18-account-registration-backend

Assignees: Samuel Ivuerah
Task: “Account registration backend”
Author: Samuel Ivuerah
Link:
https://docs.google.com/document/d/1cbm6yxeHJgw-1ND6lJzFXghYiFWVPuO1-GDzJ1Uu78g/edit?usp=sharing

Designing and Planning

Business Rules (User Accounts)

- To create an account, a user must provide a username, email, and password that adhere to the following rules:

1. The username must be at least five characters long.
2. The username must be unique across all users.
3. The email must be valid and unique across all users.
4. The password must be at least eight characters long.
5. The password must contain at least one uppercase letter, one lowercase letter, and one digit.

- The business rules for user accounts are designed to ensure a user’s account is easily identifiable yet secure:
    - Enforcing a minimum length for usernames and passwords fosters simplicity and memorability.
    - Uniqueness in email and usernames helps us maintain a clean database.
    - Character requirements safeguard against unauthorised access but ensure it’s not too complex for the user.

Validation

- We must validate the information a user will input to allow account registration and ensure data integrity/authenticity,

Pseudocode

- To validate the incoming information, we will have to expand upon the current way our API handles validation:

```
# UserAPI/validators.py
```

```
class UsernameValidator:
      function validate(username)
            Strip everything but the username
            if username == blank:
                  throw error "Username field cannot be empty."
            else:
                  return True

class EmailValidator:
      function validate(email)
            Strip everything but the email
            if email == blank:
                  throw error "Email field cannot be empty."
            else:
                  return True

class PasswordValidator:
      function validate(password)
            Strip everything but the password
            if password == blank:
                  throw error "Password field cannot be empty."
            else:
                  return True
```

- We have adapted the original validators by separating them into their respective classes. This allows us to inherit the base validation for each of these fields (that they cannot be empty) and add further validation to encompass our business rules:

```
# UserAPI/business_validators.py
import validators.py

User = Get the user model from the auth system

class UsernameBusinessValidator inherits UsernameValidator:
      function validate(username)
            super.validate(username)

            # Business rule 1
            if the length of the username < 5:
                  throw error "Username must be at least five characters
long."
            # Business rule 2
```

```
            if the database contains the username:
                  throw error "Username already exists."
            return True

class EmailBusinessValidator inherits EmailValidator:
      function validate(email)
            super.validate(username)

            # Business rule 3
            validate the email, ensuring it has an email''s standard
features.
            if the database contains the email:
                  throw error "Email already exists."
            return True

class PasswordBusinessValidator inherits PasswordValidator:
      function validate(password)
            super.validate(password)

            # Business rule 4
            if the length of the password < 8:
                  throw error "Password must be at least 8 characters
long."
            # Business rule 5
            if the password doesn''t contain an uppercase letter:
                  throw error "Password must contain at least one uppercase
letter."
            if the password doesn''t contain a lowercase letter:
                  throw error "Password must contain at least one lowercase
letter."
            if the password doesn''t contain a digit:
                  throw error "Password must contain at least one digit."
            return True
```

- Here, we've extended the base validators to enforce our business rules while keeping the base validation needed for authentication.

Serialiser

- We must make a serialiser to convert the incoming JSON containing the account credentials to the User model.

Pseudocode

```
# UserAPI/serializers.py
...
class UserRegisterSerializer inherits ModelSerializer
...
      function create(cleaned_data):
            username_field = The username field from the User model

            credentials = {
                  username_field: cleaned_data['username'],
                  'email': cleaned_data['email'],
                  'password': cleaned_data['password']
            }

            # Creating the user.
            user = use Django auth system to create a user using
credentials
            Save the user to the database
            return user
```

- Our serialiser will take in the credentials after they've been cleaned (validated), create a user, and save the user to our database.

Registration

- We will have to make a view class and function to handle all the actions necessary for registration.

Pseudocode

```
# UserAPI/views.py
import necessary modules from Django''s REST framework
...
import business_validators.py
...
class UserRegister inherits APIView:
```

```
# Allow all users to access this view.
...
function post(request):
      if the user is already logged in:
            return Response("User is already authenticated.",
status.HTTP_400_BAD_REQUEST) # The user is already logged in; hence, does
not need to register.

      data = the data from the request

      # Validating the username, email, and password.
      assert UsernameBusinessValidator(data)
      assert EmailBusinessValidator(data)
      assert PasswordBusinessValidator(data)
      # Validations will return 400/500 responses in the event of failure

      serializer = UserRegisterSerializer(data)
      if serializer.is_valid():
            serializer.create(serializer.validated_data)
            return Response(status.HTTP_201_CREATED)
      else:
            return Response(status.HTTP_400_BAD_REQUEST)
```

- Here, our view will handle the request to register a new user.

API Endpoints

- To make use of the API backend, the frontend will need to send post requests to:
    - http://localhost:8000/api/auth/register/ - for registration
    - (See Django REST API documentation for more)

Future Plans

- Review Business validators
    - We can add more business rules to our validators such as “Only alphanumeric characters can be used for usernames”. etc