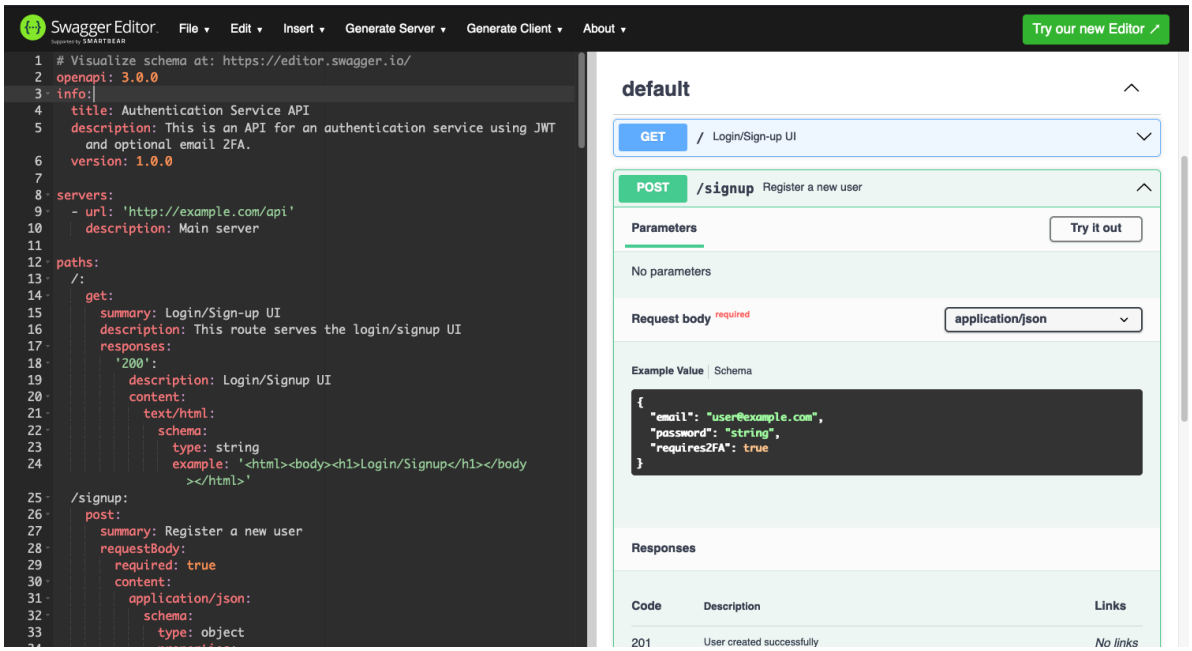
1. Review the log-in route contract
    - Steps:
        1. Open up Swagger Studio and paste the contents of auth-serivce/api_schema.yml into the editor:



            Notice that the log-in route accepts a JSON object:

```
{
  "email": "user@example.com",
  "password": "string",
}
```

            If the JSON object is missing or malformed, a 422 HTTP status code should be sent back.

            If the JSON object contains invalid credentials, a 400 HTTP status code should be sent back.

            Finally, if the JSON object contains credentials that are valid but incorrect, a 401 HTTP status code should be returned.

            We'll start by implementing these known error cases!

2. Add test for 422 case
    - Steps:
        1. Update the ~~post~~_login method in auth-service/tests/api/helpers.rs :

```
//...

impl TestApp {
    //...

    pub async fn ~~post~~_login<Body>(&self, body: &Body) -> reqwest::Response
    where
        Body: serde::Serialize,
    {
        self.http_client
            .post(&format!("{}/login", &self.address))
            .json(body)
```

```
            .send()
            .await
            .expect("Failed to execute request.")
    }
}
```

We are now passing a body argument to the post_login helper method, which is any type that implements Serde's Serialize trait. body is then added to the HTTP request as the JSON body.

2. Add 422 test case to auth-service/tests/api/login.rs :

```
use crate::helpers::{get_random_email, TestApp};

#[tokio::test]
async fn should_return_422_if_malformed_credentials() {
    todo!()
}
```

3. Remove the login_should_return_200 test case from auth-service/tests/api/login.rs. This was a placeholder test case that is no longer relevant.
4. Run tests

```
cargo test
```

Our new test should be failing at this point. Next, we'll make it pass!

3. Update log-in route
   - Steps:
     1. Update auth-service/src/routes/login.rs to only accept the appropriate JSON body.
     2. Run tests

```
cargo test
```

All our tests should now be passing!

4. Add test for 400 case
   - Steps:
     1. Add 400 test case to auth-service/tests/api/login.rs :

```
use auth_service::ErrorResponse;
//...

#[tokio::test]
async fn should_return_400_if_invalid_input() {
    // Call the log-in route with invalid credentials and assert that a
    // 400 HTTP status code is returned along with the appropriate error message.
    todo!()
}
```

Take some time to read through the code. We are

2. Run tests

```
cargo test
```

Our new test should be failing at this point. Next, we'll make it pass!

5. Update log-in route
   - Steps:
     1. Update auth-service/src/routes/login.rs to parse the email and password in the request body. Similar to the sign-up route!
     2. Run tests

```
cargo test
```

All our tests should now be passing!

6. Add test for 401 case
   - Steps:
     1. Add 401 test case to auth-service/tests/api/login.rs :

```
//...

#[tokio::test]
async fn should_return_401_if_incorrect_credentials() {
    // Call the log-in route with incorrect credentials and assert
    // that a 401 HTTP status code is returned along with the appropriate error message.
    todo!()
}
```

     2. Run tests

```
cargo test
```

Our new test should be failing at this point. Next, we'll make it pass!

7. Update log-in route
   - Steps:
     1. Add a new IncorrectCredentials error variant to AuthAPIError in auth-service/src/domain/error.rs
     2. Update the IntoResponse implementation in auth-service/src/lib.rs to account for the IncorrectCredentails error variant.

        StatusCode::UNAUTHORIZED should be returned for the IncorrectCredentails case.

     3. Update auth-service/src/routes/login.rs to check user credentials:

```
//...

pub async fn login(
    State(state): State<AppState>, // New!
    Json(request): Json<LoginRequest>,
) -> Result<impl IntoResponse, AuthAPIError> {
    //...

    let user_store = &state.user_store.read().await;

    // TODO: call `user_store.validate_user` and return
    // `AuthAPIError::IncorrectCredentials` if valudation fails.

    // TODO: call `user_store.get_user`. Return AuthAPIError::IncorrectCredentials if the
operation fails.
    let user = todo!();

    Ok(StatusCode::OK.into_response())
}
```

     4. Run tests

```
cargo test
```

All our tests should now be passing!

8. Test auth service locally

   Steps:

1. Run application through Docker:

   Run these commands in the root project directory:

```
docker compose build
docker compose up
```

2. Use Postman to call the login endpoint with various malformed/invalid/incorrect request bodies and ensure you get the expected HTTP status code back. We are essentially executing the log-in integration tests manually.