Dokumentacja projektu z przedmiotu *Wybrane technologie JavaScript*

Katarzyna Pieczonka **semestr VI - letni**
nr albumu 38760 **rok akademicki 2022/23**

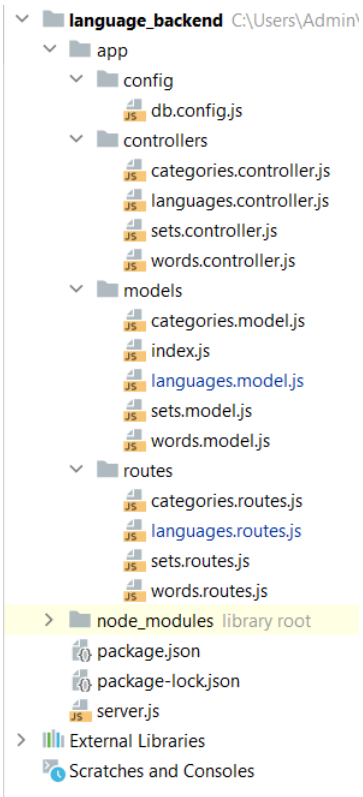
Tytuł projektu: Aplikacja internetowa do nauki języków obcych z wykorzystaniem frameworka VueJS.

1. Backend - projekt `language_backend`

Projekt wykorzystuje framework Express środowiska Node.js w celu połączenia się z bazą danych MySQL i wysyłania zapytań do serwera.
Zawiera pliki zawierające konfigurację połączenia z bazą danych, modele obiektów do których odczytywane są dane z tabeli, a także kontrolery z funkcjami pozwalającymi na operacje CRUD na tych obiektach oraz pliki ze zdefiniowanymi ścieżkami, które wywołują dane funkcje z kontrolera.

Uruchamianie aplikacji następuje poprzez wpisanie do linii poleceń: `node server.js`.

Struktura plików w aplikacji:



Konfiguracja połączenia z bazą danych znajduje się w pliku `db.config.js`. Plik ten definiuje adres serwera, na którym znajduje się baza, nazwę użytkownika, który łączy się z bazą oraz jego hasło, a także nazwę bazy danych. Ostatnia z opcji, czyli `pool` zawiera konfigurację połączenia z Sequelize.

```
module.exports = {
    HOST: "localhost",
    USER: "root",
    PASSWORD: "",
    DB: "lang",
    dialect: "mysql",
    pool: {
        max: 5,
        min: 0,
        acquire: 30000,
        idle: 10000
    }
};
```

Konfiguracja serwera, który tworzony jest przez aplikację, znajduje się w pliku `server.js`. Znajduje się tutaj port, na którym uruchamiany jest serwer, a także ścieżki do lokalizacji w projekcie plików zawierających ustawienia ścieżek uruchamiających poszczególne funkcje obsługujące obiekty z bazy danych.

```
const PORT = process.env.PORT || 8080;

require("./app/routes/words.routes.js")(app);
require("./app/routes/categories.routes.js")(app);
require("./app/routes/sets.routes.js")(app);
require("./app/routes/languages.routes.js")(app);
```

W katalogu app/models znajdują się pliki z modelami Sequelize obiektów, na których operuje użytkownik w aplikacji.
Każde z pól modeli jest odwzorowaniem kolumny w tabeli w bazie danych.
Modele stworzone są dla każdej z tabel istniejącej w bazie danych. Jeśli tabela odwzorowująca dany model nie istnieje, zostanie automatycznie utworzona w bazie danych przy uruchamianiu serwera.
Przykładowy model dla obiektu `Language`:

```
module.exports = (sequelize, Sequelize) => {
    const Language = sequelize.define("language", {
        jezyk: {
            type: Sequelize.STRING
        }
    });

    return Language;
};
```

Funkcje CRUD (tworzenie, odczytywanie, aktualizowanie i usuwanie danych) dla obiektów w aplikacji znajdują się w katalogu app/controllers. Każdy z kontrolerów odpowiada jednemu modelowi danych.
Kontrolery zawierają funkcje:

- `create` - funkcja tworząca nowy obiekt;

```
exports.create = (req, res) => {
    if (!req.body.jezyk) {
        res.status(400).send({
            message: "Content can not be empty!"
        });
        return;
    }

    const language = {
        jezyk: req.body.jezyk
    };

    Language.create(language)
        .then(data => {
            res.send(data);
        })
        .catch(err => {
            res.status(500).send({
                message:
                    err.message || "Some error occurred while creating the new language."
            });
        });
};
```

- `findAll` - funkcja zwracająca wszystkie obiekty danej klasy z tabeli;

```
exports.findAll = (req, res) => {
    const jezyk = req.query.jezyk;
    var condition = jezyk ? { jezyk: { [Op.like]: `%${jezyk}%` } } : null;

    Language.findAll( options: { where: condition })
        .then(data => {
            res.send(data);
        })
        .catch(err => {
            res.status(500).send({
                message:
                    err.message || "Some error occurred while retrieving languages."
            });
        });
};
```

- `findOne` - funkcja zwracająca tylko obiekt o podanym id;

```
exports.findOne = (req, res) => {
    const id = req.params.id;

    Language.findByPk(id)
        .then(data => {
            if (data) {
                res.send(data);
            } else {
                res.status(404).send({
                    message: `Cannot find language with id=${id}.`
                });
            }
        })
        .catch(err => {
            res.status(500).send({
                message: "Error retrieving word language id=" + id
            });
        });
};
```

- `update` - funkcja aktualizująca obiekt o podanym id;

```
exports.update = (req, res) => {
    const id = req.params.id;

    Language.update(req.body, {
        where: { id: id }
    })
        .then(num => {
            if (num == 1) {
                res.send({
                    message: "Language was updated successfully."
                });
            } else {
                res.send({
                    message: `Cannot update language with id=${id}. Maybe it was not found or
                });
            }
        })
        .catch(err => {
            res.status(500).send({
                message: "Error updating language with id=" + id
            });
        });
};
```

- `delete` - funkcja usuwająca obiekt o podanym id;

```
exports.delete = (req, res) => {
    const id = req.params.id;

    Language.destroy( options: {
        where: { id: id }
    })
        .then(num => {
            if (num == 1) {
                res.send({
                    message: "Language was deleted successfully!"
                });
            } else {
                res.send({
                    message: `Cannot delete language with id=${id}. Maybe it was not found!`
                });
            }
        })
        .catch(err => {
            res.status(500).send({
                message: "Could not delete language with id=" + id
            });
        });
};
```

- `deleteAll` - funkcja usuwająca wszystkie obiekty z danej tabeli;

```
exports.deleteAll = (req, res) => {
    Language.destroy( options: {
        where: {},
        truncate: false
    })
        .then(nums => {
            res.send({ message: `${nums} Languages were deleted successfully!` });
        })
        .catch(err => {
            res.status(500).send({
                message:
                    err.message || "Some error occurred while removing all languages."
            });
        });
};
```

Katalog app/routes zawiera pliki przyporządkowujące funkcje z kontrolerów do odpowiednich ścieżek, dzięki czemu mogą być one wywoływane w aplikacji.
Poniżej plik ze ścieżkami dla obiektu `language`:

```
module.exports = app => {
    const languages = require("../controllers/languages.controller.js");

    var router = require("express").Router();

    router.post( path: "/", languages.create);

    router.get( path: "/", languages.findAll);

    router.get( path: "/:id", languages.findOne);

    router.put( path: "/:id", languages.update);

    router.delete( path: "/:id", languages.delete);

    router.delete( path: "/", languages.deleteAll);

    app.use('/api/languages', router);
};
```

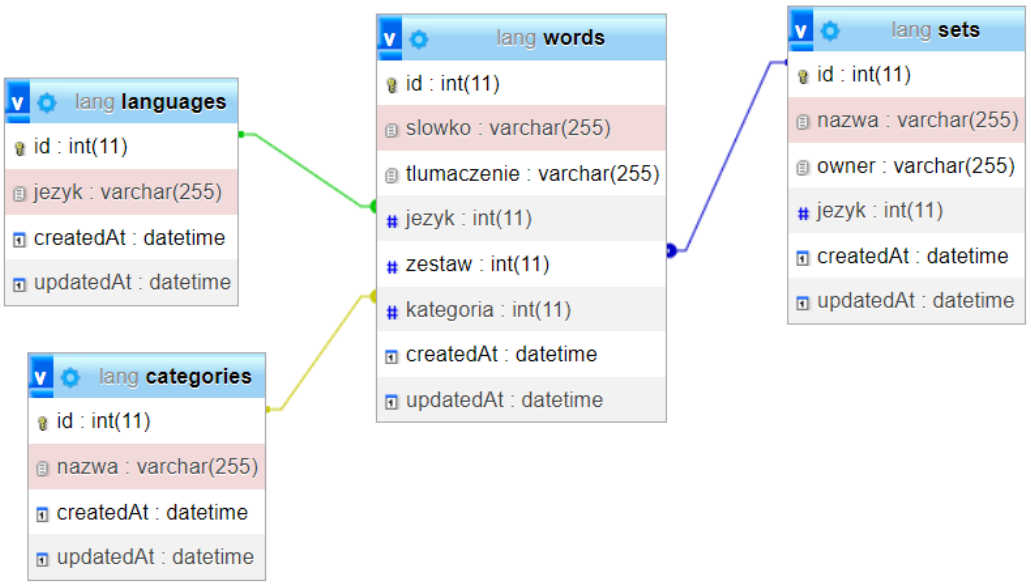
Znajdujący się w katalogu app/models plik `index.js` implementuje Sequelize czyli mapowanie obiektowo-relacyjne obiektów z bazy danych, a także łączy stworzone pliki z modelami obiektów na tabele w bazie danych MySQL.

```
db.words = require("./words.model.js")(sequelize, Sequelize);
db.categories = require("./categories.model.js")(sequelize, Sequelize);
db.sets = require("./sets.model.js")(sequelize, Sequelize);
db.languages = require("./languages.model.js")(sequelize, Sequelize);
```

2. Baza danych - MySQL

Dane wykorzystywane w aplikacji przechowywane są w relacyjnej bazie MySQL o nazwie `lang`. Zapytania do bazy danych wykonywane są poprzez API.
Konfiguracja połączenia z bazą danych znajduje się w projekcie `language_backend`, w pliku `db.config.js`, opisanym w punkcie 1.

Baza danych posiada następującą strukturę:

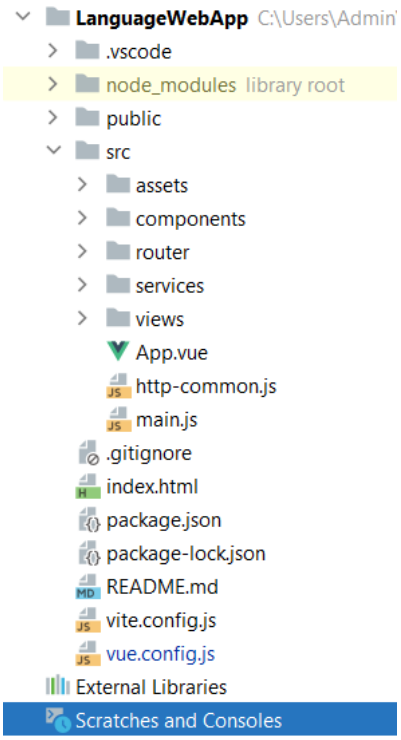


Jeśli jednak powyższe tabele nie istnieją w bazie danych podczas uruchamiania serwera, zostaną one automatycznie utworzone (jako puste).

3. Frontend - projekt `LanguageWebApp`

Frontendowa część projektu stworzona jest we framework'u Vue.js.

Struktura plików aplikacji:



Moduły vue, vue-router oraz axios zawarte są w pliku `package.json`.

```json
{
  "name": "languagewebapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "axios": "^1.4.0",
    "bootstrap": "^5.2.3",
    "bootstrap-vue": "^2.23.1",
    "jquery": "^3.7.0",
    "popper.js": "^1.16.1",
    "vue": "^3.2.47",
    "vue-router": "^4.2.0"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^4.0.0",
    "vite": "^4.1.4"
  }
}
```

W głównym katalogu projektu znajdują się pliki konfigurujące, m.in. port, na którym działa aplikacja backendowa (plik vue.config.js).

```
module.exports = {
    devServer: {
        port: 5173
    }
}
```

Znajduje się tam także plik `index.html`, zawierający główną strukturę strony internetowej, do której potem wstawiane są komponenty Vue.

Plik `main.js` implementuje do aplikacji arkusze stylów css, moduły odpowiedzialne za bootstrap oraz komponenty i router, odpowiedzialny za ścieżki w aplikacji.

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

import './assets/main.css'

import "../node_modules/bootstrap/dist/css/bootstrap.css"
import "../node_modules/bootstrap/dist/js/bootstrap.bundle"

const app = createApp(App)
app.use(router)
app.mount( rootContainer: '#app')
```

Za inicjalizację axiosa odpowiada w aplikacji plik `http-common.js`:

```
import axios from "axios";

5+ usages  Katarzyna Pieczonka
export default axios.create({
    baseURL: "http://localhost:8080/api",
    headers: {
        "Content-type": "application/json"
    }
});
```

Głównym komponentem Vue w aplikacji jest komponent `App.vue`. W nim wstawiane są pozostałe komponenty, m.in. `Header.vue`, zawierający strukturę głównego menu aplikacji w postaci paska nawigacji.

Komponent `App.vue`:

```
<script setup>
import { RouterLink, RouterView } from 'vue-router
import Header from "@/components/Header.vue";
</script>

<template>
    <Header />
</template>

<style scoped>

</style>
```

Komponent `Header.vue`:

```
<script setup>                                                    ⚠ 21  ✓ 1
defineProps({...})
</script>

<template>
    <div class="row">
        <div class="col-sm-12">
            <div id="app">
                <nav class="navbar navbar-expand navbar-dark bg-dark"...>

                <div class="container mt-3"...>
            </div>
        </div>
    </div>

</template>

<style scoped...>
```

Za główny widok strony internetowej odpowiada `HomeView.vue`, znajdujący się w katalogu src/views.

Plik `index.js`, znajdujący się w katalogu src/router, zawiera ścieżki, które wywołują poszczególne komponenty vue w aplikacji.

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'

const router = createRouter( options: {
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView
    },
    {
      path: "/words",
      name: "words",
      component: () => import('../components/WordsList.vue')
    },
    {
```

Katalog src/services zawiera pliki z metodami wysyłającymi zapytania do API.
Poniżej fragment przykładowego pliku service dla jednego z obiektów w aplikacji - obiektu `Language`:

```
import http from "../http-common";

1 usage  Katarzyna Pieczonka
class LanguageDataService {
    1 usage  Katarzyna Pieczonka
    getAll() {
        return http.get( url: "/languages");
    }

    1 usage  Katarzyna Pieczonka
    get(id) {
        return http.get( url: `/languages/${id}`);
    }

    1 usage  Katarzyna Pieczonka
    create(data) {
        return http.post( url: "/languages", data);
    }
```

Każdy z obiektów, które dostępne są w aplikacji posiada własny plik service.

Dla każdego z czterech obiektów (`Category, Language, Word, Set`) istnieją także po trzy komponenty vue. Wszystkie te komponenty zlokalizowane są w katalogu src/components.

Komponenty `Set.vue, Language.vue, Category.vue oraz Word.vue` mają za zadanie wyświetlać szczegóły o danym obiekcie, w celu edytowania go i aktualizacji. Przykładowy komponent `Language.vue`:

```
<template>
    <div v-if="currentLanguage" class="edit-form">
        <h4>Język:</h4>
        <form...>

        <button class="btn btn-dark my-3" @click="deleteLanguage">Usuń</button>

        <button type="submit" class="btn btn-dark m-3" @click="updateLanguage">Aktualizuj</button>
        <p>{{ message }}</p>
    </div>

    <div v-else...>
</template>

<script>
import LanguageDataService from "../services/LanguageDataService.js";

2 usages  Katarzyna Pieczonka
export default {
    name: "language",
    data() {...},
    methods: {
        getLanguage(id) {...},

        updateLanguage() {...},

        deleteLanguage() {...}
    },
    mounted() {...}
};
```

Komponenty `LangaugeList.vue, CategoryList.vue, SetList.vue` oraz `WordsList.vue` odpowiadają za wyświetlanie w aplikacji listy z obiektami danej klasy pobranymi z bazy danych.

Natomiast komponenty AddCategory.vue, AddLanguage.vue, AddSet.vue, AddWord.vue wyświetlają formularze służące do tworzenia nowych obiektów każdej z klas.

Aby uruchomić aplikację w trybie developerskim należy w linii poleceń wpisać: `npm run dev`.