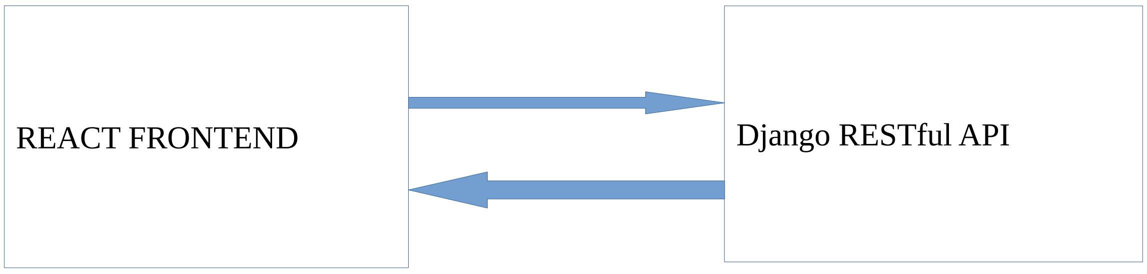
U-CAMPAS BLOG

The blog is a decoupled system with a separate frontend system consuming the django API.



API

The Django API hosts and manages the Postgres Database. The following is the structure of the database

User Table/Model: This table stores all the users of the blog. A user has an ID, username and email. All these fields are unique. Each user has a password that is used for authentication when interacting with the blog system.

Posts Table/Model: This table has all posts or blogs posted by the user. Each instance of the Post has a foreign key to the user table. This identifies the owner of the post. Other fields of this model are the date and time the post was made, the actual content of the post, title and a short excerpt of the post.

Functions on the API

The following are the functionalities of the API:

User registration: A user submits the username, password and email on the frontend. The frontend makes a post request to the API's registration function. The details are stored on the database and the password is encrypted before being stored to prevent unauthorized access by system admins.
A feedback from the API is sent to the frontend to notify the user if the registration was successful.
If the registration was successful, the user is redirected to the blogs home page which consists of a list of all blogs posted.
In case of an error, the error message is shown to the user and a possible solution to the error.

Viewing all Posts: This is a get request made to the API. It fetches all posts and sends a list of all posts to the frontend. The React system then loops and renders all posts on the user's home page.

Creating a Post: Users are allowed to create the posts. On the frontend, a form is displayed to the user. The user provides the excerpt, title and content of the post. On submit, a post request is made to the database. The user's username or id is sent as a parameter on the post request. This is to identify the user who is creating the post. Once the request is made, the post is recorded on the database, the current time and date is read and filled on the post's date time field and the user who has made the request is made the owner of the post. This is done by creating a foreign key to the user table.

Updating a post: On the frontend, a get request for all posts is made to the backend API. The response data is a list of all posts with all the fields and the owner or user who created the post. The frontend then renders these posts by looping using the map method of javascript. During looping, an if statement checks if the current user is the owner of the post being rendered. If the current logged in user owns the post, an update and delete button is displayed. The update button displays a form that enables the user to update the post. The form is pre-filled with current details of the post. The user can then change the details of the post and submit. On submit, a put request is made to the API. The API then fetches the post that is to be updated and updates the fields with the request data.

Deleting a Post: On the frontend, a delete button is displayed if the current user owns a post. If the button is clicked, the id of the post and user is sent to the API. The function first confirms if the user deleting the post actually owns the post. If this condition is True, then the post is deleted and a success message is sent to the user.

Viewing a User’s posts: If a user wants to only view posts from one user, a get request containing the user’s id is sent to the API. The API then fetches all posts owned by the user. A list of all the user’s posts is sent to the frontend and React renders all these posts through a loop. This is also the same case if the logged-in user wants to view his/her posts. The user’s id is sent to the API and a fetch request of all posts owned by the user are sent to the frontend.