Week 2 Server side

# Backend architecture

SAXION HOGESCHOOL

## What will we do today

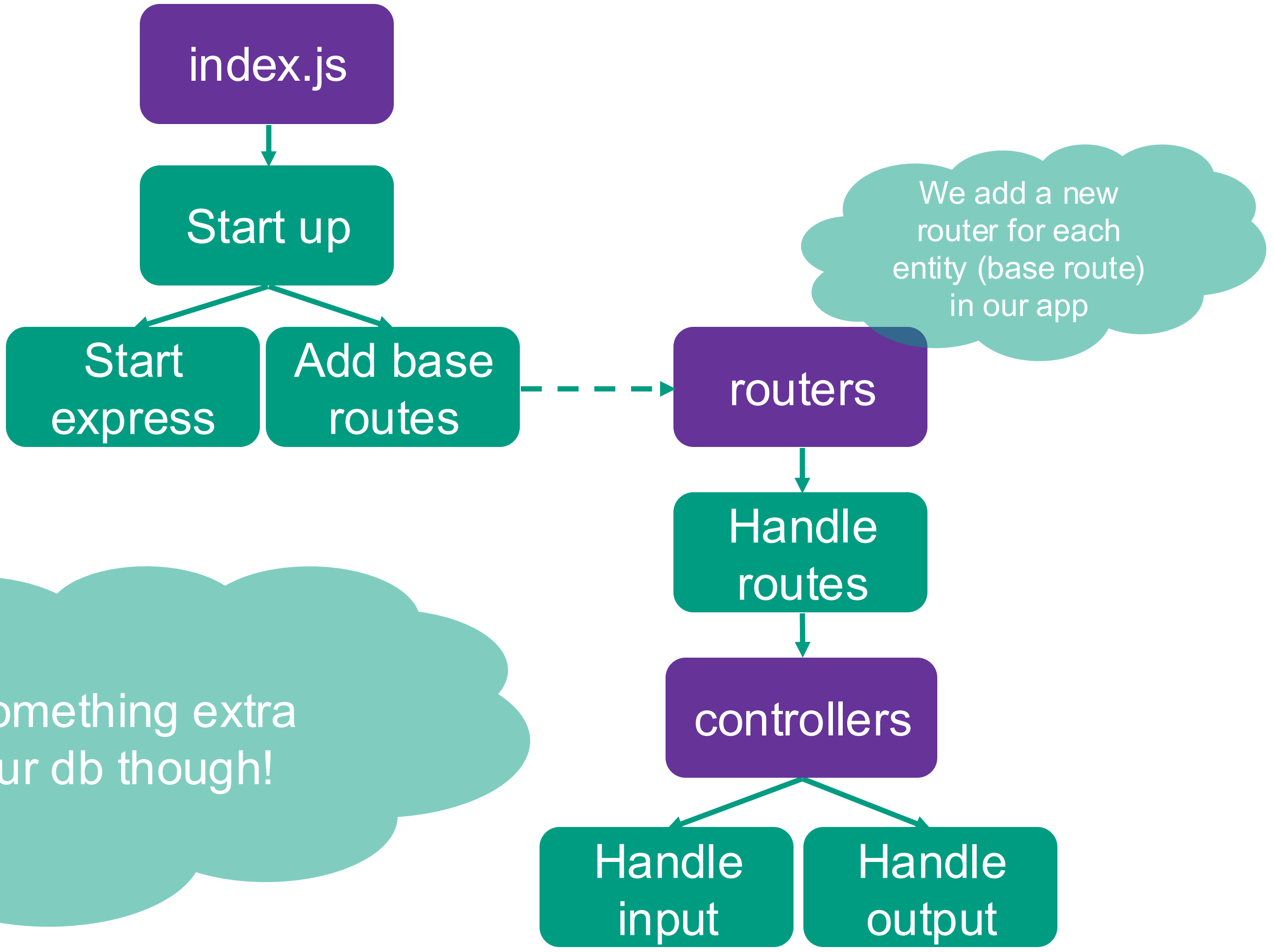
- Recap Web Basics
- ORM
- Node environments

## Let’s refresh your Web Basics backend architecture knowledge

*Let’s start with a quick brainstorm. How would we cut up our backend into multiple files again? And why?*

## Remember this?

*We'll approach it the same way for Web Advanced!*



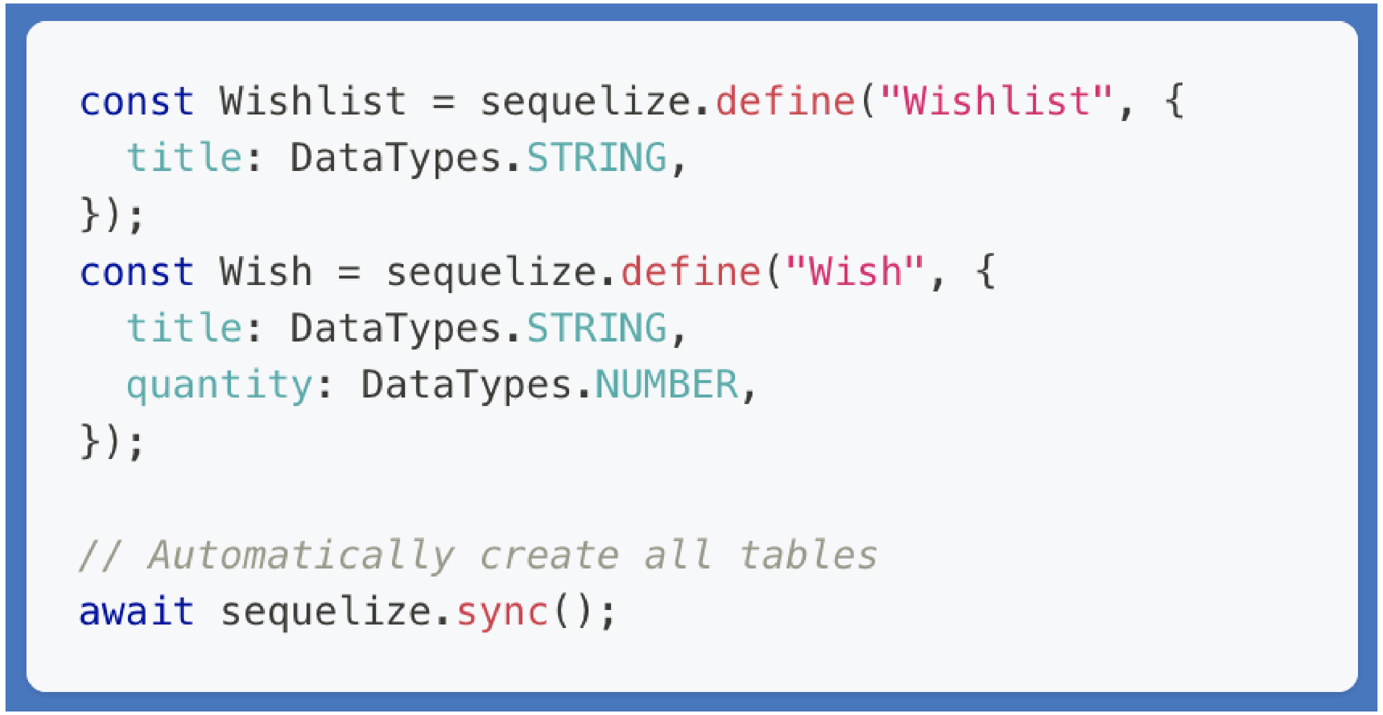
We will add something extra to manage our db though!

## ORM

- We will add an ORM to manage our database
- ORM = Object Relational Mapper
- Adding an ORM to our project means writing no, or in some cases less, SQL queries!
- It helps us set up tables, define associations between tables, read data, add or change data, and remove data
- ORMs are SQL dialect independent, so it does not matter which database you are using
- All backend languages have ORMs available. The one that we are going to use is called Sequelize

## ORM: Sequelize

- In Sequelize, we define models

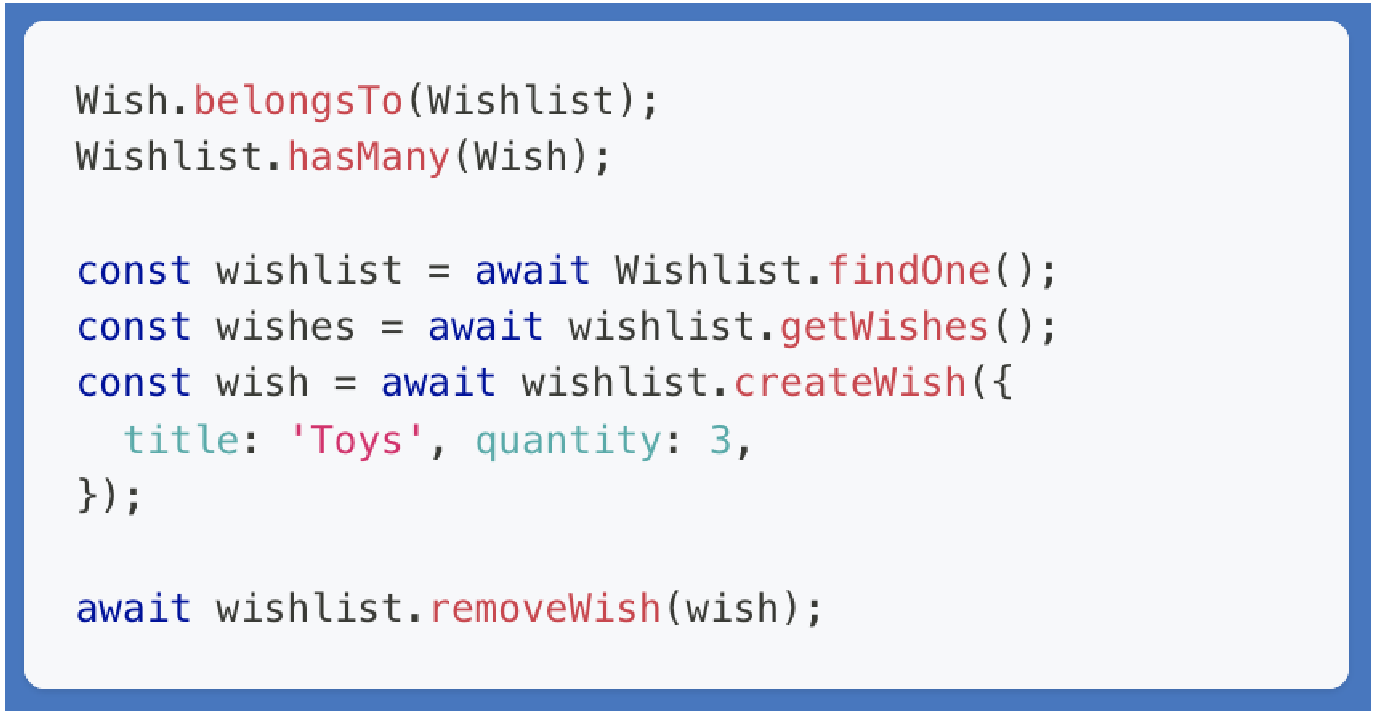


```javascript
const Wishlist = sequelize.define("Wishlist", {
  title: DataTypes.STRING,
});
const Wish = sequelize.define("Wish", {
  title: DataTypes.STRING,
  quantity: DataTypes.NUMBER,
});

// Automatically create all tables
await sequelize.sync();
```

## ORM: Sequelize

- Once defined, we can add associations and query the data



```
Wish.belongsTo(Wishlist);
Wishlist.hasMany(Wish);

const wishlist = await Wishlist.findOne();
const wishes = await wishlist.getWishes();
const wish = await wishlist.createWish({
  title: 'Toys', quantity: 3,
});

await wishlist.removeWish(wish);
```

## ORM: Sequelize

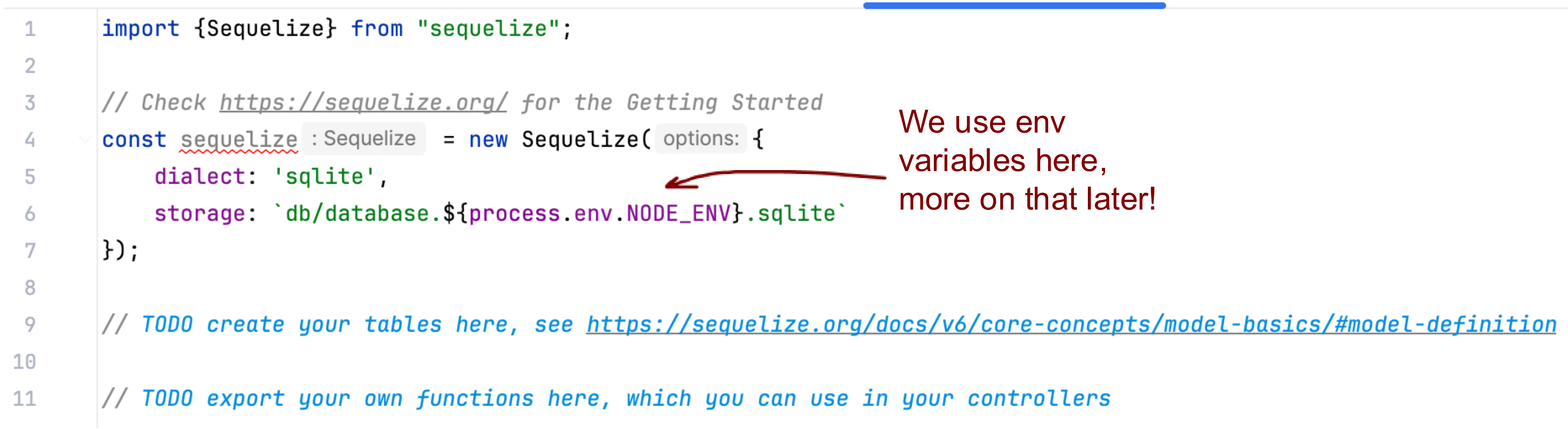
- The Sequelize website is GREAT to get you started

- Don’t trust ChatGPT too much when it comes to teaching you the basics; it’s very often trained on older versions of frameworks and therefore straight up wrong

- Read up on the documentation, and get started in your template after that

## ORM: Sequelize – template project

- We provided a basic setup for your Sequelize connection in the template



```
import {Sequelize} from "sequelize";

// Check https://sequelize.org/ for the Getting Started
const sequelize : Sequelize = new Sequelize( options: {
    dialect: 'sqlite',
    storage: `db/database.${process.env.NODE_ENV}.sqlite`
});

// TODO create your tables here, see https://sequelize.org/docs/v6/core-concepts/model-basics/#model-definition

// TODO export your own functions here, which you can use in your controllers
```

We use env variables here, more on that later!

- This setup uses sqlite, with different databases for different environments. You have to define the rest!

## Different environments

- Companies very often have multiple copies of the same application running, for different purposes

- One environment you're already familiar with: **development**
  - (Almost) always locally on your device
  - Does not matter if it breaks; no one is disadvantaged because of that (except you?)

- Another environment is often set up for **testing**
  - An online server, but often uses cheap hardware
  - Runs (automated) tests
  - Does not matter if it breaks; that's the purpose! Finding bugs early

## Different environments

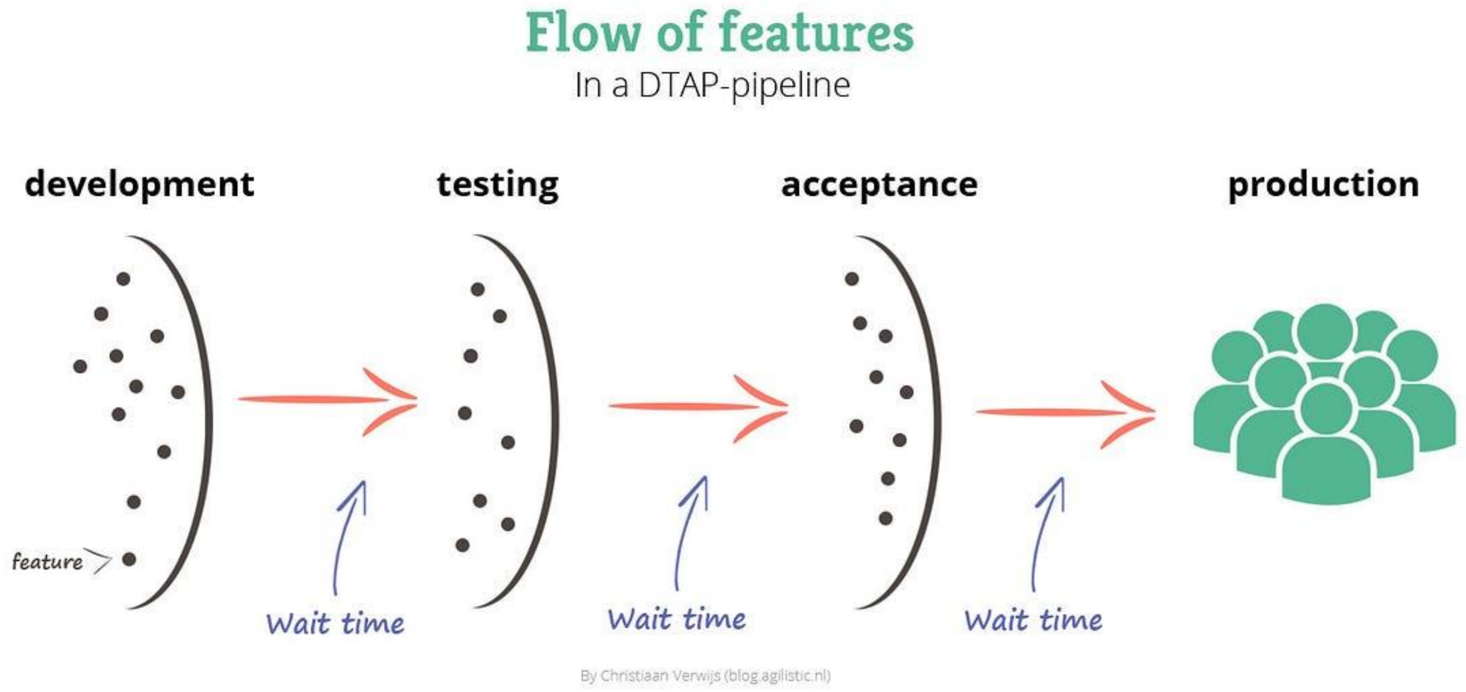
- Companies very often have multiple copies of the same application running, for different purposes

- A third environment is often set up for clients or users to test in a small group (**acceptance**)
  - Usually runs on similar server hardware as the live product
  - Is meant to replicate the live product, but without (sensitive) data
  - Shouldn’t break (testing should cover that), but if it does, no one is disadvantaged

- The final, most important copy: the live product (**production** environment)
  - Runs on (scaleable) proper hardware
  - Has all the (user)data
  - SHOULD NOT BREAK! Breaking can mean losing data, or loss of reputation

## Different environments

- Companies very often have multiple copies of the same application running, for different purposes
- So, four environments: **D**evelopment, **T**esting, **A**cceptance, **P**roduction



## Different environments – realistic case

- In our case, we are not going all-out

- We will have a **development** environment (localhost)
  - We will use this environment most. While developing, you'll use this one. The database can break, needs to be deleted on a regular basis, and the API calls change all the time

- We will have a **testing** environment, that we run tests on
  - This environment will have a predictable database, so you can actually test your API calls

- We will have a "**production**" environment (localhost)
  - The data here is valid and professional. This is what we use for grading

## Node environments

- Multiple environments often require a different setup per environment

- For example:
  - Where is the database located?
  - What is the secret we use for our tokens? (more on that in later weeks)
  - On what port do we run the server?

- This is very often done using environment variables, which Node can access easily

- Those environment variables are set manually, or often done through so-called env files

- Our template is already setup to use different env files

!! env files often contain very sensitive data, like secrets. You DO NOT commit env files to your git repo!

SAXION HOGESCHOOL

## Node environments

- When running our application, we tell node where to find the env file

```
"scripts": {
  "start": "node --env-file=.env.prod ./src/index.js",
  "dev": "node --env-file=.env.dev --watch ./src/index.js",
  "lint": "npx eslint ./src",
  "test": "vitest"
},
```

- Those env files contain key-value (String-String) pairs. Example .env.dev file:

```
NODE_ENV=dev
CUSTOM_VAR=Blabla
ANOTHER_ONE=fadsdsfsfdasfdfsdfsdasfdasfd
```

- We can then use these env variables in our code

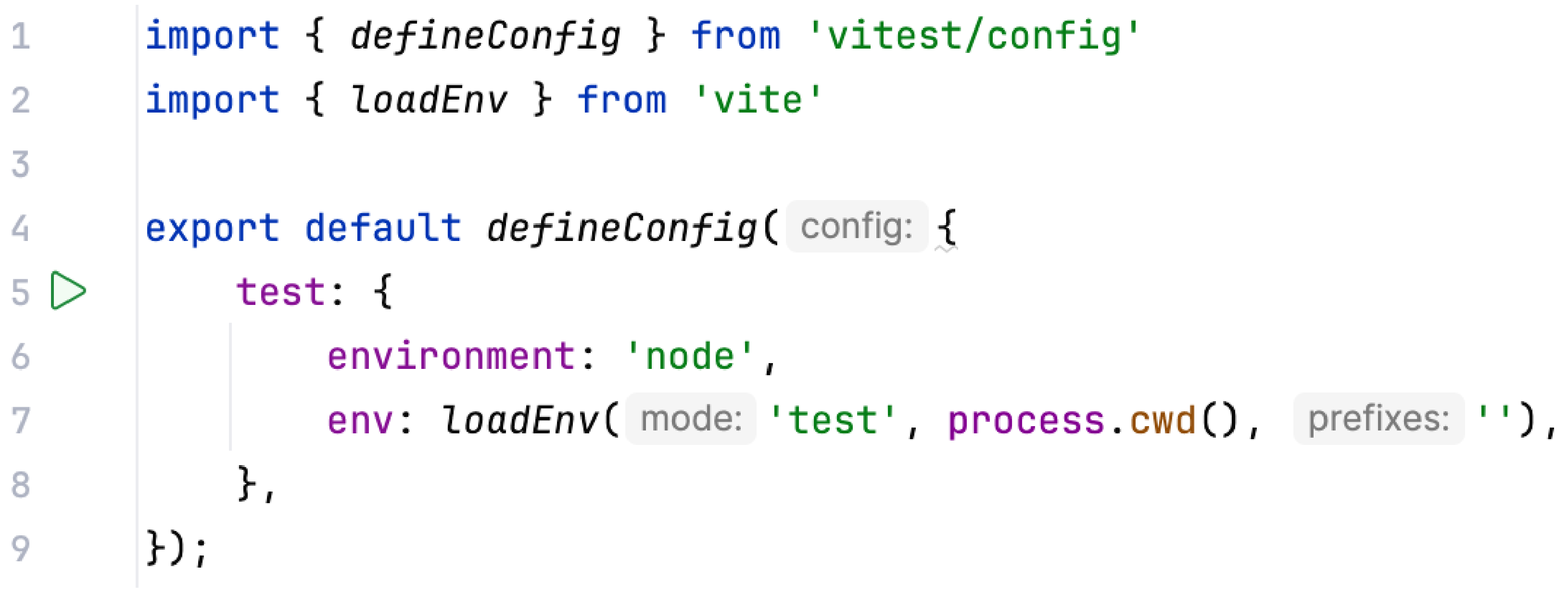
```
// Check if NODE_ENV environment variable is set, otherwise go to development mode
const nodeEnv = process.env.NODE_ENV || 'dev';
```

## Node environments – Vitest

- Vitest automatically sets the **NODE_ENV** variable to the value **test** at startup

- For other, custom values, create a .env.test file

- Vitest is setup in a way that it will automatically detect that file when running the tests.
  This is our vitest.config.js in the template:



```
import { defineConfig } from 'vitest/config'
import { loadEnv } from 'vite'

export default defineConfig( config: {
    test: {
        environment: 'node',
        env: loadEnv( mode: 'test', process.cwd(),  prefixes: ''),
    },
});
```

## Questions?

## Assignment:

Setup your .env files, start with Sequelize