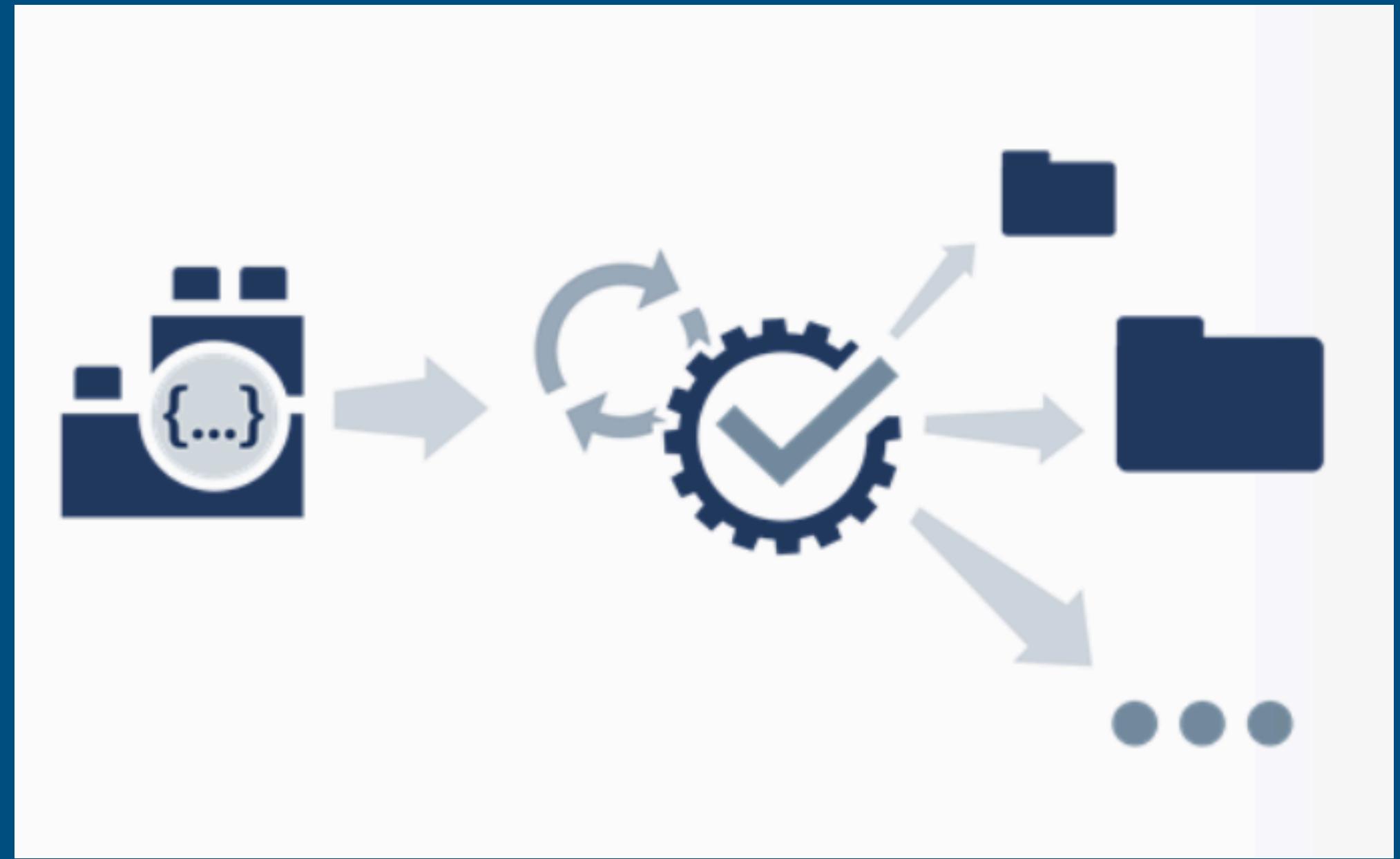


TDD Store



Full Stack Web Development

Test + Test Runner

Fixtures

Test Setup

Test Fail/Fix/Pass cycle

Exhaustive Tests

Test Runner

- Sometimes call xUnit Runners, each language/environment will usually be serviced by a tool to execute tests
- These tools provide a simplified green/red for pass/fail + support simple at a glance report as to overall status



simple, flexible, fun

Mocha is a feature-rich JavaScript test framework running on [Node.js](#) and in the browser, making asynchronous testing *simple* and *fun*. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on [GitHub](#).

<https://mochajs.org>

[Installation](#)

[Getting Started](#)

[Run Cycle Overview](#)

[Detects Multiple Calls to `done\(\)`](#)

[Assertions](#)

[Asynchronous Code](#)

[Synchronous Code](#)

[Arrow Functions](#)

[Hooks](#)

[Pending Tests](#)

[Exclusive Tests](#)

[Inclusive Tests](#)

[Retry Tests](#)

[Dynamically Generating Tests](#)

[Timeouts](#)

[Diffs](#)

[Command-Line Usage](#)

[Parallel Tests](#)

[Root Hook Plugins](#)

[Global Fixtures](#)

[Test Fixture Decision-Tree Wizard Thing](#)

Run: all tests

Test Results

- User API tests
- create a user
- delete all users
- get a user - success
- delete One User - success
- get a user - bad params
- delete One User - fail

AssertionError: expected 3 to equal 2
Expected :2
Actual :3
[Click to see difference](#)

at Context.<anonymous> (<file:///Users/edeleastar/repos/modules/hdip/2021/prj/ent-w>

Event Log

AssertionError: expected 3 to equal 2

Editor Plugins

Examples

Testing Mocha

More Information

- IDEs provide plugins to support Mocha test runners

- Webstorm (above) and VSCode supported

- In addition to a test runner we also need an assertion library
- This specifies how we ‘assert’ predicates within our tests


Chai Assertion Library
Guide
API
Plugins

Chai is a BDD / TDD assertion library for [node](#) and the browser that can be delightfully paired with any javascript testing framework.

Download Chai

[for Node](#) Another platform? [Browser](#) [Rails](#)

The `chai` package is available on npm.

[`\$ npm install chai`](#)
[View Node Guide](#)

[Issues](#) | [Fork on GitHub](#) | [Releases](#) | [Google Group](#) | [Build Status](#)



Getting Started

Learn how to install and use Chai through a series of guided walkthroughs.



API Documentation

Explore the BDD & TDD language specifications for all available assertions.



Plugin Directory

Extend Chai's with additional assertions and vendor integration.

Chai has several interfaces that allow the developer to choose the most comfortable. The chain-capable BDD styles provide an expressive language & readable style, while the TDD assert style provides a more classical feel.

Should

```
chai.should();  
  
foo.should.be.a('string');  
foo.should.equal('bar');  
foo.should.have.lengthOf(3);  
tea.should.have.property('flavors')  
  .with.lengthOf(3);
```

[Visit Should Guide ➔](#)

Expect

```
var expect = chai.expect;  
  
expect(foo).to.be.a('string');  
expect(foo).to.equal('bar');  
expect(foo).to.have.lengthOf(3);  
expect(tea).to.have.property('flavors')  
  .with.lengthOf(3);
```

[Visit Expect Guide ➔](#)

Assert

```
var assert = chai.assert;  
  
assert.typeOf(foo, 'string');  
assert.equal(foo, 'bar');  
assert.lengthOf(foo, 3)  
assert.property(tea, 'flavors');  
assert.lengthOf(tea.flavors, 3);
```

[Visit Assert Guide ➔](#)

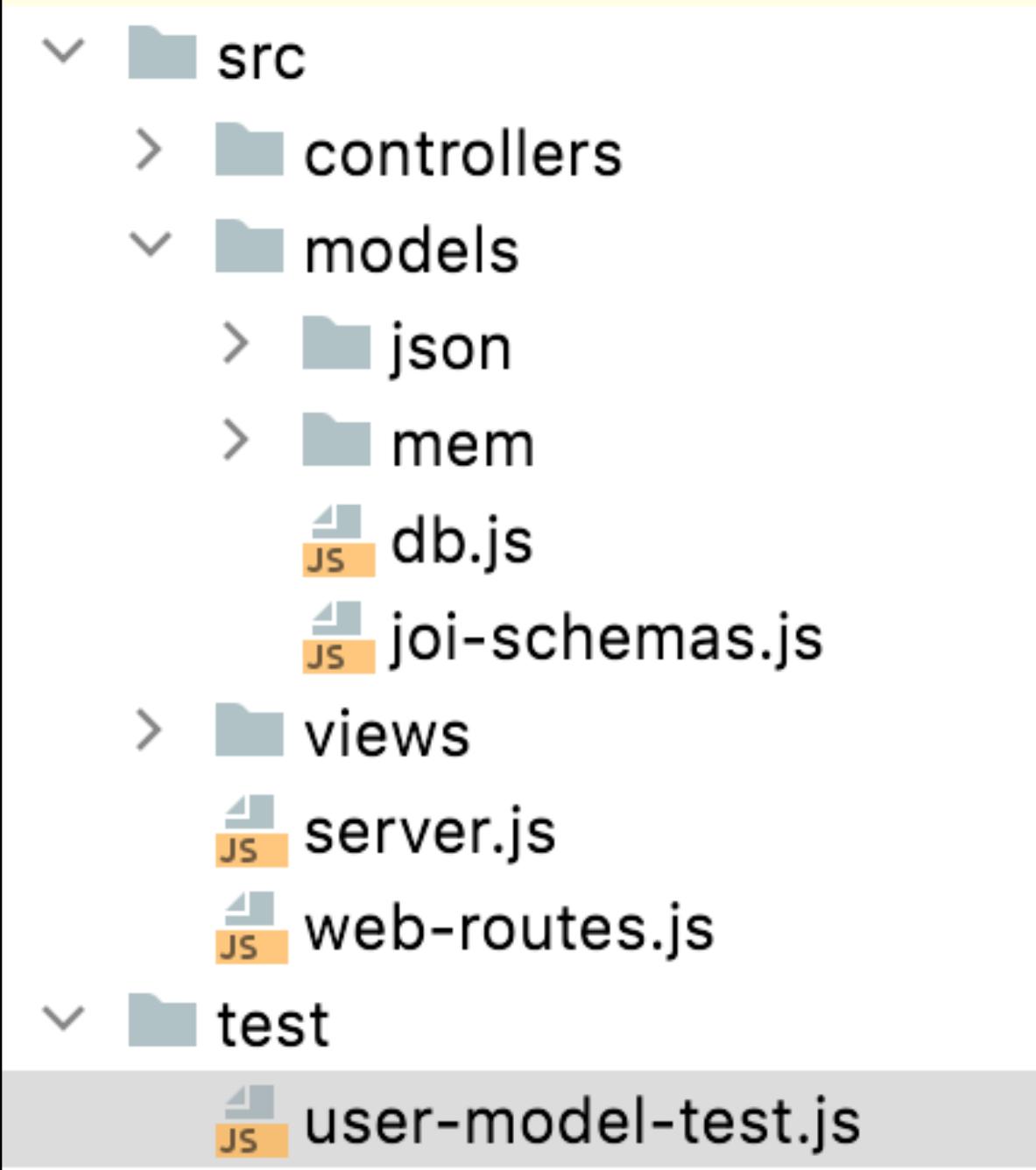


- Multiple styles supported
- We will use the Assert library

- Install as developer dependencies

```
npm install -D mocha  
npm install -D chai
```

```
"devDependencies": {  
    "chai": "^4.3.4",  
    "eslint": "^7.32.0",  
    "eslint-config-airbnb-base": "^15.0.0",  
    "eslint-config-prettier": "^8.3.0",  
    "eslint-plugin-import": "^2.25.3",  
    "mocha": "^9.1.3",  
    "prettier": "^2.5.0"  
}
```



- Introduce a new module - a “test”
- Not part of production code, so we keep in ‘test’ folder

user-model-test.js

```
import { assert } from "chai";
import { db } from "../src/models/db.js";

suite("User API tests", () => {

  const maggie = {
    firstName: "Maggie",
    lastName: "Simpson",
    email: "maggie@simpson.com",
    password: "secret",
  };

  setup(async () => {
    db.init();
  });

  test("create a user", async () => {
    const newUser = await db.userStore.addUser(maggie);
    assert.deepEqual(maggie, newUser);
  });
});
```

System we are going to test

The test suite - Mocha will run all the test in this suite:

user-model-test.js

```
import { assert } from "chai";
import { db } from "../src/models/db.js";

suite("User API tests", () => {

  const maggie = {
    firstName: "Maggie",
    lastName: "Simpson",
    email: "maggie@simpson.com",
    password: "secret",
  };

  setup(async () => {
    db.init();
  });

  test("create a user", async () => {
    const newUser = await db.userStore.addUser(maggie);
    assert.deepEqual(maggie, newUser)
  });
});
```

user-model-test.js

Some test data
(fixture)

setup function to
be run before each
test

A single test

```
import { assert } from "chai";
import { db } from "../src/models/db.js";

suite("User API tests", () => {

    const maggie = {
        firstName: "Maggie",
        lastName: "Simpson",
        email: "maggie@simpson.com",
        password: "secret",
    };

    setup(async () => {
        db.init();
    });

    test("create a user", async () => {
        const newUser = await db.userStore.addUser(maggie);
        assert.deepEqual(maggie, newUser)
    });
});
```

Mocha Test Runner (WebStorm)



callback for suite() > callback for test()

Run: all tests

Tests passed: 1 of 1 test – 2 ms

/usr/local/bin/node /Users/edeleastar/repos/modules/hdip/2021/prj/ent-web/scratch/playtime-0.4.0/node_modules/mocha/b

Test Results

- User API tests
 - create a user

2ms 2ms 2ms

Favorites

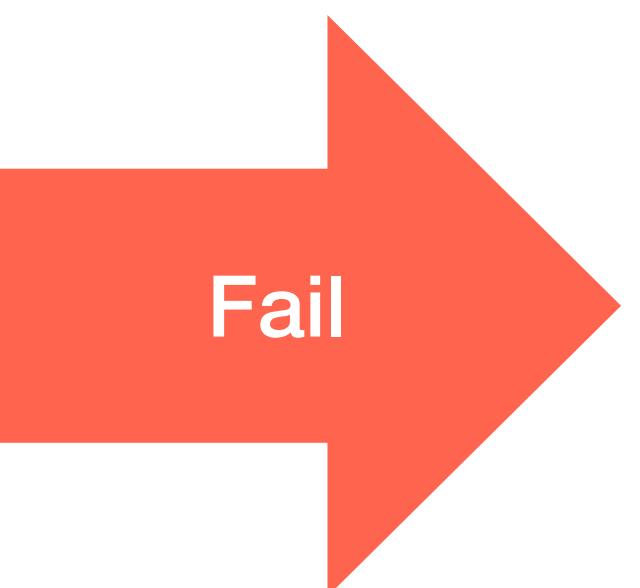
npm

Tests passed: 1

Run TODO Problems Terminal

Tests passed: 1 (moments ago)

19:35 LF UTF-8 2 spaces Event Log



callback for suite() > callback for test()

Run: all tests

Tests failed: 1, passed: 1 of 2 tests – 2 ms

AssertionError: expected 4 to equal 3

Expected :3

Actual :4

<Click to see difference>

at Context.<anonymous> (<file:///Users/edeleastar/repos/modules/hdip/2021/prj/ent-web/scratch/playtime-0.4.0/test/user-mode>

Test Results

- User API tests
 - create a user
 - delete all users

2ms 2ms 1ms 1ms

Run TODO Problems Terminal

Tests failed: 1, passed: 1

Tests failed: 1, passed: 1 (moments ago)

9:8 LF UTF-8 2 spaces Event Log

Mocha Test Runner (VSCode)

The screenshot shows the Visual Studio Code (VSCode) interface with the following components:

- Left Sidebar:** Icons for Testing, Search, Find, Snippets, Issues, Pull Requests, and Settings.
- Testing View:** Shows a tree view with "User API tests" expanded, containing "create a user" which took 5ms. A green checkmark indicates it passed.
- Code Editor:** The file "user-model-test.js" is open, showing Mocha test code. The test "create a user" has failed at line 19, indicated by a red dot.
- Terminal:** Shows a blank terminal window with the prompt "Eamonns-Mac-mini:playtime-0.4.0 edeleastar\$".
- Status Bar:** Displays file statistics: Ln 19, Col 8, Spaces: 2, UTF-8, LF, and language settings: {} JavaScript, ESLint, Prettier, [off].

```
user-model-test.js — playtime-0.4.0

user-model-test.js
test > user-model-test.js > suite("User API tests") callback > test("create a user") callback
1 import { assert } from "chai";
2 import { db } from "../src/models/db.js";
3
4 suite("User API tests", () => {
5   const maggie = {
6     firstName: "Maggie",
7     lastName: "Simpson",
8     email: "maggie@simpson.com",
9     password: "secret",
10    };
11
12 setup(async () => {
13   db.init();
14 });
15
16 test("create a user", async () => {
17   const newUser = await db.userStore.addUser(maggie);
18   assert.isDefined(newUser._id);
19   // assert.equal(1, 2);
20 });
21
22
```

Running test from the command line (Mocha)

```
npm run test
```

package.json

```
"scripts": {  
  "start": "node src/server.js",  
  "dev": "./node_modules/.bin/nodemon --verbose src/server.js",  
  "lint": "./node_modules/.bin/eslint . --ext .js",  
  "test": "./node_modules/mocha/bin/mocha --ui tdd test/**.js"  
}
```

Test script

Mocha Test Runner (Mocha command line)

All Pass

```
npm run test
```

Fail

```
> playtime@0.3.0 test
> ./node_modules/mocha/bin/mocha --ui tdd test/**.js

User API tests
  ✓ create a user

1 passing (4ms)
```

```
User API tests
  1) create a user
```

```
0 passing (4ms)
```

```
1 failing
```

```
  1) User API tests
```

```
    create a user:
```

```
      Assertion Error: expected 1 to equal 2
```

```
        + expected - actual
```

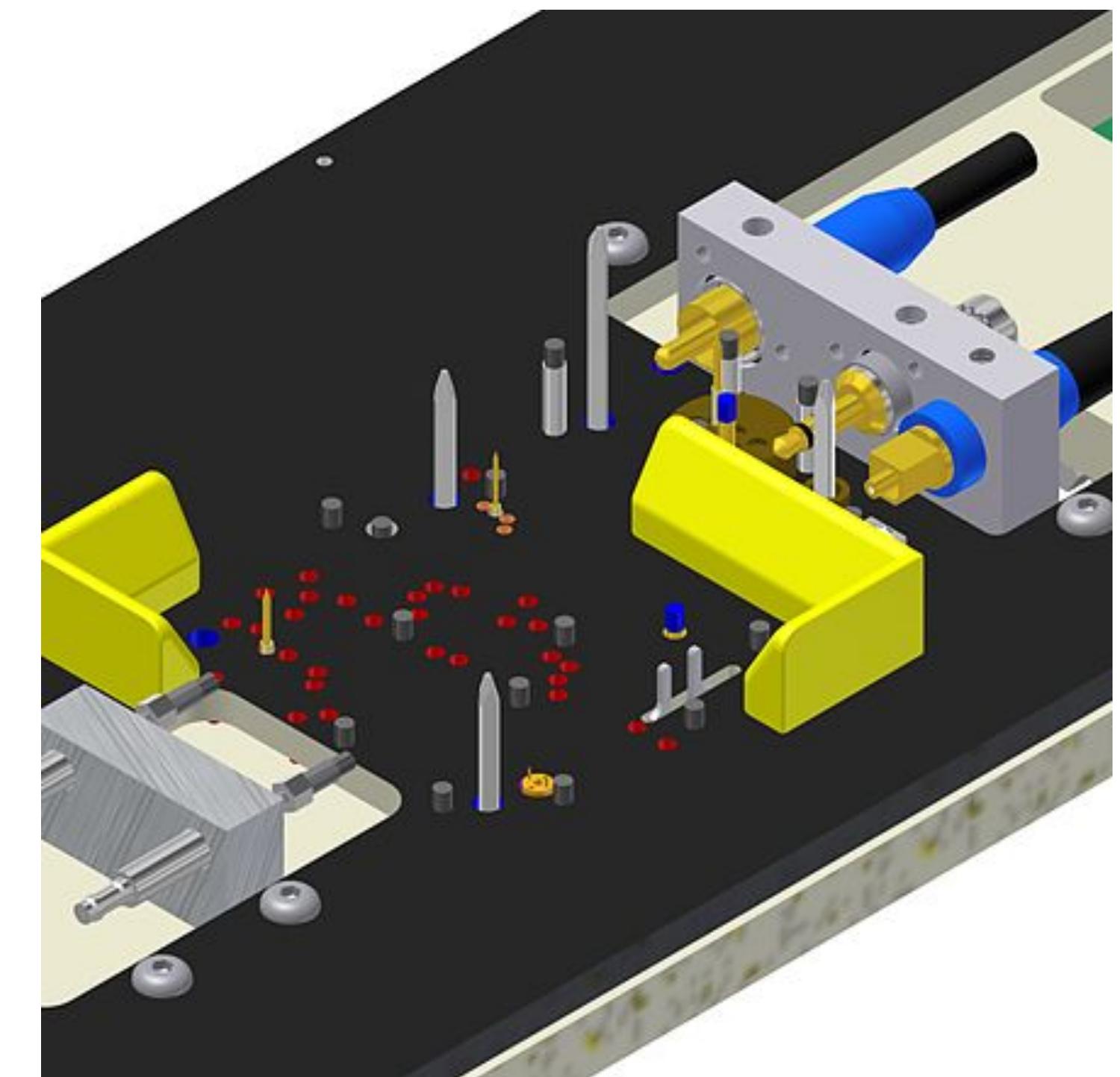
```
-1
```

```
+2
```

Fixtures

A test fixture is an environment used to consistently test some item, device, or piece of software. Test fixtures can be found when testing electronics, software and physical devices.

- Allows for tests to be repeatable since each test is always starting with the same setup.
- Ease test code design by allowing the developer to separate methods into different functions and reuse each function for other tests.
- Test fixtures preconfigure tests into a known initial state instead of working with whatever was left from a previous test run



user-model-test.js

```
import { assert } from "chai";
import { db } from "../src/models/db.js";

suite("User API tests", () => {
  const maggie = {
    firstName: "Maggie",
    lastName: "Simpson",
    email: "maggie@simpson.com",
    password: "secret",
  };

  setup(async () => {
    db.init();
  });

  test("create a user", async () => {
    const newUser = await db.userStore.addUser(maggie);
    assert.deepEqual(maggie, newUser)
  });
});
```

- Move fixtures to separate module

fixtures.js

```
export const maggie = {
  firstName: "Maggie",
  lastName: "Simpson",
  email: "maggie@simpson.com",
  password: "secret",
};
```

```
import { assert } from "chai";
import { db } from "../src/models/db.js";
import { maggie } from "./fixtures.js";

suite("User API tests", () => {

  setup(async () => {
    db.init();
  });

  test("create a user", async () => {
    const newUser = await db.userStore.addUser(maggie);
    assert.equal(newUser, maggie);
  });
});
```

- Import fixtures as needed in tests

Test Setup

```
export const testUsers = [
  {
    firstName: "Homer",
    lastName: "Simpson",
    email: "homer@simpson.com",
    password: "secret",
  },
  {
    firstName: "Marge",
    lastName: "Simpson",
    email: "marge@simpson.com",
    password: "secret",
  },
  {
    firstName: "Bart",
    lastName: "Simpson",
    email: "bart@simpson.com",
    password: "secret",
  },
];
```

Fixtures

user-model-test.js

```
test("delete all users", async () => {
  for (let i = 0; i < testUsers.length; i += 1) {
    // eslint-disable-next-line no-await-in-loop
    await db.userStore.addUser(testUsers[i]);
  }
  let returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, 3);
  await db.userStore.deleteAll();
  returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, 0);
});
```

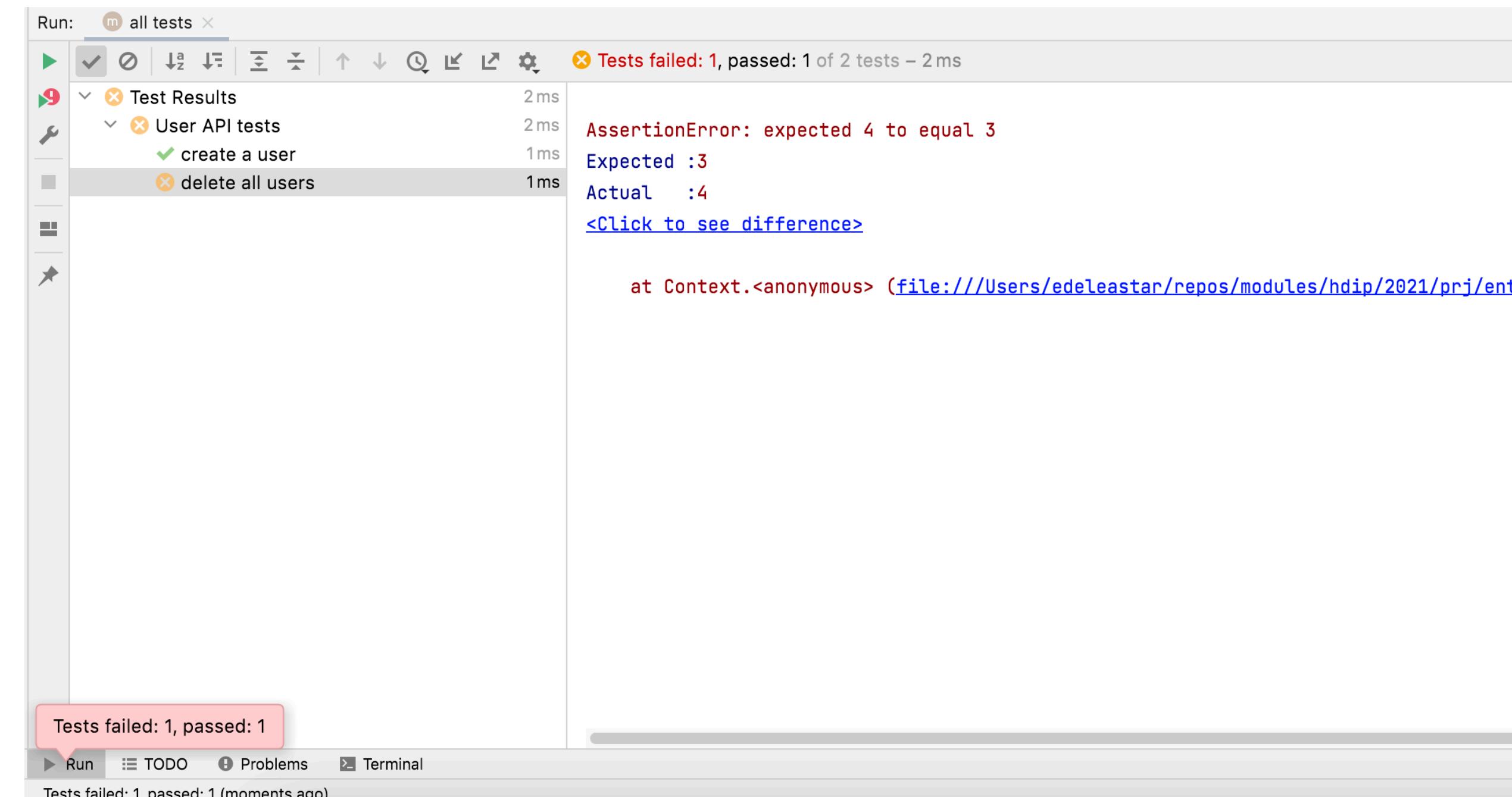
- Add some users
- Verify they have been added
- Delete all users
- Verify that there are 0 users

- The tests will fail

```
user-model-test.js

test("delete all users", async () => {
  for (let i = 0; i < testUsers.length; i += 1) {
    // eslint-disable-next-line no-await-in-loop
    await db.userStore.addUser(testUsers[i]);
  }
  let returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, 3);
  await db.userStore.deleteAll();
  returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, 0);
});

test("create a user", async () => {
  const newUser = await db.userStore.addUser(maggie);
  assert.equal(newUser, maggie);
});
```



- The tests will fail
- Failure cause by two tests interfering with each other
- "create a user" leaves behind a user in the store - causing "delete all users" to fail

user-model-test.js

```
test("delete all users", async () => {
  for (let i = 0; i < testUsers.length; i += 1) {
    // eslint-disable-next-line no-await-in-loop
    await db.userStore.addUser(testUsers[i]);
  }
  let returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, 3);
  await db.userStore.deleteAll();
  returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, 0);
});

test("create a user", async () => {
  const newUser = await db.userStore.addUser(maggie);
  assert.equal(newUser, maggie);
});
```

```
setup(async () => {
  db.init();
  await db.userStore.deleteAll();
});
```

- For each test, delete all users before the test runs

- Increase the rigour of the a test

```
test("get a user - success", async () => {
  const user = await db.userStore.addUser(maggie);
  const returnedUser1 = await db.userStore.getUserById(user._id);
  assert.deepEqual(user, returnedUser1);
  const returnedUser2 = await db.userStore.getUserByEmail(user.email);
  assert.deepEqual(user, returnedUser2);
});
```

.deepEqual(actual, expected, [message])

- @param {Mixed} actual
- @param {Mixed} expected
- @param {String} message

Asserts that `actual` is deeply equal to `expected`.

assert	
fail	
isOk	
isNotOk	
equal	
notEqual	
strictEqual	
notStrictEqual	
deepEqual	
notDeepEqual	
isAbove	
isAtLeast	
isBelow	
isAtMost	
isTrue	
isNotTrue	
isFalse	
isNotFalse	
isNull	

Test Fail/Fix/Pass cycle

```

test("delete One User - success", async () => {
  for (let i = 0; i < testUsers.length; i += 1) {
    // eslint-disable-next-line no-await-in-loop
    testUsers[i] = await db.userStore.addUser(testUsers[i]);
  }

  await db.userStore.deleteUserById(testUsers[0]._id);
  const returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, testUsers.length - 1);
  const deletedUser = await db.userStore.getUserById(testUsers[0]._id);
  assert.isNull(deletedUser);
});

```

- Delete a user test:
 - Add some users
 - Delete one of these users
 - Check if we have 1 user less in the store

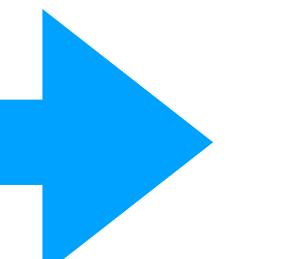
```

User API tests
  ✓ create a user
  ✓ delete all users
  ✓ get a user - success
  1) delete One User - success

  3 passing (21ms)
  1 failing

  1) User API tests
    delete One User - success:
      AssertionError: expected undefined to equal null
      (...playtime-0.4.0/test/user-model-test.js:46:12)

```

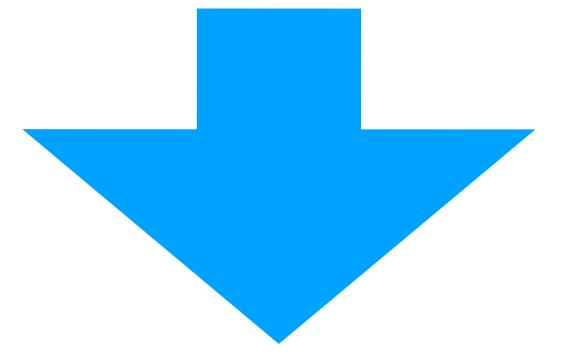


```

assert.isNull(deletedUser);

```

```
async getUserById(id) {  
  await db.read();  
  return db.data.users.find((user) => user._id === id);  
},
```



```
async getUserById(id) {  
  await db.read();  
  let u = db.data.users.find((user) => user._id === id);  
  if (u === undefined) u = null;  
  return u;  
},
```

- Bug in getUserById()
- Find return ‘undefined’, not ‘null’ if it cannot locate object
- We convert this to null, and our test will now pass

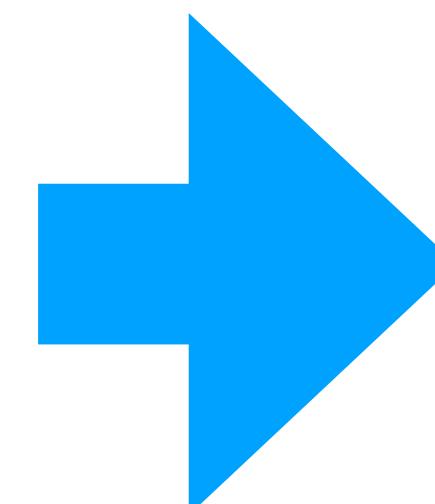
- Deliberately look for users that should not be there.

```
test("get a user - failures", async () => {
  const noUserWithId = await db.userStore.getUserById("123");
  assert.isNull(noUserWithId);
  const noUserWithEmail = await db.userStore.getUserByEmail("no@one.com");
  assert.isNull(noUserWithEmail);
});
```

- Provide invalid or null parameters

```
test("get a user - bad params", async () => {
  let nullUser = await db.userStore.getUserByEmail("");
  assert.isNull(nullUser);
  nullUser = await db.userStore.getUserById("");
  assert.isNull(nullUser);
  nullUser = await db.userStore.getUserById();
  assert.isNull(nullUser);
});
```

- Create some users
- Call delete with an incorrect ID
- Make sure no change in user store
- Error!



```
test("delete One User - fail", async () => {
  for (let i = 0; i < testUsers.length; i += 1) {
    // eslint-disable-next-line no-await-in-loop
    await db.userStore.addUser(testUsers[i]);
  }
  await db.userStore.deleteUserById("bad-id");
  const allUsers = await db.userStore.getAllUsers();
  assert.equal(testUsers.length, allUsers.length);
});
```

Tests failed: 1, passed: 5 of 6 tests – 31ms

Test	Time
create a user	2ms
delete all users	7ms
get a user - success	4ms
delete One User - success	6ms
get a user - bad params	2ms
delete One User - fail	10 ms

AssertionError: expected 3 to equal 2
 Expected :2
 Actual :3
[<Click to see difference>](#)

at Context.<anonymous> (<file:///Users/edeleastar/repos/modules/hdip/2021/prj/ent-w>

Run TODO Problems Terminal Event Log

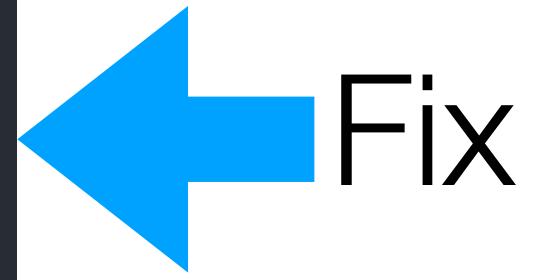
AssertionError: expected 3 to equal 2

65:12 LF UTF-8 2 spaces

```
async deleteUserById(id) {  
  await db.read();  
  const index = db.data.users.findIndex((user) => user._id === id);  
  db.data.users.splice(index, 1);  
  await db.write();  
},
```

- Call to splice removes an object - even if -1 provided!

```
async deleteUserById(id) {  
  await db.read();  
  const index = db.data.users.findIndex((user) => user._id === id);  
  if (index !== -1) db.data.users.splice(index, 1);  
  await db.write();  
},
```



Exhaustive Tests

```
import { assert } from "chai";
import { db } from "../src/models/db.js";
import { maggie, testUsers } from "./fixtures.js";

suite("User API tests", () => {
  setup(async () => {
    db.init();
    await db.userStore.deleteAll();
    for (let i = 0; i < testUsers.length; i += 1) {
      // eslint-disable-next-line no-await-in-loop
      await db.userStore.addUser(testUsers[i]);
    }
  });
  test("create a user", async () => {
    const newUser = await db.userStore.addUser(maggie);
    assert.equal(newUser, maggie);
  });
  test("delete all users", async () => {
    let returnedUsers = await db.userStore.getAllUsers();
    assert.equal(returnedUsers.length, 3);
    await db.userStore.deleteAll();
    returnedUsers = await db.userStore.getAllUsers();
    assert.equal(returnedUsers.length, 0);
  });
});
```

```
test("get a user - success", async () => {
  const user = await db.userStore.addUser(maggie);
  const returnedUser1 = await db.userStore.getUserById(user._id);
  assert.deepEqual(user, returnedUser1);
  const returnedUser2 = await db.userStore.getUserByEmail(user.email);
  assert.deepEqual(user, returnedUser2);
});

test("delete One User - success", async () => {
  await db.userStore.deleteUserById(testUsers[0]._id);
  const returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, testUsers.length - 1);
  const deletedUser = await db.userStore.getUserById(testUsers[0]._id);
  assert.isNull(deletedUser);
});

test("get a user - bad params", async () => {
  assert.isNull(await db.userStore.getUserByEmail(""));
  assert.isNull(await db.userStore.getUserById(""));
  assert.isNull(await db.userStore.getUserById());
});

test("delete One User - fail", async () => {
  await db.userStore.deleteUserById("bad-id");
  const allUsers = await db.userStore.getAllUsers();
  assert.equal(testUsers.length, allUsers.length);
});
```

```
setup(async () => {
  db.init();
  await db.userStore.deleteAll();
  for (let i = 0; i < testUsers.length; i += 1) {
    // eslint-disable-next-line no-await-in-loop
    await db.userStore.addUser(testUsers[i]);
  }
});
```

- Create users in setup
- Tests will assume these users have been created, and can then focus in a specific feature to exercise

```
test("create a user", async () => {
  const newUser = await db.userStore.addUser(maggie);
  assert.equal(newUser, maggie);
});

test("delete all users", async () => {
  let returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, 3);
  await db.userStore.deleteAll();
  returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, 0);
});
```

```
test("get a user - success", async () => {
  const user = await db.userStore.addUser(maggie);
  const returnedUser1 = await db.userStore.getUserById(user._id);
  assert.deepEqual(user, returnedUser1);
  const returnedUser2 = await db.userStore.getUserByEmail(user.email);
  assert.deepEqual(user, returnedUser2);
});

test("delete One User - success", async () => {
  await db.userStore.deleteUserById(testUsers[0]._id);
  const returnedUsers = await db.userStore.getAllUsers();
  assert.equal(returnedUsers.length, testUsers.length - 1);
  const deletedUser = await db.userStore.getUserById(testUsers[0]._id);
  assert.isNull(deletedUser);
});

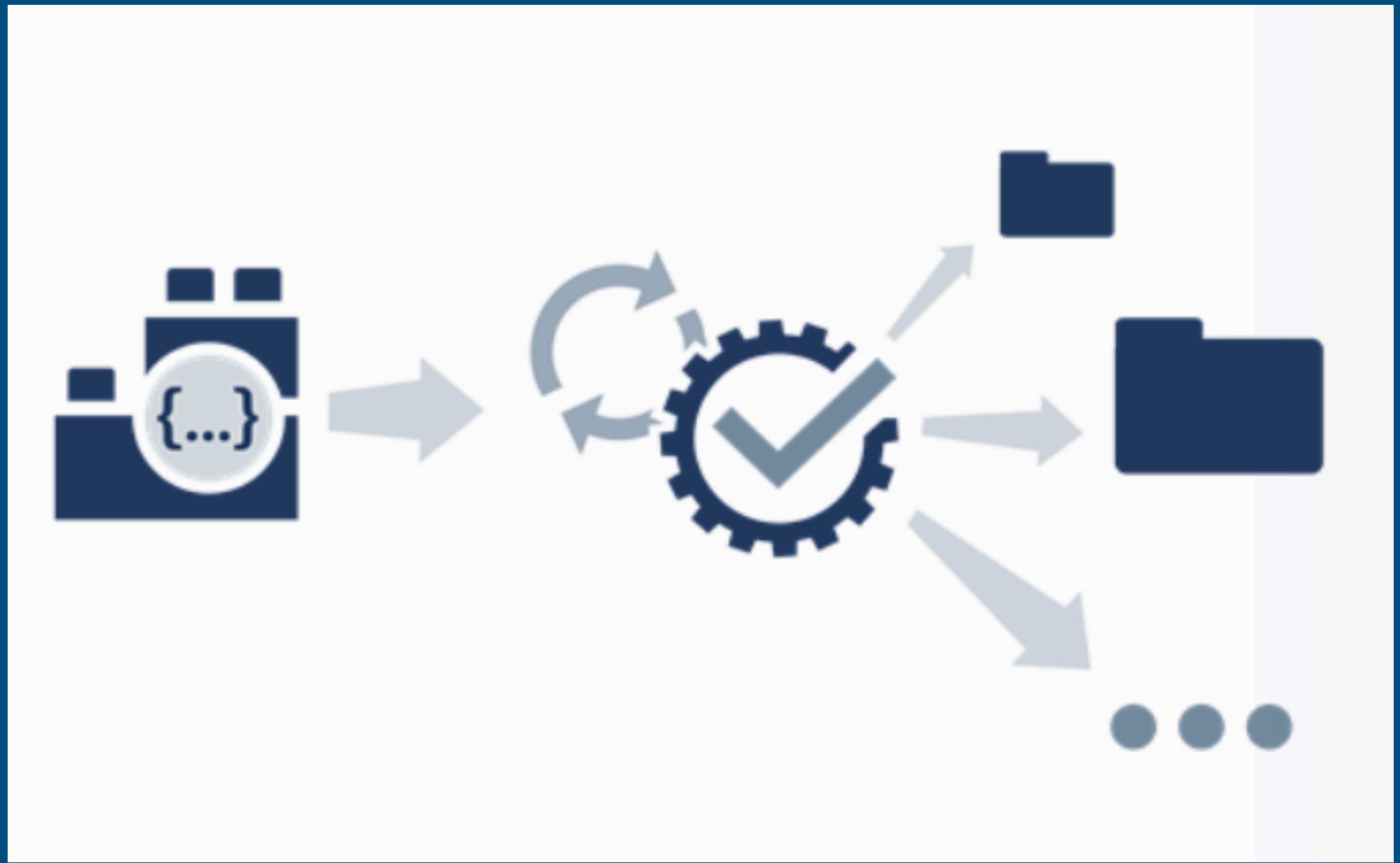
test("get a user - bad params", async () => {
  assert.isNull(await db.userStore.getUserByEmail(""));
  assert.isNull(await db.userStore.getUserById(""));
  assert.isNull(await db.userStore.getUserById());
});

test("delete One User - fail", async () => {
  await db.userStore.deleteUserById("bad-id");
  const allUsers = await db.userStore.getAllUsers();
  assert.equal(testUsers.length, allUsers.length);
});
```

Arrange-Act-Assert

- Arrange inputs and targets. Arrange steps should set up the test case. Does the test require any objects or special settings? Does it need to prep a database?
- Act on the target behaviour. Act steps should cover the main thing to be tested. This could be calling a function or method, Keep actions focused on the target behaviour.
- Assert expected outcomes. Act steps should elicit some sort of response. Assert steps verify the goodness or badness of that response. Sometimes, assertions are as simple as checking numeric or string values. Other times, they may require checking multiple facets of a system. Assertions will ultimately determine if the test passes or fails

TDD Store



Full Stack Web Development