Voorbereiding

Maken van de root van het project

- Maak een repository aan op `github` en clone die map in de workspace die jij wil
- In die map zet je vollegende bestanden

```
touch README.md;
touch .gitignore;
```

README.md

- In de `README.md` beschrijf je hoe je de applicatie start alsook het uitvoeren van testen.
- Dit is een template van hoe meestal een `README` er zal uitzien:

naam_repository

How to install

Voorbereiding

- Pull dit project met volgende githublink:

```
https://github.com/Robbe04/naam_repository
```

- In zowel de **Front-end** als de **Back-end** voer je dit uit om alle **dependancies** te installeren:

```
yarn install
```

Front-end

- Maak een `.env` file in de root van het project:

```
VITE_API_URL='http://localhost:9000'
```

Back-end

- Maak een `.env` file in de root van het project:

```
NODE_ENV=development
DATABASE_URL="mysql://USERNAME:PASSWORD@localhost:3306/NAME_DATABASE"
```

Gitgnore

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
lerna-debug.log*
.pnpm-debug.log*

# Diagnostic reports (https://nodejs.org/api/report.html)
report.[0-9]*.[0-9]*.[0-9]*.[0-9]*.json

# Runtime data
pids
*.pid
*.seed
*.pid.lock

# Directory for instrumented libs generated by jscoverage/JSCover
lib-cov

# Coverage directory used by tools like istanbul
coverage
*.lcov

# nyc test coverage
.nyc_output

# Grunt intermediate storage (https://gruntjs.com/creating-plugins#storing-task-files)
.grunt

# Bower dependency directory (https://bower.io/)
bower_components

# node-waf configuration
.lock-wscript

# Compiled binary addons (https://nodejs.org/api/addons.html)
build/Release

# Dependency directories
node_modules/
jspm_packages/
```

```
# Snowpack dependency directory (https://snowpack.dev/)
web_modules/

# TypeScript cache
*.tsbuildinfo

# Optional npm cache directory
.npm

# Optional eslint cache
.eslintcache

# Optional stylelint cache
.stylelintcache

# Microbundle cache
.rpt2_cache/
.rts2_cache_cjs/
.rts2_cache_es/
.rts2_cache_umd/

# Optional REPL history
.node_repl_history

# Output of 'npm pack'
*.tgz

# Yarn Integrity file
.yarn-integrity

# dotenv environment variable files
.env
.env.development.local
.env.test.local
.env.production.local
.env.local

# parcel-bundler cache (https://parceljs.org/)
.cache
.parcel-cache

# Next.js build output
.next
out

# Nuxt.js build / generate output
.nuxt
dist

# Gatsby files
.cache/
# Comment in the public line in if your project uses Gatsby and not Next.js
# https://nextjs.org/blog/next-9-1#public-directory-support
# public
```

```
# vuepress build output
.vuepress/dist

# vuepress v2.x temp and cache directory
.temp
.cache

# vitepress build output
**/.vitepress/dist

# vitepress cache directory
**/.vitepress/cache

# Docusaurus cache and generated files
.docusaurus

# Serverless directories
.serverless/

# FuseBox cache
.fusebox/

# DynamoDB Local files
.dynamodb/

# TernJS port file
.tern-port

# Stores VSCode versions used for testing VSCode extensions
.vscode-test

# yarn v2
.yarn/cache
.yarn/unplugged
.yarn/build-state.yml
.yarn/install-state.gz
.pnp.*
```

Front-end

Aanmaak

```
yarn create vite naamProjectFrontend --template react-swc;
yarn install;
```

- In deze cursus ga ik werken met `.tsx`-bestanden wat volgend commando nodig heeft
- Verwijder dan direct `App.tsx` want we gaat dit aanpassen via de `main.tsx`

```
yarn add -D typescript @types/react @types/react-dom
```

Mappenstructuur

- naamProjectFrontend
  - src
    - components
    - contexts
    - pages
    - api
      - index.js

Initialisatie

- Maak een `.yarnrc.yml`bestand in de root met volgende inhoud

```
nodeLinker: node-modules
```

- Installeer nu `ES-lint`

```
yarn add -D @stylistic/eslint-plugin
```

- Met inhoud:

```
import js from '@eslint/js';
import stylistic from '@stylistic/eslint-plugin';
import globals from 'globals';
import react from 'eslint-plugin-react';
import reactHooks from 'eslint-plugin-react-hooks';
import reactRefresh from 'eslint-plugin-react-refresh';

export default [
  { ignores: ['dist'] },
  {
    files: ['**/*.{js,jsx}'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
      parserOptions: {
        ecmaVersion: 'latest',
        ecmaFeatures: { jsx: true },
        sourceType: 'module',
      },
    },
    settings: { react: { version: '18.3' } },
```

```
    plugins: {
      react,
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
      '@stylistic': stylistic,
    },
    rules: {
      ...js.configs.recommended.rules,
      ...react.configs.recommended.rules,
      ...react.configs['jsx-runtime'].rules,
      ...reactHooks.configs.recommended.rules,
      'react/jsx-no-target-blank': 'off',
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
      '@stylistic/no-multiple-empty-lines': [
        'error',
        {
          max: 1,
          maxEOF: 1,
          maxBOF: 0,
        },
      ],
      '@stylistic/indent': ['error', 2, { SwitchCase: 1 }],
      '@stylistic/quotes': ['error', 'single'],
      '@stylistic/semi': ['error', 'always'],
      '@stylistic/comma-dangle': ['error', 'always-multiline'],
      '@stylistic/no-tabs': ['error'],
      '@stylistic/max-len': [
        'error',
        {
          code: 200,
          tabWidth: 2,
        },
      ],
      '@stylistic/arrow-parens': ['error', 'always'],
      '@stylistic/brace-style': ['error', '1tbs', { allowSingleLine: false }],
      '@stylistic/no-inner-declarations': 'off',
      'react/react-in-jsx-scope': 'off',
      'react/jsx-props-no-spreading': 'off',
      'react/prop-types': 'off',
    },
  },
];
```

- Zorg dat de `index.html` volgende inhoud omvat:
- Dit bevat namelijk css framework `bootstrap`

```
<!doctype html>
<html lang="en">
```

```
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Home</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.11.3/font/bootstrap-icons.min.css"></head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
  </body>
</html>
```

Helpers

- Ik vind het ook belangrijk om een `helpers/helpers.js`-bestand te maken met allerlei functies die je af en toe zult gebruiken
- Dit bestand komt ook een aantal keer in mijn project dus zorg zeker dat je dit hebt
- Dit is voorlopig hoe mijn bestand eruit ziet:

```
export const VITE_API_URL = import.meta.env.VITE_API_URL;
export const getDatumVanVandaag = new Date();
export const getMaandVanVandaag = new Date().getMonth();
export const getDagVanVandaag = new Date().getDay();
export const getJaarVanVandaag = new Date().getFullYear();
export const getUurVanVandaag = new Date().getHours();
export const getTijdInMiliseconden = new Date().getTime();
export const getTijdInUurMinutenSeconden = () => {
  return {
    uur : getDatumVanVandaag.getHours(),
    minuten : getDatumVanVandaag.getMinutes(),
    seconden : getDatumVanVandaag.getSeconds(),
    miliseconden : getDatumVanVandaag.getMilliseconds(),
  }
}

/**
 * Gaat terug naar de vorige pagina in de gebruiker zijn history
 */
export const handelTerugNaarVorigePagina = () => {
   window.history.back()
```

```
}

/**
 * Kopieert een tekst naar je klembord.
 * @param {string} text - De tekst die je wilt kopiëren naar het klembord.
 * @returns {Promise<void>}
 */
export const copyToClipboard = async (text) => {
   try {
     await navigator.clipboard.writeText(text);
     alert("Text gekopiëerd naar het klembord")
   } catch (err) {
     console.error('Fout bij kopiëren naar klembord: ', err);
   }
};

/**
 * Genereert een unieke numerieke ID op basis van timestamp en een willekeurig
getal
 * @returns {number} - Een uniek numeriek ID
 */
export const genereerUniekNummerId = () => {
   const timestamp = Date.now();
   const randomPart = Math.floor(Math.random() * 1000);
   return Number(`${timestamp}${randomPart}`);
};
```

Router

- Het is de bedoeling dat je urls er als volgende uitzien: `http://localhost:5173/voorbeeld`
- Installeer deze dependancy

```
yarn add react-router@~6.26.0 react-router-dom@~6.26.0
```

NotFound

- Maak alvast een `NotFound.jsx` in `src/pages` nog voor we de routes aanmaken:

```
import { useLocation, useNavigate } from 'react-router-dom';

const NotFound = () => {
  const { pathname } = useLocation();
  const navigate = useNavigate();

  /**
   * Gaat naar de home pagina
   */
  const handelGoHome = () => {
   navigate('/', {replace : true})
```

```
  }

  return (
    <div className="d-flex flex-column align-items-center justify-content-center
vh-100 text-center bg-primary text-white">
      <h1 className="display-1 fw-bold">Error 404</h1>
      <h2 className="mb-3">Pagina niet gevonden</h2>
      <p className="fs-5">
        Oeps! Deze url: "<span className="fw-bold text-light">{pathname}</span>"
bestaat niet.
      </p>
      <button onClick={handelGoHome} className="btn btn-light btn-lg mt-3">
        Terug naar home
      </button>
    </div>
  );
};
export default NotFound;
```

Navbar

- Maak ook alvast een navbarcomponent:

```
import { Link } from "react-router-dom";

export default function Navbar() {
  return (
    <nav className="navbar navbar-expand-lg navbar-dark bg-black shadow-lg border-
bottom border-white">
      <div className="container">
        <Link className="navbar-brand text-white fw-bold fs-4" to="/">
          React Template
        </Link>
        <button
          className="navbar-toggler border-white"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarNav"
          aria-controls="navbarNav"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span className="navbar-toggler-icon"></span>
        </button>
        <div className="collapse navbar-collapse" id="navbarNav">
          <ul className="navbar-nav ms-auto">
            <li className="nav-item">
              <Link className="nav-link text-white px-3 py-2 border border-white
rounded" to="/">
                Home
              </Link>
```

```
            </li>
            <li className="nav-item">
              <Link className="nav-link text-white px-3 py-2 border border-white
rounded ms-2" to="/login">
                Login
              </Link>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  );
}
```

Layout (voor navbar en kinderen daarvan)

- Alsook de `Layout.jsx` zodat de navbar op elke pagina komt:

```
import { Outlet, ScrollRestoration } from 'react-router-dom';
import NavBar from './Navbar';

export default function Layout() {
  return (
    <>
      <NavBar />
      <Outlet />
      <ScrollRestoration />
    </>
  );
}
```

Home

- Alsook de `Home.jsx`

```
export default function Home() {
  return (
    <div
      className="d-flex justify-content-center align-items-center text-center
text-light"
      style={{height: '100vh', background: 'linear-gradient(135deg,rgb(3, 255,
255),rgb(49, 94, 94))',}}
    >
      <div>
        <h1 className="display-1 font-weight-bold">React Template</h1>
        <h2 className="lead">Een eenvoudige React-template website met alle nodige
componenten mogelijk.</h2>
      </div>
    </div>
```

```
  );
}
```

Werken met data uit de REST-API

- Ondertussen hebben we in de `back-end` heel onze API gemaakt
- Vanaf dit punt wordt dus aangeraden dat je heel dat gedeelte hebt gemaakt
- Vanaf nu gaan we enkele pagina's maken die REST-API data gebruiken
- Voeg dit commando toe:

```
yarn add axios swr
```

- Maak een `src/api/index.js` bestand met alle endpoints die je gaat aanroepen

```
import axios from 'axios';

const baseURL = import.meta.env.VITE_API_URL + "/api";

/**
 * Haal alle items op van een bepaald endpoint.
 * @param {string} url - Het endpoint van de API dat je wil oproepen.
 * @returns {Promise<Array>} - Een array met alle opgehaalde items.
 */
export const getAll = async (url) => {
   const { data } = await axios.get(`${baseURL}/${url}`);
   return data.items;
};

 /**
 * Haal gegevens op van een specifiek endpoint via zijn ID.
 * @param {string} url - Het endpoint van de API dat je wil oproepen inclusief ID.
 * @param {Object} param - Object met de ID van het te verwijderen item.
 * @returns {Promise<Object>} - De gegevens van het opgehaalde item.
 */
export const getById = async (url, {arg : id}) => {
   const { data } = await axios.get(`${baseURL}/${url}/${id}`);
   return data;
 };

 /**
  * Verwijder een specifiek item via zijn ID.
  * @param {string} url - Het endpoint van de API dat je wil oproepen.
  * @param {Object} param - Object met de ID van het te verwijderen item.
  * @param {number|string} param.arg - De ID van het item.
  */
 export const deleteById = async (url, { arg: id }) => {
   await axios.delete(`${baseURL}/${url}/${id}`);
 };
```

```
 /**
  * Werk een specifiek item bij.
  * @param {string} url - Het endpoint van de API dat je wil oproepen.
  * @param {Object} param - Object met de bij te werken gegevens.
  * @param {Object} param.arg - De gegevens die geüpdatet moeten worden.
  * @returns {Promise<Object>} - De bijgewerkte gegevens.
  */
 export const updateById = async (url, { arg: id, body }) => {
   const { data } = await axios.put(`${baseURL}/${url}/${id}`, body);
   return data;
 };

 /**
  * Voegt iets toe aan de REST-api
  * @param url - Het endpoint van de API dat je wil oproepen.
  * @param param1 - Object met de bij te werken gegevens.
  * @returns {Promise<Object>} - Het nieuw aangemaakte item
  */
 export const post = async(url, {arg}) => {
  const {data} = await axios.post(`${baseURL}/${url}`, arg)
  return data
 }

 /**
  * Sla een nieuw item op of werk een bestaand item bij.
  * @param {string} url - Het endpoint van de API dat je wil oproepen.
  * @param {Object} param - Object met de gegevens van het item.
  * @param {Object} param.arg - Object met het ID (optioneel) en de overige
gegevens.
  */
 export const save = async (url, { arg: { id, ...data } }) => {
   await axios({
     method: id ? 'PUT' : 'POST',
     url: `${baseURL}/${url}/${id ?? ''}`,
     data,
   });
 };
```

Laadindicator

- Maak een loader bestand `components/Loader.jsx` voor het ophalen van data indien het niet lukt

```
export default function Loader() {
   return (
     <div
       className="d-flex flex-column align-items-center justify-content-center"
       style={{ minHeight: '100vh' }}
     >
       <div className="spinner-border" role="status" aria-live="polite" aria-
label="Loading...">
         <span className="visually-hidden">Loading...</span>
```

```
        </div>
      </div>
    );
  }
```

Error

- Maak ook een `components/Error` file
- Laten we starten het maken van een User en de lijst tonen:

```

```

AsyncData

- Maak nog één laatste data/errorcomponent namelijk `components/AsyncData`

```
import Loader from './Loader';
import Error from './Error';

export default function AsyncData({
  loading,
  error,
  children,
}) {
  if (loading) {
    return <Loader />;
  }

  return (
    <>
      <Error error={error} />
      {children}
    </>
  );
}
```

UserList.jsx

- Vanaf hier beginnen we met het maken van echte pagina's
- Maak `pages/user/UserList.jsx`

```
import useSWR from "swr";
import User from "../../components/user/User"
import AsyncData from "../../components/AsyncData";
import * as api from '../../api/index';
```

```
export default function UserList() {
   const { data: users = [], error, isLoading: loading } = useSWR("users",
api.getAll);
  return (
    <div className="text-center mt-4">
      <h1>All of this template&apos;s users:</h1>
      <AsyncData error={error} loading={loading}>
        <div
          style={{
            display: "grid",
            gridTemplateColumns: "repeat(auto-fit, minmax(250px, 3fr))",
            gap: "1rem",
          }}
        >
          {Array.isArray(users) ? (
            users?.map((user) => (
              <div
                key={user?.userId}
                style={{
                  border: "1px solid #ccc",
                  padding: "1rem",
                  borderRadius: "8px",
                  boxShadow: "0 2px 5px rgba(0,0,0,0.1)",
                }}
              >
                <User key={user.userId} user={user} />
              </div>
            ))
          ) : (
            <p>No users found.</p>
          )}
        </div>
      </AsyncData>
    </div>
  );
}
```

User.jsx

- Maak een `components/user/User.jsx` aan

```
export default function User({user}){

   return (
      <>
         <h2>{user?.firstName} {user?.lastName}</h2>
      </>
   )
}
```

ProductList.jsx

- Maak nu `pages/product/ProductList.jsx`

```
import useSWR from "swr";
import * as api from "../../api/index";
import Product from "../../components/products/Product";
import AsyncData from "../../components/AsyncData";

export default function ProductsList() {
  const { data: products = [], isLoading: loading, error } = useSWR("products",
api.getAll);

  return (
    <div className="container mt-4">
    <AsyncData error={error} loading={loading}>
      {Array.isArray(products) && products.length > 0 ? (
        <div style={{
          display: "grid",
          gridTemplateColumns: "repeat(auto-fit, minmax(18rem, 1fr))",
          gap: "1rem",
          justifyContent: "center"
        }}>
          {products.map((product) => (
            <Product key={product.productId} product={product} />
          ))}
        </div>
      ) : (
        <p>No products available</p>
      )}
    </AsyncData>
    </div>
  );
}
```

Product.jsx

- Gevolgt door `components/product/Product.jsx`

```
import { Link } from "react-router-dom";
import * as helpers from "../../../helpers/helpers";

const baseApiUrl = helpers.VITE_API_URL;

export default function Product({ product }) {
  return (
    <div className="card" style={{ width: "18rem" }}>
      <div className="text-center">
      <img
        src={`${baseApiUrl}/img/${product.image}`}
```

```
          alt={`Picture of product: ${product.productName}`}
          width={"200px"}
          height={"200px"}
          style={{objectFit : "cover"}}
        />
        </div>
        <div className="card-body">
          <h5 className="card-title">{product.productName}</h5>
          <p className="card-text">
            Delivered by:{" "}
            <Link to={`/products/suppliers/${product.supplier.supplierId}`}>
              {product.supplier.firstName} {product.supplier.lastName}
            </Link>
          </p>
          <p className="card-text">
            Price: ${product.unitPrice}
          </p>
          <div className="text-center">
            <Link to={`/products/${product.productId}`} className="btn btn-primary">
              View Product
            </Link>
          </div>
        </div>
      </div>
    );
}
```

ProductDetail.jsx

- Nu is het de bedoeling dat als we klikken op een `Product` dat we de details ervan kunnen zien
- Deze functie gaan we volbrengen via `pages/ProductDetail.jsx`

```
import { useParams } from 'react-router-dom';
import useSWR from 'swr';
import * as api from "../../api/index";
import AsyncData from '../../components/AsyncData';
import * as helpers from "../../../helpers/helpers";
import { Link } from 'react-router-dom';

const baseApiUrl = helpers.VITE_API_URL;

export default function ProductDetail() {
  const { id } = useParams();
  const idNumber = Number(id);

  const { data: products = [], error, isLoading } = useSWR("products",
api.getAll);
  const product = products.find(product => product.productId === idNumber);

  if (!product) {
    return <div className="text-center py-5"><p className="alert alert-
```

```
warning">Product met ID {idNumber} is niet gevonden</p></div>;
  }

  const handelTerugNaarVorigePagina = () => {
    helpers.handelTerugNaarVorigePagina();
  }

  return (
    <AsyncData error={error} loading={isLoading}>
      <div className="container d-flex align-items-center justify-content-center
min-vh-100">
        <div className="card shadow-lg p-4" style={{ maxWidth: '600px', width:
'100%' }}>
          <div className="row g-3">
            <div className="col-12 text-center">
              <img
                src={product.image ? `${baseApiUrl}/img/${product.image}` :
'/placeholder.jpg'}
                alt={`Picture of product: ${product.productName}`}
                className="img-fluid rounded border"
                style={{ maxHeight: '250px', objectFit: 'cover' }}
              />
            </div>
            <div className="col-12">
              <h3 className="border-bottom pb-2 text-center">{product.productName}
</h3>
              <p className="text-muted text-center">{product.description || 'Geen
beschrijving beschikbaar'}</p>
              <h5 className="fw-bold text-primary">Prijs:
${product.unitPrice.toFixed(2)}</h5>
              <h6 className="mt-3">Geleverd door:</h6>
              <p>
                <Link to={`/products/suppliers/${product.supplier.supplierId}`}
className="text-decoration-none fw-bold">
                  {product.supplier.firstName} {product.supplier.lastName}
                </Link>
              </p>
              <button className='text-center btn btn-primary' onClick=
{handelTerugNaarVorigePagina}>
                Back
              </button>
            </div>
          </div>
        </div>
      </div>
    </AsyncData>
  );
}
```

main.jsx

- De `main.jsx` waar we onze routes gaan vastleggen ziet er dan als volgende uit:

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import Home from './components/home/Home.jsx';
import NotFound from './pages/NotFound.jsx';
import Layout from './components/Layout.jsx';
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
import UserList from './pages/user/UserList.jsx';
import ProductsList from "./pages/products/ProductsList.jsx";
import ProductDetail from "./pages/products/ProductDetail.jsx"

const router = createBrowserRouter([
  {
    element: <Layout />, children: [
      {path: '/', element: <Home />},
      {path: '*', element: <NotFound />},
      {path: '/users', element : <UserList/>},
      {path: '/products', children : [
        {
          index : true,
          element : <ProductsList />
        },
        {
          path: ':id',
          element : <ProductDetail/>
        }
      ]}
    ]
  }
]);

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>,
);
```

Front-end Authenticatie

Inleiding

- Dit hoofdstuk staat apart van de normale front-end aangezien je hier heel wat gaat moeten aanpassen en indien je een statische website wil bouwen heb je dit niet nodig
- In dit hoofdstuk ga je zien hoe je een fully functional Login-component moet bouwen
- Je gaat in vergelijking met het vorige hoofdstuk ook heel veel moeten aanpassen!
- Je moet dit stuk **EERST** toevoegen via de Back-end voor je met dit stuk verderkan

Authenticatie Context

- We starten met het bouwen van een `API-Context` om de ingelogde user bij te houden in `src/contexts/Authentication.context.jsx`:

```
import {createContext, useState, useCallback, useMemo} from 'react';
import useSWRMutation from 'swr/mutation';
import * as api from '../api';
import useSWR from 'swr';

export const JWT_TOKEN_KEY = 'jwtToken';
export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [token, setToken] = useState(localStorage.getItem(JWT_TOKEN_KEY));

  const {data: user, loading: userLoading, error: userError} = useSWR(
    token ? 'users/me' : null, api.getById
  );

  const {trigger: doLogin, isMutating: loginLoading, error: loginError} =
useSWRMutation(
    'sessions', api.post
  );

  const login = useCallback(async (emailadres, password) => {
      try {
        const { token } = await doLogin({emailadres, password});

        setToken(token);

        localStorage.setItem(JWT_TOKEN_KEY, token);

        return true;
      } catch (error) {
          console.error(error);
          return false;
      }
    },
    [doLogin],
  );

  const logout = useCallback(() => {
    setToken(null);

    localStorage.removeItem(JWT_TOKEN_KEY);
  }, []);

  const value = useMemo(
    () => ({
      token,
      user,
      error: loginError || userError,
      loading: loginLoading || userLoading,
```

```
      isAuthed: Boolean(token),
      ready: !userLoading,
      login,
      logout,
    }),
    [
      token,
      user,
      loginError,
      loginLoading,
      userError,
      userLoading,
      login,
      logout,
    ],
  );

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
};
```

- Dit component roepen we dan op in `src/contexts/authentication.js`:

```
import {createContext, useState, useCallback, useMemo} from 'react';
import useSWRMutation from 'swr/mutation';
import * as api from '../api';
import useSWR from 'swr';

export const JWT_TOKEN_KEY = 'jwtToken';
export const AuthenticationContext = createContext();

export const AuthenticationProvider = ({ children }) => {
  const [token, setToken] = useState(localStorage.getItem(JWT_TOKEN_KEY));

  const {data: user, loading: userLoading, error: userError} = useSWR(
    token ? 'users/me' : null, api.getById
  );

  const {trigger: doLogin, isMutating: loginLoading, error: loginError} =
useSWRMutation(
    'sessions', api.post
  );

  const login = useCallback(async (email, password) => {
      try {
        const { token } = await doLogin({email, password});

        setToken(token);

        localStorage.setItem(JWT_TOKEN_KEY, token);

        return true;
      } catch (error) {
```

```
            console.error(error);
            return false;
        }
      },
      [doLogin],
    );

    const logout = useCallback(() => {
      setToken(null);

      localStorage.removeItem(JWT_TOKEN_KEY);
    }, []);

    const value = useMemo(
      () => ({
        user,
        error: loginError || userError,
        loading: loginLoading || userLoading,
        login,
        logout,
      }),
      [user, loginError, loginLoading, userError, userLoading, login, logout],
    );

    return <AuthenticationContext.Provider value={value}>{children}
</AuthenticationContext.Provider>;
};
```

- Nu wrappen we dit alles in de `main.jsx`:

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import Home from './components/home/Home.jsx';
import NotFound from './pages/NotFound.jsx';
import Layout from './components/Layout.jsx';
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
import UserList from './pages/user/UserList.jsx';
import ProductsList from "./pages/products/ProductsList.jsx";
import ProductDetail from "./pages/products/ProductDetail.jsx"
import { AuthenticationProvider } from './contexts/Authentication.context.jsx';
const router = createBrowserRouter([
  {
    element: <Layout />, children: [
      {path: '/', element: <Home />},
      {path: '*', element: <NotFound />},
      {path: '/users', element : <UserList/>},
      {path: '/products', children : [
        {
          index : true,
          element : <ProductsList />
        },
        {
```

```
          path: ':id',
          element : <ProductDetail/>
        }
      ]}
    ]
  }
]);

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <AuthenticationProvider>
        <RouterProvider router={router} />
    </AuthenticationProvider>
  </StrictMode>,
);
```

- Dankzij dit autheticatieproces moeten we ook onze `api/index.js` aanpassen voor nu rekening te houden met de bearer-tokens:

```
import axiosRoot from 'axios';
import { JWT_TOKEN_KEY } from '../contexts/Authentication.context';

const baseURL = import.meta.env.VITE_API_URL + "/api";

export const axios = axiosRoot.create({
  baseURL: baseURL,
});

axios.interceptors.request.use((config) => {
  const token = localStorage.getItem(JWT_TOKEN_KEY);

  if (token) {
    config.headers['Authorization'] = `Bearer ${token}`;
  }

  return config;
});

/**
 * Haal alle items op van een bepaald endpoint.
 * @param {string} url - Het endpoint van de API dat je wil oproepen.
 * @returns {Promise<Array>} - Een array met alle opgehaalde items.
 */
export const getAll = async (url) => {
   const { data } = await axios.get(`${baseURL}/${url}`);
   return data.items;
};

 /**
 * Haal gegevens op van een specifiek endpoint via zijn ID.
 * @param {string} url - Het endpoint van de API dat je wil oproepen inclusief ID.
```

```
 * @param {Object} param - Object met de ID van het te verwijderen item.
 * @returns {Promise<Object>} - De gegevens van het opgehaalde item.
 */
export const getById = async (url, {arg : id}) => {
  const { data } = await axios.get(`${baseURL}/${url}/${id}`);
  return data;
};

/**
 * Verwijder een specifiek item via zijn ID.
 * @param {string} url - Het endpoint van de API dat je wil oproepen.
 * @param {Object} param - Object met de ID van het te verwijderen item.
 * @param {number|string} param.arg - De ID van het item.
 */
export const deleteById = async (url, { arg: id }) => {
  await axios.delete(`${baseURL}/${url}/${id}`);
};

/**
 * Werk een specifiek item bij.
 * @param {string} url - Het endpoint van de API dat je wil oproepen.
 * @param {Object} param - Object met de bij te werken gegevens.
 * @param {Object} param.arg - De gegevens die geüpdatet moeten worden.
 * @returns {Promise<Object>} - De bijgewerkte gegevens.
 */
export const updateById = async (url, { arg: id, body }) => {
  const { data } = await axios.put(`${baseURL}/${url}/${id}`, body);
  return data;
};

/**
 * Voegt iets toe aan de REST-api
 * @param url - Het endpoint van de API dat je wil oproepen.
 * @param param1 - Object met de bij te werken gegevens.
 * @returns {Promise<Object>} - Het nieuw aangemaakte item
 */
export const post = async(url, {arg}) => {
 const {data} = await axios.post(`${baseURL}/${url}`, arg)
 return data
}

/**
 * Sla een nieuw item op of werk een bestaand item bij.
 * @param {string} url - Het endpoint van de API dat je wil oproepen.
 * @param {Object} param - Object met de gegevens van het item.
 * @param {Object} param.arg - Object met het ID (optioneel) en de overige
gegevens.
 */
export const save = async (url, { arg: { id, ...data } }) => {
  await axios({
    method: id ? 'PUT' : 'POST',
    url: `${baseURL}/${url}/${id ?? ''}`,
    data,
```

```
    });
  };
```

Gebruikers een account laten aanmaken

LabelInput-component

- Dit is een herbruikbaar stuk code dat wordt gebruikt waneer er inputvelden worden gevalideerd

```
import { useFormContext } from 'react-hook-form';

export default function LabelInput({
  label,
  name,
  type,
  validationRules,
  ...rest
}) {
  const {
    register,
    formState: { errors, isSubmitting },
  } = useFormContext();

  const hasError = name in errors;

  return (
    <div className='mb-3'>
      <label htmlFor={name} className='form-label'>
        {label}
      </label>
      <input
        {...register(name, validationRules)}
        id={name}
        type={type}
        disabled={isSubmitting}
        className='form-control'
        {...rest}
      />
      {hasError ? (
        <div className='form-text text-danger'>{errors[name].message}</div>
      ) : null}
    </div>
  );
}
```

SelectList-component

- Dit is hetzelfde principe als het LabelInput-component maar dan voor een **select-list** via `components/registration/SelectList.jsx`

Login-component

- installeer eerst volgende formulier-package:

```
yarn add react-hook-form
```

- Dit is het component waarin users zullen inloggen in `pages/registration/Login.jsx`:

```
import { useCallback } from 'react';
import { useNavigate, useLocation } from 'react-router-dom';
import { FormProvider, useForm } from 'react-hook-form';
import LabelInput from '../../components/registration/LabelInput';
import { useAuthentication } from '../../contexts/authentication';
import Error from '../../components/Error';

const validationRules = {
  emailadres: {
    required: 'Email is required',
  },
  password: {
    required: 'Password is required',
  },
};

export default function Login() {
  const { error, loading, login } = useAuthentication();
  const navigate = useNavigate();
  const { search } = useLocation();

  const methods = useForm({
    defaultValues: {
      emailadres: 'thomas.cooper@gmail.com',
      password: 'password2',
    },
  });
  const { handleSubmit, reset } = methods;
  const handleCancel = useCallback(() => {
    reset();
  }, [reset]);

  const handleLogin = useCallback(
   async ({ email, password }) => {
     const loggedIn = await login(email, password);

     if (loggedIn) {
       const params = new URLSearchParams(search);
       navigate({
         pathname: params.get('redirect') || '/',
         replace: true,
       });
```

```
      }
    },
    [login, navigate, search], // 👈
  );

  return (
    <FormProvider {...methods}>
      <div className='container'>
        <form
          className='d-flex flex-column'
          onSubmit={handleSubmit(handleLogin)}
        >
          <h1>Sign in</h1>
          <Error error={error} />
          <LabelInput
            label='emailadres'
            type='text'
            name='emailadres'
            placeholder='your@email.com'
            validationRules={validationRules.email}
          />
          <LabelInput
            label='password'
            type='password'
            name='password'
            validationRules={validationRules.password}
          />
          <div className='clearfix'>
            <div className='btn-group float-end'>
              <button
                type='submit'
                className='btn btn-primary'
                disabled={loading}
              >
                Sign in
              </button>

              <button
                type='button'
                className='btn btn-light'
                onClick={handleCancel}
              >
                Cancel
              </button>
            </div>
          </div>
        </form>
      </div>
    </FormProvider>
  );
}
```

PrivateRoute

- Nu maken we een PrivateRoute component voor onze **main.jsx** in `components/PrivateRoute.jsx`
- Wanneer we niet ingegolgd zijn, en we proberen naar een pagina te gaan die dit component omwikkel heeft, worden we verwezen naar `login.jsx`

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import Home from './components/home/Home.jsx';
import NotFound from './pages/NotFound.jsx';
import Layout from './components/Layout.jsx';
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
import UserList from './pages/user/UserList.jsx';
import ProductsList from "./pages/products/ProductsList.jsx";
import ProductDetail from "./pages/products/ProductDetail.jsx"
import { AuthenticationProvider } from './contexts/Authentication.context.jsx';
import Login from './pages/registration/Login.jsx';
import PrivateRoute from './components/PrivateRoute';
import Logout from './components/registration/Logout.jsx';

const router = createBrowserRouter([
  {
    element: <Layout />, children: [
      {path: '/', element: <Home />},

      {path: '*', element: <NotFound />},

      {path: '/users', element : <PrivateRoute />, children : [
        {
          index : true,
          element : <UserList />
        }
      ]},

      {path : '/login', element : <Login/>},

      {path : '/logout', element : <Logout/>},

      {path: '/products', element: <PrivateRoute />, children : [
        {
          index : true,
          element : <ProductsList />
        },
        {
          path: ':id',
          element : <ProductDetail/>
        }
      ]}
    ]
  }
]);
```

```
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <AuthenticationProvider>
        <RouterProvider router={router} />
    </AuthenticationProvider>
  </StrictMode>,
);
```

Navbar

- Nu maken we de `components/Navbar.jsx` af om te werken met het **login/logout**-icoon

```
import { Link } from "react-router-dom";
import { useAuthentication } from "../contexts/authentication";

export default function Navbar() {
  const { isAuthed } = useAuthentication();

  return (
    <nav className="navbar navbar-expand-lg navbar-dark bg-black shadow-lg border-
bottom border-white">
      <div className="container">
        <Link className="navbar-brand text-white fw-bold fs-4" to="/">
          React Template
        </Link>
        <button
          className="navbar-toggler border-white"
          type="button"
          data-bs-toggle="collapse"
          data-bs-target="#navbarNav"
          aria-controls="navbarNav"
          aria-expanded="false"
          aria-label="Toggle navigation"
        >
          <span className="navbar-toggler-icon"></span>
        </button>
        <div className="collapse navbar-collapse" id="navbarNav">
          <ul className="navbar-nav ms-auto">

            <li className="nav-item">
              <Link className="nav-link text-white px-3 py-2 border border-white
rounded" to="/">
                Home
              </Link>
            </li>

            <li className="nav-item">
              <Link className="nav-link text-white px-3 py-2 border border-white
rounded ms-2" to="/users">
                Users
              </Link>
```

```
            </li>

            <li className="nav-item">
              <Link className="nav-link text-white px-3 py-2 border border-white
rounded ms-2" to="/products">
                Products
              </Link>
            </li>


            {isAuthed ? (
              <li className="nav-item">
                <Link className="nav-link text-white px-3 py-2 border border-white
rounded ms-2" to="/logout">
                  Logout
                </Link>
              </li>
            ) : (
              <li className="nav-item">
                <Link className="nav-link text-white px-3 py-2 border border-white
rounded ms-2" to="/login">
                  Login
                </Link>
              </li>
            )
            }

          </ul>
        </div>
      </div>
    </nav>
  );
}
```

Logout

- Als laatste maken we we de `components/registration/Logout.jsx`-component om onszelf uit te loggen:

```
import { useEffect } from 'react';
import { useAuthentication } from '../../contexts/authentication';

export default function Logout() {
  const { isAuthed, logout } = useAuthentication();

  useEffect(() => {
    logout();
  }, [logout]);

  if (isAuthed) {
    return (
```

```
        <div className='container'>
          <div className='row'>
            <div className='col-12'>
              <h1>Logging out...</h1>
            </div>
          </div>
        </div>
      );
    }

    return (
      <div className='container'>
        <div className='row'>
          <div className='col-12'>
            <h1>You were successfully logged out</h1>
          </div>
        </div>
      </div>
    );
  }
```

Back-end

- Voor de backend heb je een database nodig
- Die kun je eenvoudig visualiseren via: (https://kroki.io/)[kroki]

Mappenstructuur

Initialisatie

- Maak de folder aan en initialiseer:

```
mkdir naamProject-backend;
cd naamProject-backend;
yarn init -p;
yarn add --dev typescript tsx @types/node
yarn add --dev @types/koa
```

- Maak een bestand `tsconfig` voor typescript te installeren:

```
yarn tsc --init
```

- Inhoud:

```

```

```
{
  "compilerOptions": {
    "target": "es2022",                                /* Set the JavaScript
language version for emitted JavaScript and include compatible library
declarations. */
    "module": "commonjs",                              /* Specify what module code
is generated. */
    "resolveJsonModule": true,                         /* Enable importing .json
files. */
    "outDir": "./build",                               /* Specify an output folder
for all emitted files. */
    "removeComments": true,                            /* Disable emitting
comments. */
    "newLine": "lf",                                   /* Set the newline character
for emitting files. */
    "noEmitOnError": true,                             /* Disable emitting files if
any type checking errors are reported. */
    "esModuleInterop": true,                           /* Emit additional
JavaScript to ease support for importing CommonJS modules. This enables
'allowSyntheticDefaultImports' for type compatibility. */
    "forceConsistentCasingInFileNames": true,          /* Ensure that casing is
correct in imports. */
    "strict": true,                                    /* Enable all strict type-
checking options. */
    "noUnusedLocals": true,                            /* Enable error reporting
when local variables aren't read. */
    "noUnusedParameters": true,                        /* Raise an error when a
function parameter isn't read. */
    "exactOptionalPropertyTypes": true,                /* Interpret optional
property types as written, rather than adding 'undefined'. */
    "noImplicitReturns": true,                         /* Enable error reporting
for codepaths that do not explicitly return in a function. */
    "noFallthroughCasesInSwitch": true,                /* Enable error reporting
for fallthrough cases in switch statements. */
    "noUncheckedIndexedAccess": true,                  /* Add 'undefined' to a type
when accessed using an index. */
    "noImplicitOverride": true,                        /* Ensure overriding members
in derived classes are marked with an override modifier. */
  },
  "include": [
    "**/*.ts"
  ],
  "exclude": [
    "test"
  ]
}
```

- Maak in de root het bestand `.yarnrc.yml`:

```
nodeLinker: node-modules
```

- maak nu een `.gitignore` aan en kijk helemaal bovenaan de cursus voor de inhoud
- Installeer ES-lint:

```
yarn add --dev eslint @eslint/js typescript-eslint @stylistic/eslint-plugin
```

- Maak een bestand `esline.config.ts`:

```
import eslint from '@eslint/js';
import tseslint from 'typescript-eslint';
import stylistic from '@stylistic/eslint-plugin';

export default tseslint.config(
  eslint.configs.recommended,
  ...tseslint.configs.recommended,
  {
    files: ['**/*.ts', '**/*.spec.ts'],
    plugins: {
      '@stylistic': stylistic,
    },
    rules: {
      '@stylistic/no-multiple-empty-lines': [
        'error',
        {
          max: 1,
          maxEOF: 1,
          maxBOF: 0,
        },
      ],
      '@stylistic/indent': ['error', 2, { SwitchCase: 1 }],
      '@stylistic/quotes': ['error', 'single'],
      '@stylistic/semi': ['error', 'always'],
      '@stylistic/comma-dangle': ['error', 'always-multiline'],
      '@stylistic/no-tabs': ['error'],
      '@stylistic/max-len': [
        'error',
        {
          code: 200,
          tabWidth: 2,
        },
      ],
      '@stylistic/arrow-parens': ['error', 'always'],
      '@stylistic/brace-style': ['error', '1tbs', { allowSingleLine: false }],
      '@stylistic/no-inner-declarations': 'off',
      '@typescript-eslint/no-explicit-any': 'off',
      '@typescript-eslint/consistent-type-imports': 'error',
      '@typescript-eslint/no-empty-object-type': 'off',
    },
  },
);
```

- Installeer koa voor de server:

```
yarn add koa
```

Voorbereiding server maken

Debugging

- Maak een map `.vscode` met een bestand `launch.json`:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Attach to server",
      "address": "localhost",
      "port": 9001,
      "request": "attach",
      "skipFiles": ["<node_internals>/**"],
      "type": "node",
      "restart": true,
      "timeout": 10000
    }
  ]
}
```

REST-API bouwen

```
yarn add config;
yarn add --dev @types/config;
```

- Maak een `.env`-bestand aan:

```
NODE_ENV=development
```

- Maak een `root/config` folder met de `development` en `production` bestand:
-

```
export default {
  log: {
    level: 'silly',
```

```
    disabled: false,
  },
  cors: {
    origins: ['http://localhost:5173'],
    maxAge: 3 * 60 * 60,
  },
  auth: {
    jwt: {
      audience: 'robbe.template.be',
      issuer: 'robbe.template.be',
      expirationInterval: 60 * 60 * 3, // 3 uur
      secret:
        'eenveeltemoeilijksecretdatniemandooitzalradenandersisdesitegehacked',
    },
    argon: {
      hashLength: 32,
      timeCost: 6,
      memoryCost: 2 ** 17,
    },
    maxDelay : 5000, // 5 seconden
  },

};
```

- Zet daar ook een ander bestand in: `custom-environment-variables.ts`

```
export default {
  env: 'NODE_ENV',
};
```

Logging

- Voeg `winston` toe hiervoor

```
yarn add winston;
yarn add config @types/config;
```

- maak een `logging.ts` aan in de map `core` in de `src`:

```
import { env } from 'node:process';
import winston from 'winston';
import config from 'config';

const NODE_ENV = config.get<string>('env');
const LOG_LEVEL = config.get<string>('log.level');
const LOG_DISABLED = config.get<boolean>('log.disabled');
```

```
console.log(
  `node env: ${NODE_ENV}, log level ${LOG_LEVEL}, logs enabled: ${
    LOG_DISABLED !== true
  }`,
);

const rootLogger: winston.Logger = winston.createLogger({
  level: LOG_LEVEL,
  format: winston.format.simple(),
  transports: [new winston.transports.Console({ silent: LOG_DISABLED })],
});

export const getLogger = () => {
  return rootLogger;
};
```

- Voeg cors toe waarmee een webapplicatie, die wordt uitgevoerd onder één domein, toegang kan krijgen tot resources in een ander domein.

```
yarn add @koa/cors;
yarn add --dev @types/koa__cors;
```

Server maken

- maak een `src`-map met een `index.ts` waar je de server zal starten
- dit moet niet per se nual aangezien we dit straks volledig zullen aanpassen
- maar je kan dit al gebruiken om te testen of je applicatie runt

```
import { Context } from "koa";
import { getLogger } from "./core/logging";
import bodyParser from 'koa-bodyparser';

const Koa = require('koa');
const app = new Koa();

app.use(async (ctx : Context) => {
  getLogger().info(JSON.stringify(ctx.request));
  ctx.body = 'Hello World from TypeScript';
});

app.use(bodyParser());


app.listen(9000, () => {
  getLogger().info('🚀 Server listening on http://127.0.0.1:9000');
});
```

- Om dit te starten voeg je dit toe aan de `package.json`:

```
"scripts": {
    "build": "tsc",
    "start:dev": "tsx watch --env-file .env --inspect=0.0.0.0:9001 src/index.ts",
    "lint": "eslint ."
}
```

Koa-Router

- Voeg volgene packages toe:

```
yarn add @koa/router koa-bodyparser;
yarn add --dev @types/koa__router @types/koa-bodyparser;
```

Database bouwen

- Installeer prisma client, een database framework:
- Daarna laten we het schema voor ons maken:
- Verplaats `schema.prisma` naar de `data`-map in de `src` en verwijder de prisma-map

```
yarn add prisma @prisma/client;
yarn prisma init --datasource-provider mysql;
```

- Omdat we het verplaaten moeten we ook iets aanpassen in de `package.json`

```
 "prisma": {
    "schema": "src/data/schema.prisma",
    "seed": "tsx ./src/data/seed.ts"
}
```

- Nu kunnen we ons schema maken:

```
model User {
  userId     Int     @id @default(autoincrement())
  firstName  String  @db.VarChar(100)
  lastName   String  @db.VarChar(100)
  emailadres String  @unique
}

model Supplier {
  supplierId Int     @id @default(autoincrement())
  firstName  String  @db.VarChar(100)
```

```
  lastName   String  @db.VarChar(100)
  company    String  @db.VarChar(100)
  products   Product[]
}

model Product {
  productId   Int       @id @default(autoincrement())
  productName String  @db.VarChar(100)
  unitPrice   Float  @db.Float()
  categoryId  Int  @db.Int()
  supplierId  Int @db.Int()
  image       String @db.VarChar(100)
  category    Category  @relation(fields: [categoryId], references: [categoryId])
  supplier    Supplier  @relation(fields: [supplierId], references: [supplierId])
}

model Category {
  categoryId   Int       @id @default(autoincrement())
  categoryName String  @db.VarChar(100)
  products     Product[]
}
```

- Ga nu naar de `.env`-file en pas de databank url aan die daar gegeneerd is
- Nu gaan we onze databank seeden via `seed.ts` en we gaan dit uitvoeren met een `migration`

```
yarn prisma migrate dev --name init
```

- Maak nu in de datalaag `index.ts` aan:

```
import { PrismaClient } from '@prisma/client';
import { getLogger } from '../core/logging';

export const prisma = new PrismaClient();

export async function initializeData(): Promise<void> {
  getLogger().info('Initializing connection to the database');

  await prisma.$connect();

  getLogger().info('Successfully connected to the database');
}

export async function shutdownData(): Promise<void> {
  getLogger().info('Shutting down database connection');

  await prisma?.$disconnect();

  getLogger().info('Database connection closed');
}
```

- Nu bouwen we een seed-bestand om data te steken in onze databank:

```
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

async function main() {
  // Voeg categorieën toe
  await prisma.category.createMany({
    data: [
      { categoryName: "iphone" },
      { categoryName: "laptop" },
    ],
  });

  // Voeg leveranciers toe
  await prisma.supplier.createMany({
    data: [
      { firstName: 'Kurt', lastName: "Mathijs", company: "Bertha NV" },
      { firstName: 'Brenda', lastName: "Mclifius", company: "Mcflow NV" },
      { firstName: 'Cooper', lastName: "Magerman", company: "Okay" },
    ],
  });

  // Voeg producten toe
  await prisma.product.createMany({
    data: [
      { productName: "Iphone 11", image: "iphone_11.jpg", unitPrice: 899.99,
categoryId: 1, supplierId: 1 },
      { productName: "Iphone 13 pro", image: "iphone_13_pro.jpg", unitPrice:
1199.99, categoryId: 1, supplierId: 1 },
      { productName: "Windows laptop", image: "windows_laptop.jpg", unitPrice:
899.99, categoryId: 2, supplierId: 2 },
    ],
  });

  // Voeg gebruikers toe
  await prisma.user.createMany({
    data: [
      { firstName: "John", lastName: "Doe", emailadres: "john.doe@gmail.com" },
      { firstName: "Thomas", lastName: "Cooper", emailadres:
"thomas.cooper@gmail.com" },
      { firstName: "Barley", lastName: "Nutorious", emailadres:
"barley.nutorious@gmail.com" },
    ],
  });
}

main()
  .then(async () => {
    await prisma.$disconnect();
```

```
  })
  .catch(async (e) => {
    console.error(e);
    await prisma.$disconnect();
    process.exit(1);
  });
```

Endpoints bouwen

Types

- Maak een `types`-map in de `src`-map
- Hiermee gaan we onze service-/restlaag types aan geven
- Maak eerst wat KoaTypes:

```
import type { ParameterizedContext } from 'koa';
import type Application from 'koa';
import type Router from '@koa/router';

export interface TemplateAppState {}

export interface TemplateAppContext<
  Params = unknown,
  RequestBody = unknown,
  Query = unknown,
> {
  request: {
    body: RequestBody;
    query: Query;
  };
  params: Params;
}

export type KoaContext<
  ResponseBody = unknown,
  Params = unknown,
  RequestBody = unknown,
  Query = unknown,
> = ParameterizedContext<
  TemplateAppState,
  TemplateAppContext<Params, RequestBody, Query>,
  ResponseBody
>;

export interface KoaApplication
  extends Application<TemplateAppState, TemplateAppContext> {}

export interface KoaRouter extends Router<TemplateAppState, TemplateAppContext> {}
```

- Nu doen we `common.ts` voor enkele responses op te vangen:

```
export interface ListResponse<T> {
   items: T[];
 }

 export interface IdParams {
   id: number;
 }
```

- Nu Koa ingesteld is, stellen we de types van onze endpoints in
- We beginnen met: `user.ts`

```
import type { ListResponse } from './common';

export interface User {
   userId : number,
   firstName : string,
   lastName : string,
   emailadres : string,
   image : string,
   password : string,
}

export interface UserCreateInput {
   firstName : string,
   lastName : string,
   emailadres : string,
   image : string,
   password : string,
}

export interface UserUpdateInput extends UserCreateInput {}

export interface CreateUserRequest extends UserCreateInput {}
export interface UpdateUserRequest extends UserUpdateInput {}

export interface GetAllUsersResponse extends ListResponse<User> {}
export interface GetUserByIdResponse extends User {}
export interface CreateUserResponse extends GetUserByIdResponse {}
export interface UpdateUserResponse extends GetUserByIdResponse {}
```

- `types/product.ts`:

```
import type { ListResponse } from './common';

export interface Product {
  productId : number
  productName : string
  unitPrice : number
```

```
  categoryId : number
  supplierId : number
  image : string,
  unitsInStock : number,
}

export interface ProductCreateInput {
  productName : string
  unitPrice : number
  categoryId : number
  supplierId : number
  image : string,
  unitsInStock : number,
}

export interface ProductUpdateInput extends ProductCreateInput {}

export interface CreateProductRequest extends ProductCreateInput {}
export interface UpdateProductRequest extends ProductUpdateInput {}

export interface GetAllProductsResponse extends ListResponse<Product> {}
export interface GetProductByIdResponse extends Product {}
export interface CreateProductResponse extends GetProductByIdResponse {}
export interface UpdateProductResponse extends GetProductByIdResponse {}
```

ServiceError

- Maak in `core/serviceError.ts` dit bestand:
- Dit zijn alle errors die kunnen voorkomen in de `servicelaag`

```
const NOT_FOUND = 'NOT_FOUND';
const VALIDATION_FAILED = 'VALIDATION_FAILED';
const UNAUTHORIZED = 'UNAUTHORIZED';
const FORBIDDEN = 'FORBIDDEN';
const INTERNAL_SERVER_ERROR = 'INTERNAL_SERVER_ERROR';
const CONFLICT = 'CONFLICT';

export default class ServiceError extends Error {
  code: string;

  constructor(code: string, message: string) {
    super(message);
    this.code = code;
    this.name = 'ServiceError';
  }

  static notFound(message: string) {
    return new ServiceError(NOT_FOUND, message);
  }

  static validationFailed(message: string) {
```

```
    return new ServiceError(VALIDATION_FAILED, message);
  }

  static unauthorized(message: string) {
    return new ServiceError(UNAUTHORIZED, message);
  }

  static forbidden(message: string) {
    return new ServiceError(FORBIDDEN, message);
  }

  static internalServerError(message: string) {
    return new ServiceError(INTERNAL_SERVER_ERROR, message);
  }

  static conflict(message: string) {
    return new ServiceError(CONFLICT, message);
  }

  get isNotFound(): boolean {
    return this.code === NOT_FOUND;
  }

  get isValidationFailed(): boolean {
    return this.code === VALIDATION_FAILED;
  }

  get isUnauthorized(): boolean {
    return this.code === UNAUTHORIZED;
  }

  get isForbidden(): boolean {
    return this.code === FORBIDDEN;
  }

  get isInternalServerError(): boolean {
    return this.code === INTERNAL_SERVER_ERROR;
  }

  get isConflict(): boolean {
    return this.code === CONFLICT;
  }
}
```

Database error handeling

- Om deze errors op te vangen gebruiken we `service/_handleDBError.ts`:
- Dit bestand moet jij aanpassen naar gelang jouw `schema.prisma`

```
import ServiceError from '../core/serviceError';
```

```
const handleDBError = (error: any) => {
  const { code = '', message } = error;

  // code voor unieke waarden
  if (code === 'P2002') {
    switch (true) {
      case message.includes('user.emailadres'):
        throw ServiceError.validationFailed('There is already a user with this
email address');
      case message.includes('supplier.company'):
        throw ServiceError.validationFailed('A supplier with this company name
already exists');
      case message.includes('product.productName'):
        throw ServiceError.validationFailed('A product with this name already
exists');
      case message.includes('category.categoryName'):
        throw ServiceError.validationFailed('A category with this name already
exists');
      default:
        throw ServiceError.validationFailed('This item already exists');
    }
  }

  // code voor niet bestaande referenties
  if (code === 'P2025') {
    switch (true) {
      case message.includes('fk_product_supplier'):
        throw ServiceError.notFound('This supplier does not exist');
      case message.includes('fk_product_category'):
        throw ServiceError.notFound('This category does not exist');
      case message.includes('product'):
        throw ServiceError.notFound('No product with this id exists');
      case message.includes('supplier'):
        throw ServiceError.notFound('No supplier with this id exists');
      case message.includes('category'):
        throw ServiceError.notFound('No category with this id exists');
      case message.includes('user'):
        throw ServiceError.notFound('No user with this id exists');
    }
  }

  // code voor foreign key conflicten
  if (code === 'P2003') {
    switch (true) {
      case message.includes('categoryId'):
        throw ServiceError.conflict('This category does not exist or is still
linked to products');
      case message.includes('supplierId'):
        throw ServiceError.conflict('This supplier does not exist or is still
linked to products');
    }
  }

  throw error;
```

```
};

export default handleDBError;
```

Service-laag

- Nu onze types gebouwd zijn kunnen we beginnen met de endpoints zelf te bouwen
- De API communiceert eerst via de servicelaag en dat wordt dan doorgegeven naar de REST-laag

```
import {prisma} from "../data"
import { User, UserCreateInput, UserUpdateInput } from "../types/user";
import ServiceError from '../core/serviceError';
import handleDBError from './_handelDBError';

export const getAll = async () : Promise<User[]>=> {
   return prisma.user.findMany();
}

export const getById = async (id: number) : Promise<User> => {
   const user = await prisma.user.findUnique({
     where: {
       userId : id,
     }
   });

   if (!user){
      throw ServiceError.notFound("No user with this Id exists");
   }

   return user
}

export const create = async (user : UserCreateInput) : Promise<User> => {
   try {
      return prisma.user.create({
         data : user
      })
   } catch (error : any){
      throw handleDBError(error)
   }

}

export const updateById = async (id: number, userChanges: UserUpdateInput) :
Promise<User> => {
   const user = await prisma.user.findUnique({
     where: {
       userId : id,
     }
   });
```

```
    if (!user){
       throw ServiceError.notFound("No user with this Id exists")
    }

    const updatedUser = await prisma.user.update({
       where: {
          userId: id,
       },
       data: userChanges
    });

    return updatedUser;
 };

 export const deleteById = async (id: number) : Promise<void> => {
    const user = await prisma.user.findUnique({
       where: {
         userId : id,
       }
     });

     if (!user){
       throw ServiceError.notFound("No user with this Id exists")
    }

    await prisma.user.delete({
       where: {
          userId: id,
       }
    });
};
```

REST-laag

- Voor we de routes zelf vastleggen, voegen we eerst het `validatiegedeelte` al toe

```
yarn add joi
```

- Daarna maken we een validatiebestand aan in de `core` voor validatie op requests die worden gestuurd:

```
import type { Schema, SchemaLike } from 'joi';
import Joi from 'joi';
import type { KoaContext } from '../types/koa';
import type { Next } from 'koa';

const JOI_OPTIONS: Joi.ValidationOptions = {
  abortEarly: true, // stop when first error occured
  allowUnknown: false, // disallow unknown fields
  convert: true, // convert values to their types (number, Date, ...)
```

```
  presence: 'required', // default require all fields
};

type RequestValidationSchemeInput = Partial<
  Record<'params' | 'body' | 'query', SchemaLike>
>;
type RequestValidationScheme = Record<'params' | 'body' | 'query', Schema>;

const validate = (scheme: RequestValidationSchemeInput | null) => {
  const parsedSchema: RequestValidationScheme = {
    body: Joi.object(scheme?.body || {}),
    params: Joi.object(scheme?.params || {}),
    query: Joi.object(scheme?.query || {}),
  };

  return (ctx: KoaContext, next: Next) => {
    const errors = new Map();

    const { error: paramsErrors, value: paramsValue } =
      parsedSchema.params.validate(ctx.params, JOI_OPTIONS);

    if (paramsErrors) {
      errors.set('params', cleanupJoiError(paramsErrors));
    } else {
      ctx.params = paramsValue;
    }

    if (errors.size > 0) {
      ctx.throw(400, 'Validation failed, check details for more information', {
        code: 'VALIDATION_FAILED',
        details: Object.fromEntries(errors),
      });
    }

    return next();
  };
};

export default validate;
```

- Nu kunnen we in `src/rest/user.ts` de effectieve API bouwen:

```
import { KoaContext, KoaRouter, TemplateAppContext, TemplateAppState } from
"../types/koa";
import * as userService from "../service/user"
import { CreateUserRequest, CreateUserResponse, GetAllUsersResponse,
GetUserByIdResponse, UpdateUserRequest, UpdateUserResponse } from "../types/user";
import { IdParams } from "../types/common";
import Router from "@koa/router";
import Joi from "joi";
import validate from '../core/validation';
```

```
const getAllUsers = async (ctx: KoaContext<GetAllUsersResponse>) => {
   const users = await userService.getAll();
   ctx.body = {
     items: users,
   };
 };

getAllUsers.validationScheme = null;

const getUserById = async (ctx : KoaContext<GetUserByIdResponse, IdParams>) => {
   const user = await userService.getById(Number(ctx.params.id));
   ctx.body = user
}

getUserById.validationScheme = {
   params: {
      id: Joi.number().integer().positive(),
   },
}

const createUser = async (ctx : KoaContext<CreateUserResponse, void,
CreateUserRequest>) => {
   const user = await userService.create(ctx.request.body);
   ctx.body = user;
   ctx.status = 201;
}

createUser.validationScheme = {
   body : {
      firstName : Joi.string(),
      lastName : Joi.string(),
      emailadres : Joi.string().email(),
   }
}

const updateUser = async (ctx : KoaContext<UpdateUserResponse, IdParams,
UpdateUserRequest>) => {
   const user = await userService.updateById(Number(ctx.params.id),
ctx.request.body);
   ctx.body = user
}

updateUser.validationScheme = {
   params : {
      id : Joi.number().integer().positive()
   },
   body : {
      firstName : Joi.string(),
      lastName : Joi.string(),
      emailadres : Joi.string().email(),
   }
}
```

```
const deleteUser = async (ctx : KoaContext<void, IdParams>) => {
   await userService.deleteById(Number(ctx.params.id));
   ctx.status = 204;
}

deleteUser.validationScheme = {
   params : {
      id : Joi.number().integer().positive()
   }
}

export default (parent: KoaRouter) => {
   const router = new Router<TemplateAppState, TemplateAppContext>({
     prefix: '/users',
   });

   router.get("/", validate(getAllUsers.validationScheme), getAllUsers);
   router.get("/:id", validate(getUserById.validationScheme), getUserById);
   router.post("/", validate(createUser.validationScheme), createUser);
   router.put("/:id", validate(updateUser.validationScheme), updateUser);
   router.delete("/:id", validate(deleteUser.validationScheme), deleteUser);

   parent.use(router.routes()).use(router.allowedMethods());
};
```

installMiddlewares

- Installeer koa-helmet dit beheert het proces van het instellen en beheren van beveiligingsheaders in Koa-applicaties:

```
yarn add koa-helmet helmet;
```

- Maak een bestand `core/installMiddelwares.ts`

```
import config from 'config';
import bodyParser from 'koa-bodyparser';
import koaCors from '@koa/cors';
import type { KoaApplication } from '../types/koa';
import { getLogger } from './logging';
import ServiceError from './serviceError';
import serve from 'koa-static';
import koaHelmet from 'koa-helmet';

const CORS_ORIGINS = config.get<string[]>('cors.origins');
const CORS_MAX_AGE = config.get<number>('cors.maxAge');
const NODE_ENV = config.get<string>('env');

export default function installMiddlewares(app: KoaApplication) {
  app.use(
```

```
    koaCors({
      origin: (ctx) => {
        if (CORS_ORIGINS.indexOf(ctx.request.header.origin!) !== -1) {
          return ctx.request.header.origin!;
        }
        // Not a valid domain at this point, let's return the first valid as we
should return a string
        return CORS_ORIGINS[0] || '';
      },
      allowHeaders: ['Accept', 'Content-Type', 'Authorization'],
      maxAge: CORS_MAX_AGE,
    }),
  );

  // Ophalen fotos uit de public map
  app.use(serve('public'));

  app.use(async (ctx, next) => {
      getLogger().info(`⏭ ${ctx.method} ${ctx.url}`);

      const getStatusEmoji = () => {
        if (ctx.status >= 500) return '💀';
        if (ctx.status >= 400) return '❌';
        if (ctx.status >= 300) return '🔀';
        if (ctx.status >= 200) return '✅';
        return '🔄';
      };

      await next();

      getLogger().info(
        `${getStatusEmoji()} ${ctx.method} ${ctx.status} ${ctx.url}`,
      );
    });

    app.use(async (ctx, next) => {
      try {
        await next();
      } catch (error: any) {
        getLogger().error('Error occured while handling a request', { error });

        let statusCode = error.status || 500;
        const errorBody = {
          code: error.code || 'INTERNAL_SERVER_ERROR',
          // Do not expose the error message in production
          message:
            error.message || 'Unexpected error occurred. Please try again later.',
          details: error.details,
          stack: NODE_ENV !== 'production' ? error.stack : undefined,
        };

        if (error instanceof ServiceError) {
          errorBody.message = error.message;
```

```
          if (error.isNotFound) {
            statusCode = 404;
          }

          if (error.isValidationFailed) {
            statusCode = 400;
          }

          if (error.isUnauthorized) {
            statusCode = 401;
          }

          if (error.isForbidden) {
            statusCode = 403;
          }

          if (error.isConflict) {
            statusCode = 409;
          }
        }

        ctx.status = statusCode;
        ctx.body = errorBody;
      }
    });

    app.use(async (ctx, next) => {
      await next();

      if (ctx.status === 404) {
        ctx.status = 404;
        ctx.body = {
          code: 'NOT_FOUND',
          message: `Unknown resource: ${ctx.url}`,
        };
      }
    });

  app.use(bodyParser());

  app.use(koaHelmet());
}
```

CreateServer

- Indien je ergens in de cursus `src/index.ts` hebt gemaakt, moet je dit nu hernoemen naar `createServer.ts`

```
import Koa from 'koa';

import { getLogger } from './core/logging';
```

```
import { initializeData, shutdownData } from './data';
import installMiddlewares from './core/installMiddeleware';
import installRest from './rest';
import type {KoaApplication, TemplateAppContext,TemplateAppState,} from
'./types/koa';

export interface Server {
  getApp(): KoaApplication;
  start(): Promise<void>;
  stop(): Promise<void>;
}
export default async function createServer(): Promise<Server> {
  const app = new Koa<TemplateAppState, TemplateAppContext>();
  installMiddlewares(app);
  await initializeData();
  installRest(app);

  return {
    getApp() {
      return app;
    },

    start() {
      return new Promise<void>((resolve) => {
        app.listen(9000, () => {
          getLogger().info('🚀 Server listening on http://localhost:9000 ☑');
          resolve();
        });
      });
    },

    async stop() {
      app.removeAllListeners();
      await shutdownData();
      getLogger().info('Goodbye! 👋');
    },
  };
}
```

index.ts

- Maak nu een nieuwe `index.ts` met de volgende inhoud:

```
import createServer from './createServer';

async function main() {
  try {
    const server = await createServer();
    await server.start();

    async function onClose() {
```

```
      await server.stop();
      process.exit(0);
    }

    process.on('SIGTERM', onClose);
    process.on('SIGQUIT', onClose);
  } catch (error) {
    console.log('\n', error);
    process.exit(-1);
  }
}

main();
```

Back-end Authenticatie

- Dit hoofdstuk staat apart van de normale back-end aangezien je hier heel wat gaat moeten aanpassen en indien je een statische website wil bouwen heb je dit niet nodig
- In dit hoofdstuk ga je zien hoe je een fully functional authenticatieproces moet bouwen in de REST-API
- Je gaat in vergelijking met het vorige hoofdstuk ook heel veel moeten aanpassen!
- Je moet dit stuk **EERST** toevoegen via de Back-end voor je dit kan toepassen in de frontend

User-type

- We starten met het toevoegen van `roles` aan ons schema
- Een user kan namelijk een gewonen user zijn of een **admin**
- Voeg jsonwebtoken toe om JWT's te ondertekenen en verifiëren
- Voeg ook argon2 toe om zijn functies voor `hashing` te gebruiken

```
yarn add jsonwebtoken;
yarn add argon2
```

- Maak in een bestand `core/roles.ts` voor de rollen die we gaan toevoegen aan de User:

```
export default {
  USER: 'user',
  ADMIN: 'admin',
};
```

- Nu breiden we ons `schema.prisma` uit met deze rollen:

```
model User {
  userId     Int     @id @default(autoincrement())
  firstName  String  @db.VarChar(100)
```

```
  lastName   String  @db.VarChar(100)
  emailadres String  @unique
  password   String  @db.VarChar(100)
  image      String  @db.VarChar(100)
  roles      Json

  @@map("users")
}
```

JWT-webtoken

- Maak `core/jwt.ts`, een module met een aantal helpers om een JWT te maken/controleren:

```
import config from 'config';
import type {
  JwtPayload,
  Secret,
  SignOptions,
  VerifyOptions,
} from 'jsonwebtoken';
import jwt from 'jsonwebtoken';
import util from 'node:util';
import type { User } from '../types/user';

const JWT_AUDIENCE = config.get<string>('auth.jwt.audience');
const JWT_SECRET = config.get<string>('auth.jwt.secret');
const JWT_ISSUER = config.get<string>('auth.jwt.issuer');
const JWT_EXPIRATION_INTERVAL = config.get<number>(
  'auth.jwt.expirationInterval',
);

const asyncJwtSign = util.promisify<JwtPayload, Secret, SignOptions, string>(
  jwt.sign,
);
const asyncJwtVerify = util.promisify<
  string,
  Secret,
  VerifyOptions,
  JwtPayload
>(jwt.verify);

export const generateJWT = async (user: User): Promise<string> => {
  const tokenData = { roles: user.roles };

  const signOptions = {
    expiresIn: Math.floor(JWT_EXPIRATION_INTERVAL),
    audience: JWT_AUDIENCE,
    issuer: JWT_ISSUER,
    subject: `${user.id}`,
  };
```

```
  return asyncJwtSign(tokenData, JWT_SECRET, signOptions);
};

export const verifyJWT = async (authToken: string): Promise<JwtPayload> => {
  const verifyOptions = {
    audience: JWT_AUDIENCE,
    issuer: JWT_ISSUER,
  };

  return asyncJwtVerify(authToken, JWT_SECRET, verifyOptions);
};
```

- We kunnen deze token testen met het volgende testbestand `src/testjwt.ts`:
- Je zal wel jouw `user`-object moeten aanpassen naar jouw `schema.prisma`
- Run `yarn tsx src/testjwt.ts` om dit uit te voeren

```
import Role from "./core/roles"

process.env.NODE_CONFIG = JSON.stringify({
   env: 'development',
 });

 import { generateJWT, verifyJWT } from './core/jwt';

 function messWithPayload(jwt: string) {
   const [header, payload, signature] = jwt.split('.');
   const parsedPayload = JSON.parse(
     Buffer.from(payload!, 'base64url').toString(),
   );

   // make me admin please ^^
   parsedPayload.roles.push('admin');

   const newPayload = Buffer.from(
     JSON.stringify(parsedPayload),
     'ascii',
   ).toString('base64url');
   return [header, newPayload, signature].join('.');
 }

 async function main() {
   const fakeUser = {
      userId : 1,
      firstName: "John",
      lastName: "Doe",
      emailadres: "john.doe@gmail.com",
      image : "johndoe.jpg",
      password : "password1",
      roles : [Role.USER]
   };
```

```
  const jwt = await generateJWT(fakeUser);
  // copy and paste the JWT in the textfield on https://jwt.io
  // inspect the content
  console.log('The JWT:', jwt);

  let valid = await verifyJWT(jwt);
  console.log('This JWT is', valid ? 'valid' : 'incorrect');
  console.log('\n');

  // Let's mess with the payload
  const messedUpJwt = messWithPayload(jwt);
  console.log('Messed up JWT:', messedUpJwt);

  try {
    console.log('Verifying this JWT will throw an error:');
    valid = await verifyJWT(messedUpJwt);
  } catch (err: any) {
    console.log('We got an error:', err.message);
  }
}

main();
```

Hashpassword

- Nu maken we `core/password.ts` om paswoorden te `hashen` en te controleren:

```
import config from 'config';
import argon2 from 'argon2';

const ARGON_HASH_LENGTH = config.get<number>('auth.argon.hashLength');
const ARGON_TIME_COST = config.get<number>('auth.argon.timeCost');
const ARGON_MEMORY_COST = config.get<number>('auth.argon.memoryCost');

export const hashPassword = async (password: string): Promise<string> => {
  return argon2.hash(password, {
    type: argon2.argon2id,
    hashLength: ARGON_HASH_LENGTH,
    timeCost: ARGON_TIME_COST,
    memoryCost: ARGON_MEMORY_COST,
  });
};

export const verifyPassword = async (
  password: string,
  passwordHash: string,
): Promise<boolean> => {
  return argon2.verify(passwordHash, password);
};
```

- We kunnen dit bestand testen via `src/testpassword.ts`:
- Run `yarn tsx src/testpassword.ts` om dit uit te voeren

```
process.env.NODE_CONFIG = JSON.stringify({
   env: 'development',
 });

 import { hashPassword, verifyPassword } from './core/password';

 async function main() {
   const password = 'verydifficult';
   const wrongPassword = 'verywrong';
   console.log('The password:', password);

   const hash = await hashPassword(password);
   // bekijk hoe de hash opgebouwd is, wat herken je?
   // waar staat de timeCost, memoryCost, salt en de hash zelf?
   console.log('The hash:', hash);

   let valid = await verifyPassword(password, hash);
   console.log('The password', password, 'is', valid ? 'valid' : 'incorrect');
   console.log('');

   valid = await verifyPassword(wrongPassword, hash);
   console.log(
     'The password',
     wrongPassword,
     'is',
     valid ? 'correct/valid' : 'incorrect/invalid',
   );
 }

 main();
```

Endpoints

Seed

- Omdat we ons schema updaten moeten we ook onze `seed` aanpassen:

```
import { PrismaClient } from '@prisma/client';
import Role from "../core/roles"
import { hashPassword } from '../core/password';

const prisma = new PrismaClient();

async function main() {
  // Voeg categorieën toe
  await prisma.category.createMany({
    data: [
```

```
      { categoryName: "iphone" },
      { categoryName: "laptop" },
    ],
  });

  // Voeg leveranciers toe
  await prisma.supplier.createMany({
    data: [
      { firstName: 'Kurt', lastName: "Mathijs", company: "Bertha NV" },
      { firstName: 'Brenda', lastName: "Mclifius", company: "Mcflow NV" },
      { firstName: 'Cooper', lastName: "Magerman", company: "Okay" },
    ],
  });

  // Voeg producten toe
  await prisma.product.createMany({
    data: [
      { productName: "Iphone 11", image: "iphone_11.jpg", unitPrice: 899.99,
categoryId: 1, supplierId: 1, unitsInStock : 50 },
      { productName: "Iphone 13 pro", image: "iphone_13_pro.jpg", unitPrice:
1199.99, categoryId: 1, supplierId: 1, unitsInStock : 30 },
      { productName: "Windows laptop", image: "windows_laptop.jpg", unitPrice:
899.99, categoryId: 2, supplierId: 2, unitsInStock : 25 },
    ],
  });

  // Wachtwoorden hashen voor de gebruikers
  const password1 = await hashPassword('password1')
  const password2 = await hashPassword('password2')
  const password3 = await hashPassword('password3')

  // Voeg gebruikers toe
  await prisma.user.createMany({
    data: [
      { firstName: "John", lastName: "Doe", emailadres: "john.doe@gmail.com",
image : "johndoe.jpg", password : password1, roles : JSON.stringify([Role.ADMIN,
Role.USER]), },
      { firstName: "Thomas", lastName: "Cooper", emailadres:
"thomas.cooper@gmail.com", image : "thomascooper.jpg", password : password2, roles
: JSON.stringify([Role.USER]), },
      { firstName: "Barley", lastName: "Nutorious", emailadres:
"barley.nutorious@gmail.com", image : "barleynutorious.jpg", password : password3,
roles : JSON.stringify([Role.USER]),  },
    ],
  });
}

main()
  .then(async () => {
    await prisma.$disconnect();
  })
  .catch(async (e) => {
    console.error(e);
    await prisma.$disconnect();
```

```
    process.exit(1);
  });
```

Voorbereiding

- Maak `core/authentication.ts`
- De eerste helper dwingt af dat de gebruiker moet aangemeld zijn om een endpoint uit te voeren. Dit is een middleware.
- De tweede helper dwingt af dat de gebruiker de juiste rollen heeft om een endpoint uit te voeren. Deze helper geeft een middleware terug (= currying).

```
import type { Next } from 'koa';
import type { KoaContext } from '../types/koa';
import * as userService from '../service/user';
import config from 'config';

const AUTH_MAX_DELAY = config.get<number>('authentication.maxDelay');

export const requireAuthentication = async (ctx: KoaContext, next: Next) => {
  const { authorization } = ctx.headers;

  ctx.state.session = await userService.checkAndParseSession(authorization);

  return next();
};

export const makeRequireRole = (role: string) => async (ctx: KoaContext, next:
Next) => {
    const { roles = [] } = ctx.state.session;

    userService.checkRole(role, roles);

    return next();
  };

  export const authDelay = async (_: KoaContext, next: Next) => {
    await new Promise((resolve) => {
      const delay = Math.round(Math.random() * AUTH_MAX_DELAY);
      setTimeout(resolve, delay);
    });
    return next();
  };
```

Types

- Nu moeten we in onze `service/rest/types`-laag allerlei dingen aanpassen om te werken met hashpasswoorden etc ...
- We maken hier een `publicUser` aan dat gebruik maakt van een normaal password en geen hash

```
import { Prisma } from '@prisma/client';
import type { ListResponse } from './common';

export interface User {
   userId : number,
   firstName : string,
   lastName : string,
   emailadres : string,
   image : string,
   password : string,
   roles : Prisma.JsonValue;
}

export interface UserCreateInput {
   firstName : string,
   lastName : string,
   emailadres : string,
   image : string,
   password : string,
}

export interface PublicUser extends Pick<User, 'userId' | 'firstName' | 'lastName'
| 'emailadres' | 'image'>{}

export interface UserUpdateInput extends UserCreateInput {}

export interface CreateUserRequest extends UserCreateInput {}
export interface UpdateUserRequest extends UserUpdateInput {}

export interface GetAllUsersResponse extends ListResponse<User> {}
export interface GetUserByIdResponse extends User {}
export interface CreateUserResponse extends GetUserByIdResponse {}
export interface UpdateUserResponse extends GetUserByIdResponse {}

export interface LoginRequest {
  emailadres: string;
  password: string;
}

export interface LoginResponse {
  token: string;
}

export interface GetUserRequest {
   id: number | 'me';
 }
```

- Maak een nieuw types `types/authentication.ts`:

```
export interface SessionInfo {
  userId: number;
```

```
  roles: string[];
}
```

- Met deze interface passen we `types/koa.ts` aan:

```
import type { ParameterizedContext } from 'koa';
import type Application from 'koa';
import type Router from '@koa/router';
import type { SessionInfo } from './authentication';

export interface TemplateAppState {
  session: SessionInfo;
}

export interface TemplateAppContext<
  Params = unknown,
  RequestBody = unknown,
  Query = unknown,
> {
  request: {
    body: RequestBody;
    query: Query;
  };
  params: Params;
}

export type KoaContext<
  ResponseBody = unknown,
  Params = unknown,
  RequestBody = unknown,
  Query = unknown,
> = ParameterizedContext<
  TemplateAppState,
  TemplateAppContext<Params, RequestBody, Query>,
  ResponseBody
>;

export interface KoaApplication
  extends Application<TemplateAppState, TemplateAppContext> {}

export interface KoaRouter extends Router<TemplateAppState, TemplateAppContext> {}
```

Service

- Nu gaan we werken met deze PublicUser in `service/user.ts`
- We maken ook andere de `login`-functie

```
import {prisma} from "../data"
import { User, UserUpdateInput, PublicUser, CreateUserRequest } from
```

```
"../types/user";
import ServiceError from '../core/serviceError';
import handleDBError from './_handelDBError';
import { hashPassword, verifyPassword } from "../core/password";
import Role from "../core/roles";
import jwt from 'jsonwebtoken';
import { getLogger } from '../core/logging';
import { generateJWT, verifyJWT } from '../core/jwt';
import type { SessionInfo } from '../types/authentication';

const makeExposedUser = ({userId, firstName, lastName, emailadres, image} : User)
: PublicUser => ({
   userId,
   firstName,
   lastName,
   emailadres,
   image,
});

export const checkAndParseSession = async (
   authHeader?: string,
 ): Promise<SessionInfo> => {
   if (!authHeader) {
     throw ServiceError.unauthorized('You need to be signed in');
   }

   if (!authHeader.startsWith('Bearer ')) {
     throw ServiceError.unauthorized('Invalid authentication token');
   }

   const authToken = authHeader.substring(7);

   try {
     const { roles, sub } = await verifyJWT(authToken);

     return {
       userId: Number(sub),
       roles,
     };
   } catch (error: any) {
     getLogger().error(error.message, { error });

     if (error instanceof jwt.TokenExpiredError) {
       throw ServiceError.unauthorized('The token has expired');
     } else if (error instanceof jwt.JsonWebTokenError) {
       throw ServiceError.unauthorized(
         `Invalid authentication token: ${error.message}`,
       );
     } else {
       throw ServiceError.unauthorized(error.message);
     }
   }
 };
```

```
 export const checkRole = (role: string, roles: string[]): void => {
   const hasPermission = roles.includes(role);

   if (!hasPermission) {
     throw ServiceError.forbidden(
       'You are not allowed to view this part of the application',
     );
   }
 };

export const login = async (
   emailadres: string,
   password: string,
 ): Promise<string> => {
   const user = await prisma.user.findUnique({ where: { emailadres } });

   if (!user) {
     throw ServiceError.unauthorized(
       'The given email and password do not match',
     );
   }

   const passwordValid = await verifyPassword(password, user.password);

   if (!passwordValid) {
     // DO NOT expose we know the user but an invalid password was given
     throw ServiceError.unauthorized(
       'The given email and password do not match',
     );
   }

   return await generateJWT(user);
 };

export const getAll = async () : Promise<PublicUser[]>=> {
   const users = await prisma.user.findMany();
   return users.map(makeExposedUser);
}

export const getById = async (id: number) : Promise<PublicUser> => {
   const user = await prisma.user.findUnique({
     where: {
       userId : id,
     }
   });

   if (!user){
      throw ServiceError.notFound("No user with this Id exists");
   }

   return makeExposedUser(user);
}

export const create = async ({firstName, lastName, emailadres, image, password} :
```

```
CreateUserRequest) : Promise<String> => {
   try {
      const passwordHash = await hashPassword(password);
      const user =  await prisma.user.create({
         data : {
            firstName,
            lastName,
            emailadres,
            image,
            password : passwordHash,
            roles : [Role.USER]
         }
      })

      if (!user) {
         throw ServiceError.internalServerError(
            'An unexpected error occured when creating the user',
         );
      }

      return await generateJWT(user);
   } catch (error : any){
      throw handleDBError(error)
   }
}

export const updateById = async (id: number, userChanges: UserUpdateInput) :
Promise<PublicUser> => {
   try {
   const user = await prisma.user.findUnique({
     where: {
       userId : id,
     }
   });

   if (!user){
      throw ServiceError.notFound("No user with this Id exists")
   }

   const updatedUser = await prisma.user.update({
      where: {
         userId: id,
      },
      data: userChanges
   });

   return updatedUser;
   } catch (error : any){
      throw handleDBError(error);
   }
 };

 export const deleteById = async (id: number) : Promise<void> => {
   try {
```

```
    const user = await prisma.user.findUnique({
       where: {
         userId : id,
       }
     });

     if (!user){
       throw ServiceError.notFound("No user with this Id exists")
    }

    await prisma.user.delete({
       where: {
          userId: id,
       }
    });
    } catch (error : any){
       throw handleDBError(error);
    }
};
```

Rest

- Nu veranderen we `rest/user.ts`:

```
import { KoaContext, KoaRouter, TemplateAppContext, TemplateAppState } from
"../types/koa";
import * as userService from "../service/user"
import { CreateUserRequest, GetAllUsersResponse, GetUserByIdResponse,
LoginResponse, UpdateUserRequest, UpdateUserResponse, GetUserRequest } from
"../types/user";
import { IdParams } from "../types/common";
import Router from "@koa/router";
import Joi from "joi";
import validate from '../core/validation';
import { requireAuthentication, makeRequireRole } from '../core/authentication';
import Role from '../core/roles';
import { Next } from "koa";
import { authDelay } from "../core/authentication";

const checkUserId = (ctx: KoaContext<unknown, GetUserRequest>, next: Next) => {
   const { userId, roles } = ctx.state.session;
   const { id } = ctx.params;

   if (id !== 'me' && id !== userId && !roles.includes(Role.ADMIN)) {
     return ctx.throw(
       403,
       "You are not allowed to view this user's information",
       { code: 'FORBIDDEN' },
     );
   }
   return next();
```

```
 };

const getAllUsers = async (ctx: KoaContext<GetAllUsersResponse>) => {
   const users = await userService.getAll();
   ctx.body = {
     items: users,
   };
 };

getAllUsers.validationScheme = null;

const getUserById = async (ctx : KoaContext<GetUserByIdResponse, GetUserRequest>)
=> {
   const user = await userService.getById(
      ctx.params.id === 'me' ? ctx.state.session.userId : ctx.params.id,
   );
   ctx.status = 200;
   ctx.body = user;
}

getUserById.validationScheme = {
   params: {
      id: Joi.alternatives().try(Joi.number().integer().positive(),
Joi.string().valid('me')),
   },
}

const createUser = async (ctx : KoaContext<LoginResponse, void,
CreateUserRequest>) => {
   const token : string = await userService.create(ctx.request.body);

   ctx.status = 200;
   ctx.body = { token };
}

createUser.validationScheme = {
   body : {
      firstName : Joi.string(),
      lastName : Joi.string(),
      emailadres : Joi.string().email(),
      password : Joi.string(),
      image : Joi.string(),
   }
}

const updateUser = async (ctx : KoaContext<UpdateUserResponse, IdParams,
UpdateUserRequest>) => {
   const user = await userService.updateById(Number(ctx.params.id),
ctx.request.body);
   ctx.body = user
}

updateUser.validationScheme = {
   params : {
```

```
      id : Joi.number().integer().positive()
   },
   body : {
      firstName : Joi.string(),
      lastName : Joi.string(),
      emailadres : Joi.string().email(),
   }
}

const deleteUser = async (ctx : KoaContext<void, IdParams>) => {
   await userService.deleteById(Number(ctx.params.id));
   ctx.status = 204;
}

deleteUser.validationScheme = {
   params : {
      id : Joi.number().integer().positive()
   }
}

export default (parent: KoaRouter) => {
   const router = new Router<TemplateAppState, TemplateAppContext>({
     prefix: '/users',
   });

   const requireAdmin = makeRequireRole(Role.ADMIN);

   router.get("/", requireAuthentication, validate(getAllUsers.validationScheme),
getAllUsers);
   router.get("/:id", requireAuthentication,
validate(getUserById.validationScheme), checkUserId, getUserById);
   router.post("/", authDelay, validate(createUser.validationScheme), createUser);
   router.put("/:id", requireAuthentication,
validate(updateUser.validationScheme), checkUserId, updateUser);
   router.delete("/:id", requireAdmin, requireAuthentication,
validate(deleteUser.validationScheme), checkUserId, deleteUser);

   parent.use(router.routes()).use(router.allowedMethods());
};
```

- We beginnen met het maken van `rest/session.ts` voor de sessies aan te maken bij het inloggen

```
import Router from '@koa/router';
import Joi from 'joi';
import validate from '../core/validation';
import * as userService from '../service/user';
import type {KoaContext, KoaRouter, TemplateAppState, TemplateAppContext} from
'../types/koa';
import type { LoginResponse, LoginRequest } from '../types/user';
import { authDelay } from '../core/authentication';

const login = async (ctx: KoaContext<LoginResponse, void, LoginRequest>) => {
```

```
  const { emailadres, password } = ctx.request.body;
  const token = await userService.login(emailadres, password);

  ctx.status = 200;
  ctx.body = { token };
};

login.validationScheme = {
  body: {
    emailadres: Joi.string().email(),
    password: Joi.string(),
  },
};

export default function installSessionRouter(parent: KoaRouter) {
  const router = new Router<TemplateAppState, TemplateAppContext>({
    prefix: '/sessions',
  });

  router.post('/', authDelay, validate(login.validationScheme), login);

  parent.use(router.routes()).use(router.allowedMethods());
}
```

- Nu moeten we deze route nog toevoegen aan de restlaag `index.ts`:

```
import Router from "@koa/router";
import type { TemplateAppContext, TemplateAppState, KoaApplication } from
"../types/koa";
import userRoutes from "../rest/user";
import productRouter from "../rest/product"
import loginRouter from "../rest/session"

export default (app: KoaApplication) => {
  const router = new Router<TemplateAppState, TemplateAppContext>({
    prefix: "/api",
  });

  userRoutes(router);
  productRouter(router);
  loginRouter(router);

  app.use(router.routes()).use(router.allowedMethods());
};
```