10-create-login-backend

Assignees: Matthew Dawson, Samuel Ivuerah
Task: "So create the API backend that react will call. Eg, login, so email and password verification."
Author: Samuel Ivuerah
Link:
https://docs.google.com/document/d/1qjE2kjMd8bSA_M7QX7tuUvFluIXl0Q_ZBPQow0rELls/edit?usp=sharing

Designing and Planning

- Based on the database user models implemented in Issue #2, we will create an API backend to handle login and logout functionality.
- We are using the in-built “AbstractUser” to make our ‘User’ table; thus, the possible fields are
    - id
    - first_name
    - last_name
    - username
    - email
    - password
    - is_superuser
    - is_staff
    - is_active
    - last_login
    - date_joined
    - (and also any custom field we add)
- Since we are focusing on logging (and hence, logging out), the following fields will be focused on:
    - username
    - password
- Since the models haven’t been set up yet, I will just implement a username & password system. (Django’s default user model only uses username and password authentication)

Serialisers

- Serialisers are used in Django’s REST framework to convert Django model instances, such as our ‘User’ class, into Python data types. This makes it easier for said types to be rendered into other types, like JSON, which React uses.

- Deserialisation is possible, meaning we can parse simple data types back through and convert them into complex ones.
- We must validate the incoming data when taking in inputs, such as the email and password fields.

Pseudocode

- To make serialisers for this task, I will create a new file in the API app named ‘serializers.py.’

```
# UserAPI/serializers.py
import necessary modules from Django's REST framework and Django's
authentication system

User = Get the user model from the auth system

class UserSerializer inherits ModelSerializer # from REST framework
      class Meta
            model = User
            fields = ['username', 'email', 'password']
      endclass
endclass

class UserLoginSerializer inherits Serializer # from REST framework
      username = serializers.CharField()
      password = serializers.CharField()

      function auth_user(cleaned_data)
            user = authenticate(cleaned_data['username'],
cleaned_data['password']) # authenticate is a function of the auth system
that will return the user if the credentials are valid

            if user:
                  return user
            else:
                  raise validation error # Incorrect credentials
```

- UserSerializer - This class will serialise and deserialise the User model. It will only operate on the ‘username’, ‘email’, and ‘password’ fields. We can update it for a profile page later, but it won’t be used as of this sprint. e.g. placeholder
- UserLoginSerializer - This class will validate the user’s username and password. It does this using the function ‘auth_user’, which takes validated data and checks whether the credentials are valid. If they are, it will return the user. Otherwise, it will raise an error.

- The validation above will occur in ‘validators.py’ and be used in ‘views.py’. It will be pretty basic for now.

Authentication

- When it comes to authentication, a method can be ‘Stateful’ or ‘Stateless’:
    - Stateful —The server stores information about the client’s state. Upon login, a session is created to authenticate a request from the client. The server recalls session data from memory or in a database. The session data is used to authenticate requests from the same client.
    - Stateless — The server stores no information about the client’s state. For the server to authenticate a request from the client, each request must include the authentication information. The server handles authentication on a per-request basis, so it doesn’t need to remember previous requests.
- There are many ways to handle authentication for a web application:

Method	**Benefits**	**Drawbacks**
Token-Based Authentication - A stateless method: Tokens act as a lookup to server-side data. The token needs to be stored client-side.	● It is simple and easy to implement (as it doesn’t require cookies). ● Scalable and easier for distributed systems. ● It’s stateless; therefore, it’s scalable and easier to use in distributed systems.	● Tokens are stored client-side - if a token is leaked, it would be easy to impersonate someone.
Session-Based Authentication - A stateful method: The server creates and tracks user sessions, and the client stores the session IDs in cookies.	● Simple and easy to implement (although it requires cookies, Django supports it. ● It’s stateful; therefore, the security of the session is up to us.	● It could be more scalable for large applications and distributed systems. ● Users can block cookies, making them impossible to use unless enabled by the user.
JSON Web Tokens (JWT) - a stateless authentication method that uses self-contained tokens — the token itself contains user data.	● Additional user information can be embedded. ● It’s stateless; therefore, it’s scalable and easier to use in distributed systems.	● It is more complex to implement than Session-Based Authentication. ● Token size can become large, endangering space constraints for client-side storage.
Third-Party Authentication Services - Services that	● It removes the complexity and saves	● It can be expensive. ● Unforeseen downtime -

Method	**Benefits**	**Drawbacks**
provide authentication, e.g. Firebase, Auth0, etc.	time. ● It provides additional features, e.g. MFA - Multi-factor authentication, password recovery, etc.	the service provider reserves the right to close for maintenance ● Inconvenient policies

I have chosen to use session-based authentication:

- Statefulness: It allows the server to keep track of the user's state, which can benefit applications where user context is essential.
- Built-in Support: Django, the framework we're using, supports session-based authentication out of the box, making it easier to implement.
- Simplicity: It is straightforward to understand and use, as it follows the traditional model of user sessions in web applications.
- Security: Django's session framework provides a secure and flexible system for managing user sessions, with features like session expiration and cookie handling.
- Price: It's free c:

Pseudocode

- To attempt a session-based authentication system, we can use the Django REST Framework to help make an API backend. We will use CBVs - Class-based views - as they make our backend more flexible.

```
# UserAPI/views.py
import necessary modules from Django's REST framework and Django's
authentication system
import serializers.py
import validators.py

class UserLogin inherits APIView # from REST framework
      function post(request)
            data = data from the request
            assert email_validator(data) # validate the email
            assert password_validator(data) # validate the password

            serializer = UserLoginSerializer(data)
            if serializer.is_valid():
                  user = serializer.auth_user(serializer.validated_data)
                  login(request, user) # make this user persistent, e.g.
create session id, cookie, etc.
                  return Response(status.HTTP_200_OK)
            else:
                  return Response(status.HTTP_400_BAD_REQUEST)
      endfunction
endclass

class UserLogout inherits APIView # from REST framework
      function post(request)
            logout(request) # remove session id
            return Response(status.HTTP_200_OK)
      endfunction
endclass
```

- UserLogin - This class will handle login requests, making sure to validate the entries, deserialise, giving the client a session ID and cookie when provided a valid credentials. Also, it will give an HTTP response depending on the success or failure of attempting to authorise the user.
- UserLogout - This class will handle logout requests by removing the session ID and giving an HTTP response.

API Endpoints

- To make use of the API backend, the frontend will need to send post requests to:
    - http://localhost:8000/api/auth/login/ - for logging in
    - http://localhost:8000/api/auth/logout/ - for logging out