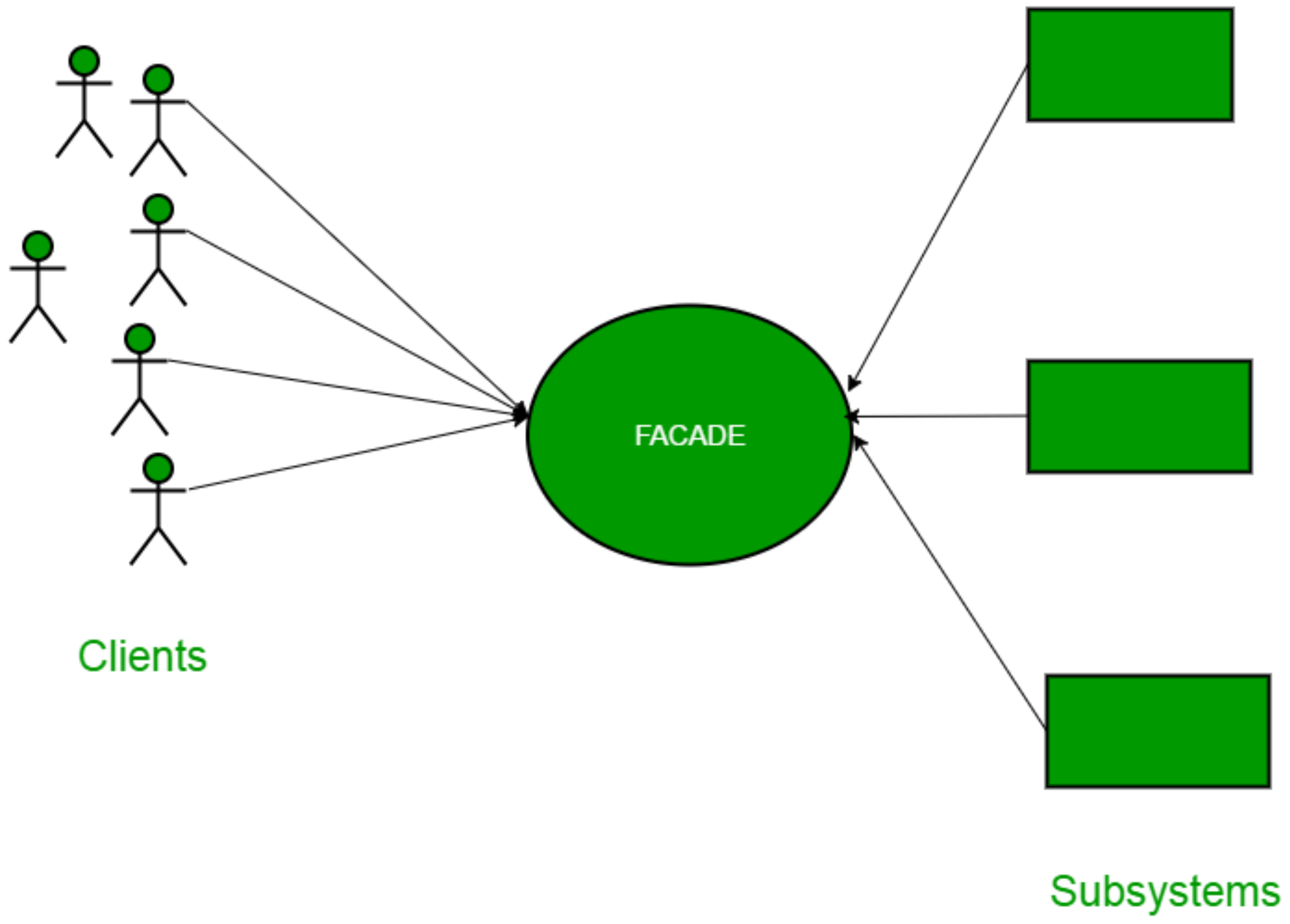
Mongo Stores

Full Stack Web Development

Facade Pattern

- The facade pattern (also spelled façade) is a software-design pattern commonly used in object-oriented programming.

- Analogous to a facade in architecture, a facade is an object that serves as a front-facing interface masking more complex underlying or structural code



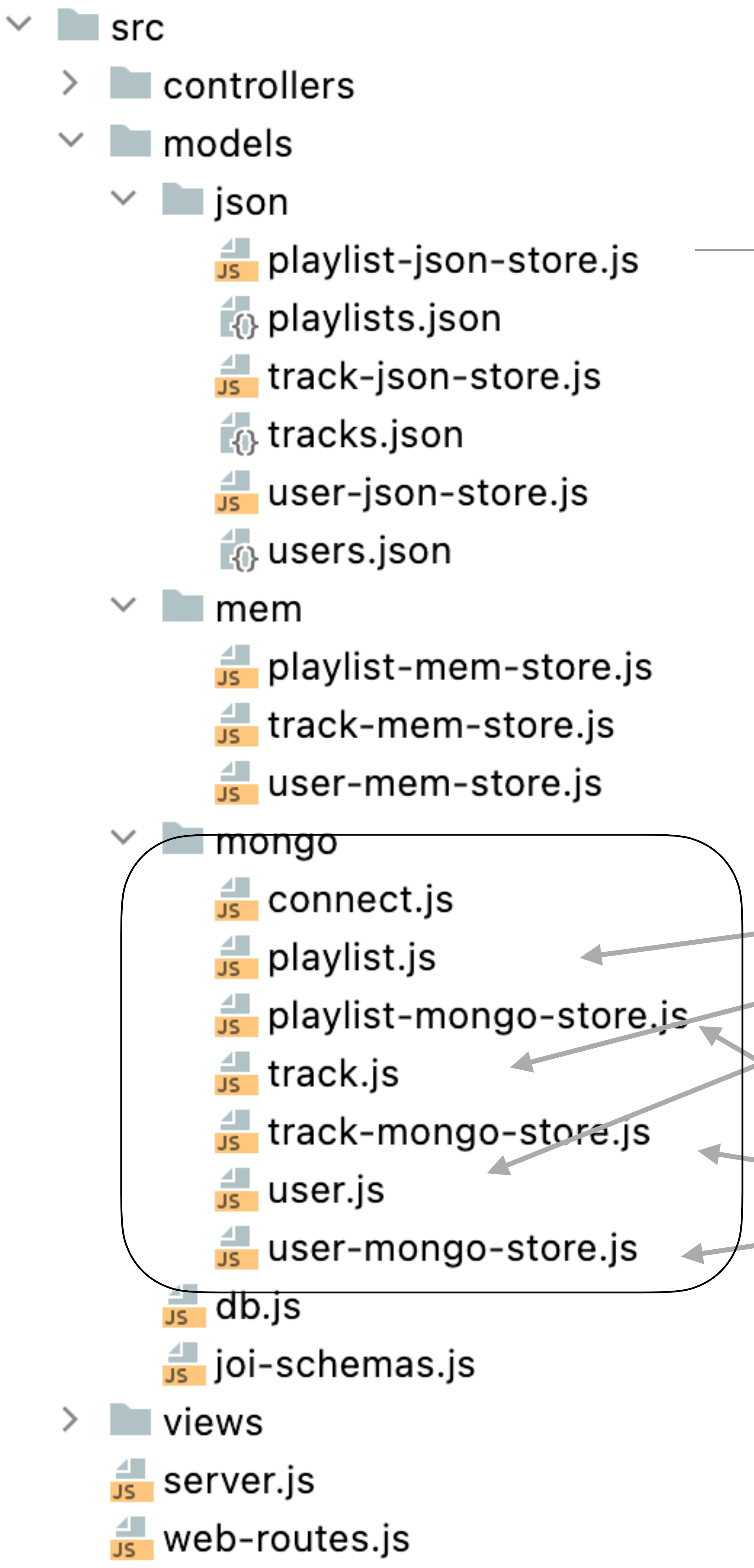
- improve the readability and usability of a software library by masking interaction with more complex components behind a single API

- provide a context-specific interface to more generic functionality

- serve as a launching point for a broader refactor of monolithic or tightly-coupled systems in favour of more loosely-coupled code

https://en.wikipedia.org/wiki/Facade_pattern

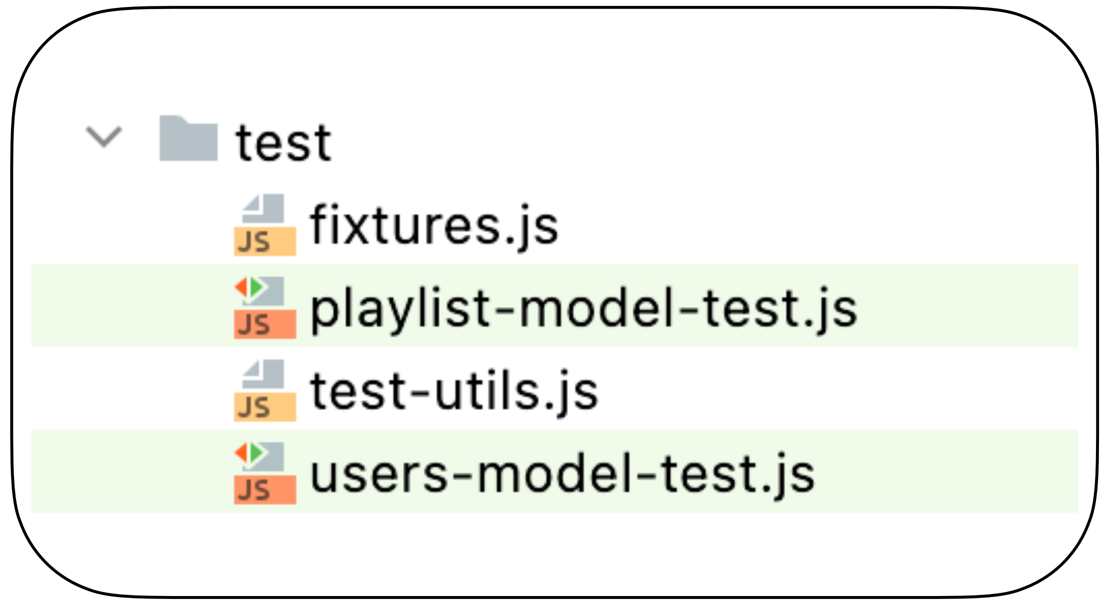
Mongo Model



```
src
  controllers
  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.js
      playlist-mongo-store.js
      track.js
      track-mongo-store.js
      user.js
      user-mongo-store.js
    db.js
    joi-schemas.js
  views
  server.js
  web-routes.js
```

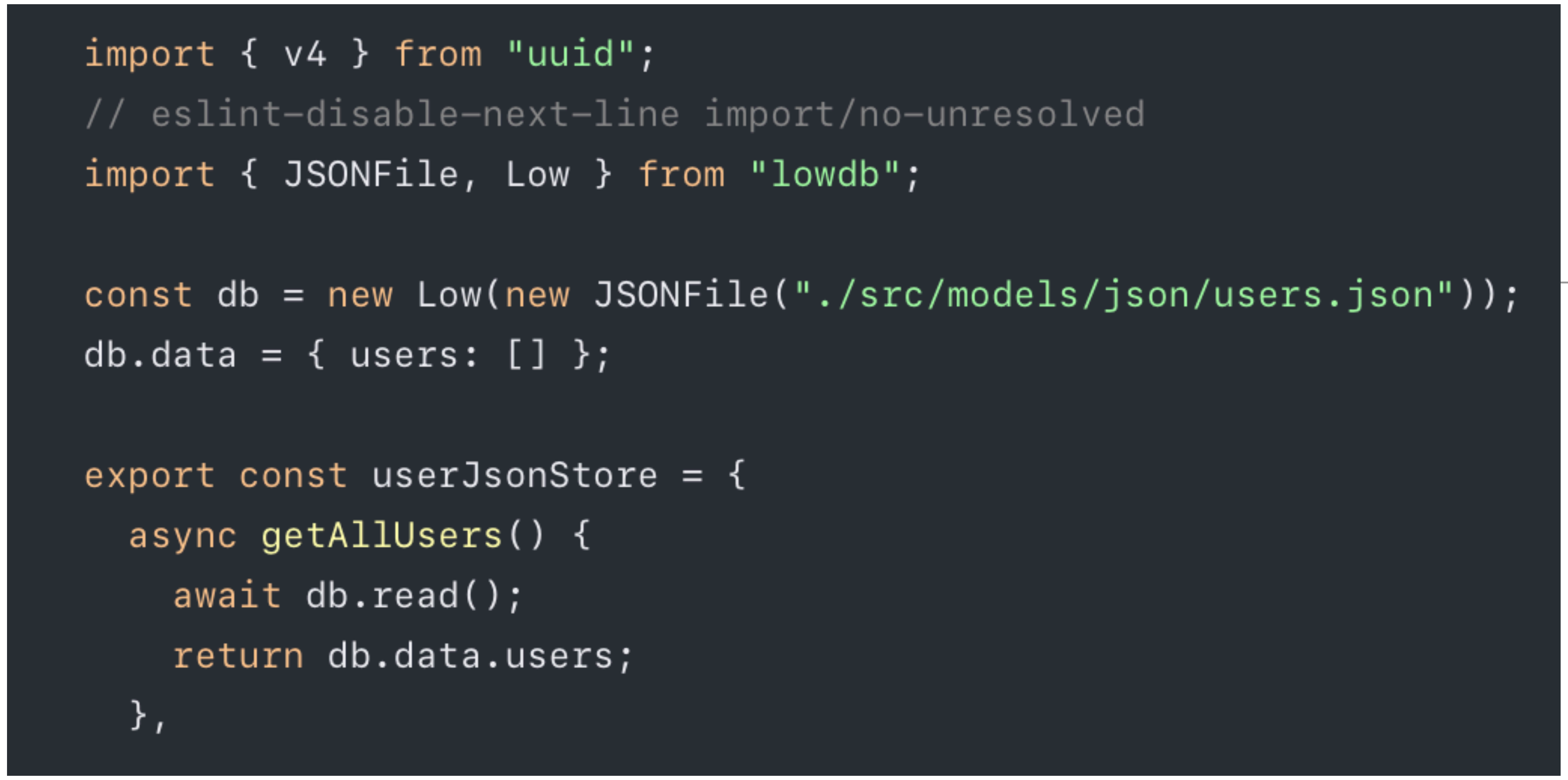
That manage these mongo collections

Introduce new set of stores



```
test
  fixtures.js
  playlist-model-test.js
  test-utils.js
  users-model-test.js
```

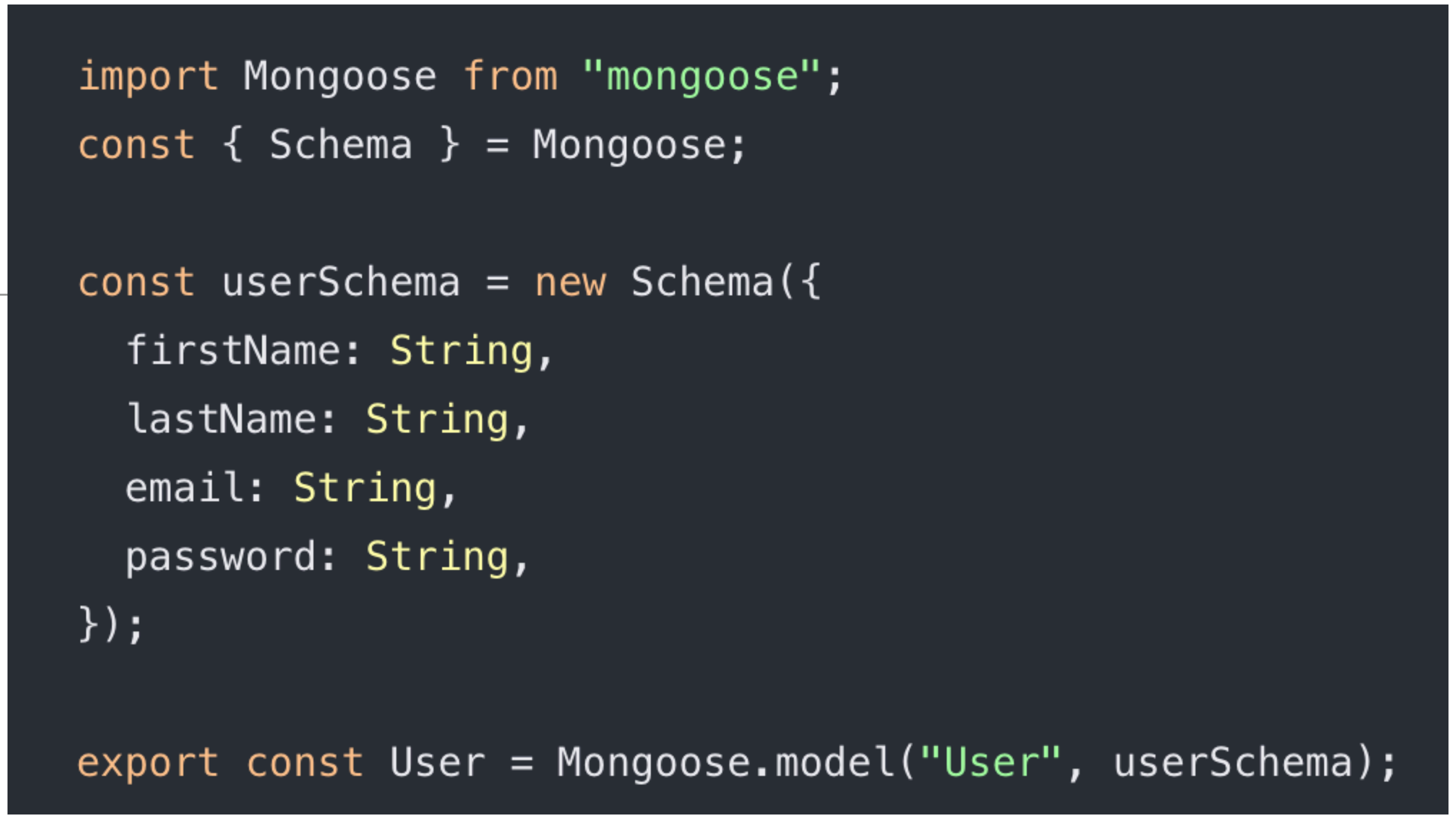
These already developed tests should accelerate development user-json-store.js



```
import { v4 } from "uuid";
// eslint-disable-next-line import/no-unresolved
import { JSONFile, Low } from "lowdb";

const db = new Low(new JSONFile("./src/models/json/users.json"));
db.data = { users: [] };

export const userJsonStore = {
  async getAllUsers() {
    await db.read();
    return db.data.users;
  },
```

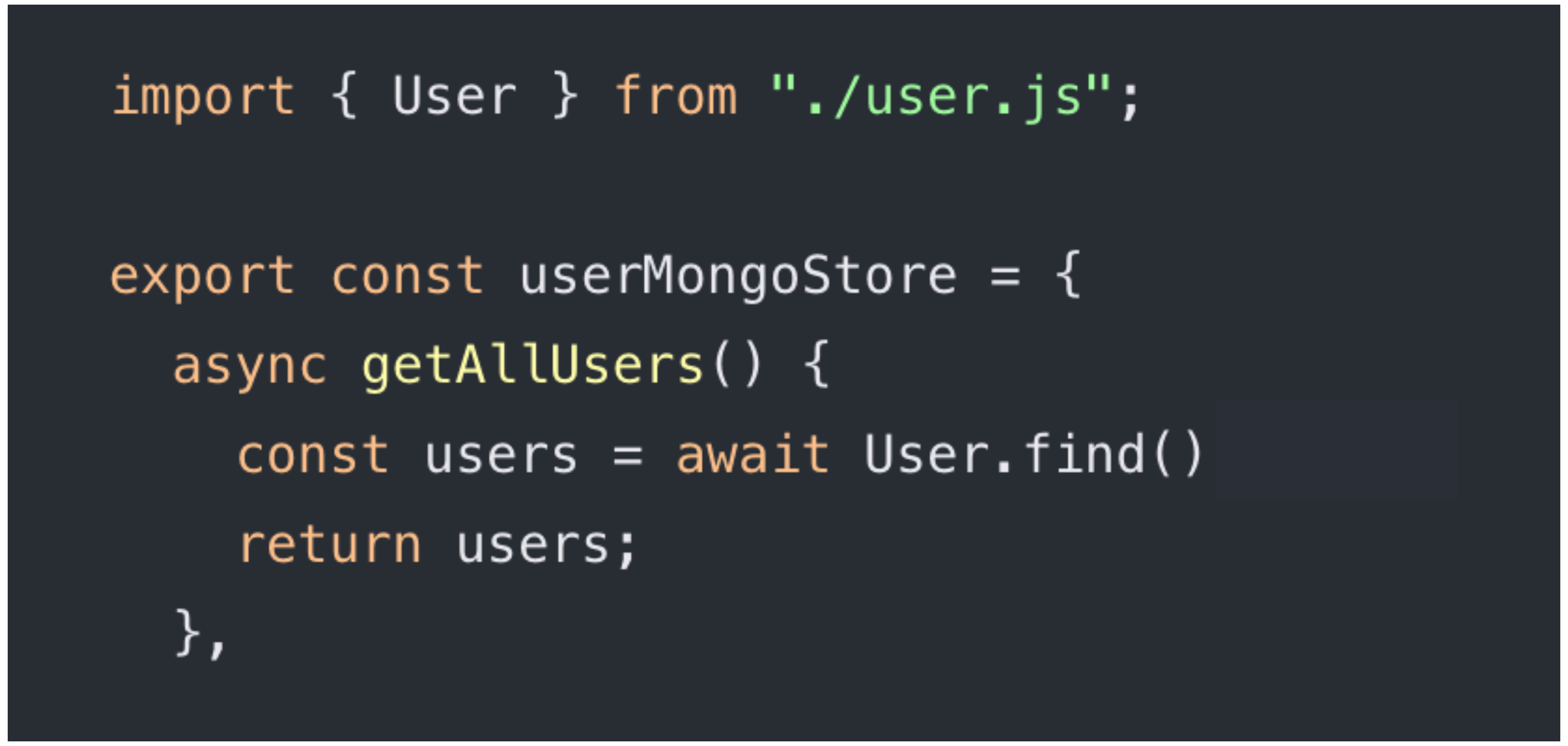
Define schema

user.js



```
import Mongoose from "mongoose";
const { Schema } = Mongoose;

const userSchema = new Schema({
  firstName: String,
  lastName: String,
  email: String,
  password: String,
});

export const User = Mongoose.model("User", userSchema);
```

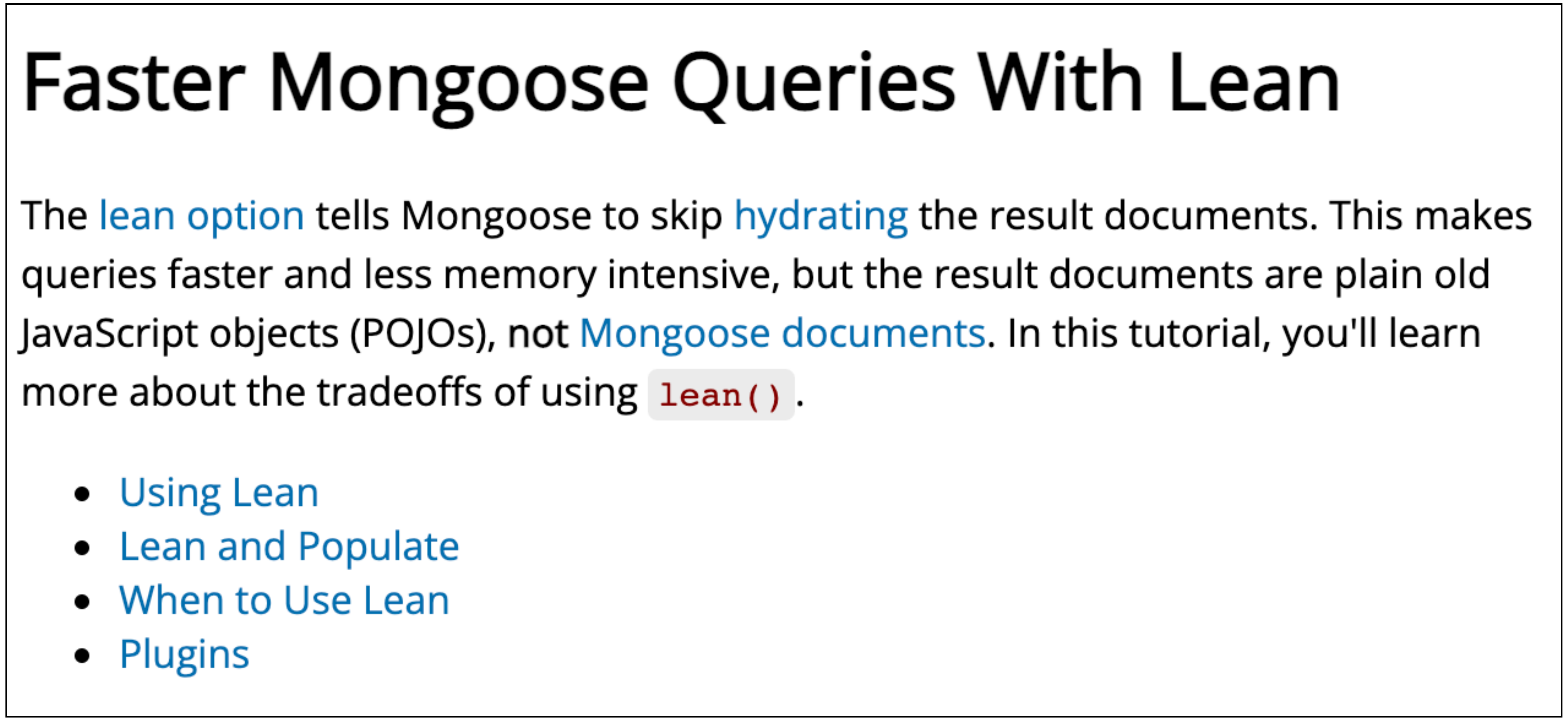
Use Mongo Model to access db



```
import { User } from "./user.js";

export const userMongoStore = {
  async getAllUsers() {
    const users = await User.find()
    return users;
  },
```

user-mongo-store.js

- Mongo queries return rich mongoose document objects
- These documents support a range or further query and access features
- *lean()* produces a POJO - Plain Old Javascript Object

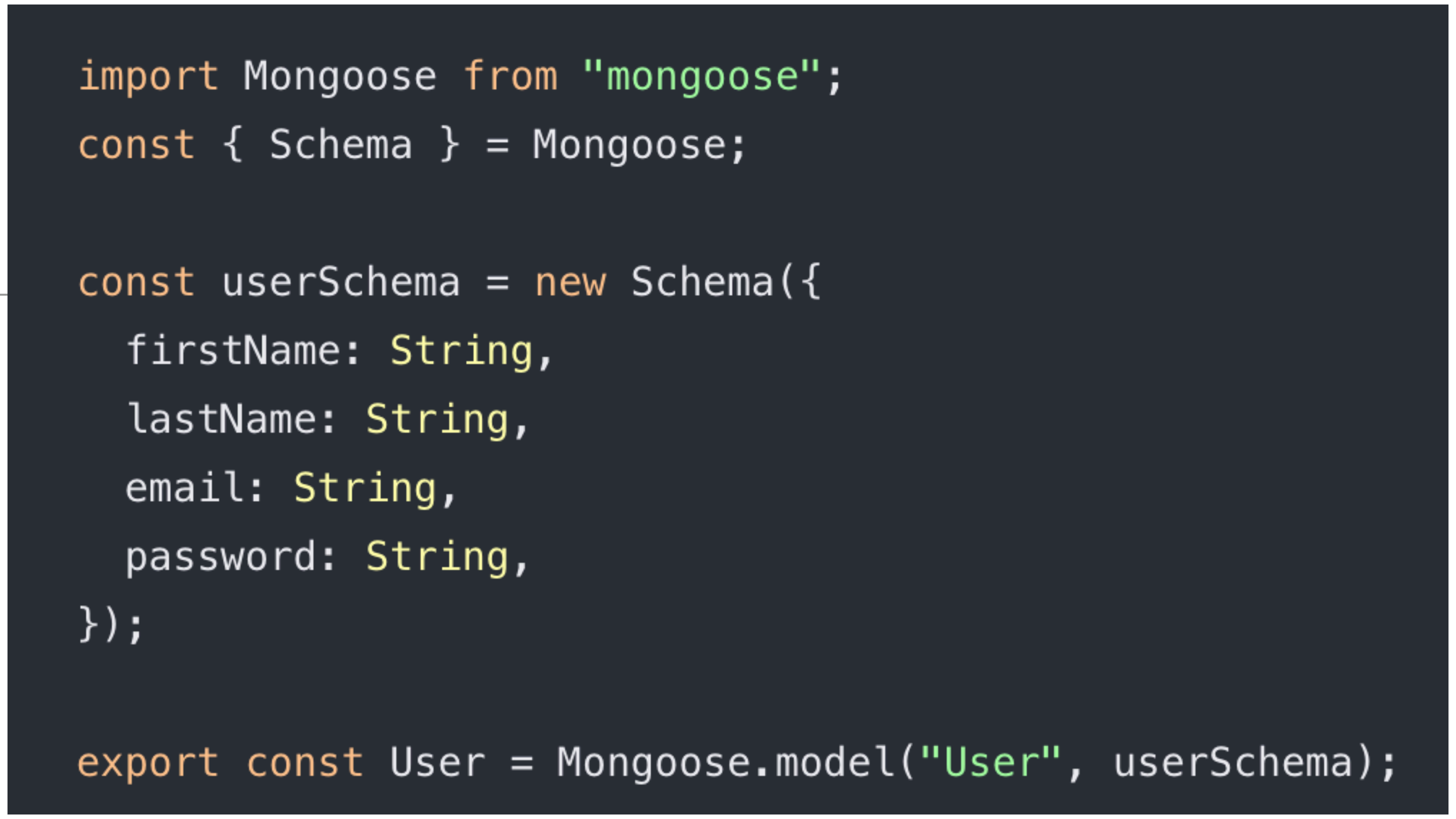


Faster Mongoose Queries With Lean

The lean option tells Mongoose to skip hydrating the result documents. This makes queries faster and less memory intensive, but the result documents are plain old JavaScript objects (POJOs), not Mongoose documents. In this tutorial, you'll learn more about the tradeoffs of using `lean()`.

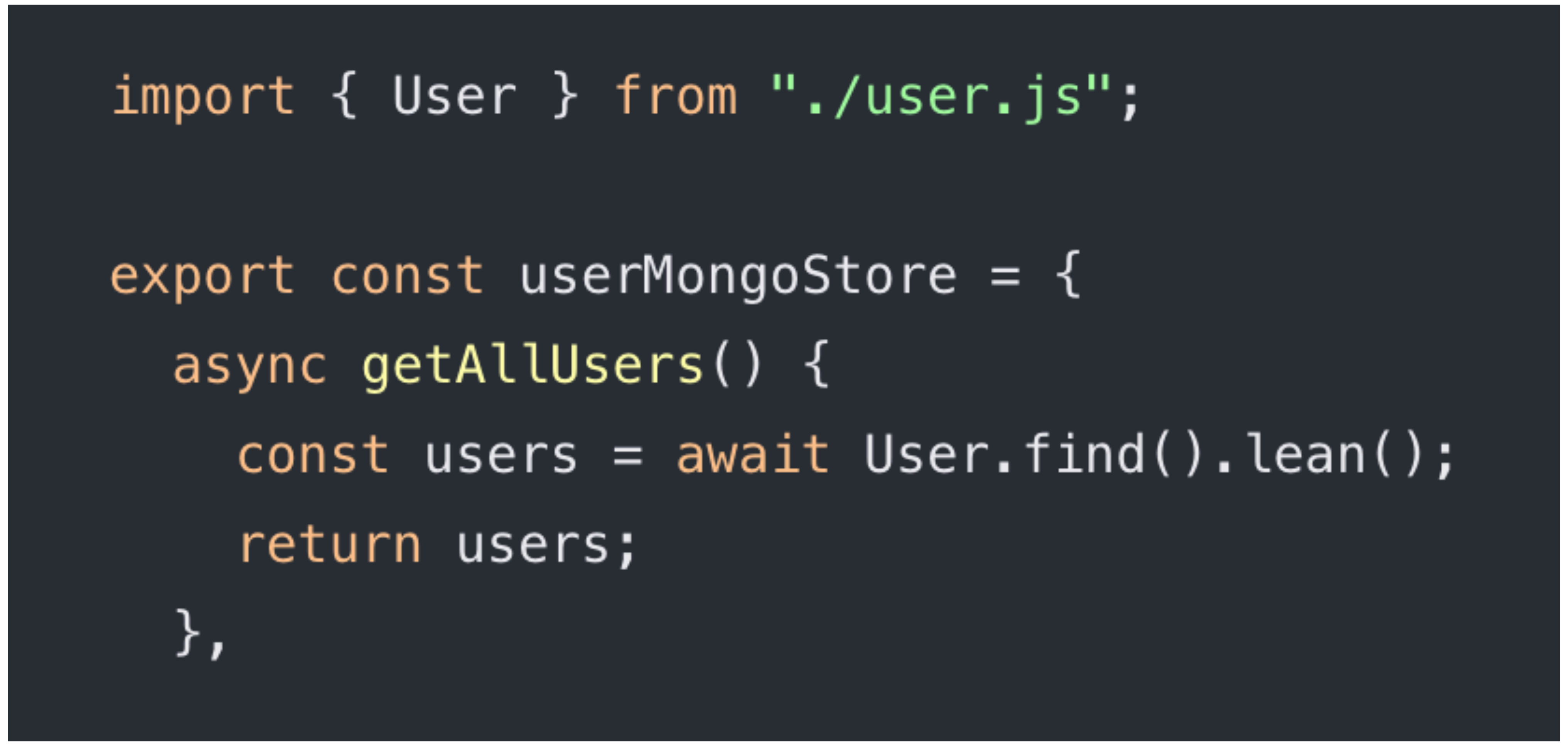
- Using Lean
- Lean and Populate
- When to Use Lean
- Plugins

user.js



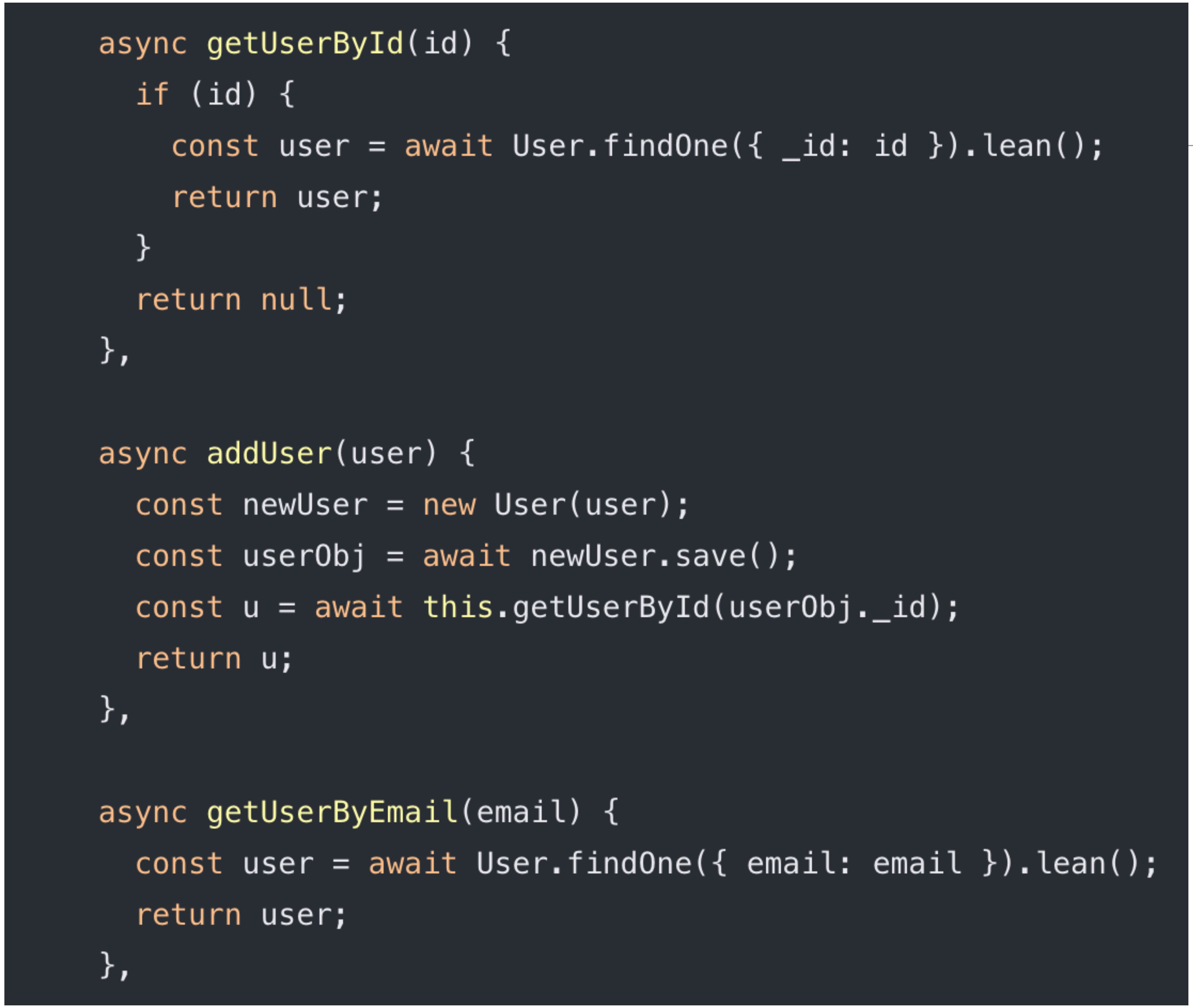
```
import Mongoose from "mongoose";
const { Schema } = Mongoose;

const userSchema = new Schema({
  firstName: String,
  lastName: String,
  email: String,
  password: String,
});

export const User = Mongoose.model("User", userSchema);
```

Use Mongo Model to access db

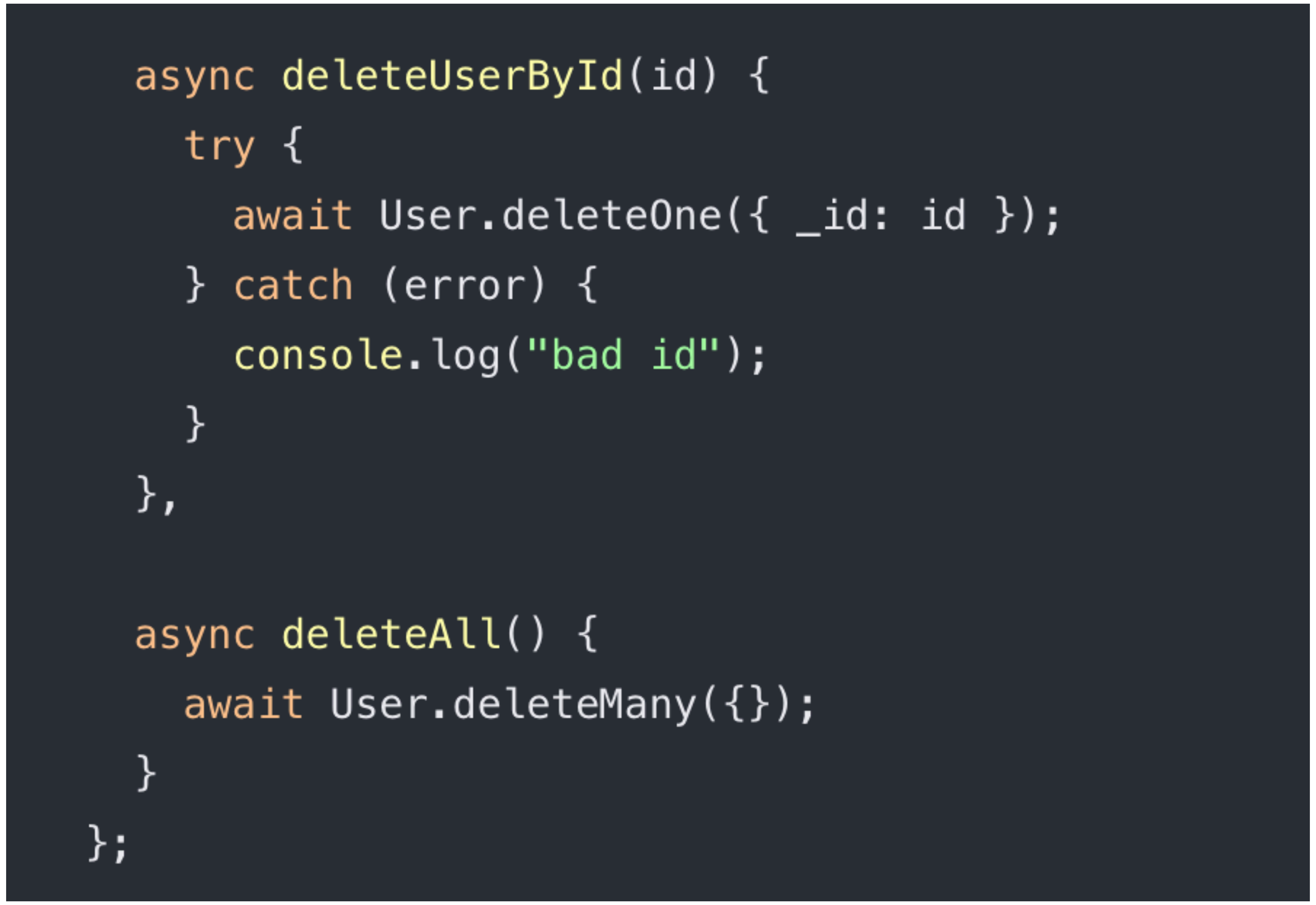


```
import { User } from "./user.js";

export const userMongoStore = {
  async getAllUsers() {
    const users = await User.find().lean();
    return users;
  },
```

user-mongo-store.js



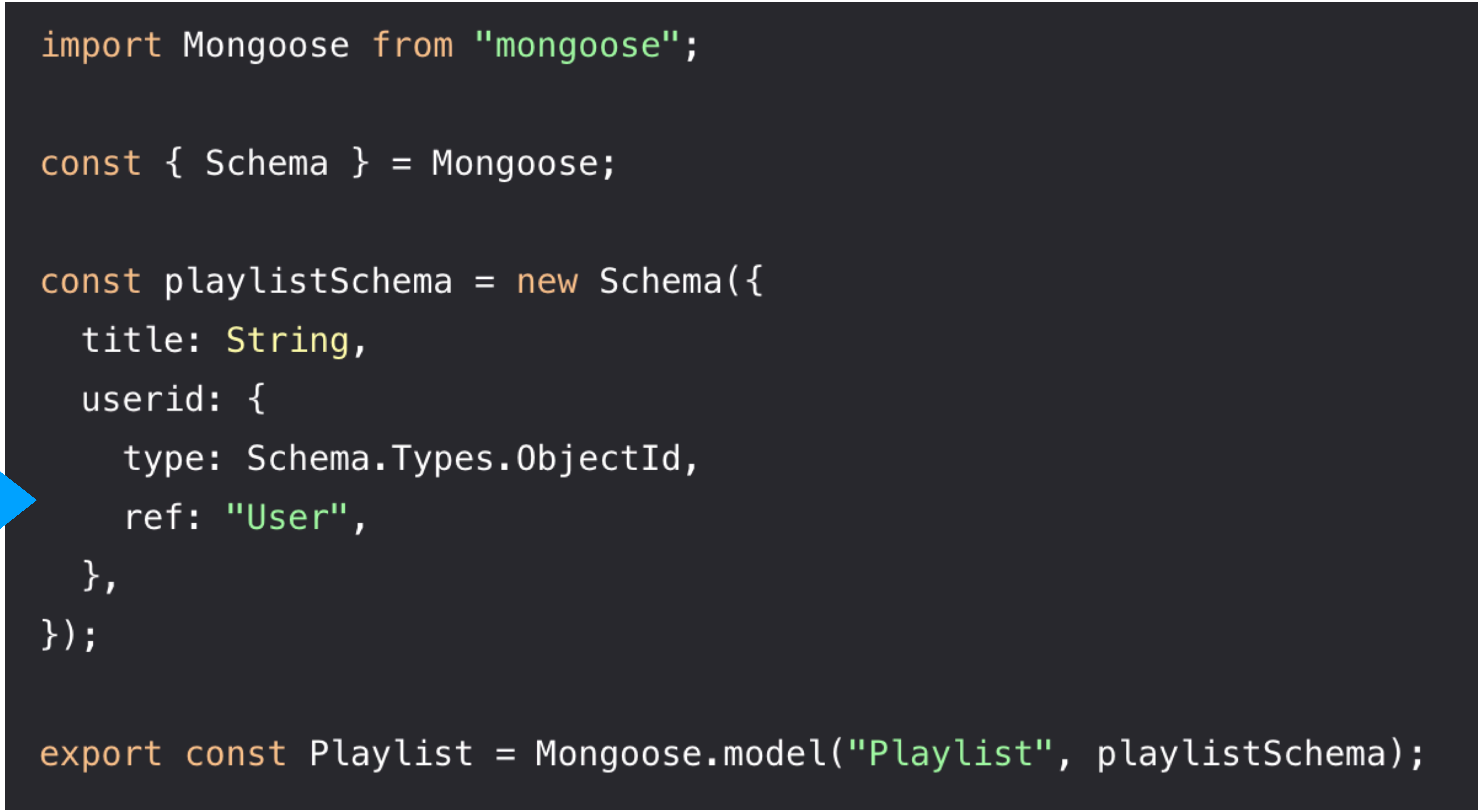
```
  async getUserById(id) {
    if (id) {
      const user = await User.findOne({ _id: id }).lean();
      return user;
    }
    return null;
  },

  async addUser(user) {
    const newUser = new User(user);
    const userObj = await newUser.save();
    const u = await this.getUserById(userObj._id);
    return u;
  },

  async getUserByEmail(email) {
    const user = await User.findOne({ email: email }).lean();
    return user;
  },
```



```
  async deleteUserById(id) {
    try {
      await User.deleteOne({ _id: id });
    } catch (error) {
      console.log("bad id");
    }
  },

  async deleteAll() {
    await User.deleteMany({});
  }
};
```

Playlist Model

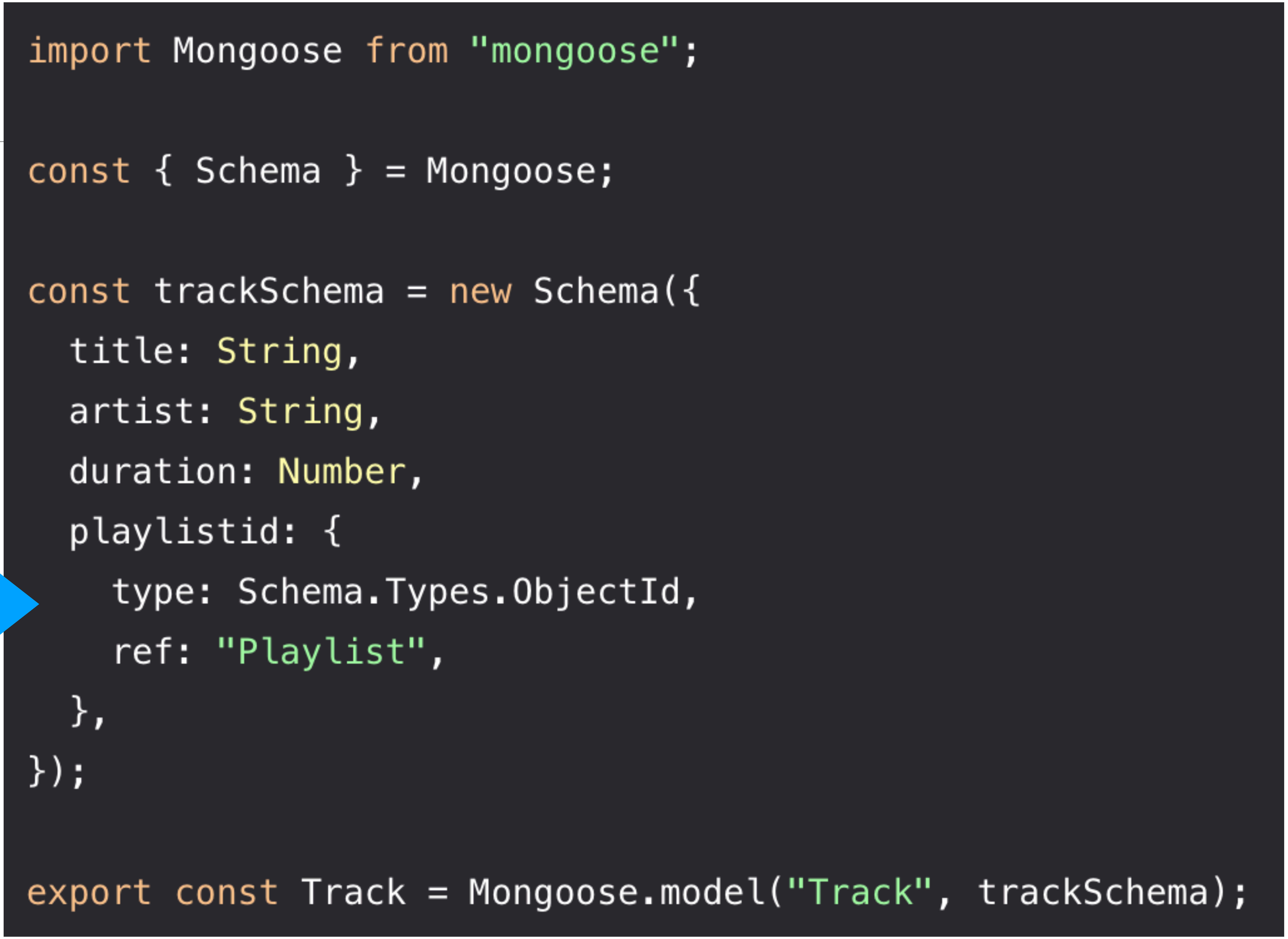
- Reference to an object in another collection



```
import Mongoose from "mongoose";

const { Schema } = Mongoose;

const playlistSchema = new Schema({
  title: String,
  userid: {
    type: Schema.Types.ObjectId,
    ref: "User",
  },
});

export const Playlist = Mongoose.model("Playlist", playlistSchema);
```

Track Model

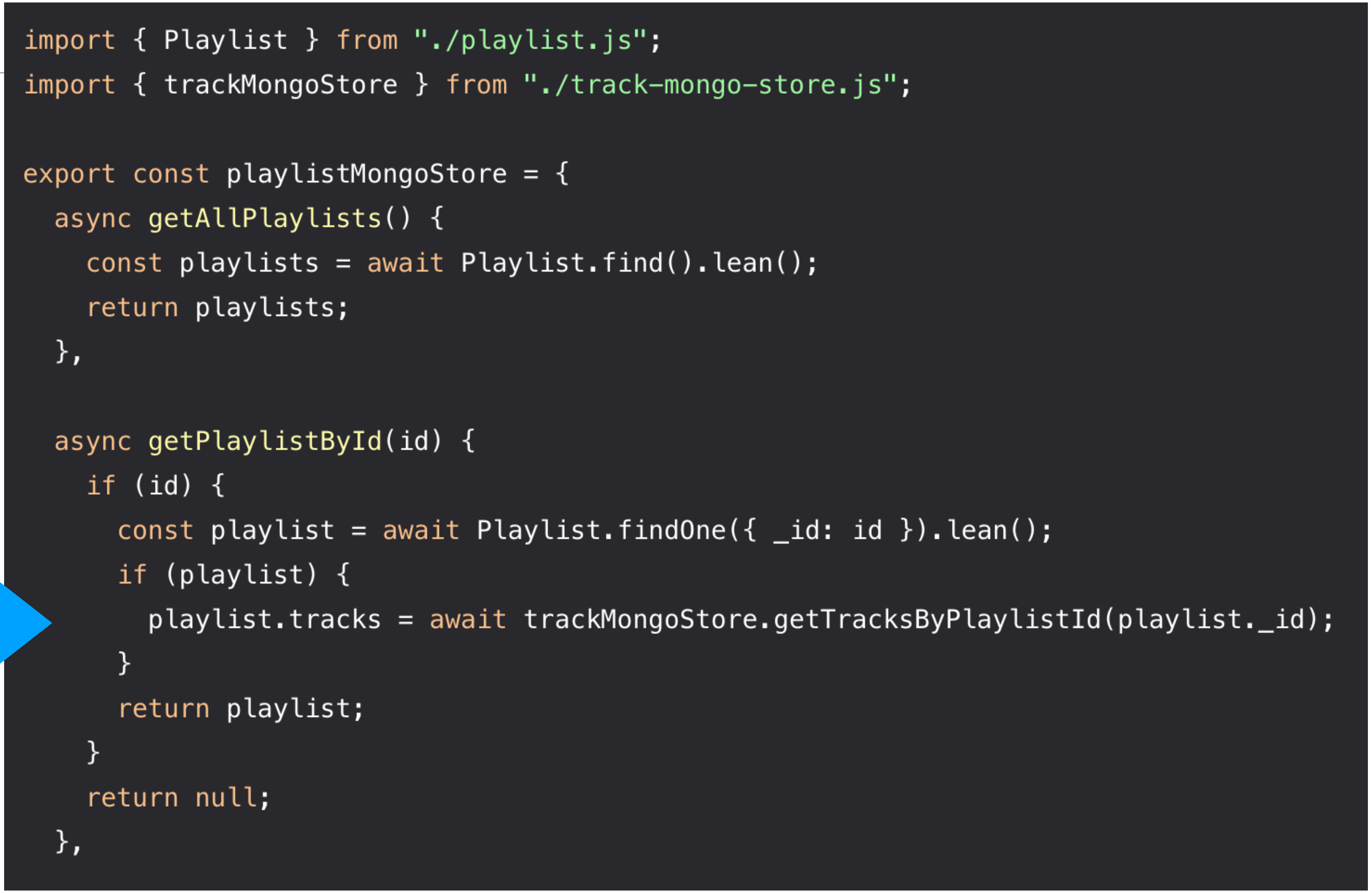
- Reference to an object in another collection



```
import Mongoose from "mongoose";

const { Schema } = Mongoose;

const trackSchema = new Schema({
  title: String,
  artist: String,
  duration: Number,
  playlistid: {
    type: Schema.Types.ObjectId,
    ref: "Playlist",
  },
});

export const Track = Mongoose.model("Track", trackSchema);
```

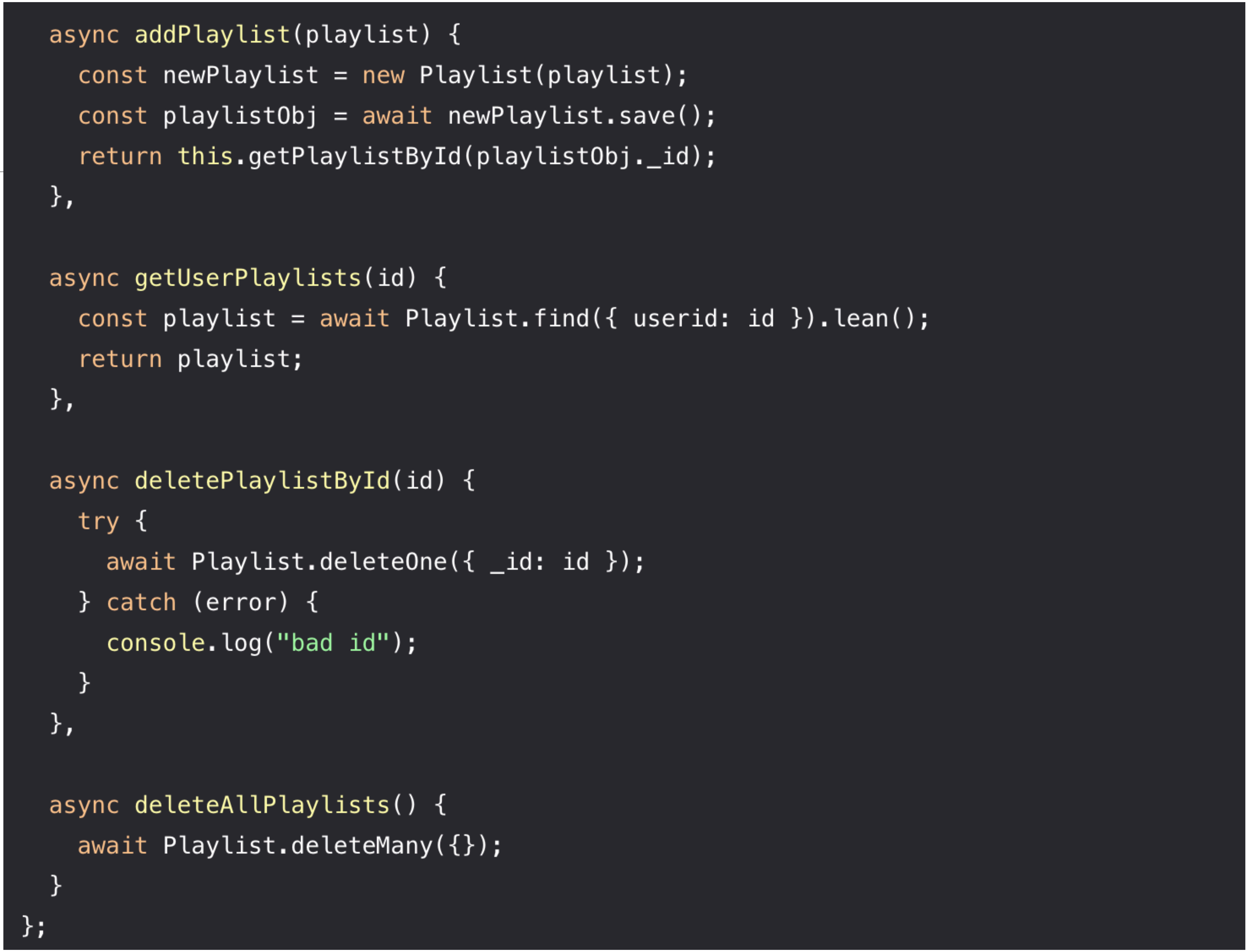
Playlist Store

- Include tracks fetched from track store



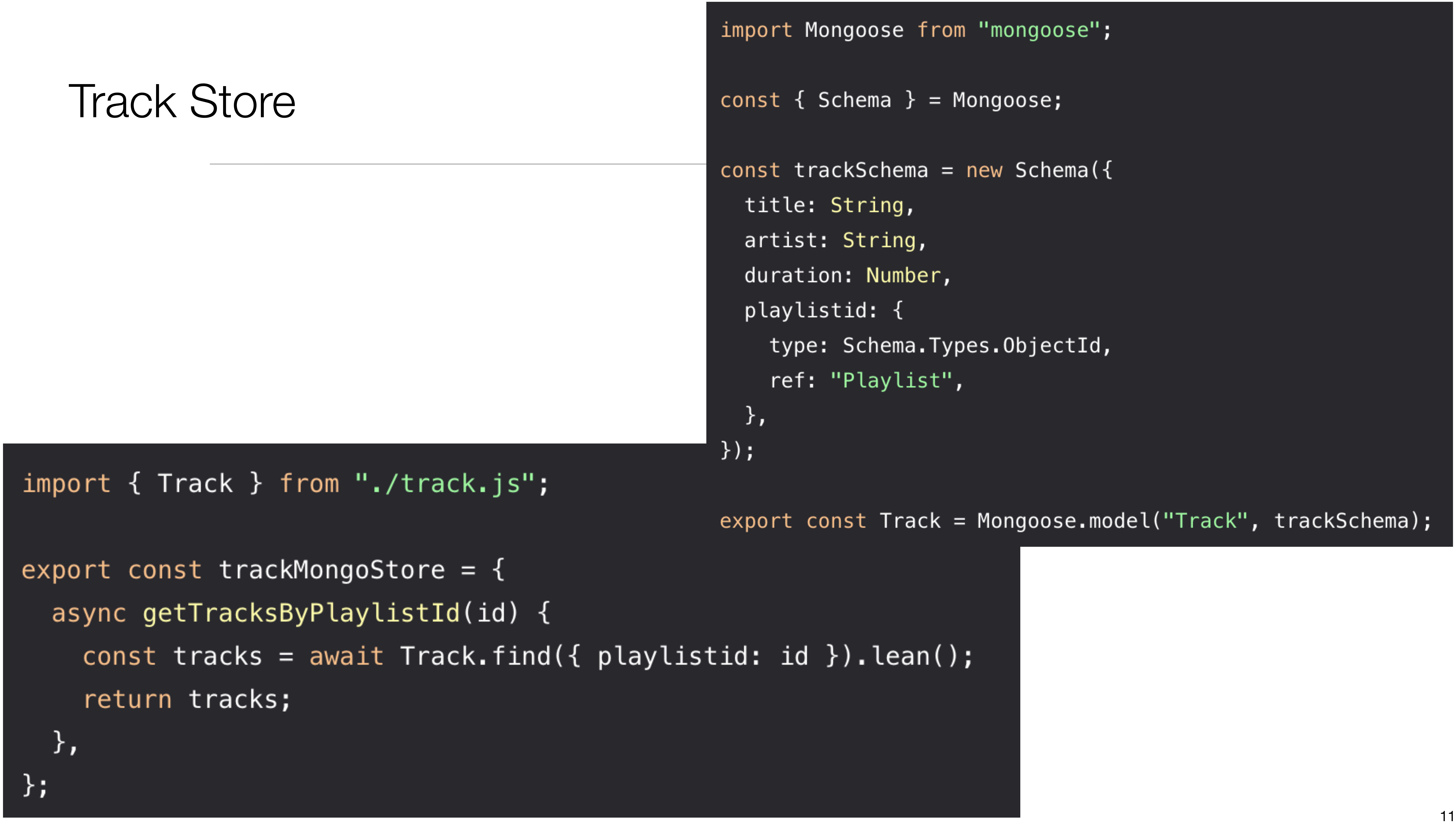
```
import { Playlist } from "./playlist.js";
import { trackMongoStore } from "./track-mongo-store.js";

export const playlistMongoStore = {
  async getAllPlaylists() {
    const playlists = await Playlist.find().lean();
    return playlists;
  },

  async getPlaylistById(id) {
    if (id) {
      const playlist = await Playlist.findOne({ _id: id }).lean();
      if (playlist) {
        playlist.tracks = await trackMongoStore.getTracksByPlaylistId(playlist._id);
      }
      return playlist;
    }
    return null;
  },
```

Playlist Store



```
  async addPlaylist(playlist) {
    const newPlaylist = new Playlist(playlist);
    const playlistObj = await newPlaylist.save();
    return this.getPlaylistById(playlistObj._id);
  },

  async getUserPlaylists(id) {
    const playlist = await Playlist.find({ userid: id }).lean();
    return playlist;
  },

  async deletePlaylistById(id) {
    try {
      await Playlist.deleteOne({ _id: id });
    } catch (error) {
      console.log("bad id");
    }
  },

  async deleteAllPlaylists() {
    await Playlist.deleteMany({});
  }
};
```

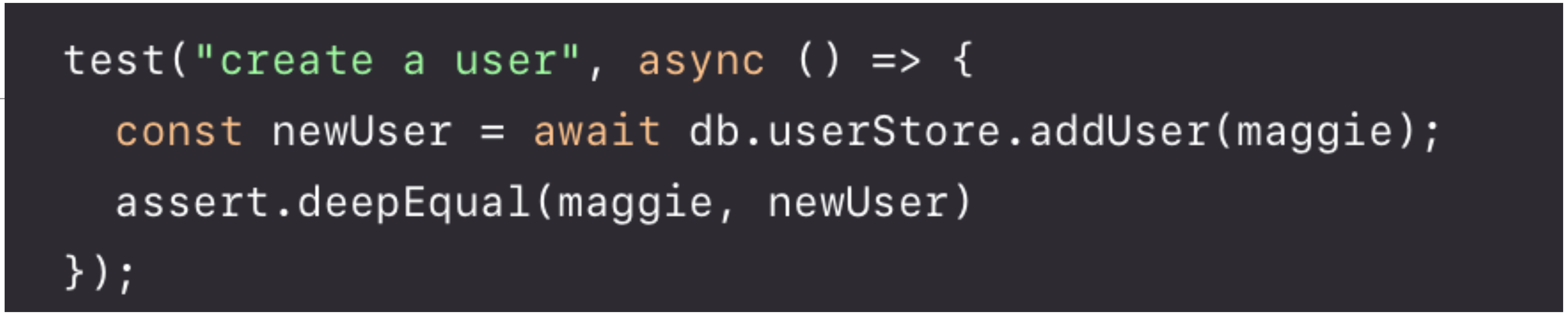
Track Store



```
import Mongoose from "mongoose";

const { Schema } = Mongoose;

const trackSchema = new Schema({
  title: String,
  artist: String,
  duration: Number,
  playlistid: {
    type: Schema.Types.ObjectId,
    ref: "Playlist",
  },
});

export const Track = Mongoose.model("Track", trackSchema);
```

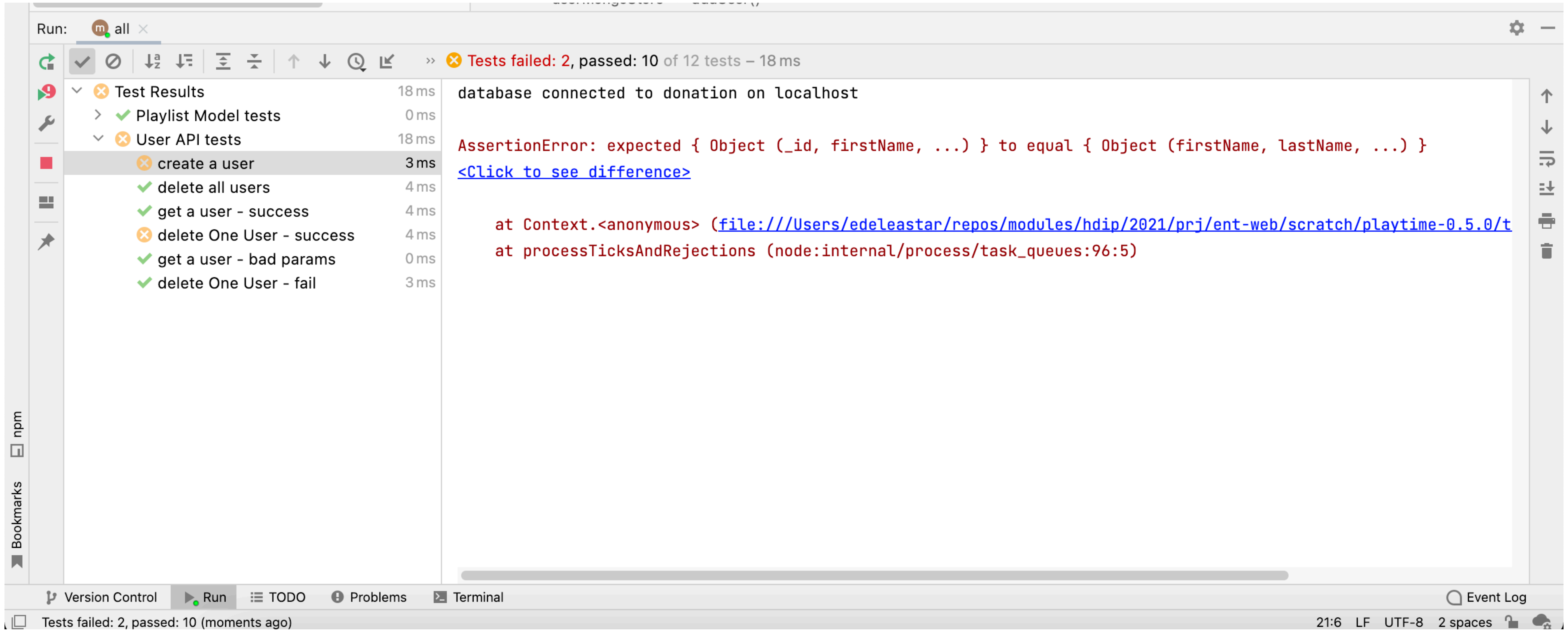
```
import { Track } from "./track.js";

export const trackMongoStore = {
  async getTracksByPlaylistId(id) {
    const tracks = await Track.find({ playlistid: id }).lean();
    return tracks;
  },
};
```

User Tests



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

- Some tests will fail initially



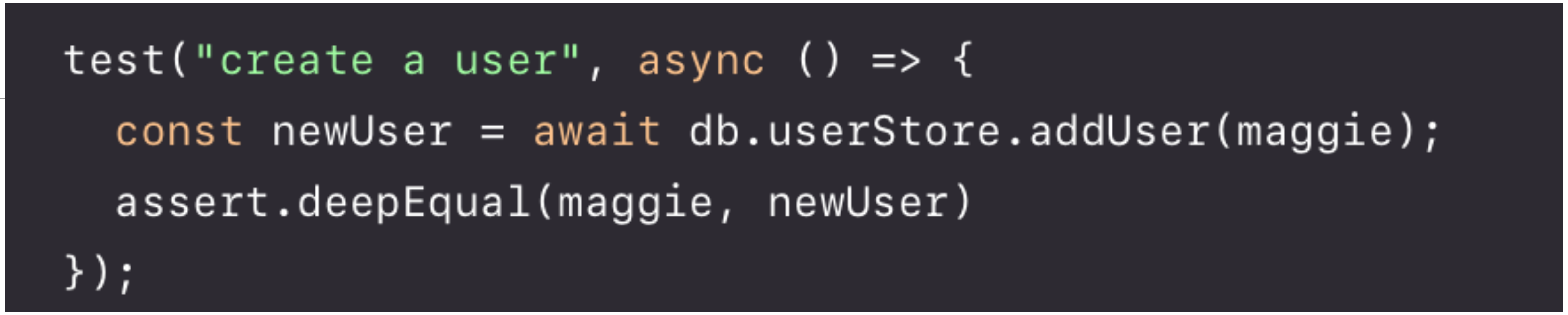
```
Test Results                     18 ms
  Playlist Model tests            0 ms
  User API tests                 18 ms
    create a user                 3 ms
    delete all users              4 ms
    get a user - success          4 ms
    delete One User - success     4 ms
    get a user - bad params       0 ms
    delete One User - fail        3 ms
```

```
database connected to donation on localhost

AssertionError: expected { Object (_id, firstName, ...) } to equal { Object (firstName, lastName, ...) }
<Click to see difference>

    at Context.<anonymous> (file:///Users/edeleastar/repos/modules/hdip/2021/prj/ent-web/scratch/playtime-0.5.0/t
    at processTicksAndRejections (node:internal/process/task_queues:96:5)
```

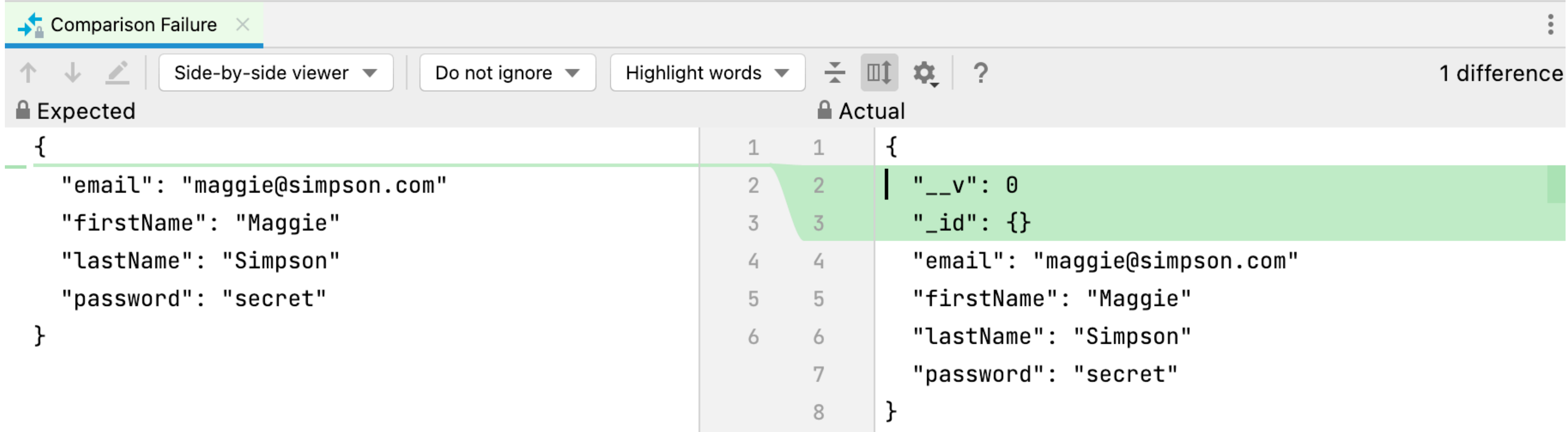
```
Tests failed: 2, passed: 10 (moments ago)                21:6  LF  UTF-8  2 spaces
```

User Tests

- Even with *lean()*, mongo will always include additional fields
  - _id : an object instead of a string
  - __v : an additional field



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



Comparison Failure

Side-by-side viewer   Do not ignore   Highlight words   1 difference

Expected

```
{
  "email": "maggie@simpson.com"
  "firstName": "Maggie"
  "lastName": "Simpson"
  "password": "secret"
}
```

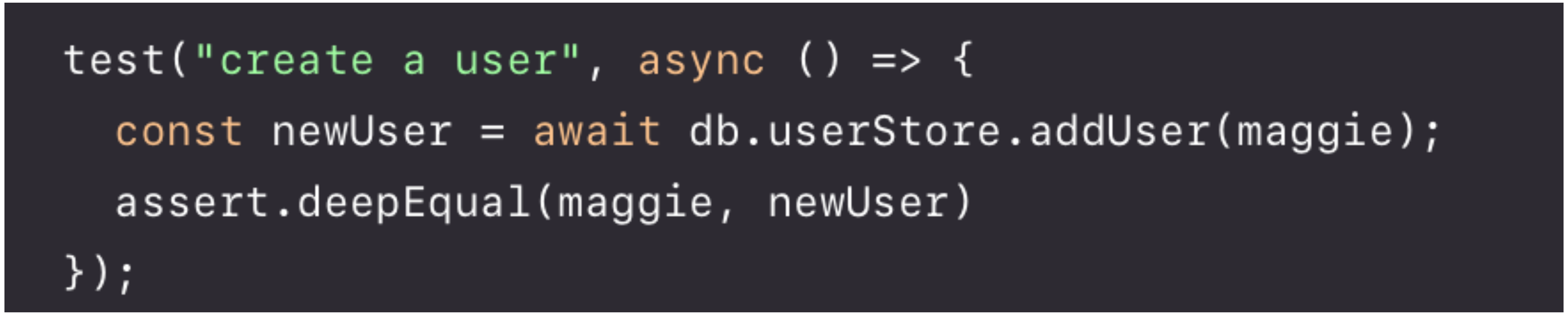
Actual

```
{
  "__v": 0
  "_id": {}
  "email": "maggie@simpson.com"
  "firstName": "Maggie"
  "lastName": "Simpson"
  "password": "secret"
}
```

assertSubset

Replace

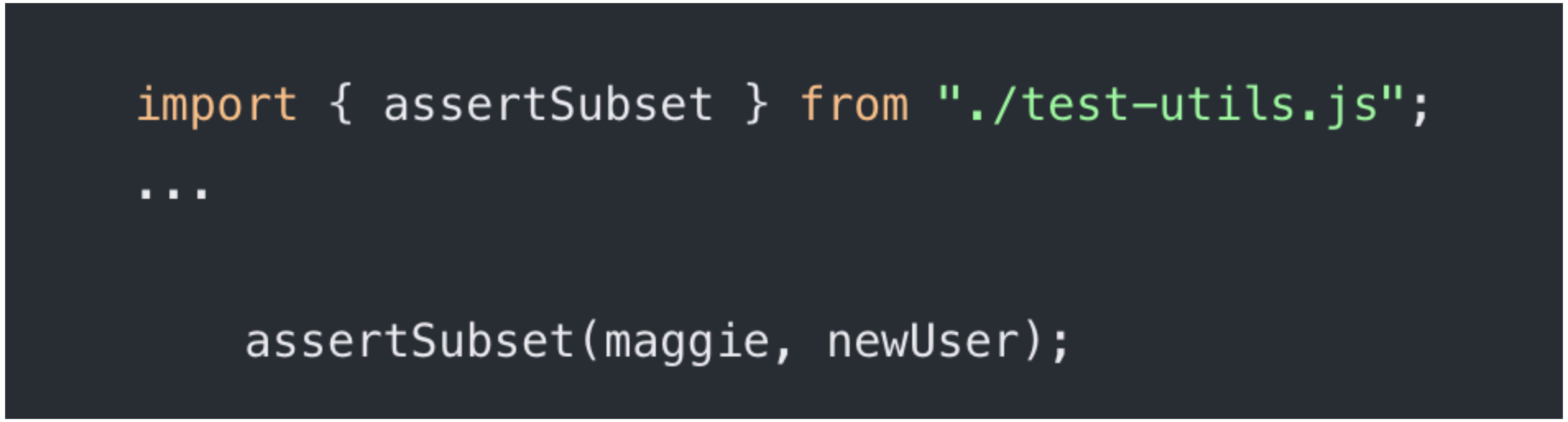
assert.deepEqual



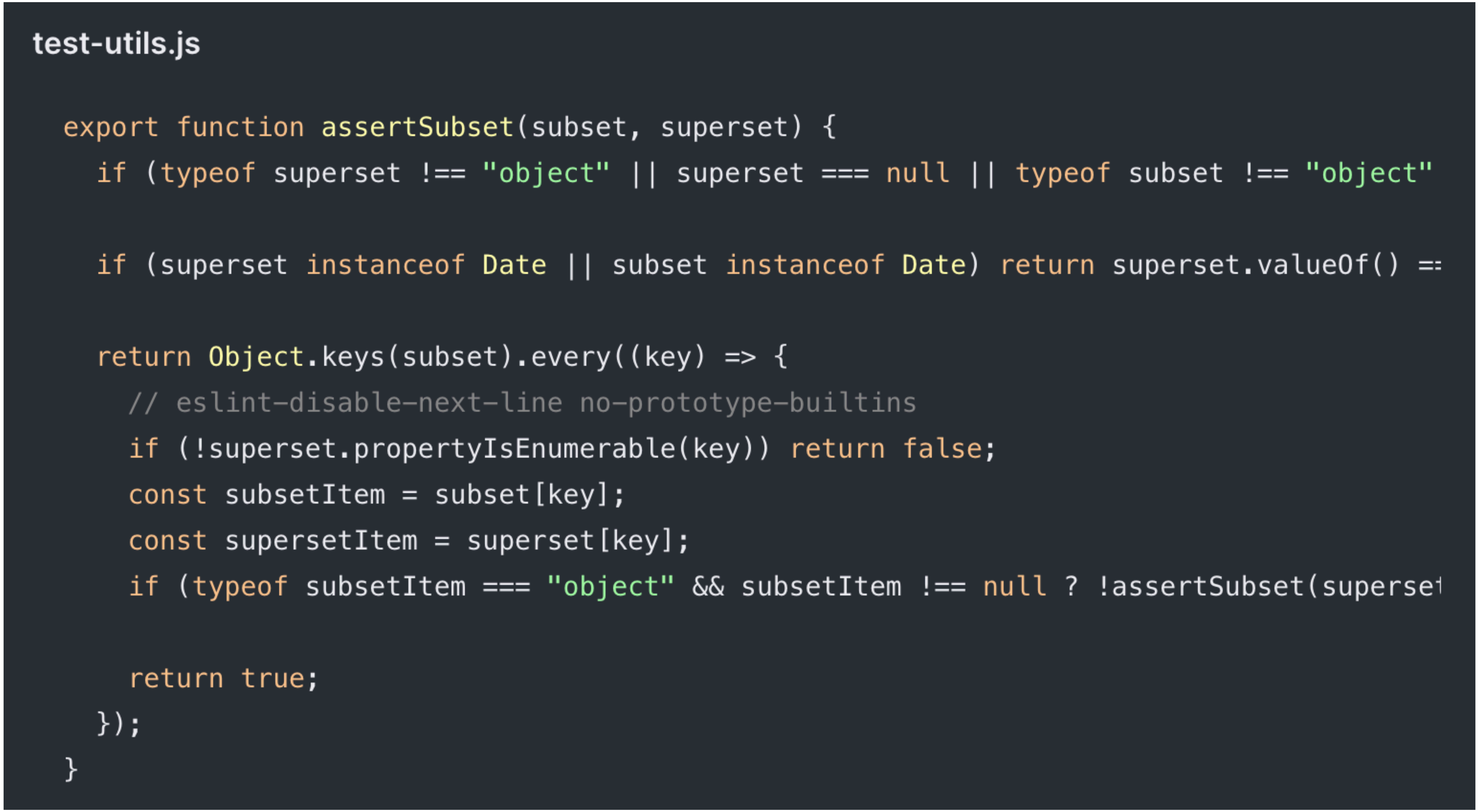
```
test("create a user", async () => {
  const newUser = await db.userStore.addUser(maggie);
  assert.deepEqual(maggie, newUser)
});
```

with

assertSubset



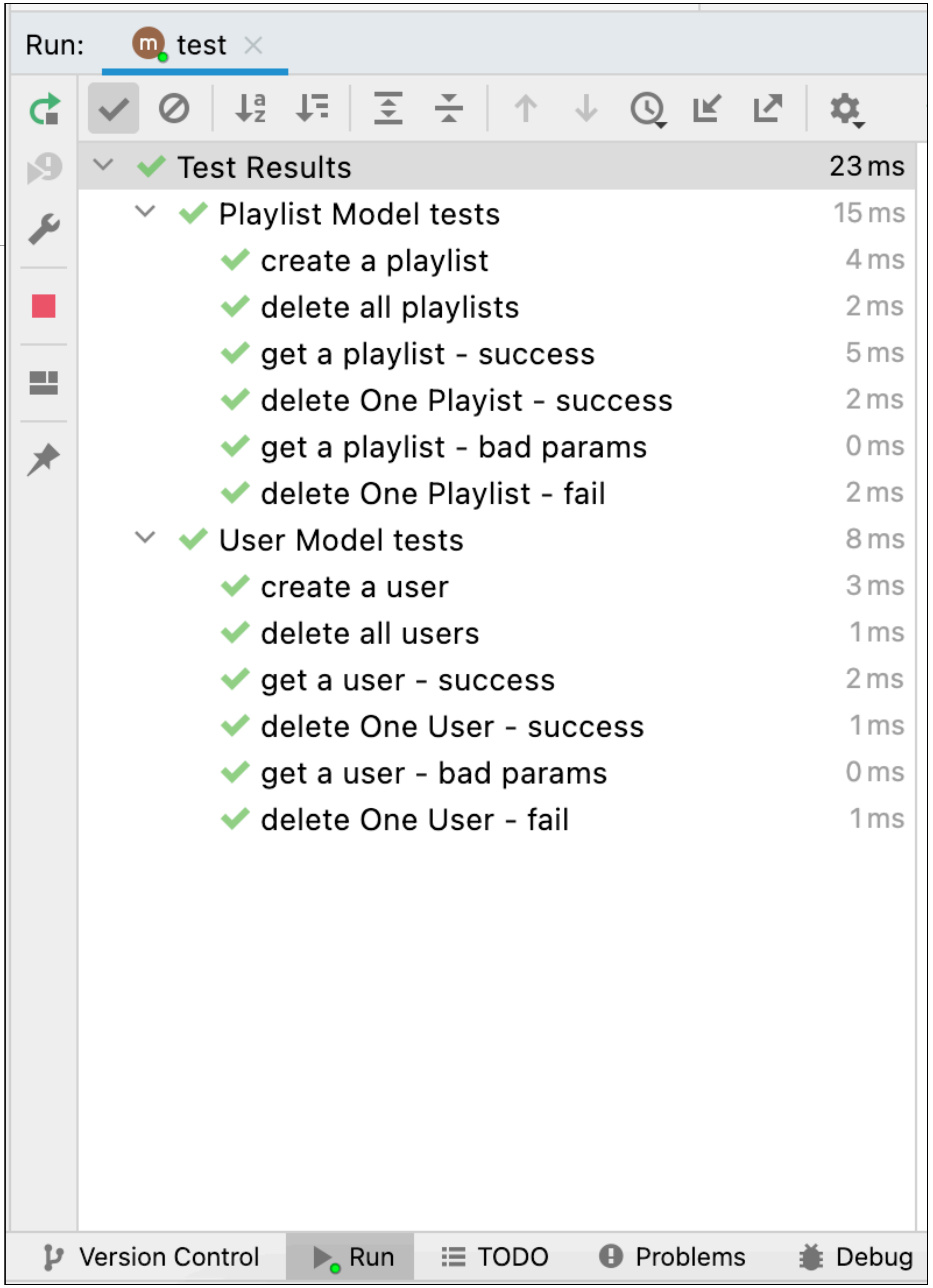
```
import { assertSubset } from "./test-utils.js";
...

    assertSubset(maggie, newUser);
```



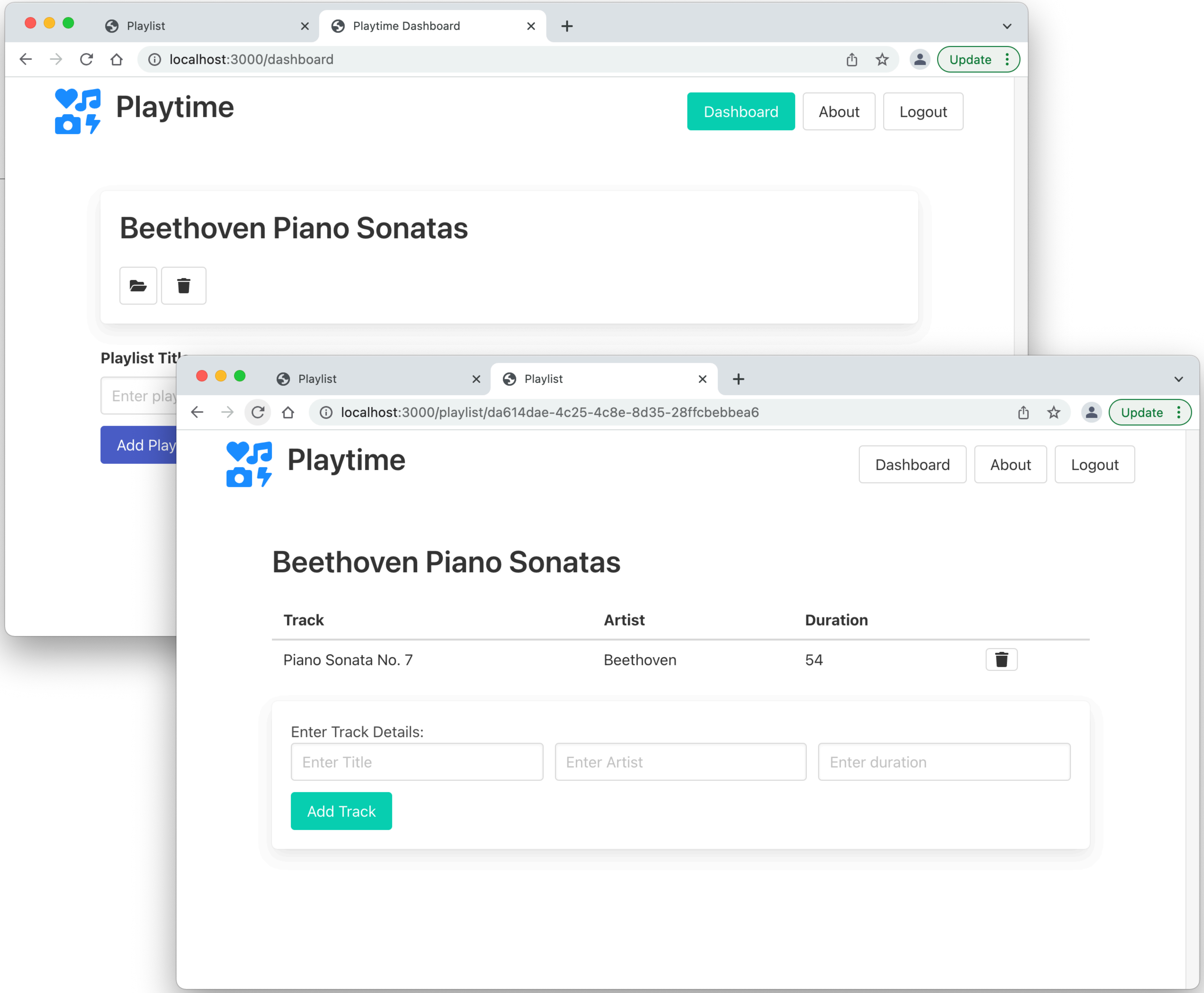
test-utils.js

```
export function assertSubset(subset, superset) {
  if (typeof superset !== "object" || superset === null || typeof subset !== "object"

  if (superset instanceof Date || subset instanceof Date) return superset.valueOf() ==

  return Object.keys(subset).every((key) => {
    // eslint-disable-next-line no-prototype-builtins
    if (!superset.propertyIsEnumerable(key)) return false;
    const subsetItem = subset[key];
    const supersetItem = superset[key];
    if (typeof subsetItem === "object" && subsetItem !== null ? !assertSubset(superset

    return true;
  });
}
```

- Confidence in Mongo store implementation significantly enhanced by successful unit tests



```
Test Results                              23 ms
  Playlist Model tests                    15 ms
    create a playlist                      4 ms
    delete all playlists                   2 ms
    get a playlist - success               5 ms
    delete One Playist - success           2 ms
    get a playlist - bad params            0 ms
    delete One Playlist - fail             2 ms
  User Model tests                         8 ms
    create a user                          3 ms
    delete all users                       1 ms
    get a user - success                   2 ms
    delete One User - success              1 ms
    get a user - bad params                0 ms
    delete One User - fail                 1 ms
```



Beethoven Piano Sonatas

Playlist Titl

Enter play

Add Play
Playlist
Playlist
localhost:3000/playlist/da614dae-4c25-4c8e-8d35-28ffcbebbea6
Update
Playtime
Dashboard
About
Logout

Beethoven Piano Sonatas

Track	Artist	Duration	
Piano Sonata No. 7	Beethoven	54	

Enter Track Details:

Enter Title

Enter Artist

Enter duration

Mongo Stores

Full Stack Web Development