title	varchar(200)
text	text
start_time	varchar(30)
end_time	varchar(30)
location	varchar(255)
postal_code	varchar(30)
author_id	integer unsigned
price	integer unsigned

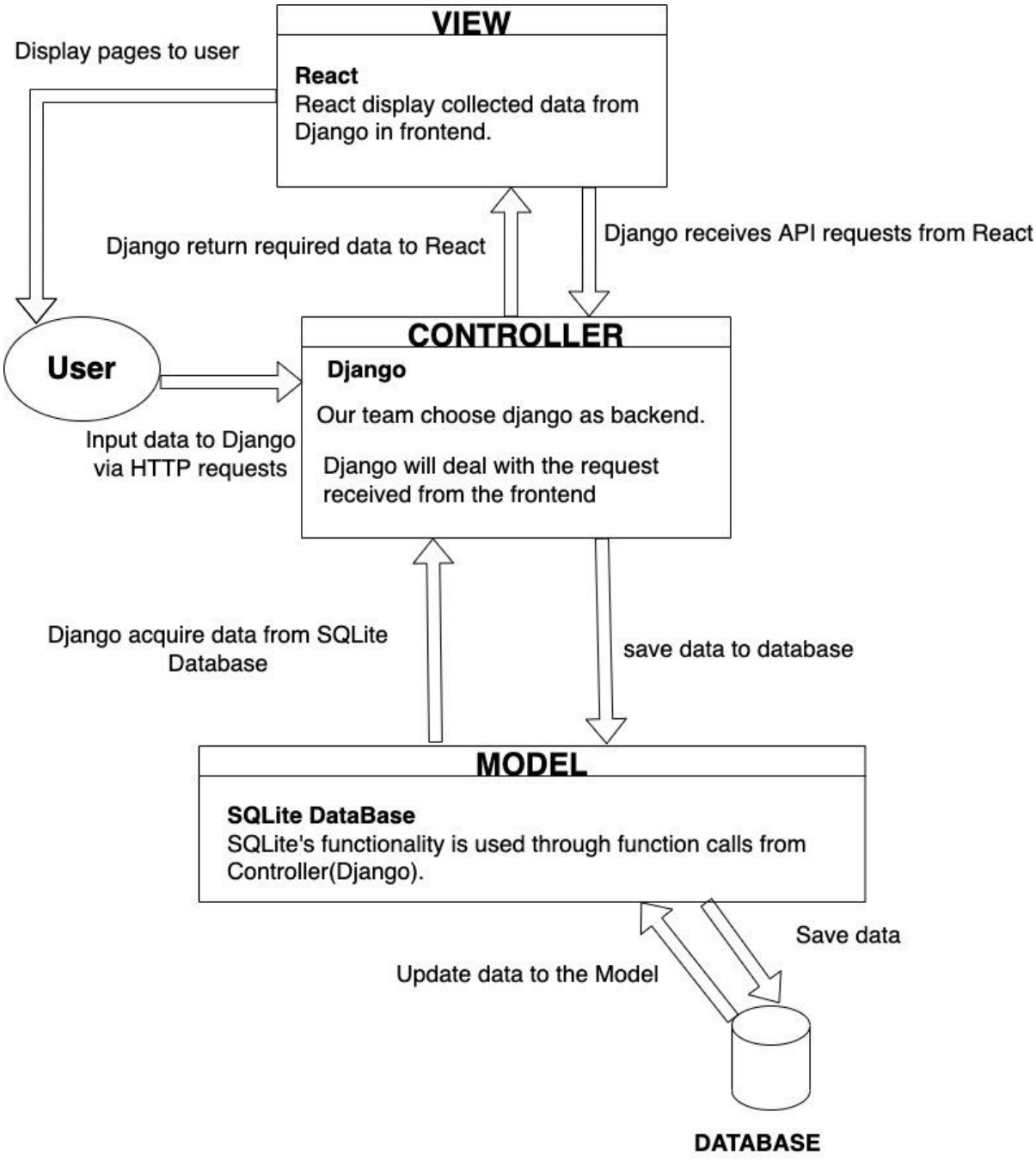
System Architecture

We use MVC architecture for our project. Architecture Description and System Interaction Description are included in the diagram.

Our view section uses Reaction and communicates through http requests to Django. React will display pages to the user. The user will then input data to the frontend. Django, as the controller section, will deal with the request received from the frontend and return required data or save data to the database. We choose SQLite DataBase as our Model part. SQLite's functionality is used through function calls from Controller which enables the backend to store the data.

Example:

For AM-9 Client Search, the user can choose to search keywords and the result can be sorted by distance or price which are displayed in the frontend. Users will input the data to Django. Django, as the controller section, will filter and sort those posts stored in the database which is the Model part..


VIEW
Display pages to user
React
React display collected data from Django in frontend.
Django return required data to React
Django receives API requests from React
User
CONTROLLER
Django
Our team choose django as backend.
Input data to Django via HTTP requests
Django will deal with the request received from the frontend
Django acquire data from SQLite Database
save data to database
MODEL
SQLite DataBase
SQLite's functionality is used through function calls from Controller(Django).
Save data
Update data to the Model
DATABASE

System Decomposition

Frontend:

The user can send api requests through the presentation tier. The frontend is responsible for gathering and displaying information. React will display the signup/login forms and send the information(each max length is 32 letters) as json to the frontend. Before redirecting to other pages, the frontend will send a request to the backend to check whether the user is logging in. If not, the frontend will redirect to the login page.

Backend

After Django receives the requests, it deals with the request and then query the SQL database. The application layer will then send http requests to the frontend so that it can display information to the user. If the backend receives an empty username, it will return status failure and false. Other errors like network failure will be thrown to exception.

SQL database

SQLdatabase will store the data it received and send back the data to Django. We choose to add salt into a MD5 which can help protect our information.