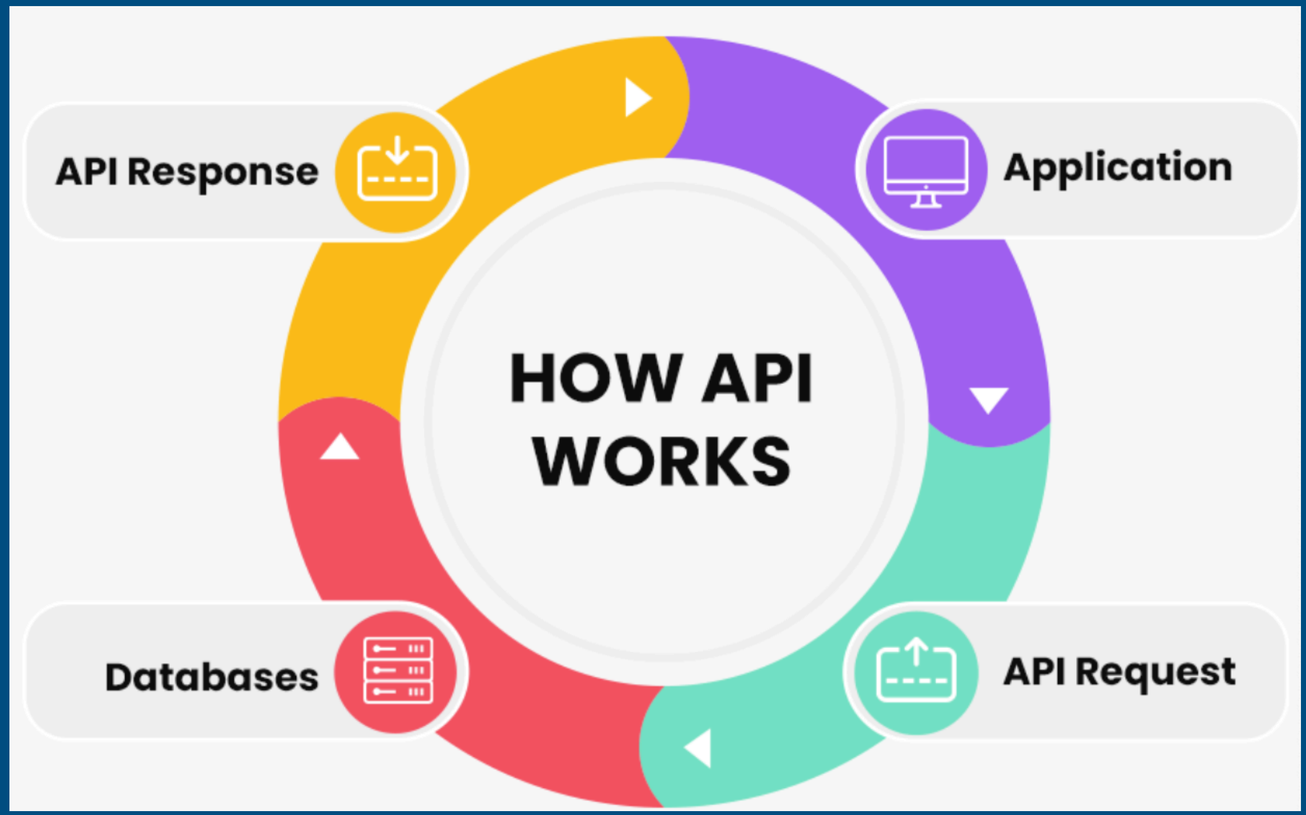
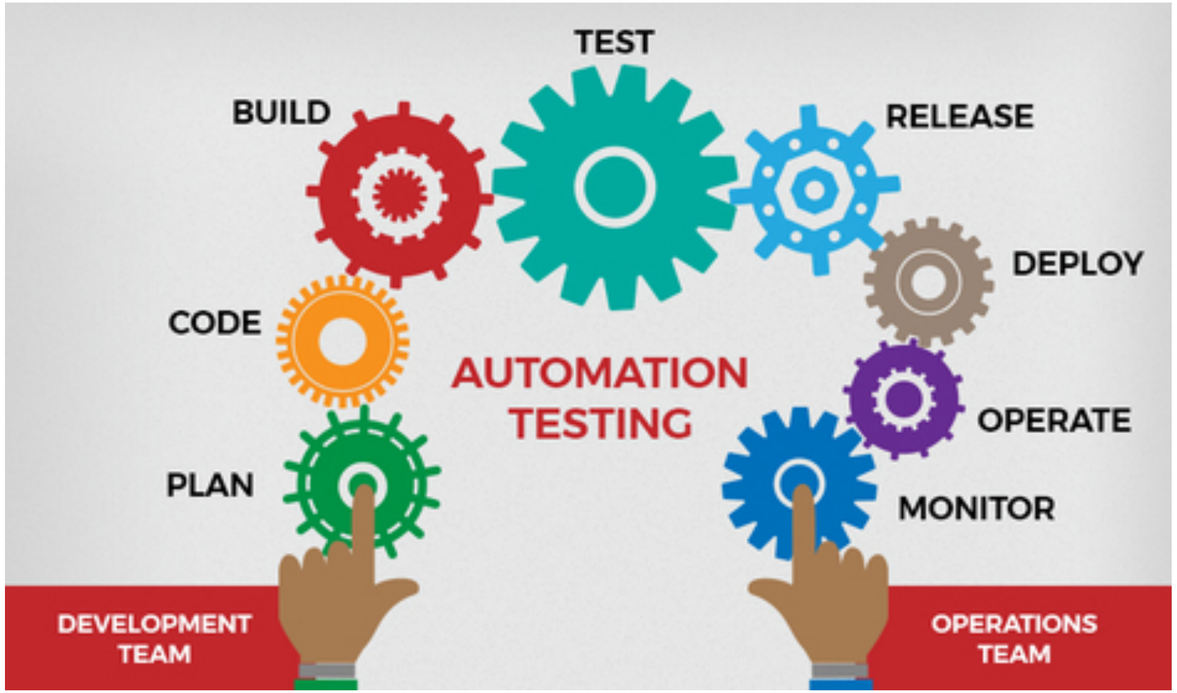
Testing End Points



Full Stack Web Development

Automated Testing

- Using Postman is a useful for exploring APIs
- However, it is very limited tools for developing APIs
- Automated testing, where we manipulate the API as a Javascript client are considerably more useful
- For this, we need xUnit test frameworks and associated test runner tools.



simple, flexible, fun

https://mochajs.org/

http://chaijs.com/

Mocha is a feature-rich JavaScript t
asynchronous testing *simple* and *fu*
reporting, while mapping uncaught

gitter join chat | backers 153 | sponsors 124

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 4.2.0 / 2018-09-25

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

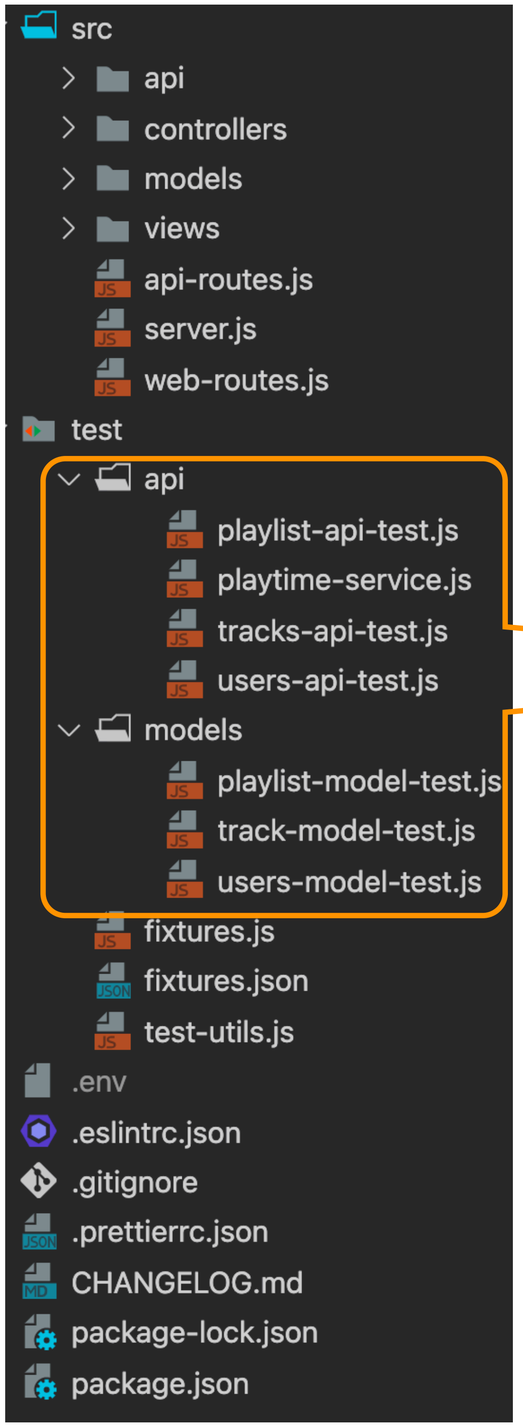
Learn how to install and use Chai through a series of guided walkthroughs.

API Documentation

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

Plugin Directory

Extend Chai's with additional assertions and vendor integration.



Restructure Tests

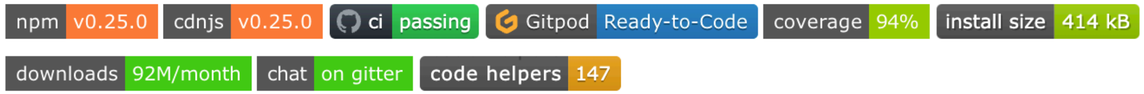
- Refactor existing tests in ‘models’
- Develop new test folder for exercising the API

axios

0.25.0 • Public • Published 13 days ago

Readme | Explore BETA | 1 Dependency | 70,526 Dependents | 53 Versions

axios



Promise based HTTP client for the browser and node.js

> New axios docs website: **click here**

Table of Contents

Install

```
> npm i axios
```

Repository

github.com/axios/axios

Homepage

axios-http.com

Weekly Downloads

24,678,164

Version	License
0.25.0	MIT

Unpacked Size	Total Files
396 kB	49

Issues	Pull Requests
215	83

Last publish

13 days ago

AXIOS

Get Started

Promise based HTTP client for the browser and node.js

Axios is a simple promise based HTTP client for the browser and node.js. Axios provides a simple to use library in a small package with a very extensible interface.

Get Started

View on GitHub

```
import axios from "axios";
axios.get('/users')
 .then(res => {
  console.log(res.data);
 });
```

https://axios-http.com/

Getting Started

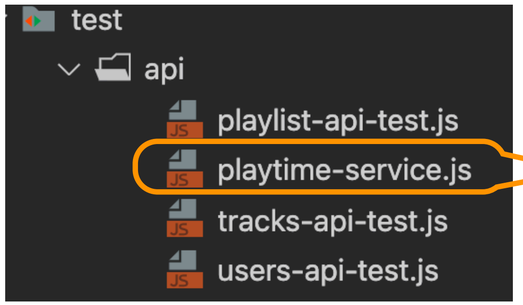
Promise based HTTP client for the browser and node.js

What is Axios?

Axios is a *promise-based* HTTP Client for `node.js` and the browser. It is *isomorphic* (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js `http` module, while on the client (browser) it uses XMLHttpRequests.

Features

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Automatic transforms for JSON data
- Client side support for protecting against XSRF



playtime-service.js

```
import axios from "axios";

import { serviceUrl } from "../fixtures.js";

export const playtimeService = {
  playtimeUrl: serviceUrl,

  async createUser(user) {
    const res = await axios.post(`${this.playtimeUrl}/api/users`, user);
    return res.data;
  }
}
```

fixtures.js

```
export const serviceUrl = "http://localhost:3000";
```

- “Gateway” to API
- Uses Axios to create dispatch HTTP requests
- Also to accept HTTP responses

- Create a user
- Verify that returned user is subset of fixture
- Verify that _id was generated

users-api-test.js

```
import { assert } from "chai";
import { playtimeService } from "./playtime-service.js";
import { assertSubset } from "../test-utils.js";
import { maggie } from "../fixtures.js";

suite("User API tests", () => {
  setup(async () => {
  });
  teardown(async () => {
  });

  test("create a user", async () => {
    const newUser = await playtimeService.createUser(maggie);
    assertSubset(maggie, newUser);
    assert.isDefined(newUser._id);
  });
});
```

Testing Endpoints

- Build test as a programmatic client of the service
- Exercise the REST endpoints
- Write tests that are
  - consistent
  - methodical
  - exhaustive

Rest Endpoints Verbs

- Comparing database (sql) and HTTP Verbs

SQL	REST
SELECT	GET
INSERT	POST
UPDATE	PUT
DELETE	DELETE

Action varies with HTTP Method

URI	HTTP METHOD	ACTION PERFORMED
/status/	GET	Get all status
/status/3	GET	Get status with id 3
/status/	POST	Add a new status
/status/4	PUT	Edit status with id 4
/status/4	DELETE	Delete status with id 4

HTTP Response Codes

HTTP Status Codes	Informational
200	OK
201	Resource created
204	No content
400	Bad Request
401	Unauthorised
404	Not found
405	Method Not allowed
500	Internal Server Error

Users API Routes

api-routes.js

```
import { usersApi } from "./api/users-api.js";

export const apiRoutes = [
  { method: "GET", path: "/api/users", config: usersApi.find },
  { method: "POST", path: "/api/users", config: usersApi.create },
  { method: "DELETE", path: "/api/users", config: usersApi.deleteAll },
  { method: "GET", path: "/api/users/{id}", config: usersApi.findOne },
];
```

users-api.js

```
import Boom from "@hapi/boom";
import { db } from "../models/db.js";

export const usersApi = {
  find: {
    auth: false,
    handler: async function (request, h) {
      try {
        const users = await db.userStore.getAllUsers();
        return users;
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
  },

  findOne: {
    auth: false,
    handler: async function (request, h) {
      try {
        const user = await db.userStore.getUserById(request.params.id);
        if (!user) {
          return Boom.notFound("No User with this id");
        }
        return user;
      } catch (err) {
        return Boom.serverUnavailable("No User with this id");
      }
    },
  },

  create: {
    auth: false,
    handler: async function (request, h) {
      try {
        const user = await db.userStore.addUser(request.payload);
        if (user) {
          return h.response(user).code(201);
        }
        return Boom.badImplementation("error creating user");
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
  },

  deleteAll: {
    auth: false,
    handler: async function (request, h) {
      try {
        await db.userStore.deleteAll();
        return h.response().code(204);
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
  },
};
```

- playtimeService expanded to cover additional features of the API

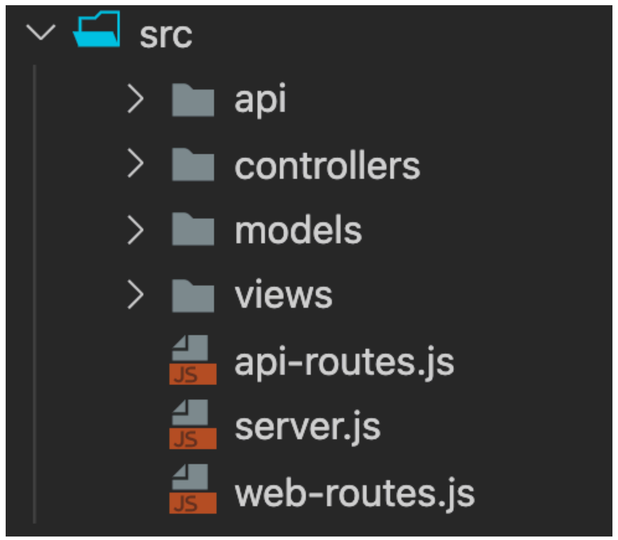
playtime-service.js

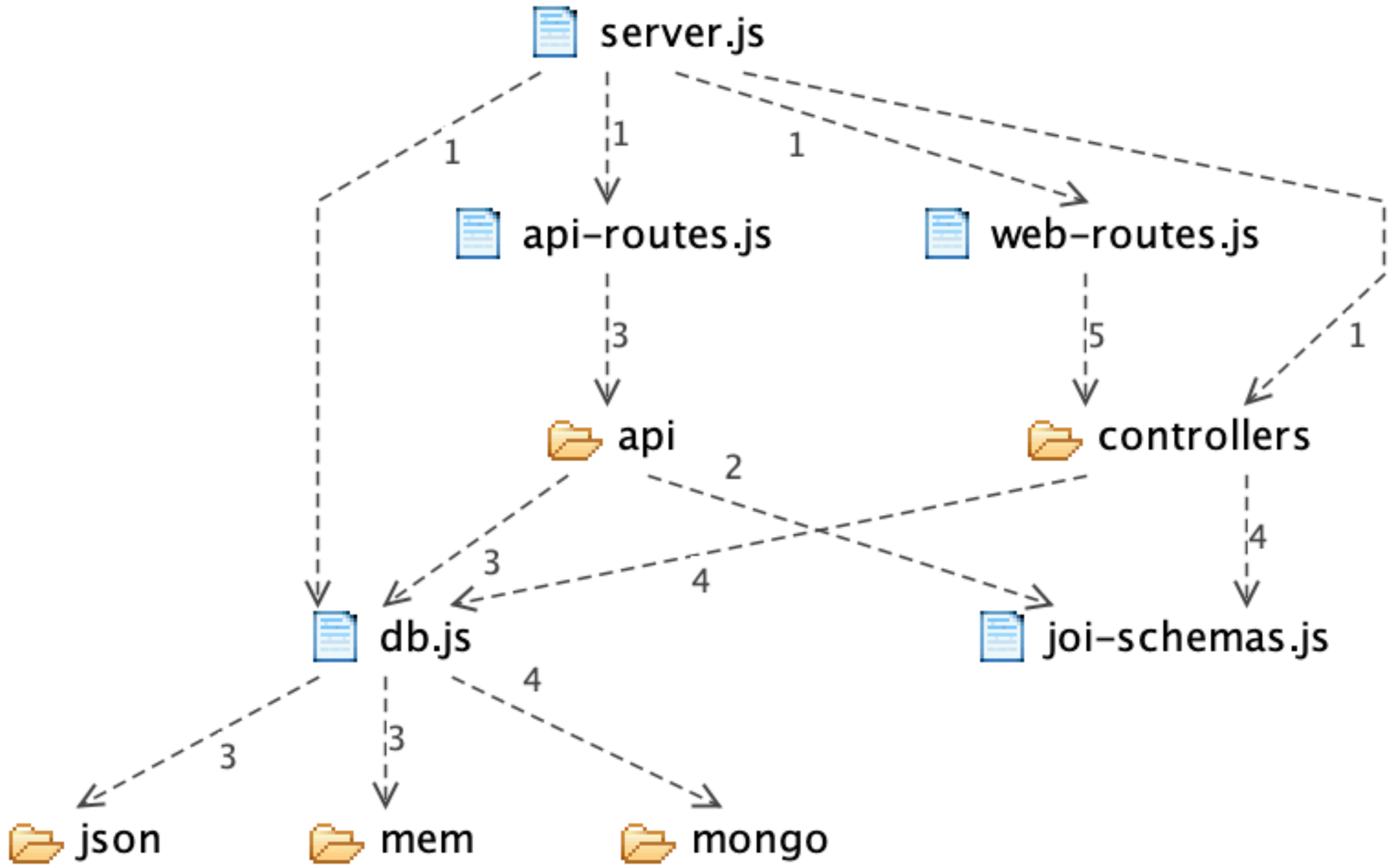
```
import axios from "axios";

import { serviceUrl } from "../fixtures.js";

export const playtimeService = {
  playtimeUrl: serviceUrl,

  async createUser(user) {
    const res = await axios.post(`${this.playtimeUrl}/api/users`, user);
    return res.data;
  },

  async getUser(id) {
    const res = await axios.get(`${this.playtimeUrl}/api/users/${id}`);
    return res.data;
  },

  async getAllUsers() {
    const res = await axios.get(`${this.playtimeUrl}/api/users`);
    return res.data;
  },

  async deleteAllUsers() {
    const res = await axios.delete(`${this.playtimeUrl}/api/users`);
    return res.data;
  },
};
```

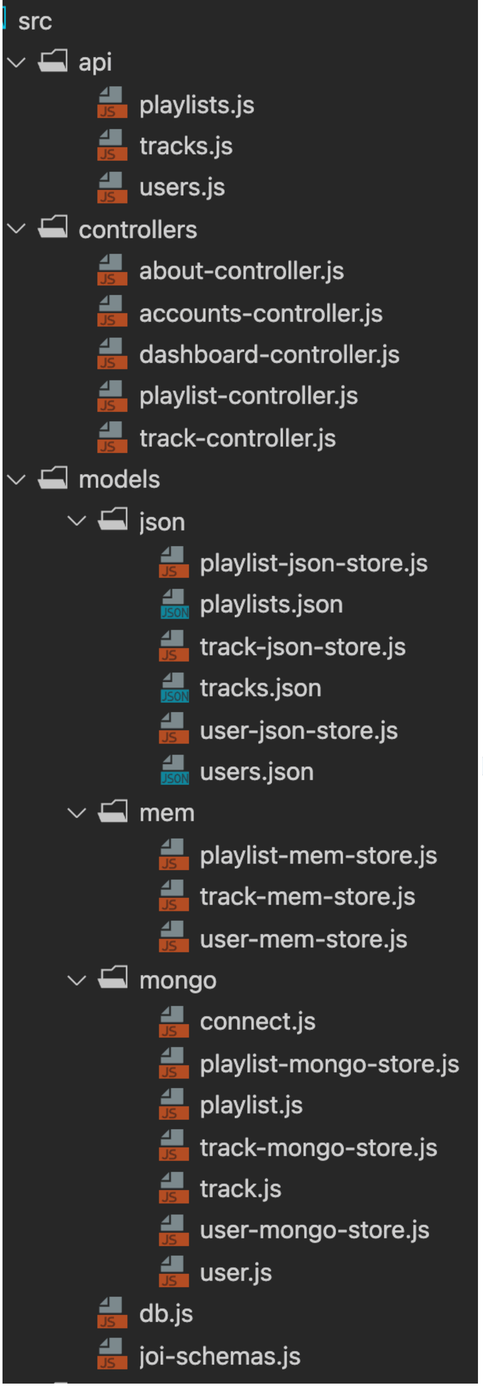
- Setup Creates test users
- Test POST
- Test DELETE
- Test GET

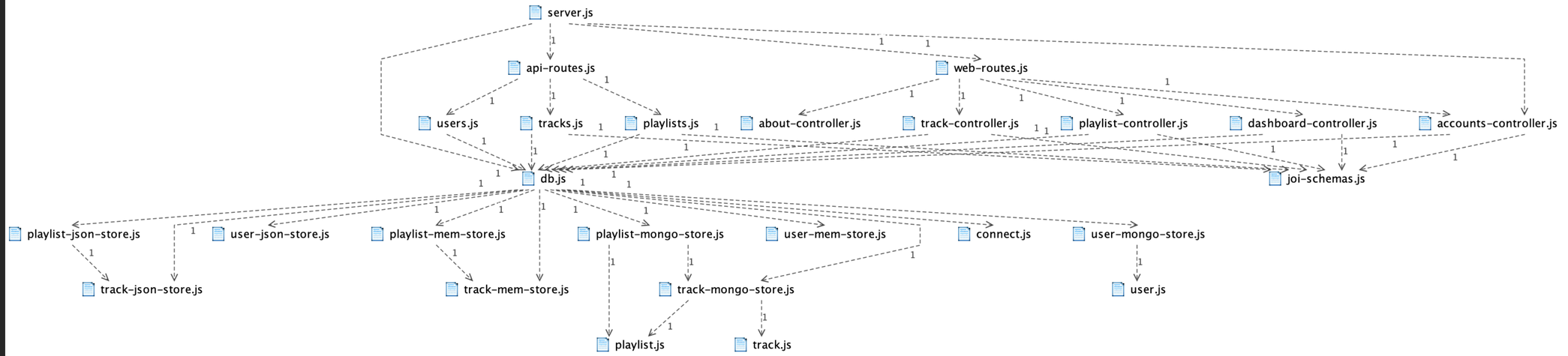
users-apis-test.js

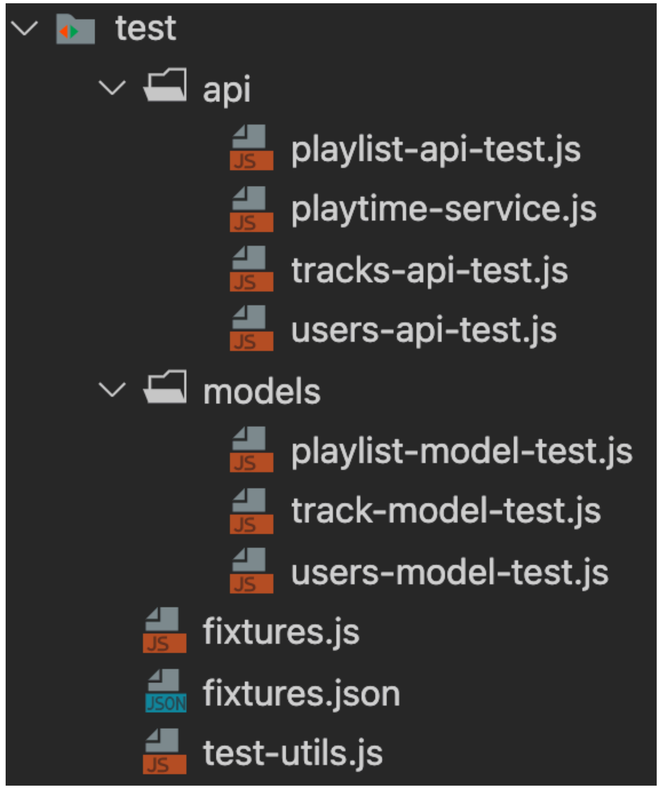
```
import { assert } from "chai";
import { playtimeService } from "./playtime-service.js";
import { assertSubset } from "../test-utils.js";
import { maggie, testUsers } from "../fixtures.js";

suite("User API tests", () => {
  setup(async () => {
    await playtimeService.deleteAllUsers();
    for (let i = 0; i < testUsers.length; i += 1) {
      // eslint-disable-next-line no-await-in-loop
      testUsers[0] = await playtimeService.createUser(testUsers[i]);
    }
  });
  teardown(async () => {
  });

  test("create a user", async () => {
    const newUser = await playtimeService.createUser(maggie);
    assertSubset(maggie, newUser);
    assert.isDefined(newUser._id);
  });

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

  test("get a user - success", async () => {
    const returnedUser = await playtimeService.getUser(testUsers[0]._id);
    assert.deepEqual(testUsers[0], returnedUser);
  });
});
```

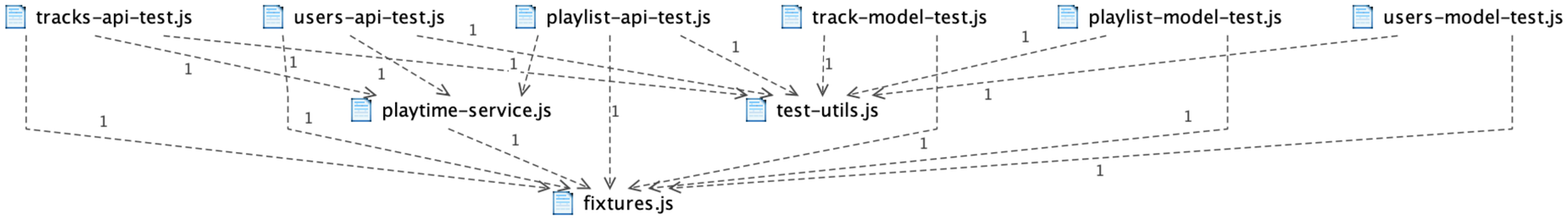

src
api
controllers
models
views
api-routes.js
server.js
web-routes.js


server.js
api-routes.js
web-routes.js
api
controllers
db.js
joi-schemas.js
json
mem
mongo
1
1
1
3
5
1
2
3
4
4
4
3
3


src
api
playlists.js
tracks.js
users.js
controllers
about-controller.js
accounts-controller.js
dashboard-controller.js
playlist-controller.js
track-controller.js
models
json
playlist-json-store.js
playlists.json
track-json-store.js
tracks.json
user-json-store.js
users.json
mem
playlist-mem-store.js
track-mem-store.js
user-mem-store.js
mongo
connect.js
playlist-mongo-store.js
playlist.js
track-mongo-store.js
track.js
user-mongo-store.js
user.js
db.js
joi-schemas.js


server.js
api-routes.js
web-routes.js
users.js
tracks.js
playlists.js
about-controller.js
track-controller.js
playlist-controller.js
dashboard-controller.js
accounts-controller.js
db.js
joi-schemas.js
playlist-json-store.js
user-json-store.js
playlist-mem-store.js
playlist-mongo-store.js
user-mem-store.js
connect.js
user-mongo-store.js
track-json-store.js
track-mem-store.js
track-mongo-store.js
user.js
playlist.js
track.js


test
api
playlist-api-test.js
playtime-service.js
tracks-api-test.js
users-api-test.js
models
playlist-model-test.js
track-model-test.js
users-model-test.js
fixtures.js
fixtures.json
test-utils.js

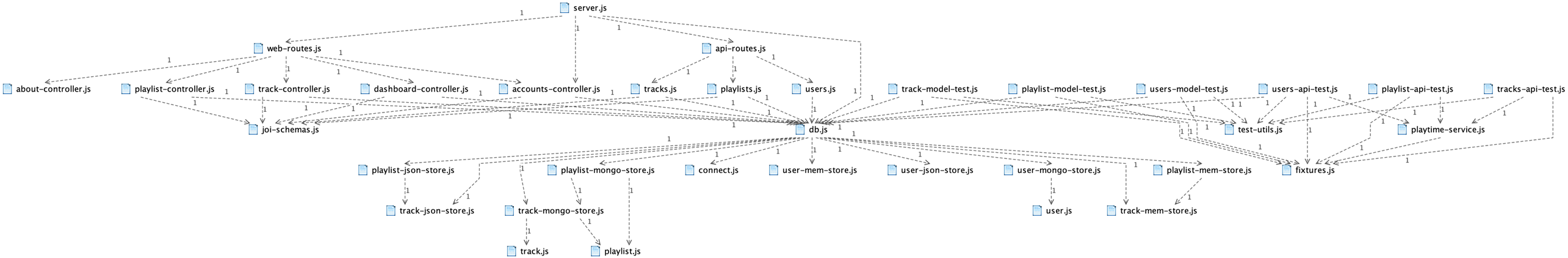

tracks-api-test.js
users-api-test.js
playlist-api-test.js
track-model-test.js
playlist-model-test.js
users-model-test.js
playtime-service.js
test-utils.js
fixtures.js

Products Resources Download Store About Blog

structure101

PUT YOUR SOFTWARE STRUCTURE TO WORK

DOWNLOAD NOW



Visualize

Visualizing the structure that emerges from source code to instantly boost architectural accuracy and developer productivity.

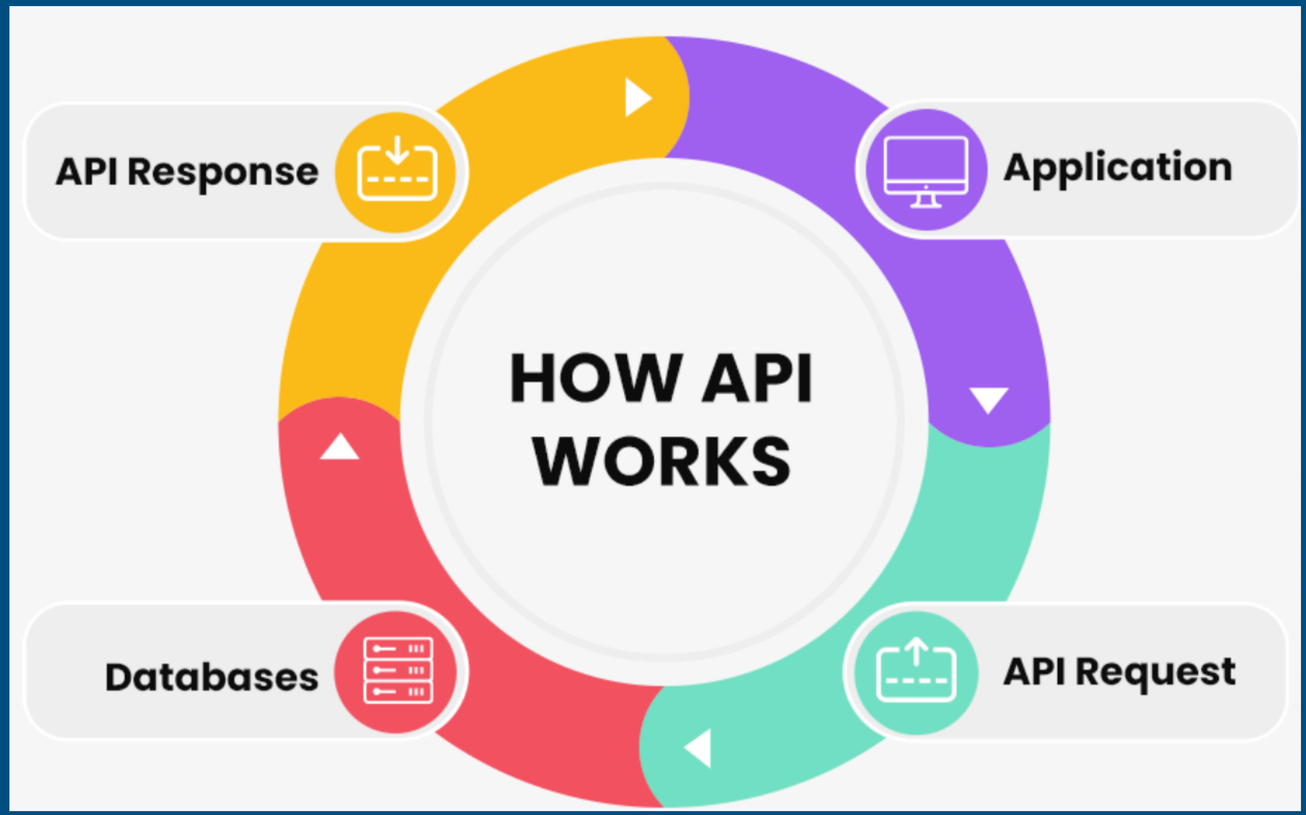
Specify

Add API and dependency specs to the emergent model to guide the team as they edit code, and to catch violations at build time.

Simplify

Manipulate the visual model to disentangle monoliths, making the visualized, specified codebase more modular and even easier to adapt and develop.

Testing End Points



Full Stack Web Development