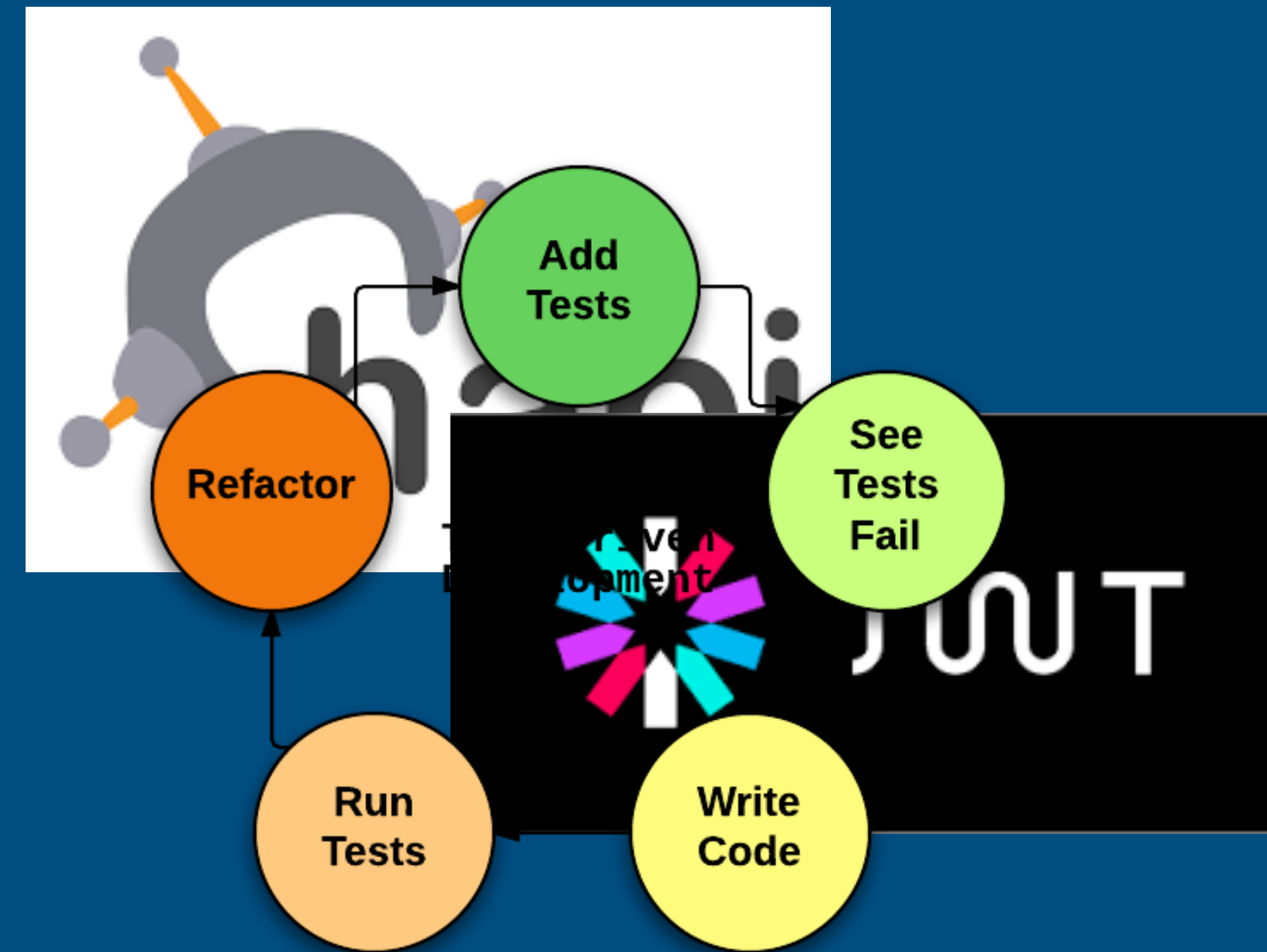
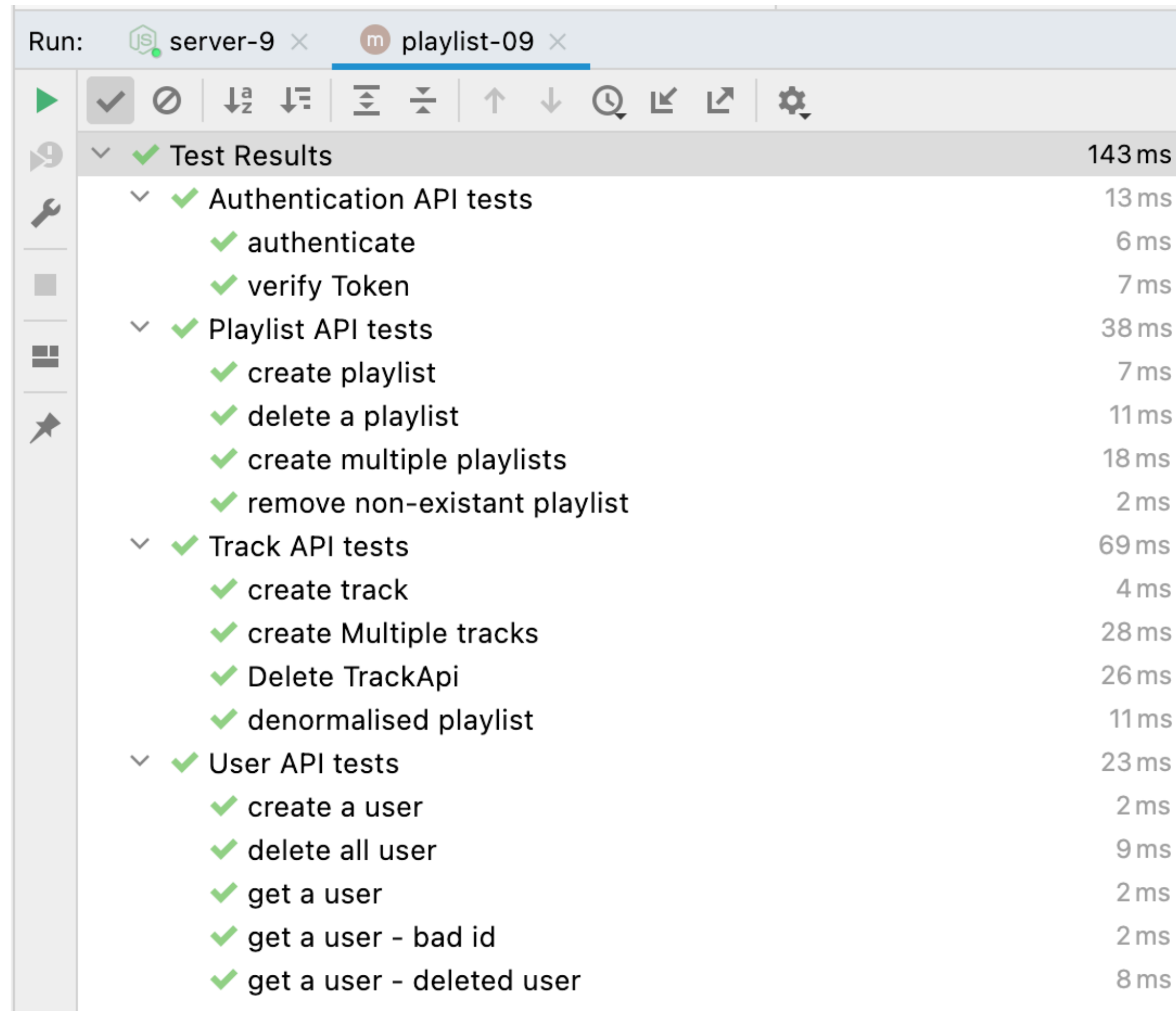


JWT/TDD



Full Stack Web Development

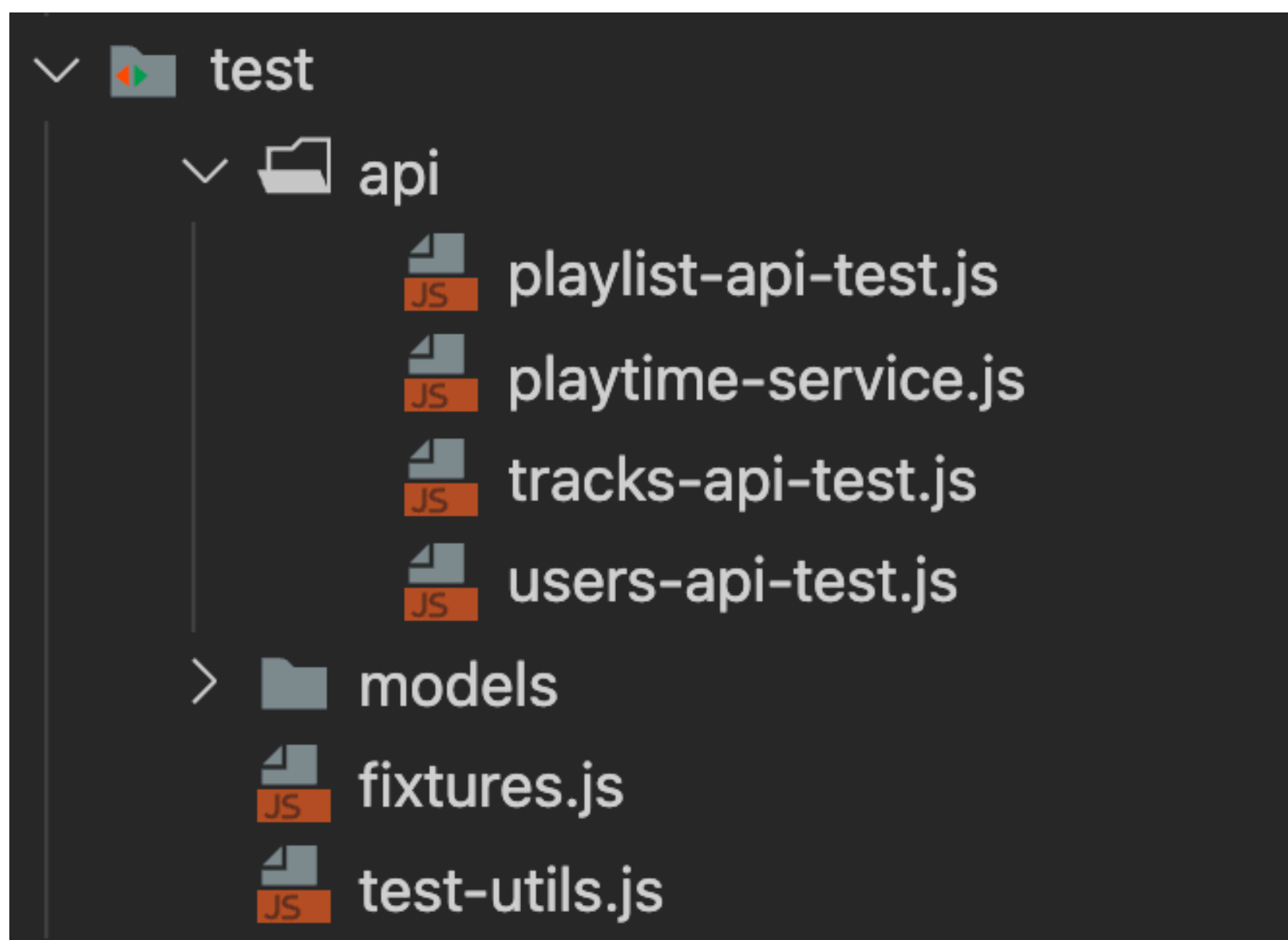
Testing Authenticated Routes



The screenshot shows a Jest test runner interface with a sidebar on the left containing icons for running tests, viewing results, and other functions. The main area displays a tree of test results for a file named 'playlist-09'. All tests are marked with green checkmarks, indicating they passed. The tests are organized into categories: Authentication API tests, Playlist API tests, Track API tests, and User API tests. Each category and individual test has an associated execution time in milliseconds.

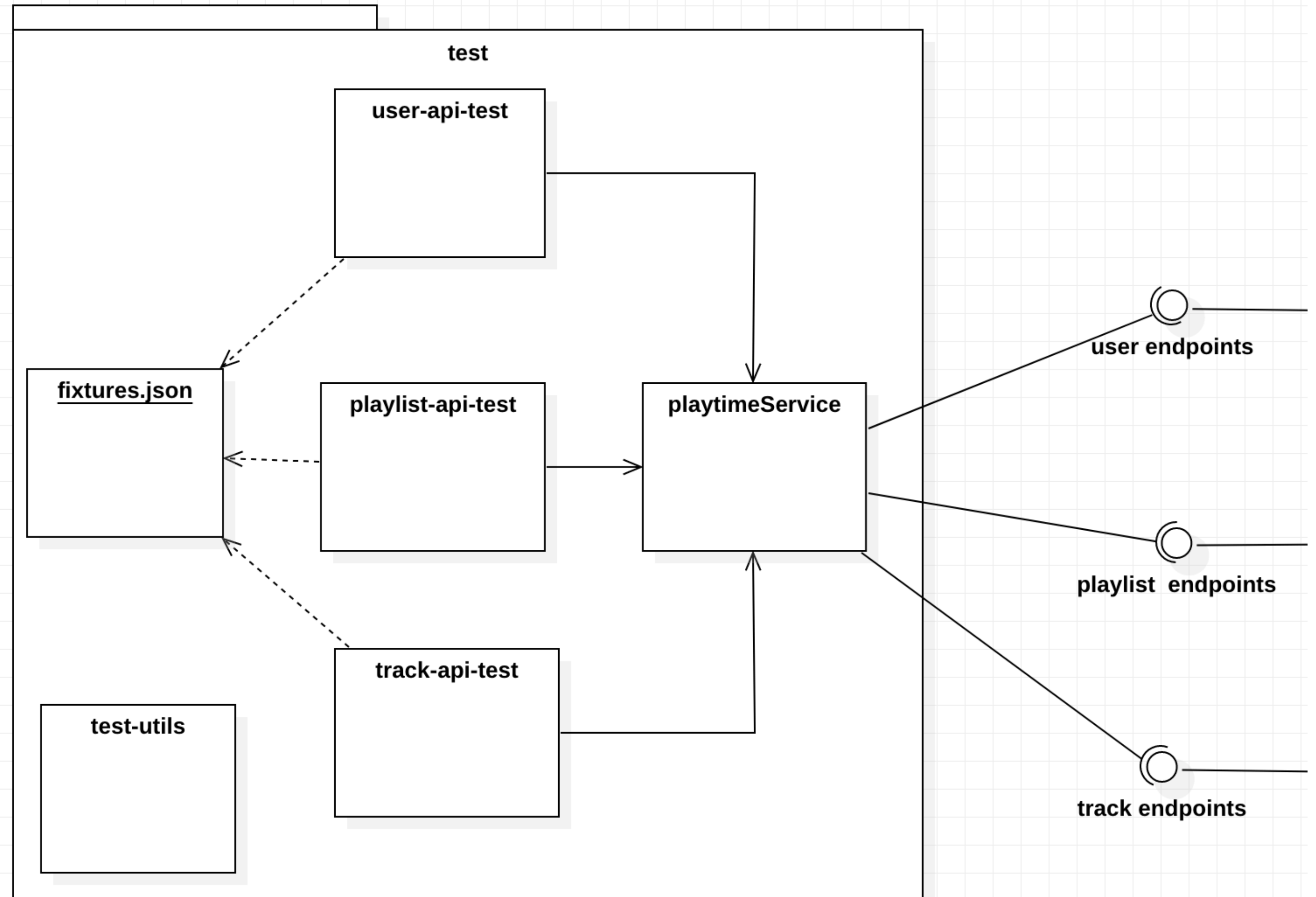
Test Category	Test Name	Execution Time
Test Results		143 ms
Authentication API tests		13 ms
	authenticate	6 ms
	verify Token	7 ms
Playlist API tests		38 ms
	create playlist	7 ms
	delete a playlist	11 ms
	create multiple playlists	18 ms
	remove non-existent playlist	2 ms
Track API tests		69 ms
	create track	4 ms
	create Multiple tracks	28 ms
	Delete TrackApi	26 ms
	denormalised playlist	11 ms
User API tests		23 ms
	create a user	2 ms
	delete all user	9 ms
	get a user	2 ms
	get a user - bad id	2 ms
	get a user - deleted user	8 ms

- All of these tests will fail when annotated with “jwt” strategy
- Do we need to rewrite all tests to retrieve and manage the tokens?



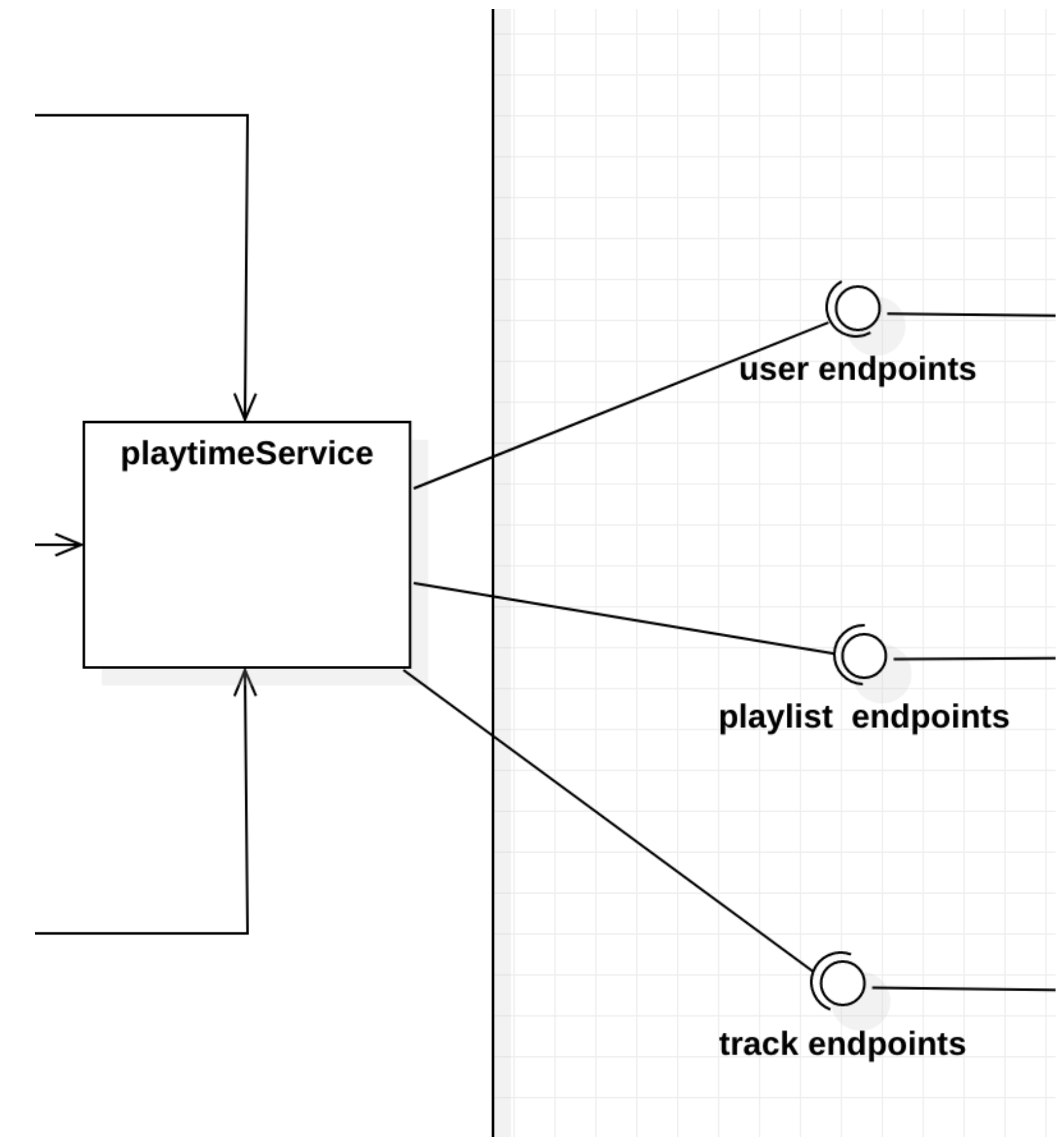
- All API access is via playtimeService
- Logical place to encapsulate authentication strategy

playtimeService



- Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens.
- The name “Bearer authentication” can be understood as “give access to the bearer of this token.”
- The bearer token is generated by the server in response to a login request via the Authenticate route.
- The client must send this token in the Authorization header in order to access API endpoints

Bearer Authentication



authenticate method

- Invokes authenticate route to retrieve token
- Set the token as an “Authorisation” header

playtime-service.js

```
async authenticate(user) {  
  const response = await axios.post(`${this.playtimeUrl}/api/users/authenticate`, user);  
  axios.defaults.headers.common["Authorization"] = "Bearer " + response.data.token;  
  return response.data;  
},
```

- All subsequent requests will have this header
 - => all requests will carry the token, and hence be validated to access the endpoint

clearAuth method

- Reset the Authorisation header
- Equivalent to logging out of the service

```
async clearAuth(user) {  
  axios.defaults.headers.common["Authorization"] = "";  
}
```


Separate test to exercise the authenticate route

Check that the token is
defined

Check that the token
contains the user id + email

auth-api-test.js

```
import { assert } from "chai";
import { playtimeService } from "../playtime-service.js";
import { decodeToken } from "../../src/api/jwt-utils.js";
import { maggie } from "../fixtures.js";

suite("Authentication API tests", function() {

  setup(async function() {
    await playtimeService.deleteAllUsers();
  });

  test("authenticate", async function() {
    const returnedUser = await playtimeService.createUser(maggie);
    const response = await playtimeService.authenticate(maggie);
    assert(response.success);
    assert.isDefined(response.token);
  });

  test("verify Token", async function() {
    const returnedUser = await playtimeService.createUser(maggie);
    const response = await playtimeService.authenticate(maggie);

    const userInfo = decodeToken(response.token);
    assert.equal(userInfo.email, returnedUser.email);
    assert.equal(userInfo.userId, returnedUser._id);
  });
});
```

- Check that a secured route cannot be accessed if user not authenticated

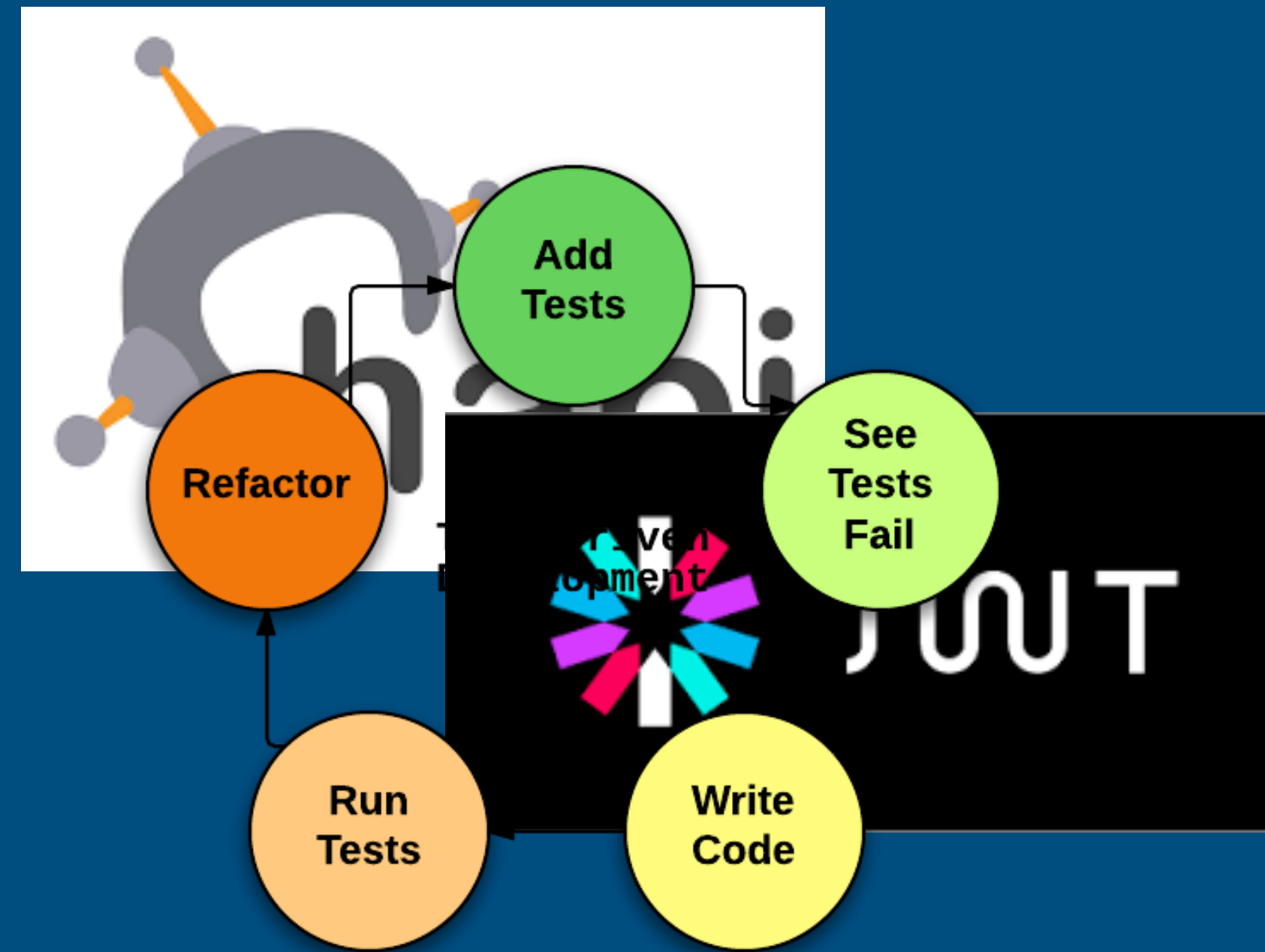
```
test("check Unauthorized", async function() {  
  playtimeService.clearAuth()  
  try {  
    await playtimeService.deleteAllUsers();  
    assert.fail("Route not protected");  
  } catch (error) {  
    assert.equal(error.response.data.statusCode, 401);  
  }  
});
```


Fixture Setup

- Each test needs an authenticated user to set up the fixture
- When we delete all users, we need to create and authenticate again

```
setup(async () => {  
    playtimeService.clearAuth();  
    user = await playtimeService.createUser(maggie);  
    await playtimeService.authenticate(maggie);  
    await playtimeService.deleteAllPlaylists();  
    await playtimeService.deleteAllUsers();  
    user = await playtimeService.createUser(maggie);  
    await playtimeService.authenticate(maggie);  
    mozart.userid = user._id;  
});
```

JWT/TDD



Full Stack Web Development