

PROJEKT: Railway Adventure Lite

Wiktor Golicz | 25.11.2024 – 27.01.2025

Pomysł

Chcę stworzyć ciekawą grę przeglądarkową, która będzie polegać na budowaniu oraz zarządzaniu sieci kolejowej województwa śląskiego. Użytkownik będzie mógł budować tory między stacjami oraz tworzyć linie kolejowe, na których będzie mógł rozmieszczać pociągi różnych typów. Celem gry będzie zrealizowanie wszystkich zadań (np. przewiez x pasażerów, zgromadź y funduszy). Warunkiem porażki będzie bankructwo (brak funduszy).

Dlaczego lite?

Pomysł na stworzenie takiej gry od dawna był w mojej głowie. Ze względu na ograniczony czas realizacji projektu zdecydowałem się uprościć założenia gry. Prawdopodobnie w przyszłości stworzę pełną wersję tej gry, która będzie miała znacznie bardziej rozwinięte mechaniki.

Technologie

- ReactJS (v18) – Projekt ręcznie tworzony, z wykorzystaniem bundlera *webpack* (własna konfiguracja, customowe wtyczki)
- SCSS
- Pakiety NPM
 - leaflet, react-leaflet – Świecka biblioteka do tworzenia interaktywnych map
 - react-leaflet-component-marker – Wtyczka do biblioteki *react-leaflet* pozwalająca na użycie komponentów reacta jako elementy mapy (pinzki)
 - mobx, mobx-react – Biblioteka do zarządzania stanem aplikacji
 - pako – Biblioteka do kompresji plików zapisu
 - react-router – Biblioteka do obsługi routera/ścieżek/adresu url
 - react-custom-scrollbars-2 – Biblioteka pozwalająca na implementację ładnych suwaków do przewijania

Mechaniki / Funkcjonalności

- Budowanie torów między stacjami kolejowymi
 - Różne rodzaje torów (w zależności od czasu)
- Tworzenie linii kolejowych z użyciem zbudowanej infrastruktury
- Przewożenie pasażerów
 - Różne rodzaje pociągów (w zależności od czasu)
- Wyzwania jako warunek ukończenia gry

Możliwości rozwoju

- Dodatkowe plansze – inne województwa
- Inne tryby rozgrywki
- Pełna wersja gry

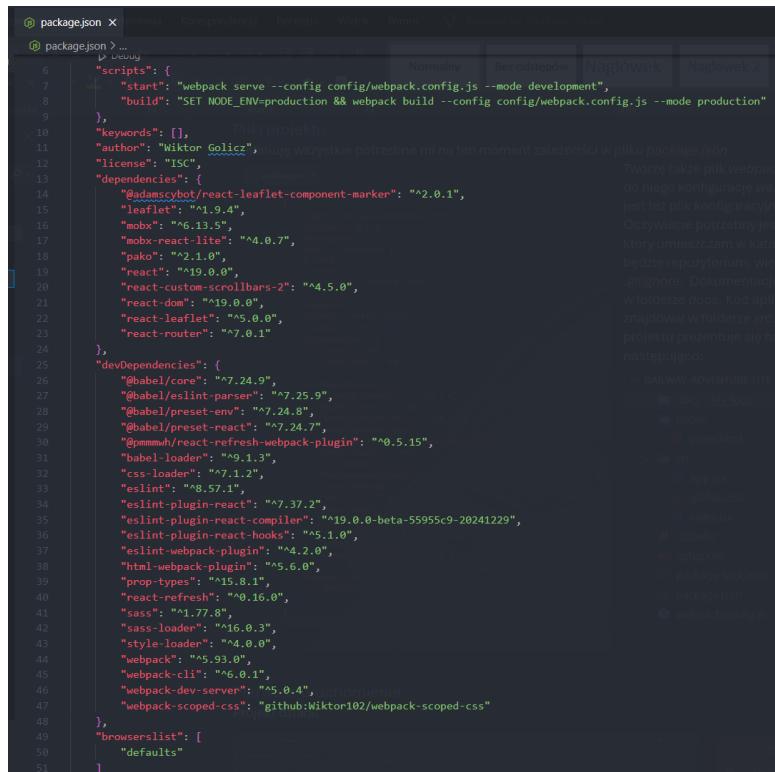
Uwaga! Sprawozdanie jest wykonywane po skończeniu projektu. Wszystkie zrzuty ekranu kodu zostały wykonane z kompletnym kodem. Niektóre zrzuty ekranu wyników (przeglądarki) zostały zrobione wcześniej (tj. podczas pracy na projektem) aby pokazać stan gry w danym momencie.

Tworzenie projektu

Mimo możliwości stworzenia automatycznego projektu (czy to z wykorzystaniem *create-react-app* czy *vite*), zdecydowałem się zainstalować wszystkie zależności samodzielnie i samodzielnie skonfigurować bundler – *webpack*'a. Tym sposobem będę miał pełną kontrolę nad wtyczkami. Przyda się to, ponieważ zamierzam użyć własnej wtyczki do lepszego zarządzania stylami tak by były odizolowane style konkretnych komponentów - github.com/Wiktor102/webpack-scoped-css.

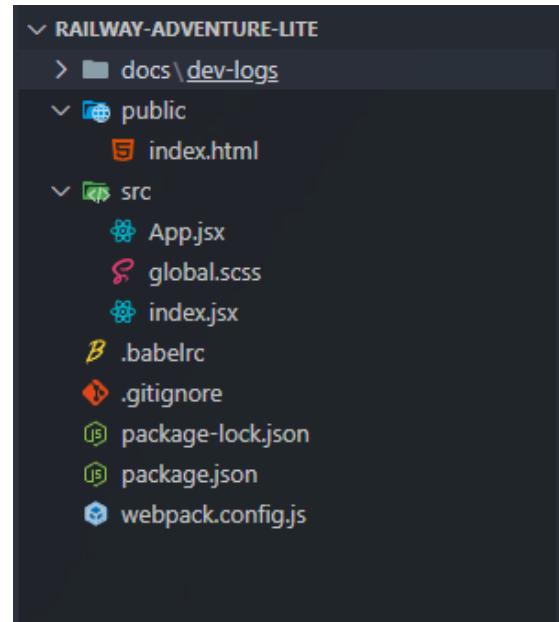
Konfiguracja projektu

Definiuję wszystkie potrzebne mi na ten moment zależności w pliku *package.json* (o zależnościach więcej napisałem w akapicie „[technologie](#)”).



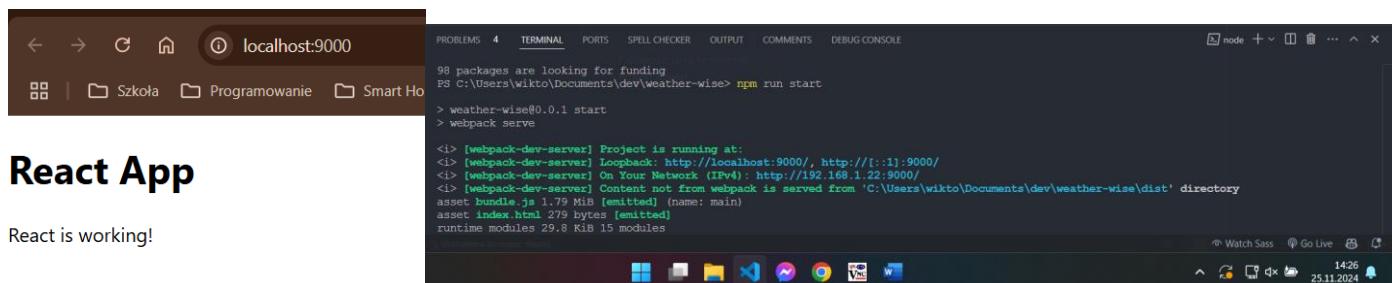
```
package.json
{
  "scripts": {
    "start": "webpack serve --config config/webpack.config.js --mode development",
    "build": "SET NODE_ENV=production && webpack build --config config/webpack.config.js --mode production"
  },
  "keywords": [],
  "author": "Wiktor Golicz", // Wszystkie potrzebne mi na ten moment zależności w pliku package.json.
  "license": "ISC",
  "dependencies": {
    "@adamszybot/react-leaflet-component-marker": "^2.0.1",
    "leaflet": "^1.9.4",
    "mobx": "6.13.5",
    "mobx-react-lite": "^4.0.7",
    "pako": "^2.1.0",
    "react": "^19.0.0",
    "react-custom-scrollbars-2": "^4.5.0",
    "react-dom": "^19.0.0",
    "react-leaflet": "^5.0.0",
    "react-router": "^7.0.1"
  },
  "devDependencies": {
    "@babel/core": "^7.24.9",
    "@babel/eslint-parser": "^7.25.0",
    "@babel/preset-env": "^7.24.8",
    "@babel/preset-react": "^7.24.7",
    "@commonh/react-refresh-webpack-plugin": "^0.5.15",
    "babel-loader": "9.1.3",
    "css-loader": "7.1.2",
    "eslint": "8.57.1",
    "eslint-plugin-react": "^7.37.2",
    "eslint-plugin-react-compiler": "19.0.0-beta-55955c9-20241229",
    "eslint-plugin-react-hooks": "^5.1.0",
    "eslint-webpack-plugin": "4.2.0",
    "html-webpack-plugin": "5.6.0",
    "prop-types": "15.8.1",
    "react-refresh": "^0.16.0",
    "sass": "1.77.8",
    "sass-loader": "16.0.3",
    "style-loader": "4.0.0",
    "webpack": "5.93.0",
    "webpack-cli": "6.0.1",
    "webpack-dev-server": "5.0.4",
    "webpack-scoped-css": "github:Wiktor102/webpack-scoped-css"
  },
  "browserslist": [
    "defaults"
  ]
}
```

Tworzę także plik *webpack.config.js* i dodaję do niego konfigurację *webpacka*. Potrzebny jest też plik konfiguracyjny *.babelrc* oraz *.eslintrc.json*. Oczywiście potrzebny jest też plik *index.html*, który umieszczam w katalogu *public*. Projekt będzie repozytorium, więc tworzę plik *.gitignore*. Dokumentację będę przechowywał w folderze *docs*. Kod aplikacji będzie się znajdował w folderze *src*. Cała struktura projektu prezentuje się na ten moment następująco:



Pierwsze uruchomienie

Projekt działa!



```
localhost:9000
PROBLEMS 4 TERMINAL PORTS SPELL CHECKER OUTPUT COMMENTS DEBUG CONSOLE
98 packages are looking for funding
PS C:\Users\wikto\Documents\dev\weather-wise> npm run start
> weather-wise@0.0.1 start
> webpack serve

<!-- [webpack-dev-server] Project is running at: -->
<!-- [webpack-dev-server]  Logstash: http://localhost:9000/, http://[::1]:9000/
<!-- [webpack-dev-server] On Your Network (IPv4): http://192.168.1.22:9000/
<!-- [webpack-dev-server] Content not from webpack is served from 'C:\Users\wikto\Documents\dev\weather-wise\dist' directory
asset bundle.js 1.79 MiB [emitted] (name: main)
asset index.html 279 bytes [emitted]
runtime modules 29.8 KiB 15 modules
Watch SASS Go Live
14:26 25.11.2024
```

Routing

Do zarządzania routingiem (ścieżką w adresie URL) zdecydowałem się użyć biblioteki *react-router*. Jest to bardzo popularna biblioteka, która ułatwia większość zadań. Definicja wszystkich ścieżek następuje w komponencie *App*. Wygląda to następująco:

```
App.jsx
src > App.jsx > ...
6 // not lazy-loaded components
7 import { GameStoreProvider } from "./store/GameStoreProvider";
8 import PassengerMenu from "./Ui/pages/PassengerMenu/PassengerMenu";
9
10 // lazy loaded components
11 const TracksMenu = lazy(() => import("./Ui/pages/TracksMenu"));
12 const RoutesMenu = lazy(() => import("./Ui/pages/RoutesMenu/RoutesMenu"));
13 const RouteDetails = lazy(() => import("./Ui/pages/RoutesMenu/RouteDetails"));
14 const TrainsMenu = lazy(() => import("./Ui/pages/TrainsMenu/TrainsMenu"));
15 const BuyTrain = lazy(() => import("./Ui/pages/TrainsMenu/BuyTrain"));
16 const AssignRoute = lazy(() => import("./Ui/pages/TrainsMenu/AssignRoute"));
17
18 function App() {
19   return (
20     <BrowserRouter basename={window.location.hostname === "wiktorgolicz.pl" ? "/ral" : undefined}>
21       <Routes>
22         <Route index element={<HomePage />} />
23         <Route
24           path="game"
25           element={
26             <GameStoreProvider>
27               <Game />
28             </GameStoreProvider>
29           }
30         >
31           <Route path="tracks/*" element={<TracksMenu />} />
32           <Route path="build/*" />
33         </Route>
34         <Route
35           path="routes"
36           element={<RoutesMenu />} />
37           <Route path="routes/create" element={<RouteDetails />} />
38           <Route path="routes/details/:routeId" element={<RouteDetails />} />
39         </Route>
40         <Route
41           path="trains"
42           element={<TrainsMenu />} />
43           <Route path="trains/:trainId/assign-route" element={<AssignRoute />} />
44           <Route path="trains/buy/*" element={<BuyTrain />} />
45         </Route>
46         <Route path="passengers/*" element={<PassengerMenu />} />
47       </Routes>
48     </BrowserRouter>
49   );
50 }
51 }
```

Dodatkowo, zastosowana tutaj ciekawa funkcjonalność react'a – *lazy-loading*. Pozwala on wczytywać wybrane komponenty dopiero gdy są potrzebne (optymalizacja). Gdy komponenty się ładują wyświetlna jest animacja ładowania podana jako prop do komponentu *Suspense*. Ten jest wykorzystany wewnątrz komponentu *Ui* i okala *Outlet* – będzie to widoczne na kolejnych zrzutach ekranu.

Strona główna

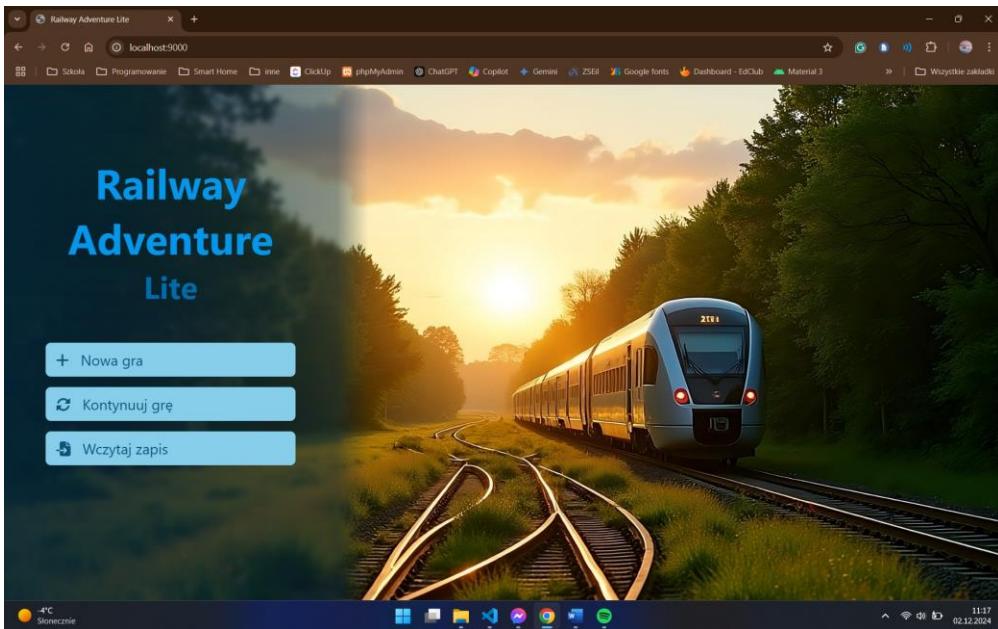
Komponent *Home* zawiera 3 przyciski:

- Nowa gra
- Kontynuuj grę
- Wczytaj grę

Wszystkie przekierowują do ścieżki */game*. Sposób wczytania jest przekazywany jako *queryParameter*, czyli po znaku ? w adresie url. O samym wczytywaniu będzie później. Oczywiście przycisk kontynuacji będzie nie aktywny jeśli w przeglądarce nie ma zapisu gry.

```
1 import { useMemo } from "react"; 4.6k (gzipped: 1.9k)
2 import { NavLink } from "react-router"; 148.9k (gzipped: 47.2k)
3
4 // services
5 import PersistenceService from "../services/PersistenceService";
6
7 import background from "../assets/icons/background-bigger.jpeg";
8
9 import styles from "./Home.component.scss";
10
11 function HomePage() {
12   const hasSave = useMemo(() => PersistenceService.loadFromLocalStorage() != null, []);
13
14   return (
15     <div className="home" data-style={styles}>
16       <img src={background} alt="" className="background" />
17       <div className="left-panel">
18         <h1>Railway Adventure</h1>
19         <h2>Lite</h2>
20         <NavLink to="/game?load=new">
21           <i className="fas fa-plus"></i> Nowa gra
22         </NavLink>
23         <NavLink
24           to={hasSave ? "/game?load=browser" : ""}
25           className={hasSave ? "" : "disabled"}
26           aria-disabled={!hasSave}
27         >
28           <i className="fas fa-refresh"></i> Kontynuuj grę
29         </NavLink>
30         <NavLink to="/game?load=file">
31           <i className="fas fa-file-import"></i> Wczytaj zapis
32         </NavLink>
33       </div>
34     </div>
35   );
36 }
37
38 export default HomePage;
```

Po zastosowaniu stylów, strona główna prezentuje się tak:



Główny komponent gry

Jest to bardzo prosty komponent. Wyświetla mapę oraz ui. Jeśli stan aplikacji nie został jeszcze wczytany, wyświetlany jest ekran ładowania. Implementacja widoczna obok.

Ekran ładowania

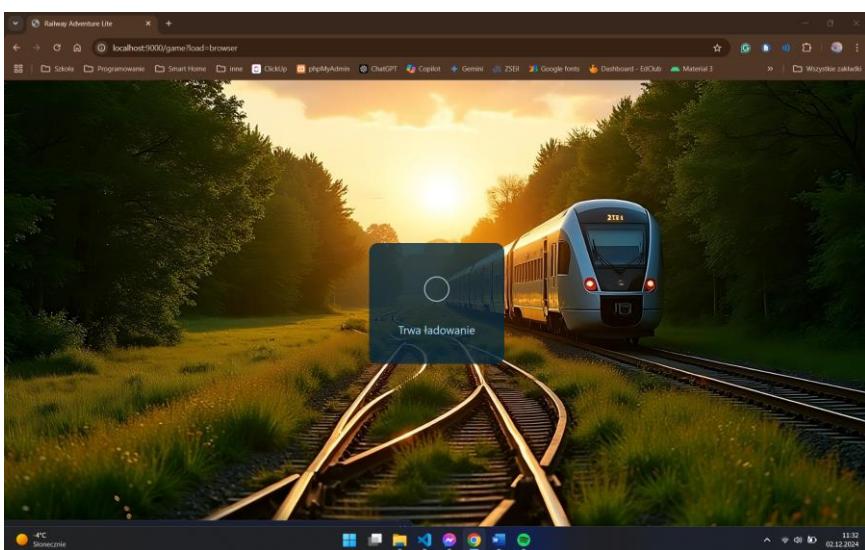
Style dla animacji ładowania pochodzą z biblioteki *loaders.css*.

Całość komponentu prezentuje się tak:

```
1 import background from "../../assets/icons/background-bigger.jpeg";
2 import style from "./LoadingScreen.component.scss";
3
4 function LoadingScreen() {
5   return (
6     <div className="loading-screen" data-style={style}>
7       <img src={background} alt="Tlo z pociągiem jadącym podczas wschodu słońca" />
8       <div className="spinner-container">
9         <div className="ball-clip-rotate">
10           <div></div>
11           <div></div>
12           <p>Trwa ładowanie</p>
13         </div>
14       </div>
15     );
16   }
17
18 export default LoadingScreen;
```

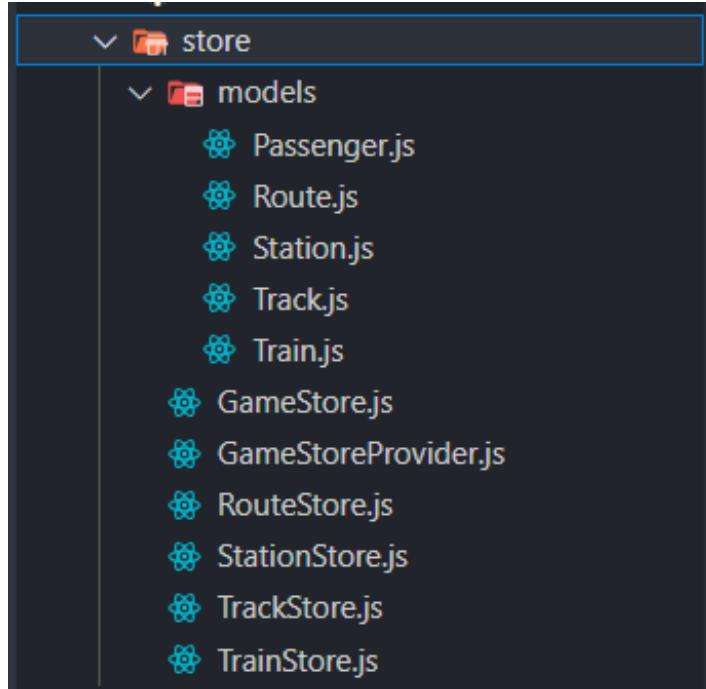
```
1 // hooks
2 import { useIsStoreLoading } from "../store/GameStoreProvider"
3
4 // components
5 import Ui from "../Ui/Ui";
6 import Map from "./Map/Map";
7 import LoadingScreen from "./LoadingScreen>LoadingScreen";
8
9 import style from "./Game.component.scss";
10
11 function Game() {
12   const loading = useIsStoreLoading();
13   if (loading) return <LoadingScreen />;
14
15   return (
16     <div className="game" data-style={style}>
17       <Ui />
18       <Map />
19     </div>
20   );
21 }
22
23 export default Game;
24
```

A w przeglądarce wygląda następująco:



Stan aplikacji

Ten projekt jest dość złożony i będzie wymagał łączenia ze sobą danych na temat różnych obiektów w grze. Najlepszym sposobem na uproszczenie wszystkich zależności jest *abstrakcja*, której najłatwiej dokonać w programowaniu obiektowym. React jest raczej biblioteką, która obrała inne podejście – niezmienność (immutability) danych. Oznacza to, że podejście OOP jest utrudnione, ale nie niemożliwe. Dlatego, do zarządzania stanem, wykorzystam bibliotekę *MobX*. Pozwala ona na przechowywanie danych w klasach a jej integracja z React'em sprawia, że wszystko działa jak należy.



GameStoreProvider

Tak jak wspomniałem wyżej, komponenty react'a będą miały dostęp do *GameStore* za pomocą kontekstu. Ten komponent zajmuje się jego definicją oraz zarządzaniem/wczytywaniem. Logikę wczytującą opiszę później przy okazji mechanizmu zapisywania. Reszta kodu jest prosta:

```
GameStoreProvider.js  App.js
src > store > GameStoreProvider.js > ...
You 5 days ago | author (You)
1 import { createContext, useContext, useEffect, useRef, useState } from "react"; 5k (gzipped: 2.1k)
2 import { useNavigate, useSearchParams } from "react-router"; 148.9k (gzipped: 47.2k)
3 import PropTypes from "prop-types"; 1.4k (gzipped: 778)
4
5 import GameStore from "./GameStore";
6
7 const GameStoreContext = createContext();
8
9 You 5 days ago | author (You)
10 function GameStoreProvider({ children }) {
11   const [store, setStore] = useState(null);
12   const [searchParams] = useSearchParams();
13   const redirectedRef = useRef(false);
14   const navigate = useNavigate();
15
16   useEffect(() => {
17     if (store) {
18       const handler = e => {
19         e.preventDefault();
20       };
21
22       addEventListener("beforeunload", handler);
23       return () => removeEventListener("beforeunload", handler);
24     }
25   }, []);
26
27   useEffect(() => {
28     if (store) {
29       store.dispose();
30       setStore(null);
31     }
32   }, [store]);
33
34   return <GameStoreContext.Provider value={{ store, loading: store == null }}>{children}</GameStoreContext.Provider>;
35 }
36
37 GameStoreProvider.propTypes = {
38   children: PropTypes.node.isRequired
39 };
40 }
```

W tym projekcie architektura będzie następującą: główny store *GameStore* oraz mieszkające w nim *StationStore*, *TrackStore*, *RouteStore*, oraz *TrainStore*. Dostęp do *GameStore* komponenty react'a uzyskają poprzez *context api* a do pozostałych store'ów jako argumenty konstruktora.

GameStore

Główny store będzie instancjonował wszystkie pozostałe oraz będzie je przechowywać. Dodatkowo będzie też przechowywać dane o funduszach gracza i ustaloną prędkość gry ora zajmować się obsługą błędów. Poniżej nagłówek tej klasy (implementacja metod jest trywialna, więc nie będę jej tutaj opisywał):

```
GameStore.js x
src > store > GameStore.js > ...
You 4 days ago | author (You)
1 class GameStore {
2   stationStore;
3   trackStore;
4   routeStore;
5   trainStore;
6
7   /* @type {number} */
8   gameSpeed = 1;
9
10  /* @type {number} */
11  money = 30_000;
12
13  /* @type {string|null} */
14  error = null;
15  _errorTimeout = null;
16
17  _passengerSpawnInterval = { id: null, time: 0 };
18
19 /**
20  * @param {GameStoreSerialized|undefined} data
21  */
22 constructor(data) {
23   makeAutoObservable(this, { _passengerSpawnInterval: false });
24
25   if (data) {
26     this.stationStore = StationStore.fromJSON(data.stations, this);
27     this.trackStore = TrackStore.fromJSON(data.tracks, this);
28     this.routeStore = RouteStore.fromJSON(data.routes, this);
29     this.trainStore = TrainStore.fromJSON(data.trains, this);
30
31     this.gameSpeed = data.gameSpeed;
32     this.money = data.money;
33   } else {
34     this.stationStore = new StationStore(this);
35     this.trackStore = new TrackStore(this);
36     this.routeStore = new RouteStore(this);
37     this.trainStore = new TrainStore(this);
38   }
39
40   this._passengerSpawnInterval.id = setInterval(() => {
41     this.stationStore.generatePassengers();
42   }, 1000 * 60 * 3);
43
44 }
45
46 // this._passengerSpawnInterval.clearAutorun = autorun(() => { ... })
47 }
```

Komponent ten jest użyty w komponencie *App* i okala komponent *Game* (w ścieżce url */game*).

Ponieważ importowanie kontekstu, używanie hook'a `useContext` oraz wyciąganie z niego atrybutu `gameStore` byłoby uciążliwe, stworzyłem własne hooki `useGameStore` oraz `useIsStoreLoading`:

```
70  /**
71   * @returns {GameStore}
72  */
73 function useGameStore() {
74   const context = useContext(GameStoreContext);
75
76   if (!context) {
77     throw new Error("useGameStore must be used within a GameStoreProvider!");
78   }
79
80   return context.store;
81 }
82
83 function useIsStoreLoading() {
84   const context = useContext(GameStoreContext);
85
86   if (!context) {
87     throw new Error("useIsStoreLoading must be used within a GameStoreProvider!");
88   }
89
90   return context.loading;
91 }
92
93 export { GameStoreProvider, useGameStore, useIsStoreLoading };
```

StationStore

```
constructor(gameStore) {
  makeAutoObservable(this, { gameStore: false });

  this.gameStore = gameStore;
  this.stationsMap = stationsData.features.reduce((map, station) => {
    if (station.properties.NAZWA_POS.includes("Gr")) return map;
    const s = new Station(station, gameStore);
    map.set(s.name, s);
    return map;
  }, new Map());
}
```

Jest to klasa zajmująca się zarządzaniem stacjami, które przechowuje w formie mapy. Umożliwia to szybki dostęp do obiektu każdej stacji po jej nazwie ze złożonością obliczeniową O^1 . Stacje są reprezentowane przez klasę *Station* o której za chwilę. Obiekty te są tworzone w konstruktorze na bazie pliku w formacie *json*, który zawiera wszystkie konieczne dane.

Klasa ta zawiera również liczne metody do pobierania stacji spełniających różne warunki (np. według nazwy, albo tylko takie które mają połączenia z innymi stacjami). Dodatkowo, *StationStore* zajmuje się też przechowywaniem informacji o aktualnie „przyciągniętej” stacji – więcej o tym przy opisie budowania torów. Ostatnią ważną funkcją tej klasy jest generowanie pasażerów, ale o tym też później.

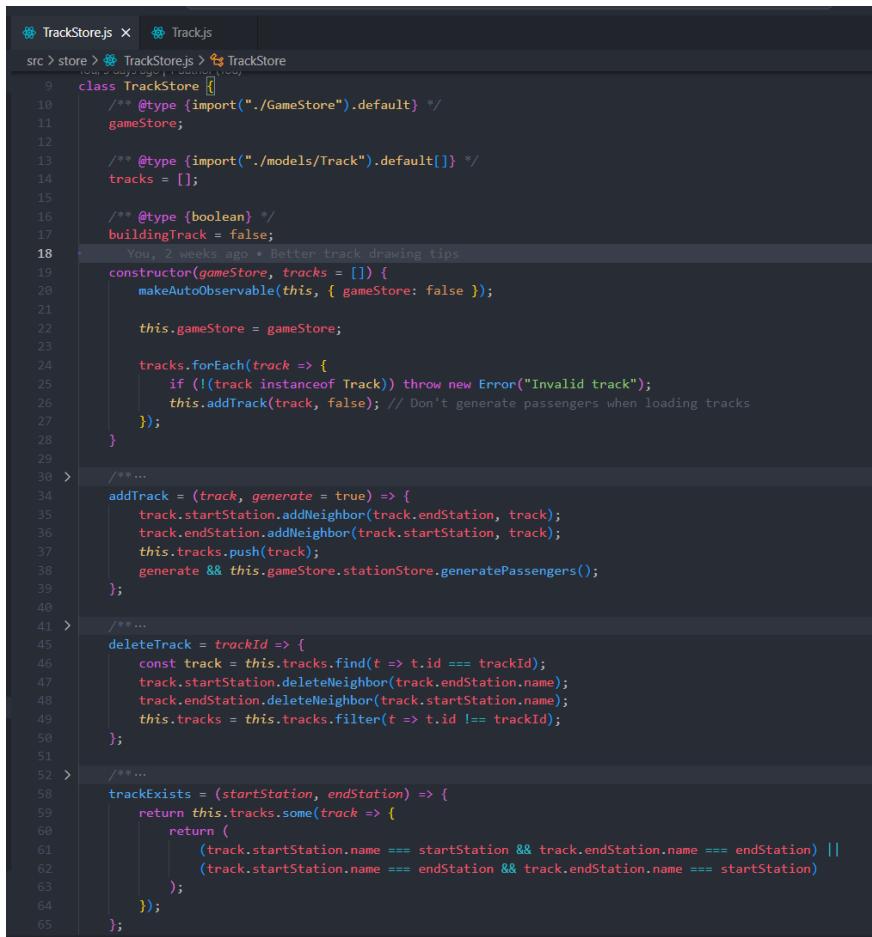
Station – klasa reprezentująca poszczególne stacje

Najważniejszymi atrybutami są nazwa, rozmiar i koordynaty oraz oczywiście tablica z oczekującymi pasażerami. Ponadto każda stacja posiada „mapę” z sąsiadami – pozwala to na bardziej efektywny *pathfinding*. Metody aktualizujące tą mapę – `addNeighbor` oraz `deleteNeighbor` – są wywoływane przy okazji budowy lub usuwania torów (czyli z poziomu *TrackStore*). Zarządzanie pasażerami (ich wsiadaniem oraz częścią generacji) też jest zadaniem każdej stacji – omówienie metod tego dotyczących nieco dalej. Tymczasem przedstawiam implementację najważniejszych elementów tej klasy:

```
1 import { makeAutoObservable } from "mobx";  55.5k (gzipped: 15.5k)
2 import Passenger from "./Passenger";
3
4 > /**
5  * You.5 days ago | 1 author (You)
6  * class Station {
7  *   /**
8  *    * @type {import("../GameStore").default}
9  *    */
10 *   gameStore;
11
12 *   name;
13 *   size;
14 *   coordinates;
15
16 *   /**
17 *    * @type {Map<String,import("./Track").default>}
18 *   neighbors = new Map();
19
20 *   /**
21 *    * @type {Array<import("./Passenger").default>}
22 *   waitingPassengers = [];
23
24 *   /**
25 *    * @param {geometry, properties, waitingPassengers = []} args
26 *    */
27 *   constructor(args, gameStore) {
28 *     makeAutoObservable(this, { coordinates: false, name: false, gameStore: false });
29
30 *     this.gameStore = gameStore;
31 *     this.name = properties.NAZWA_POS;
32 *     this.size = properties.size;
33 *     this.coordinates =
34 *       geometry.coordinates[0] > geometry.coordinates[1] ? geometry.coordinates : geometry.coordinates.reverse();
35 *     this.waitingPassengers = waitingPassengers;
36 *   }
37
38 *   /**
39 *    * @param {station, track}
40 *    */
41 *   addNeighbor(station, track) {
42 *     this.neighbors.set(station.name, track);
43 *   }
44
45 *   /**
46 *    * @param {stationName}
47 *    */
48 *   deleteNeighbor(stationName) {
49 *     this.neighbors.delete(stationName);
50 *   }
51 }
```

Format importowanego pliku *json* nie jest idealny, więc wewnątrz konstruktora są wymagane pewne korekty takie jak sprawdzenie i przetworzenie współrzędnych geograficznych.

TrackStore



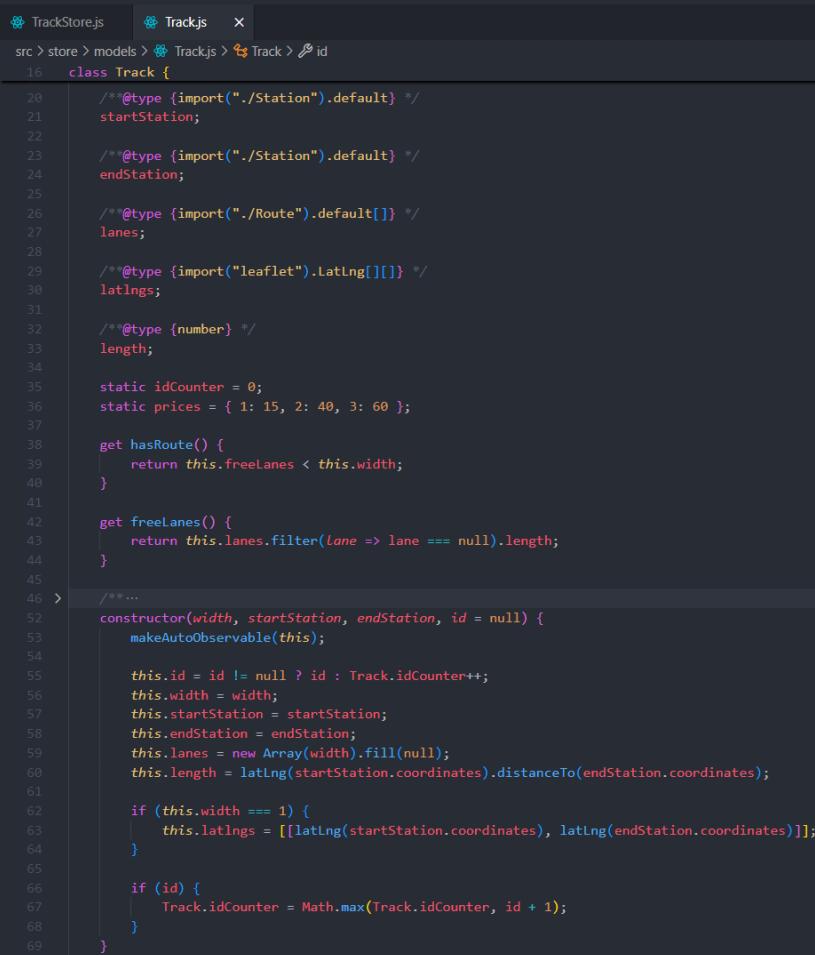
```
src > store > TrackStore.js > TrackStore.js
src > store > TrackStore.js > TrackStore.js
9  class TrackStore {
10    /* @type {import("./GameStore").default} */
11    gameStore;
12
13    /* @type {import("./models/Track").default[]} */
14    tracks = [];
15
16    /* @type {boolean} */
17    buildingTrack = false;
18    You, 2 weeks ago * Better track drawing tips
19    constructor(gameStore, tracks = []) {
20      makeAutoObservable(this, { gameStore: false });
21
22      this.gameStore = gameStore;
23
24      tracks.forEach(track => {
25        if (!track instanceof Track) throw new Error("Invalid track");
26        this.addTrack(track, false); // Don't generate passengers when loading tracks
27      });
28    }
29
30    /* ... */
31    addTrack = (track, generate = true) => {
32      track.startStation.addNeighbor(track.endStation, track);
33      track.endStation.addNeighbor(track.startStation, track);
34      this.tracks.push(track);
35      generate && this.gameStore.stationStore.generatePassengers();
36    };
37
38    /* ... */
39    deleteTrack = trackId => {
40      const track = this.tracks.find(t => t.id === trackId);
41      track.startStation.deleteNeighbor(track.endStation.name);
42      track.endStation.deleteNeighbor(track.startStation.name);
43      this.tracks = this.tracks.filter(t => t.id !== trackId);
44    };
45
46    /* ... */
47    trackExists = (startStation, endStation) => {
48      return this.tracks.some(track => {
49        return (
50          (track.startStation.name === startStation && track.endStation.name === endStation) ||
51          (track.startStation.name === endStation && track.endStation.name === startStation)
52        );
53      });
54    };
55
56    /* ... */
57    /* @type {number} */
58    width;
59
60    static idCounter = 0;
61    static prices = { 1: 15, 2: 40, 3: 60 };
62
63    get hasRoute() {
64      return this.freelanes < this.width;
65    }
66
67    get freelanes() {
68      return this.lanes.filter(lane => lane === null).length;
69    }
70
71    /* ... */
72    constructor(width, startStation, endStation, id = null) {
73      makeAutoObservable(this);
74
75      this.id = id != null ? id : Track.idCounter++;
76      this.width = width;
77      this.startStation = startStation;
78      this.endStation = endStation;
79      this.lanes = new Array(width).fill(null);
80      this.length = latLng(startStation.coordinates).distanceTo(endStation.coordinates);
81
82      if (this.width === 1) {
83        this.latlngs = [[latLng(startStation.coordinates), latLng(endStation.coordinates)]];
84      }
85
86      if (id) {
87        Track.idCounter = Math.max(Track.idCounter, id + 1);
88      }
89    }
90  
```

Zadaniem tej klasy jest przechowywanie torów, dodawanie i usuwanie ich oraz sprawdzanie czy już nie istnieją (by nie istniała możliwość połączenia tej samej pary stacji więcej niż jednym torem).

Obok widoczna większość jej implementacji.

Na tym etapie warto jest wyjaśnić istnienie atrybutu *buildingTrack*. Jest on używany do wyświetlanego użytkownikowi odpowiednich w danym momencie podpowiedzi sterowania w rogu ekranu.

Track – klasa reprezentująca tor



```
src > store > models > Track.js > Track.js
src > store > models > Track.js > Track.js
16  class Track {
17
18    /* @type {import("./Station").default} */
19    startStation;
20
21    /* @type {import("./Station").default} */
22    endStation;
23
24    /* @type {import("./Route").default[]} */
25    lanes;
26
27    /* @type {import("leaflet").LatLng[][]} */
28    latlngs;
29
30    /* @type {number} */
31    length;
32
33    static idCounter = 0;
34    static prices = { 1: 15, 2: 40, 3: 60 };
35
36    get hasRoute() {
37      return this.freelanes < this.width;
38    }
39
40    get freelanes() {
41      return this.lanes.filter(lane => lane === null).length;
42    }
43
44    /* ... */
45    constructor(width, startStation, endStation, id = null) {
46      makeAutoObservable(this);
47
48      this.id = id != null ? id : Track.idCounter++;
49      this.width = width;
50      this.startStation = startStation;
51      this.endStation = endStation;
52      this.lanes = new Array(width).fill(null);
53      this.length = latLng(startStation.coordinates).distanceTo(endStation.coordinates);
54
55      if (this.width === 1) {
56        this.latlngs = [[latLng(startStation.coordinates), latLng(endStation.coordinates)]];
57      }
58
59      if (id) {
60        Track.idCounter = Math.max(Track.idCounter, id + 1);
61      }
62    }
63
64    /* ... */
65  
```

Oprócz podstawowych atrybutów jak stacja początkowa i końcowa, istniała potrzeba przechowywania dodatkowych danych:

- *width* – „szerokość toru”, w tej grze każdy obiekt tej klasy reprezentuje tak naprawdę jedynie połączenie między dwoma stacjami. Mogą istnieć 1, 2 lub 3 równoległe tory i ten atrybut przechowuje tą liczbę.
- *lanes* – „pasy” tym mianem określam każdy z „prawdziwych” indywidualnych torów. Jest to tablica, która przechowuje referencje do trasy zajmujących konkretnie tory. Długość tej tablicy jest = *width*.
- *latlngs* – tablica przechowywująca koordynaty każdego z poszczególnych torów. Idealnie by było gdyby te były obliczane wewnątrz tej klasy, jednak z uwagi na ograniczony czas zdecydowałem się że będą obliczane wewnątrz komponentów react'a i będą przekazywane do opisywanej struktury danych.

Klasa definiuje też 2 proste gettery: *hasRoutes* i *freeLanes*.

```

71 >     /**
72      * @param {Route} route
73      */
74      addRoute(route) {
75          const emptyLaneIndex = this.lanes.findIndex(lane => lane === null);
76          if (emptyLaneIndex === -1) return [null, `Osiągnięto maksymalną liczbę tras na tym torze (${this.width})`];
77          this.lanes[emptyLaneIndex] = route;
78          return [emptyLaneIndex, null];
79      }
80
81 >     /**
82      * @param {String} routeId
83      */
84      removeRoute(routeId) {
85          this.lanes = this.lanes.map(route => (route?.id === routeId ? null : route));
86      }
87
88 >     /**
89      * @param {Number} width
90      */
91      updateWidth(width) {
92          if (width > this.width) {
93              this.lanes.splice(1, 0, null);
94          } else if (width < this.width) {
95              const freeIndex = this.lanes.findIndex(lane => lane === null);
96
97              if (freeIndex === -1) {
98                  return "Nie można zmniejszyć szerokości toru, ponieważ znajdują się na nim trasy";
99              }
100
101             this.lanes = this.lanes.splice(freeIndex, 1);
102         }
103
104         this.width = width;
105     }
106
107     /**
108      * @param {import("leaflet").LatLng[][]} latlngs
109      * @returns {void}
110      */
111     setLatlngs(latlngs) {
112         this.latlngs = latlngs;
113     }
114
115
116
117

```

Metody pozwalają na dopisanie/odpisanie trasy. Wraz z opisany wyżej atrybutem *lanes* jest to konieczne w celu kolorowego zaznaczenia torów z trasą na mapie.

Możliwa jest także zmiana szerokości toru – pod warunkiem, że nie przeszkadza w tym liczba przypisanych tras.

Pasażerowie

```

src > store > models > ✎ Passenger.js ...
  1  class Passenger {
  2      /* @type {import("../GameStore").default} */
  3      gameStore;
  4
  5      /*@type {number}*/
  6      id;
  7
  8      /*@type {string}*/
  9      originName;
 10
 11      /*@type {string}*/
 12      destinationName;
 13
 14      static idCounter = 0;
 15
 16      constructor(origin, destination, gameStore) {
 17          makeAutoObservable(this, { gameStore: false });
 18
 19          this.gameStore = gameStore;
 20          this.id = Passenger.idCounter++;
 21          this.originName = origin;
 22          this.destinationName = destination;
 23      }
 24
 25      /**
 26      * @param {Route} route
 27      */
 28      pay(route) {
 29          const stationIndex = route.stations.indexOf(this.originName);
 30          if (stationIndex === -1) return;
 31
 32          const destinationIndex = route.stations.indexOf(this.destinationName);
 33          if (destinationIndex === -1) return;
 34
 35          const segments = route.path.slice(
 36              Math.min(stationIndex, destinationIndex),
 37              Math.max(stationIndex, destinationIndex)
 38          );
 39
 40          const distance = segments.reduce((acc, segment) => acc + segment.track.length, 0);
 41          const price = (distance * route.pricePerKm) / 1000; // Convert to km
 42
 43          this.gameStore.addMoney(Math.round(price));
 44      };
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55

```

Klasa Passenger

Jest to bardzo prosta klasa. Jej konstruktor przyjmuje jedynie nazwy stacji początkowej i końcowej (oraz referencję do *GameStore*). Jedyną metodą jest natomiast *pay*, która oblicza pokonany przez pasażera dystans i na jego podstawie dodaje fundusze (cena za kilometr jest pobierana z właściwości trasy).

Wsiadanie pasażerów

Gdy pociąg odjeżdża ze stacji, wywołuje metodę `processPassengers` na tej stacji. Wybiera ona pasażerów, którzy mają wsiąść do odjeżdżającego właśnie pociągu i wywołuje z nimi metodę `train.embarcPassengers`. Następnie aktualizuje ona tablicę pasażerów oczekujących na stacji usuwając tych który właśnie wsiedli.

Aktualnie przesiadki nie są obsługiwane, więc wsiadają jedynie pasażerowie z połączeniami bezpośrednimi.

Oto implementacje tych metod:

```
105  /**
106   * @param {import("./Train").default} train
107   * @returns {void}
108   */
109  processPassengers = train => {
110    const [newWaiting, passengersToEmbark] = this.waitingPassengers.reduce(
111      ([waiting, passengers], passenger) => {
112        const stations = train.route.stations;
113        if (this.direction === -1) stations.reverse();
114
115        const i = stations.indexOf(passenger.destinationName);
116        const currentIndex = stations.indexOf(train.route.currentStation);
117        if (i === -1 || i < currentIndex) {
118          waiting.push(passenger);
119        } else {
120          passengers.push(passenger);
121        }
122
123        return [waiting, passengers];
124      },
125      [[], []]
126    );
127
128    const remaining = train.embarcPassengers(passengersToEmbark);
129    this.waitingPassengers = [...newWaiting, ...remaining];
130  };
131
132
133  /**
134   * @param {import("./Passenger").default[]} passengers
135   * @returns {import("./Passenger").default[]} remaining passengers
136   */
137  embarkPassengers = (passengers) => {
138    const availableSeats = this.seats - this.passengers.length;
139    const fittingPassengers = passengers.slice(0, availableSeats);
140    const remainingPassengers = passengers.slice(availableSeats);
141    this.passengers.push(...fittingPassengers);
142
143    fittingPassengers.forEach(p => p.pay(this.route));
144
145    return remainingPassengers;
146  }
147
148
149  /**
150   * @param {import("./Station").default} station
151   * @returns {void}
152   */
153  disembarkPassengers = (station) => {
154    const [_, remaining] = this.passengers.reduce(
155      (acc, p) => {
156        if (p.destinationName === station.name) {
157          acc[0].push(p);
158        } else {
159          acc[1].push(p);
160        }
161
162        return acc;
163      },
164      [[], []]
165    );
166
167    this.passengers = remaining;
168
169    /* We assume these passengers are already at their destination
170     * station.addPassengers(disembarked);
171   }
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
623
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
153
```

Generowanie pasażerów

Station.js GameStore.js StationStore.js

src > store > StationStore.js > StationStore > constructor > stationsData.features.reduce() callback

```
13 class StationStore {
14
15     /**
16      * Returns stations that have a connected track or are close to a such station
17      * @returns {Station[]}
18     */
19     get activeStations() {
20         const stationsToGenerateAt = [...this.connectedStations.values()];
21         const closeToConnected = stationsToGenerateAt.flatMap(station => {
22             const loc = latLng(station.coordinates);
23             return this.stations.filter(other => {
24                 if (station.name === other.name || this.connectedStations.has(other.name)) return false;
25                 const distance = loc.distanceTo(latlng(other.coordinates));
26                 return distance <= 6_500;
27             });
28         });
29
30         return [...stationsToGenerateAt, ...closeToConnected];
31     }
32
33     /**
34      * Returns stations that have a connected track or are close to a such station
35      * @returns {Station[]}
36     */
37     get connectedStationsBySize() {
38         return this.stations
39             .filter(station => station.neighbors.size > 0)
40             .reduce((map, station) => {
41                 if (!map.has(station.size)) map.set(station.size, []);
42                 map.get(station.size).push(station);
43                 return map;
44             }, new Map());
45     }
46
47     /**
48      * Returns stations that have a connected track or are close to a such station
49      * @returns {Station[]}
50     */
51     get stationsWithConnectedTrack() {
52         return this.stations.filter(station => station.neighbors.size > 0);
53     }
54
55     /**
56      * Returns stations that have a connected track or are close to a such station
57      * @returns {Station[]}
58     */
59     get stationsWithCloseToConnected() {
60         const stationsToGenerateAt = [...this.connectedStations.values()];
61         const closeToConnected = stationsToGenerateAt.flatMap(station => {
62             const loc = latLng(station.coordinates);
63             return this.stations.filter(other => {
64                 if (station.name === other.name || this.connectedStations.has(other.name)) return false;
65                 const distance = loc.distanceTo(latlng(other.coordinates));
66                 return distance <= 6_500;
67             });
68         });
69
70         return [...stationsToGenerateAt, ...closeToConnected];
71     }
72 }
```

Generowanie pasażerów następuje w 2 etapach. Najpierw obliczana jest liczba pasażerów dla *wybranych* stacji biorąc pod uwagę ich wielkość. Dzieje się to w metodzie *generatePassengers* wewnątrz klasy *StationStore*. Stacje do generacji są nazywane przeze mnie jako „aktywne”.
Są to takie stacje, które:

Są to takie stacje, które:

- Mają podłączony co najmniej jeden tor, lub
 - Są w obrębie 6,5 km od takich „podłączonych” stacji

Obok implementacja tych get'ów, a poniżej metody generującej.

```
Station.js GameStore.js StationStore.js X
src > store > StationStores > StationStore > constructor > stationsData.features.reduce() callback
13 class StationStore {
95     generatePassengers = () => {
96         console.log("Passenger spawn");
97         const sizeMap = [0, 100, 500, 1000, 3000, 5000];
98
99         this.activeStations.forEach(station => {
100             if (station.waitingPassengers.length > sizeMap[station.size]) return true;
101             const maxNumber = sizeMap[station.size] / 100;
102             const number = Math.floor(Math.random() * (maxNumber + 1));
103             // console.log(`Spawning ${number} passengers at ${station.name}`);
104
105             for (let i = 0; i < number; i++) {
106                 station.addPassenger();
107             }
108         });
109     };
}
```

W kolejnym etapie wywoływana jest metoda `addPassenger` na instancji stacji. Ta metoda decyduje o miejscu docelowym generowanych podróżnych. Większa jest szansa, że będzie to większa stacja, ale stacje te są wybierane wyłącznie spośród stacji mających połączenie torowe (nie jest przy tym sprawdzanie czy to połączenie jest bezpośrednie). Z poziomu tej metody tworzone są rzeczywiste instancje klasy `Passenger`, które dodawane są do tablicy oczekujących pasażerów. Tak to wygląda:

```
Station.js X GameStore.js StationStore.js
src > store > models > Station.js > Station
10  class Station {
11      addPassenger = () => {
12          function getDestinationSize() {
13              const weight = [0, 10, 15, 25, 30, 20];
14              const random = Math.random() * 100;
15              let sum = 0;
16              let size = 1;
17
18              for (let i = 1; i <= 5; i++) {
19                  sum += weight[i];
20                  if (random <= sum) {
21                      size = i;
22                      break;
23                  }
24              }
25
26              return size;
27          }
28
29
30          let destinationSize = getDestinationSize();
31          if (!this.gameStore.stationStore.connectedStationsBySize.has(destinationSize)) {
32              destinationSize = 1;
33              while (destinationSize < 6 && !this.gameStore.stationStore.connectedStationsBySize.has(destinationSize)) {
34                  destinationSize++;
35              }
36
37              if (destinationSize === 6) destinationSize = 1;
38          }
39
40          const availableDestinations = this.gameStore.stationStore.connectedStationsBySize
41              .get(destinationSize)
42              .filter(station => station.name !== this.name);
43          if (availableDestinations.length === 0) return;
44
45          const destinationIndex = Math.floor(Math.random() * availableDestinations.length);
46          const destination = availableDestinations[destinationIndex];
47          const passenger = new Passenger(this.name, destination.name, this.gameStore);
48          this.waitingPassengers.push(passenger);
49
50      };
51  
```

Kiedy są generowani pasażerowie?
Obecnie metoda *generatePassengers*
jest wywoływana w 3 przypadkach:

- Dowolny pociąg właśnie zatrzymał się na dowolnej stacji
 - Zbudowano nowy tor
 - Zawsze co 3 minuty – implementacja na końcu konstruktora klasy *GameStore*

RouteStore

Ten store zajmuje się trasami w 3 zakresach:

- Przechowywanie tras w tablicy *routes*, dodawanie oraz usuwanie ich.
- Przechowywanie w atrybucie *currentRoute* referencji do aktualnie tworzonej trasy oraz 4 metody do manipulacji nią.
- Przechowywanie referencji do aktualnie podświetlonej/wyróżnionej trasy w atrybucie *highlightedRoute*.

Konstruktor i podstawowe metody

```
RouteStore.js
src > store > RouteStore.js > RouteStore
5  * @typedef {Object} RouteStoreSerialized
6  * @property {import("./models/Route").RouteSerialized[]} routes
7  */
8
9 You 4 days ago | 1 author (You)
10 class RouteStore {
11   gameStore;
12
13   /**
14    * @type {Route[]}
15   */
16   routes = [];
17
18   /**
19    * @type {Route|null}
20   */
21   currentRoute = null;
22
23   /**
24    * @type {Route|null}
25   */
26   highlightedRoute = null;
27
28   /**
29    * @param {import("./GameStore").default} gameStore
30    * @param {RouteSerialized[]} [routes]
31    */
32   constructor(gameStore, routes = []) {
33     makeAutoObservable(this, { gameStore: false, initCurrentRoute: action });
34
35     this.gameStore = gameStore;
36
37     routes.forEach(r => {
38       if (!(r instanceof Route)) throw new Error("Invalid route data");
39     });
39     this.routes = routes;
40
41     addRoute = route => {
42       this.tracks.push(route);
43     };
44
45     deleteRoute = routeId => {
46       const routeToDeleteIndex = this.routes.findIndex(r => r.id === routeId);
47       if (routeToDeleteIndex === -1) return;
48
49       this.routes[routeToDeleteIndex].cleanup();
50       this.routes.splice(routeToDeleteIndex, 1);
51     };
52   }
53 }
```

Kod widoczny obok jest trywialny i nie wymaga szczegółowego wyjaśnienia.

Metody obecnie tworzonej trasy

```
RouteStore.js
src > store > RouteStore.js > RouteStore
9  class RouteStore {
10
11   // Current Route related methods
12   // -----
13
14   /**
15    * Creates a new route
16    * @returns {void}
17    */
18   initCurrentRoute = () => {
19     this.currentRoute = new Route([], this.gameStore);
20   };
21
22   /**
23    * @param {Station} station
24    * @returns {string|null} error
25    */
26   addToCurrentRoute = station => {
27     return this.currentRoute.addStation(station.name, this.gameStore.stationStore.stationsMap);
28   };
29
30   /**
31    * @param {Station} station
32    * @returns {void}
33    */
34   removeFromCurrentRoute = station => {
35     this.currentRoute.removeStation(station.name);
36   };
37
38   /**
39    * Accepts current route and adds it to the routes list
40    * @returns {boolean} success
41    */
42   acceptCurrentRoute = () => {
43     if (this.currentRoute.stations.length < 2) return false;
44
45     this.currentRoute.draft = false;
46     this.routes.push(this.currentRoute);
47     this.currentRoute = null;
48     return true;
49   };
50
51   discardCurrentRoute = () => {
52     this.currentRoute.cleanup();
53     this.currentRoute = null;
54   };
55 }
```

Metoda *initCurrentRoute* tworzy nową trasę.

Metody *addToCurrentRoute* oraz *removeFromCurrentRoute* po prostu wywołują odpowiednie metody na obecnej trasie.

Najciekawszą z tych metod jest *acceptCurrentRoute*, która sprawdza poprawność trasy i dodaje ją do wszystkich pozostałych.

Jeśli użytkownik zechce natomiast odrzucić trasę to wywoływana jest metoda *discardCurrentRoute*.

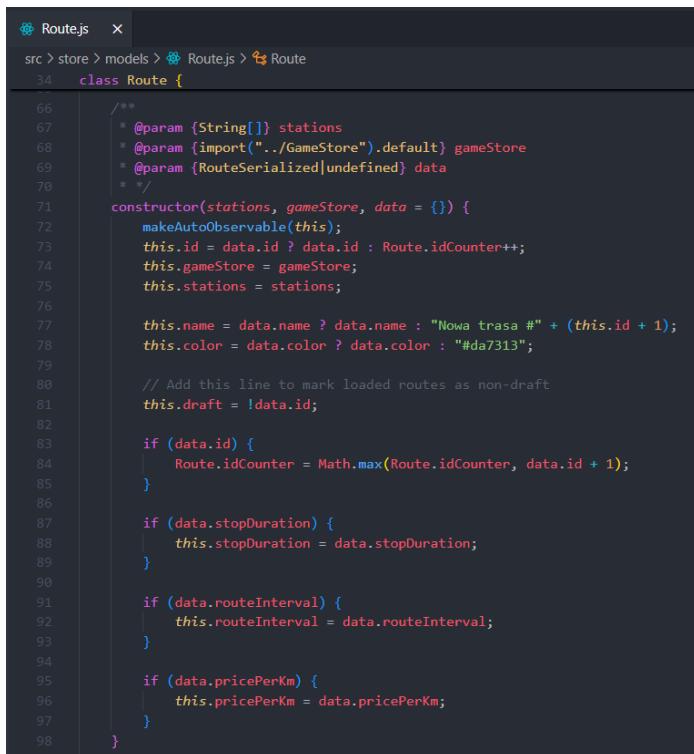
Route – klasa reprezentująca trasę

Jest to jedna z najbardziej rozbudowanych klas w tym projekcie. Omówienie zacznę od najbardziej logicznych atrybutów:

- *name* – Nazwa trasy nadana przez użytkownika.
- *color* – Użytkownik będzie miał możliwość wybrania koloru dla każdej trasy. Pozwoli to na prostsze rozróżnienie tras. Kolor przechowywany jest jako *string* w formie szesnastkowej, razem z początkowym hashtagiem.
- *stations* – Tablica przechowywająca nazwy przystanków na trasie (jedynie nazwy stacji, na których pociąg się zatrzymuje).
- *stopDuration* – Czas postoju na stacjach pośrednich w sekundach.
- *routeInterval* – Czas postoju na stacjach końcowych w sekundach. Nie dotyczy na samym początku zaraz po przypisaniu pociągu do trasy.
- *pricePerKm* – Cena za kilometr trasy. To właśnie tego atrybutu używa [metoda *pay* w klasie *Passenger*](#).
- *draft* – Wskazuje czy trasa jest zapisana ([wewnątrz tablicy routes klasy RouteStore](#)). Domyślnie wartość to *false* (chyba, że wczytujemy z zapisu).

Większość z pól z powyższej listy posiada proste setery do ich modyfikacji. Bardziej skomplikowane są metody modyfikujące tablicę ze stacjami – o nich za chwilę.

Konstruktor



```
Route.js  X
src > store > models > Route.js > Route
34 class Route {
  ...
  /**
   * @param {String[]} stations
   * @param {import("./GameStore").default} gameStore
   * @param {RouteSerialized|undefined} data
   */
  constructor(stations, gameStore, data = {}) {
    makeAutoObservable(this);
    this.id = data.id ? data.id : Route.idCounter++;
    this.gameStore = gameStore;
    this.stations = stations;

    this.name = data.name ? data.name : "Nowa trasa #" + (this.id + 1);
    this.color = data.color ? data.color : "#da7313";

    // Add this line to mark loaded routes as non-draft
    this.draft = !data.id;

    if (data.id) {
      Route.idCounter = Math.max(Route.idCounter, data.id + 1);
    }

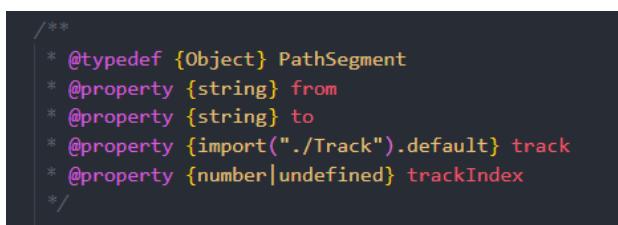
    if (data.stopDuration) {
      this.stopDuration = data.stopDuration;
    }

    if (data.routeInterval) {
      this.routeInterval = data.routeInterval;
    }

    if (data.pricePerKm) {
      this.pricePerKm = data.pricePerKm;
    }
  }
}
```

Konstruktor tej klasy jest nieco długi – wpływ na to ma logika wczytująca z zapisu, a jest dużo parametrów. Obok widoczna jest dokładna implementacja.

Ścieżka



```
/*
 * @typedef {Object} PathSegment
 * @property {string} from
 * @property {string} to
 * @property {import("./Track").default} track
 * @property {number|undefined} trackIndex
 */
```

Przechowywanie przystanków to nie wszystko. Może przecież istnieć wiele dróg łączących dane 2 stacje. Dlatego każda trasa przechowuje „ścieżkę”, czyli dokładny opis wszystkich punktów trasy. Służy do tego atrybut *path*. Przechowuje on tablicę zawierającą poszczególne fragmenty ścieżki. Obok widoczna jest definicja typu takiego fragmentu w formacie *JsDoc*.

Poszukiwanie ścieżki

Poszukiwanie ścieżki jest dość skomplikowane, więc kod ten znajduje się w osobnym pliku – *pathfinding.js*. Wykorzystałem algorytm „[przeszukiwania w szerz \(ang. breadth-first search, BFS\)](#)”. Działa on na grafie reprezentującym stacje, czyli *stationMap* (więcej o tym atrybutie przy [opisie StationStore](#)). Implementacja tego algorytmu opiera się na dwóch funkcjach: szukającej – *findPath*, oraz rekonstruującej – *reconstructPath*. Implementacja obu tych funkcji poniżej.

```
pathfinding.js
src > utils > pathfinding.js > reconstructPath > parentInfo
1  /**
2   * @typedef {Array<{ from: string, to: string, track: Track }>} Path */
3
4  /**
5   * Find path
6   * @param {Map<String, import("../store/models/Station").default} graph
7   * @param {string} startName
8   * @param {string} endName
9   * @returns {Path|null}
10 */
11
12 function findPath(graph, startName, endName) {
13     // Initialize BFS
14     const queue = [];
15     const visited = new Set();
16     const parent = new Map();
17
18     const startStation = graph.get(startName);
19     if (!startStation) return null; // Start station not found
20
21     queue.push(startStation);
22     visited.add(startName);
23
24     // BFS main loop
25     while (queue.length > 0) {
26         const currentStation = queue.shift();
27
28         // Found the destination
29         if (currentStation.name === endName) {
30             return reconstructPath(parent, startName, endName);
31         }
32
33         // Check all neighbors
34         for (const [neighborStationName, track] of currentStation.neighbors.entries()) {
35             if (visited.has(neighborStationName)) continue;
36             visited.add(neighborStationName);
37             parent.set(neighborStationName, {
38                 station: currentStation.name,
39                 track: track
40             });
41             queue.push(graph.get(neighborStationName));
42         }
43
44     }
45     return null; // No path found
46 }
47
48 /**
49  * Reconstruct path
50  * @param {Map<String, { station: string, track: Track }>} parent
51  * @param {string} startName
52  * @param {string} endName
53  * @returns {Path}
54 */
55
56 function reconstructPath(parent, startName, endName) {
57     const path = [];
58     let currentName = endName;
59
60     while (currentName !== startName) {
61         const parentInfo = parent.get(currentName);
62         if (!parentInfo) return null; // Should never happen if path was found
63
64         path.unshift({
65             from: parentInfo.station,
66             to: currentName,
67             track: parentInfo.track
68         });
69
70         currentName = parentInfo.station;
71     }
72
73     return path;
74 }
75
76 export default findPath;
```

Aktualizacja ścieżki

Aktualizacja ścieżki musi zawsze odbywać się poprzez metodę *updatePath*. Zapewnia ona integrację z torami – przypisuje trasę do odpowiedniego „pasa” toru (i ten index jest zapisywany w ścieżce jako atrybut *trackIndex*).

```
Route.js
src > store > models > Route.js > Route > toJSON > path > path.map() callback
34 class Route {
35
36     /**
37      * @param {Path} newPath
38      * @returns {string|undefined} error message
39     */
40
41     updatePath(newPath) {
42         let error;
43
44         // Add route to new tracks
45         let i = 0;
46         const tracksWithAddedRoute = [];
47         for (const segment of newPath) {
48             const isNew = !this.path.some(oldSegment => oldSegment.track.id === segment.track.id);
49             if (isNew) {
50                 [newPath[i].trackIndex, error] = segment.track.addRoute(this);
51                 if (!error) tracksWithAddedRoute.push(segment.track);
52             }
53
54             if (error) break;
55             i++;
56         }
57
58         if (error) {
59             // If there was an error, revoke changes
60             for (const track of tracksWithAddedRoute) {
61                 track.removeRoute(this.id);
62             }
63
64             return error;
65         }
66
67         // Find tracks that are no longer used
68         this.path.forEach(segment => {
69             const stillExists = newPath.some(newSegment => newSegment.track.id === segment.track.id);
70             if (!stillExists) segment.track.removeRoute(this.id);
71         });
72
73         this.path = newPath;
74     }
75 }
```

Metody modyfikujące przystanki

Dodawanie przystanku

Dodawanie przystanku jest niezwykle skomplikowaną operacją. Istnieją różne scenariusze dodania stacji:

- dodanie stacji na koniec trasy (lub ustanowienie stacji początkowej);
- dodanie jako przystanku stacji, przez którą pociąg już przejeżdża (już uwzględniona w ścieżce);
- dodanie stacji na początek trasy.

Dodatkowo, konieczna jest obsługa błędów, jeśli nie uda się znaleźć ścieżki lub pociąg już przejeżdża tą ścieżką (gra nie obsługuje takiej możliwości). Te wszystkie scenariusze zostały zaimplementowane wewnątrz metody `addStation`. Jej kod jest na poniższych zrzutach:

The screenshot displays two code editors side-by-side, both titled "Route.js M".

Left Editor (Implementation):

```
src > store > models > Route.js > Route > addStation
34 class Route {
174     /**
175      * @param {string} stationName
176      * @param {Object} graph
177      * @returns {string|null} error
178     */
179     addStation(stationName, graph = null) {
180         if (this.stations.includes(stationName)) return "Trasa już zawiera tę stację";
181         if (this.stations.length === 0) {
182             this.stations.push(stationName);
183             return;
184         }
185         if (this.stations.length === 1) {
186             const path = findPath(graph, this.stations.at(-1), stationName);
187             if (path == null) return "Nie można odnaleźć ścieżki";
188             const error = this.updatePath(path);
189             if (error) return error;
190             this.stations.push(stationName);
191             return;
192         }
193         const segmentIndex = this.path.findIndex(segment => segment.to === stationName);
194         if (segmentIndex === -1) {
195             const nextExistingStation = this.path.slice(segmentIndex).find(segment => this.stations.includes(segment.to));
196             if (!nextExistingStation) {
197                 console.error("Route.addStation: nextExistingStation is null! This should've never happened.");
198                 this.stations.push(stationName);
199                 return;
200             }
201             const insertIndex = this.stations.indexOf(nextExistingStation);
202             this.stations.splice(insertIndex, 0, stationName);
203         }
204     }
205 }
```

Right Editor (Test Implementation):

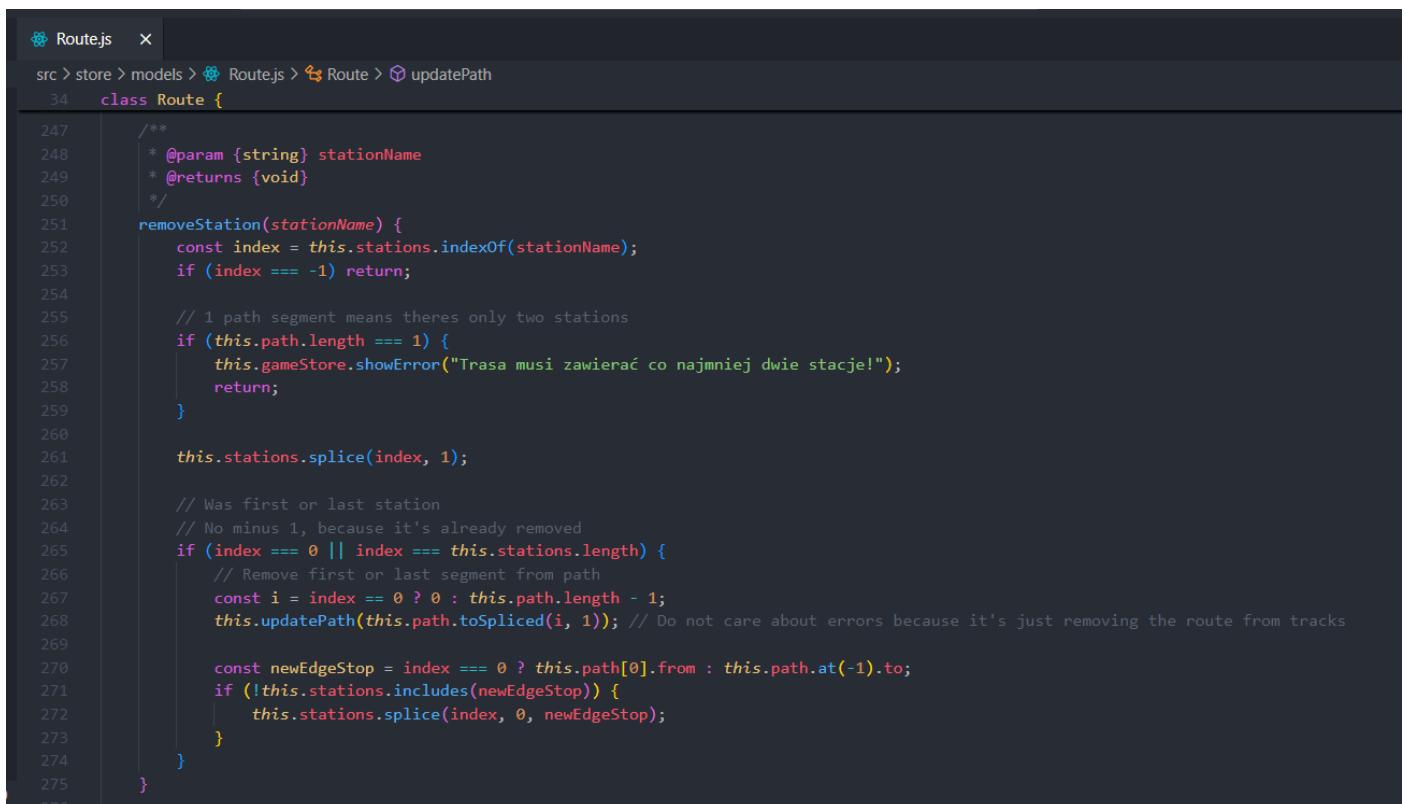
```
src > store > models > Route.js > Route > addStation
34 class Route {
179     addStation(stationName, graph = null) {
180         const segmentIndex = this.path.findIndex(segment => segment.to === stationName);
181         if (segmentIndex === -1) {
182             // Add new station at the end of the route
183             const newPathSegments = findPath(graph, this.stations.at(-1), stationName);
184             if (newPathSegments == null) return "Nie można odnaleźć ścieżki";
185
186             const overlaps = newPathSegments.some(newSegment =>
187                 this.path.some(existingSegment => existingSegment.track.id === newSegment.track.id)
188             );
189
190             if (overlaps) {
191                 // Check if all current tracks are included in the new path
192                 const allTracksIncluded = this.path.every(existingSegment =>
193                     newPathSegments.some(newSegment => newSegment.track.id === existingSegment.track.id)
194                 );
195
196                 if (!allTracksIncluded) return "Trasa już zawiera ten odcinek";
197
198                 const reverseNewPathSegments = newPathSegments
199                     .map(segment => ({ from: segment.to, to: segment.from, track: segment.track }))
200                     .reverse();
201
202                 // Prepend the new path and station
203                 const error = this.updatePath([
204                     ...reverseNewPathSegments.slice(0, reverseNewPathSegments.length - this.path.length),
205                     ...this.path
206                 ]);
207                 if (error) return error;
208                 this.stations.unshift(stationName);
209                 return;
210             }
211
212             const error = this.updatePath([...this.path, ...newPathSegments]);
213             if (error) return error;
214
215             this.stations.push(stationName);
216             return;
217         }
218     }
219 }
```

Usuwanie przystanku

Usuwanie przystanku jest znacznie prostsze. Istnieją 2 scenariusze:

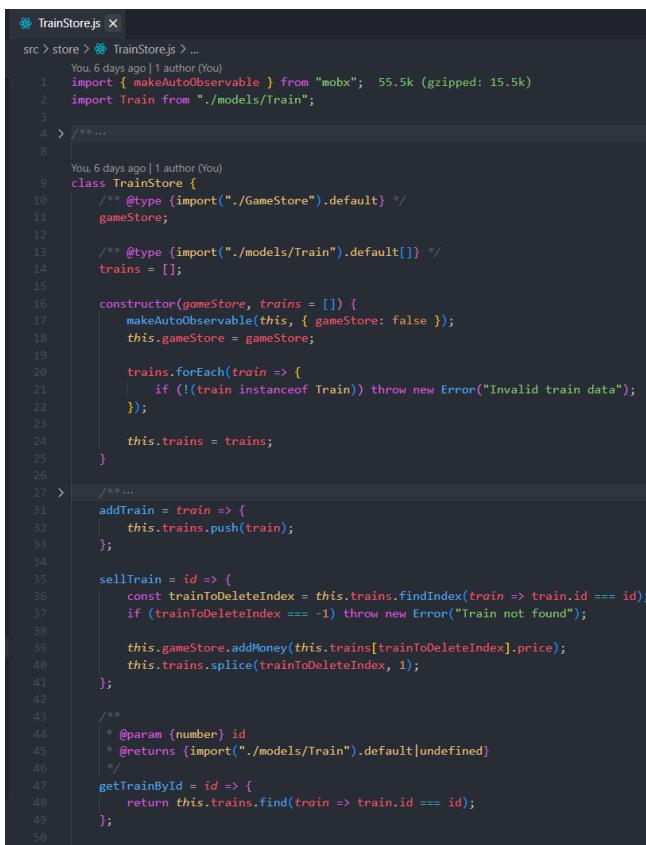
- Usuwanie stacji początkowej lub końcowej;
- Usuwanie przystanku pośredniego – pociąg i tak będzie tamtędy przejeżdżał.

Jeśli mamy doczynienia z pierwszym z tych scenariuszy to trzeba zaktualizować ścieżkę oraz upewnić się że nowa stacja końcowa/początkowa jest dodana jako przystanek. Wspólnym kodem jest sprawdzenie warunku poprawności trasy (co najmniej 1 segment ścieżki) oraz usunięcie stacji za pomocą metody *splice*.



```
Route.js
src > store > models > Route.js > Route > updatePath
34 class Route {
247     /**
248      * @param {string} stationName
249      * @returns {void}
250      */
251     removeStation(stationName) {
252         const index = this.stations.indexOf(stationName);
253         if (index === -1) return;
254
255         // 1 path segment means there's only two stations
256         if (this.path.length === 1) {
257             this.gameStore.showError("Trasa musi zawierać co najmniej dwie stacje!");
258             return;
259         }
260
261         this.stations.splice(index, 1);
262
263         // Was first or last station
264         // No minus 1, because it's already removed
265         if (index === 0 || index === this.stations.length) {
266             // Remove first or last segment from path
267             const i = index === 0 ? 0 : this.path.length - 1;
268             this.updatePath(this.path.toSpliced(i, 1)); // Do not care about errors because it's just removing the route from tracks
269
270             const newEdgeStop = index === 0 ? this.path[0].from : this.path.at(-1).to;
271             if (!this.stations.includes(newEdgeStop)) {
272                 this.stations.splice(index, 0, newEdgeStop);
273             }
274         }
275     }
276 }
```

TrainStore



```
TrainStore.js
src > store > TrainStore.js > ...
You, 6 days ago | 1 author (You)
1 import { makeAutoObservable } from "mobx"; 55.5k (gzipped: 15.5k)
2 import Train from "./models/Train";
3
4 /**
5 * You, 6 days ago | 1 author (You)
6 */
7 class TrainStore {
8     /**
9      * @type {import("./GameStore").default}
10     */
11     gameStore;
12
13     /**
14      * @type {import("./models/Train").default[]}
15     */
16     trains = [];
17
18     constructor(gameStore, trains = []) {
19         makeAutoObservable(this, { gameStore: false });
20         this.gameStore = gameStore;
21
22         trains.forEach(train => {
23             if (!(train instanceof Train)) throw new Error("Invalid train data");
24         });
25
26         this.trains = trains;
27     }
28
29     /**
30      * @param {Train} train
31      */
32     addTrain = train => {
33         this.trains.push(train);
34     };
35
36     /**
37      * @param {number} id
38      */
39     sellTrain = id => {
40         const trainToDeleteIndex = this.trains.findIndex(train => train.id === id);
41         if (trainToDeleteIndex === -1) throw new Error("Train not found");
42
43         this.gameStore.addMoney(this.trains[trainToDeleteIndex].price);
44         this.trains.splice(trainToDeleteIndex, 1);
45     };
46
47     /**
48      * @param {number} id
49      */
50     getTrainById = id => {
51         return this.trains.find(train => train.id === id);
52     };
53 }
```

Klasa zarządzająca pociągami. Przechowuje je w tablicy *trains*, pozwala na ich dodawanie (*addTrain*) oraz sprzedawanie (*sellTrain*). Dodatkowa metoda *getTrainById* pozwala na pobranie pociągu po ID, który jest jedynym atrybutem unikalnie identyfikującym każdy pociąg.

Pociągi

W grze istnieją 2 rodzaje pociągów:

- Elektryczne zespoły trakcyjne – pociągi złożone z określonej liczby członów, zintegrowane z 2 członami napędowymi.
- Składy wagonowe – osobny zakup lokomotyw oraz wagonów. Pozwala to na większą elastyczność. *Z uwagi na ograniczony czas realizacji projektu, zabrakło czasu na ich pełną implementację. Nie została zaimplementowana możliwość kupowania i doczepiania wagonów.*

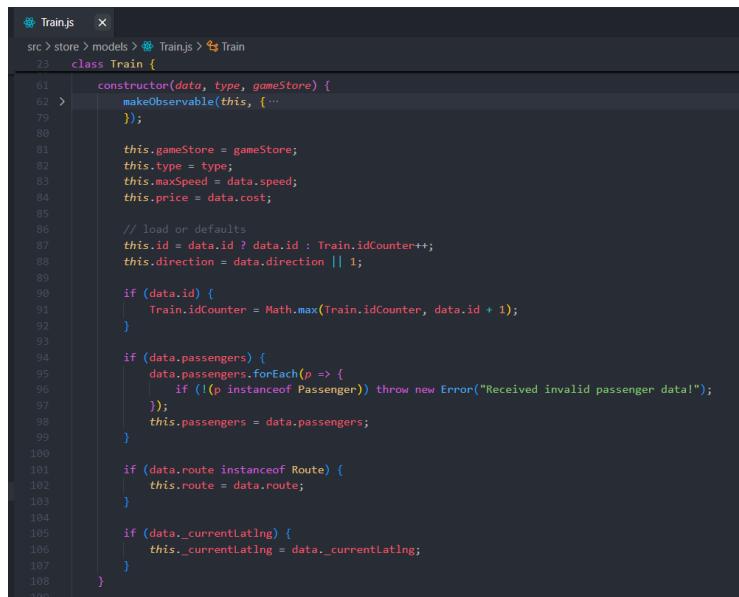
By uniknąć powtarzania części implementacji w klasie EKT oraz pociągów klasycznych, zdecydowałem się użyć dziedziczenia. Główną klasą jest klasa *Train*. Po niej dziedziczą *UnitTrain* oraz *CarriageTrain*.

Klasa podstawowa Train

Klasa ta posiada wiele ważnych pól:

- *Type* – pole do identyfikacji która subklaśa jest użyta;
- *maxSpeed* – maksymalna prędkość (liczba w km/h);
- *price* – cena, za którą zakupiono pociąg;
- *route* – referencja do trasy, do której jest aktualnie przypisany pociąg;
- *direction* – kierunek pociągu na trasie;
- *passengers* – tablica z referencjami do pasażerów, którzy są aktualnie w pojeździe;
- *currentStop* – jeśli pociąg aktualnie stoi na stacji, pole to przechowuje obiekt z danymi tego przyjazdu (nazwa stacji, godzina przyjazdu oraz czy jest końcowa);
- *_currentLatLang* – obecna pozycja pociągu, która jest używana wyłącznie w celu zapisywania i wczytywania stanu gry.

Konstruktor

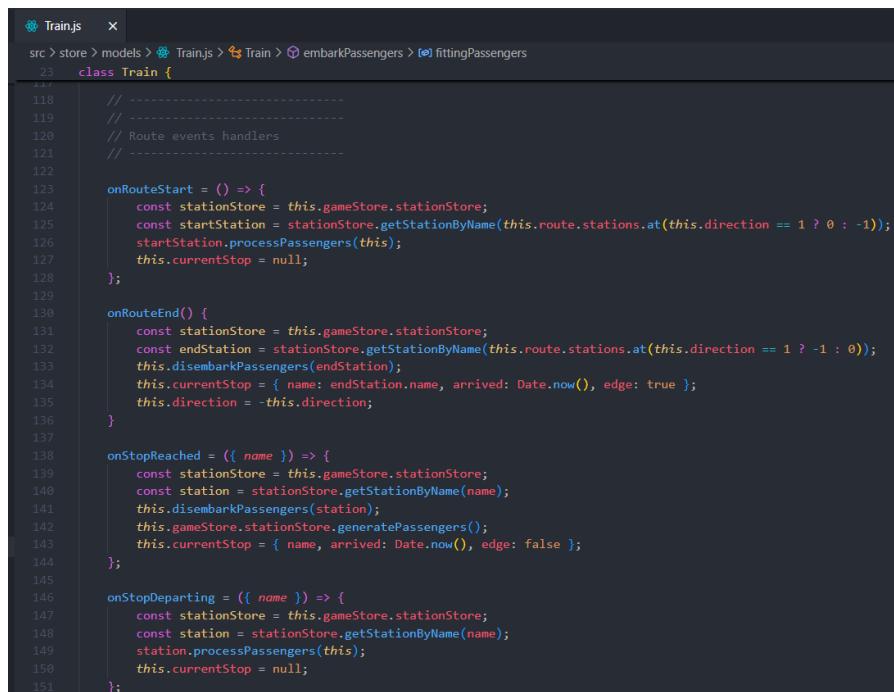


```
Train.js
src > store > models > Train.js > Train
23  class Train {
61    constructor(data, type, gameStore) {
62      makeObservable(this, [...]);
63      this.gameStore = gameStore;
64      this.type = type;
65      this.maxSpeed = data.speed;
66      this.price = data.cost;
67
68      // load or defaults
69      this.id = data.id ? data.id : Train.idCounter++;
70      this.direction = data.direction || 1;
71
72      if (data.id) {
73        Train.idCounter = Math.max(Train.idCounter, data.id + 1);
74      }
75
76      if (data.passengers) {
77        data.passengers.forEach(p => {
78          if (!(p instanceof Passenger)) throw new Error("Received invalid passenger data!");
79        });
80        this.passengers = data.passengers;
81      }
82
83      if (data.route instanceof Route) {
84        this.route = data.route;
85      }
86
87      if (data._currentLatLang) {
88        this._currentLatLang = data._currentLatLang;
89      }
90    }
91  }
```

Z uwagi, że po tej klasie będą dziedziczyć inne klasy należy użyć *makeObservable* zamiast *makeAutoObservable*. Niestety wydłuża to kod konstruktora (wewnętrz wywołania tej funkcji przekazuję obiekt definiujący rodzaj każdego pola i metody). Poza tym konstruktor, widoczny obok, jest bardzo podobny do innych klas.

Metody

Metody dotyczące pasażerów (*embarkPassengers*, *disembarkPassengers*) opisałem wcześniej, więc tu podam jedynie szczegóły metod dotyczących „zdarzeń”. Są to 4 metody wywoływane w momencie gdy pociąg osiągnie pewien stan:



```
Train.js
src > store > models > Train.js > Train > embarkPassengers > fittingPassengers
23  class Train {
118  // -----
119  // -----
120  // Route events handlers
121  // -----
122
123  onRouteStart = () => {
124    const stationStore = this.gameStore.stationStore;
125    const startStation = stationStore.getStationByName(this.route.stations.at(this.direction == 1 ? 0 : -1));
126    startStation.processPassengers(this);
127    this.currentStop = null;
128  };
129
130  onRouteEnd() {
131    const stationStore = this.gameStore.stationStore;
132    const endStation = stationStore.getStationByName(this.route.stations.at(this.direction == 1 ? -1 : 0));
133    this.disembarkPassengers(endStation);
134    this.currentStop = { name: endStation.name, arrived: Date.now(), edge: true };
135    this.direction = -this.direction;
136  }
137
138  onStopReached = ({ name }) => {
139    const stationStore = this.gameStore.stationStore;
140    const station = stationStore.getStationByName(name);
141    this.disembarkPassengers(station);
142    this.gameStore.stationStore.generatePassengers();
143    this.currentStop = { name, arrived: Date.now(), edge: false };
144  };
145
146  onStopDeparting = ({ name }) => {
147    const stationStore = this.gameStore.stationStore;
148    const station = stationStore.getStationByName(name);
149    station.processPassengers(this);
150    this.currentStop = null;
151  };
}
```

- *onRouteStart* – Metoda wywoływana gdy pociąg rozpoczyna trasę. Pobiera stację startową i „wsiada” pasażerów.
- *onRouteEnd* – Metoda wywoływana po osiągnięciu stacji docelowej. „Wysiada” pasażerów.
- *onStopReached* – Metoda wywoływana zaraz po zatrzymaniu się pociągu na przystanku. Również „wysiada” pasażerów.
- *onStopDeparting* – Metoda wywoływana gdy pociąg odjeżdża z przystanku. „Wsiada” pasażerów.

Implementacja wszystkich 4 metod widoczna obok.

Klasa UnitTrain

```
305 class UnitTrain extends Train {  
306     /** @type {number} */  
307     segments;  
308  
309     /** @type {number} */  
310     seats;  
311  
312     constructor(data, gameStore) {  
313         super(data, "unit", gameStore);  
314  
315         makeObservable(this, {  
316             segments: observable,  
317             seats: observable  
318         });  
319  
320         this.segments = data.segments;  
321         this.seats = data.seats;  
322     }  
323  
324     /**  
325      * @returns {TrainSerialized}  
326     */  
327     toJSON() {...  
328 }  
329 }
```

Klasa CarriageTrain

```
255 class CarriageTrain extends Train {  
256     /** @type {number} */  
257     strength;  
258  
259     /** @type {number} */  
260     maxCarriages;  
261  
262     /** @type {Carriage[]} */  
263     carriages = [...];  
264  
265     get speed() {  
266         return Math.min(this.maxSpeed, ...this.carriages.map(c => c.speed));  
267     }  
268  
269     get seats() {  
270         return this.carriages.reduce((acc, c) => acc + c.seats, 0);  
271     }  
272  
273     constructor(data, gameStore) {  
274         super(data, "carriage", gameStore);  
275  
276         makeObservable(this, {  
277             strength: observable,  
278             maxCarriages: observable,  
279             carriages: observable,  
280             seats: computed  
281         });  
282  
283         this.strength = data.strength;  
284         this.maxCarriages = data.maxCarriages;  
285     }  
286  
287     /**  
288      * @returns {TrainSerialized}  
289     */  
290     toJSON() {...  
291 }  
292 }
```

Klasa Carriage

Klasa ta posiada jedynie zdefiniowane atrybuty i nie jest jeszcze użyta w żadnym miejscu gry (zostanie to zmienione w przyszłości). Wygląda to tak:

```
236 // eslint-disable-next-line no-unused-vars  
237 You, 2 weeks ago | author (You)  
238 class Carriage {  
239     /** @type {number} */  
240     seats;  
241  
242     /** @type {number} */  
243     speed;  
244  
245     /** @type {number} */  
246     price;  
247  
248     constructor(data) {  
249         makeAutoObservable(this);  
250         this.speed = data.speed;  
251         this.seats = data.seats;  
252         this.price = data.cost;  
253     }  
254 }
```

Elektryczny zespół trakcyjny dziedziczy po zwykłym pociągu. Posiada jedynie 2 dodatkowe atrybuty:

- *segments* – liczba członów pociągu,
- *seats* – łączna liczba siedzeń we wszystkich członach.

Tak jak zostało wspomniane przed chwilą, nie zostało to skończone, ale zarys jest gotowy. Oto różnice od pozostałych klas „pociągowych”:

- *seats* jest getterem, zamiast polem jak w *UnitTrain*. Jest to spowodowane możliwością podłączenia różnego typu wagonów. Ten getter oblicza sumę używając metody *reduce* na tablicy *carriages* (przechowywającej wagony)
- *speed* jako getter. Plan rozwoju gry zakłada, że wagony mogą mieć mniejszą prędkość maksymalną od lokomotyw. W takim przypadku maksymalna prędkość pociągu byłaby równa właśnie maksymalnej prędkości najwolniejszego wagonu.

Zapisywanie i wczytywanie gry

Istnieją 2 rodzaje zapisu różniące się miejscem przechowywania danych: w *localStorage* przeglądarki lub na komputerze użytkownika. Wszystkie dane zamieniane są do formatu json. W tym celu każda klasa posiada 2 metody:

- *toJson* – zamienia instancję klasy do formatu json,
- *fromJson* – metoda statyczna tworząca instancję klasy na podstawie otrzymanych danych w formacie json.

Nie uwzględniałem tych metod w powyższym opisie klas, ponieważ są bardzo powtarzalne. Warto wspomnieć, że część wczytywania jest też dokonywana w konstruktorach – to dla tego przyjmuję one dodatkowe parametry opcjonalne.

Samym zapisywaniem oraz wczytywaniem zajmuje się specjalna (abstrakcyjna) klasa – *PersistenceService*. Z uwagi na dużą liczbę wygenerowanych pasażerów w dalszych momentach rozgrywki, otrzymany z serializacji json jest poddawany kompresji. Dzięki temu rozmiar pliku zapisu jest mniejszy około 10-cio krotnie. Ponadto utrudnia to modyfikację pliku zapisu.

Metody dotyczące localStorage

```
5  class PersistenceService {  
6      static saveToLocalStorage(gameStore) {  
7          const saveData = gameStore.toJson();  
8          const jsonString = JSON.stringify(saveData);  
9          const compressed = pako.deflate(jsonString);  
10         const base64 = btoa(String.fromCharCode.apply(null, compressed));  
11         localStorage.setItem(LOCAL_STORAGE_KEY, base64);  
12     }  
13  
14     static loadFromLocalStorage() {  
15         const saveData = localStorage.getItem(LOCAL_STORAGE_KEY);  
16         if (!saveData) return null;  
17  
18         try {  
19             // First try parsing as regular JSON (backwards compatibility)  
20             return JSON.parse(saveData);  
21         } catch {  
22             try {  
23                 // If that fails, try decompressing  
24                 const binary = atob(saveData);  
25                 const bytes = new Uint8Array(binary.length);  
26                 for (let i = 0; i < binary.length; i++) {  
27                     bytes[i] = binary.charCodeAt(i);  
28                 }  
29                 const decompressed = pako.inflate(bytes, { to: "string" });  
30                 return JSON.parse(decompressed);  
31             } catch (error) {  
32                 return null;  
33             }  
34         }  
35     }  
36 }
```

Nie ma możliwości zapisu danych binarnych w *localStorage*, więc skompresowane dane są zamieniane na ciąg znaków *base64*. Implementacja widoczna jest obok.

Metody dotyczące pobierania i wczytywania pliku zapisu

```
36  
37     static downloadSaveFile(gameStore) {  
38         const saveData = gameStore.toJson();  
39         const jsonString = JSON.stringify(saveData);  
40         const compressed = pako.deflate(jsonString);  
41         const blob = new Blob([compressed], { type: "application/x-compressed" });  
42         const url = URL.createObjectURL(blob);  
43         const a = document.createElement("a");  
44         const dateString = new Date()  
45             .toISOString()  
46             .replace(/\D\d\d\d\d/g, "")  
47             .slice(2, 14);  
48  
49         a.href = url;  
50         a.download = `ralsave-${dateString}.ralsave`;  
51         a.click();  
52         URL.revokeObjectURL(url);  
53     }  
54  
55 }
```

Zapisywanie pliku jest bardzo proste. Można zapisać dane binarne, więc użyte rozszerzenie nie ma znaczenia (zdecydowałem się na *.ralsave*). Nazwa pliku zawiera datę i godzinę zapisu (bez formatowania).

```
56     static async loadFromFile() {  
57         return new Promise((resolve, reject) => {  
58             // Create file input element  
59             const fileInput = document.createElement("input");  
60             fileInput.type = "file";  
61             fileInput.accept = ".json, ralsave";  
62  
63             // Handle file selection  
64             fileInput.onchange = event => { ... };  
65  
66             fileInput.onerror = () => {  
67                 reject(new Error("File selection cancelled"));  
68             };  
69  
70             fileInput.click();  
71         });  
72     }  
73 }
```

Przeciwnie jest jeśli chodzi o wczytywanie zapisu. Wymaga to stworzenia niewidocznego pola *input* o typie *file*, a następnie wywołanie kliknięcia na nie. Należało też obsłużyć scenariusz, w którym użytkownik nie wybiera pliku (klik „anuluj”). W efekcie kod się znacznie wydłużał. Wnętrze implementacji *onChange* jest widoczne na następnej stronie.

```

PersistenceService.js X
src > services > PersistenceService.js > PersistenceService > loadFromFile > <function>
  5   class PersistenceService {
  55     static async loadFromFile() {
  56       return new Promise((resolve, reject) => {
  62         // Handle file selection
  63         inputFile.onchange = event => {
  64           const file = event.target.files[0];
  65
  66           // Check if a file was selected
  67           if (!file) {
  68             reject(new Error("No file selected"));
  69             return;
  70           }
  71
  72           // Create FileReader instance
  73           const reader = new FileReader();
  74           const isCompressed = file.name.endsWith(".ralsave");
  75
  76           // Handle successful file read
  77           reader.onload = e => {
  78             try {
  79               // Try to parse the file contents as JSON
  80               const jsonData = JSON.parse(e.target.result);
  81               resolve(jsonData);
  82             } catch (error) {
  83               try {
  84                 // Try to decompress the file contents
  85                 const decompressed = pako.inflate(e.target.result, { to: "string" });
  86                 const jsonData = JSON.parse(decompressed);
  87                 resolve(jsonData);
  88               } catch (error) {
  89                 reject(new Error("Invalid save file: " + error.message));
  90               }
  91             }
  92           };
  93
  94           reader.onerror = () => {
  95             reject(new Error("Error reading file"));
  96           };
  97
  98           if (isCompressed) {
  99             reader.readAsArrayBuffer(file);
 100           } else {
 101             reader.readAsText(file);
 102           }
 103         };
 104       };
 105     }
 106   };
 107 }

```

Wywoływanie metod wczytujących

Metoda wczytania gry wybierana jest przez użytkownika z poziomu menu głównego i jest przekazywana przez *queryParameter*. Jest to następnie odczytywane przez komponent *GameStoreProvider*, a dokładniej przez przedstawione obok wywołanie hook'a *useEffect*.

Ten sam komponent wyświetla również informację o nie zapisaniu gry, gdy użytkownik spróbuje zamknąć kątę przeglądarki:

```

useEffect(() => {
  const handler = e => {
    e.preventDefault();
  };

  addEventListener("beforeunload", handler);
  return () => removeEventListener("beforeunload", handler);
}, []);

```

```

useEffect(() => {
  const loadType = useParams().get("load");

  if (!redirectedRef.current) {
    if (loadType === "new") {
      setStore(new GameStore());
    } else if (loadType === "browser") {
      // If no data in local storage, it behaves as "new"
      setStore(GameStore.loadFromLocalStorage());
    } else if (loadType === "file") {
      GameStore.loadSaveFile()
        .then(store => {
          setStore(store);
        })
        .catch(err => {
          console.error(err);
          navigate("/");
        });
    }
  }

  if (loadType !== "browser") {
    redirectedRef.current = true;
    const url = new URL(window.location.href);
    url.searchParams.set("load", "browser");
    window.history.replaceState(null, "", url.toString()); // So the browser back button skips that change
  }
}, [params, navigate]);

```

Wywoływanie metod zapisujących

Zapisywanie jest dokonywane przez użytkownika z poziomu okienka dialogowego. Więcej o nim tutaj.

Ui – interfejs użytkownika

Interfejs wyświetlany jest nad mapą. Składa się z paru „paneli” rozmieszczonych w różnych lokalizacjach. Do czego użyłem *position: absolute*; wewnątrz arkusza stylów. Aby całość była spójna, zdecydowałem się stworzyć parę komponentów, które wykorzystywałyby się w całości ui. Ich implementacje poniżej:

```
IconButton.jsx | ElevatedButton.jsx
src > Ui > common > IconButton > IconButton.jsx > ...
1 import PropTypes from "prop-types"; 1.4k (gzipped: 778)
2
3 import style from "./IconButton.component.scss";
4 import { Link } from "react-router"; 148.9k (gzipped: 47.2k)
5
6 function IconButton({ onClick, children, active = false, disabled = false, inverted = false, ...props }) {
7   const classList = ["icon-button"];
8   if (active) classList.push("active");
9   if (disabled) classList.push("disabled");
10  if (inverted) classList.push("inverted");
11  if (props.className) classList.push(...props.className.split(" "));
12
13  return (
14    <button onClick={onClick} disabled={disabled} data-style={style} {...props} className={classList.join(" ")>
15      {children}
16    </button>
17  );
18}
19
20 IconButton.propTypes = {
21   children: PropTypes.node.isRequired,
22   onClick: PropTypes.func,
23   active: PropTypes.bool,
24   disabled: PropTypes.bool,
25   inverted: PropTypes.bool,
26   className: PropTypes.string
27 };
28
29 function IconLinkButton({ to, children, active = false, disabled = false, inverted = false, ...props }) {
30   const classList = ["icon-button"];
31   if (active) classList.push("active");
32   if (disabled) classList.push("disabled");
33   if (inverted) classList.push("inverted");
34   if (props.className) classList.push(...props.className.split(" "));
35
36  return (
37    <Link to={to} data-style={style} {...props} className={classList.join(" ")>
38      {children}
39    </Link>
40  );
41}
42
43 IconLinkButton.propTypes = {
44   children: PropTypes.node.isRequired,
45   to: PropTypes.string.isRequired,
46   active: PropTypes.bool,
47   disabled: PropTypes.bool,
48   inverted: PropTypes.bool
49 };
50
```

```
IconButton.jsx | ElevatedButton.jsx
src > Ui > common > ElevatedButton > ElevatedButton.jsx > ...
1 import PropTypes from "prop-types"; 1.4k (gzipped: 778)
2 import style from "./ElevatedButton.component.scss";
3 import { Link } from "react-router"; 148.9k (gzipped: 47.2k)
4
5 function ElevatedButton({ children, onClick = () => {}, disabled = false }) {
6   return (
7     <button
8       className={"elevated-button ${disabled ? "disabled" : ""}"}
9       onClick={onClick}
10      data-style={style}
11      disabled={disabled}
12    >
13      {children}
14    </button>
15  );
16}
17
18 ElevatedButton.propTypes = {
19   children: PropTypes.node.isRequired,
20   onClick: PropTypes.func,
21   disabled: PropTypes.bool
22 };
23
24 function ElevatedLinkButton({ children, to, disabled = false }) {
25   return (
26     <Link to={to} className={"elevated-button ${disabled ? "disabled" : ""}" data-style={style}>
27       {children}
28     </Link>
29   );
30}
31
32 ElevatedLinkButton.propTypes = {
33   children: PropTypes.node.isRequired,
34   to: PropTypes.string.isRequired,
35   disabled: PropTypes.bool
36 };
37
38 export { ElevatedLinkButton, ElevatedButton };
39 export default ElevatedButton;
40 *
```

```

src > Ui > common > StatusBox > StatusBox.jsx ...
1 import PropTypes from "prop-types"; 1.4k (gzipped: 778)
2
3 import style from "./StatusBox.component.scss";
4
5 function StatusBox({ icon, value }) {
6   return (
7     <div className="status-box" data-style={style}>
8       <img src={icon} alt="" />
9       <span>{value}</span>
10    </div>
11  );
12}
13
14 StatusBox.propTypes = {
15   icon: PropTypes.string.isRequired,
16   value: PropTypes.number.isRequired
17 };
18
19 export default StatusBox;
20

```

```

src > Ui > common > StatusBox > StatusBox.component.scss ...
1 .status-box {
2   display: flex;
3   align-items: center;
4   background-color: var(--on-surface);
5   color: var(--on-surface);
6   border-radius: 10px;
7   box-shadow: -2px 2px 2px #093d5e77;
8
9   img {
10     height: 2.5rem;
11     margin: 0.3rem;
12     margin-right: 0;
13   }
14
15   span {
16     min-width: 5ch;
17     padding: 0.5rem;
18     margin: 0.4rem;
19     border-radius: calc(10px - 0.3rem);
20     background-color: var(--surface-lighter);
21     line-height: 1;
22     text-align: center;
23   }
24 }

```

Większość samego komponentu *Ui* stanowi kod *jsx*:

```

src > Ui > UI.jsx ...
28 const UI = observer(() => {
29   return (
30     <div className="game-ui" data-style={style}>
31       <div className="game-ui-top">
32         <SpeedChanger />
33         <StatusBox icon={moneyImg} value={money.toLocaleString("en-US").replace(/,/g, " ")}/>
34         <IconButton onClick={handleQuit}>
35           <i className="fa-solid fa-arrow-right-from-bracket"/>
36         </IconButton>
37       </div>
38       <div className="game-ui-right">
39         {routes.map(route => (
40           <IconLinkButton
41             to={`${pathname.includes(`/${route.id}`) ? "" : route.id}`}
42             key={route.id}
43             onClick={() => {}}
44             active={pathname.includes(`/${route.id}`)}
45           >
46             {route.icon}
47           </IconLinkButton>
48         ))
49       </div>
50       <div className="game-ui-left">
51         <div className={panel ${pathname === "" ? "collapsed" : ""}}>
52           <Suspense
53             fallback={
54               <div className="ball-grid-pulse">...
55             </div>
56           }
57           >
58             <Link to="/game">
59               <i className="fas fa-times"/>
60             </Link>
61             <Outlet />
62           </Suspense>
63         </div>
64       </div>
65       <div className="game-ui-bottom-right">
66         <TipRenderer />
67       </div>
68       <div className="game-ui-bottom-center">
69         <div className="error-banner">{error}</div>
70       </div>
71       <QuitDialog dialogRef={quitDialogRef} onClose={handleCloseDialog} />
72     </div>
73   );
74 }

```

W prawym górnym rogu znajdują się 3 elementy:

- kontrolka zmiany prędkości gry (*SpeedChanger*);
- kontrolka wyświetlająca liczbę dostępnych funduszy;
- przycisk wyjścia, który wyświetla okienko dialogowe, o którym dalej.

Główne menu znajduje się w panelu lewym. Jego zakładki mogą być przełączane za pomocą przycisków wyświetlanych w panelu prawym.

Prawy dolny róg został poświęcony na wyświetlanie podpowiedzi. Zarządza tym komponent *TipRenderer*.

Po środku dolnej krawędzi będą wyświetlane informacje o wszelkich błędach.

Na koniec renderowany jest *QuitDialog*, który pozwala wybrać tryb zapisu.

```

const UI = observer(() => {
  const { error, money } = useGameStore();
  const quitDialogRef = useRef(null);
  let { pathname } = useLocation();
  pathname = pathname.substring(5);

  const handleQuit = () => {
    quitDialogRef.current?.showModal();
  };

  const handleCloseDialog = () => {
    quitDialogRef.current?.close();
  };
});

```

Widoczna obok logika tego komponentu, dotyczy wyświetlania wspomnianego okienka dialogowego oraz błędów.

SpeedChanger

Ponieważ gra działa w czasie rzeczywistym, użytkownik ma możliwość jej przyspieszenia (x2 lub x3). Pozwala na to ten komponent.

The screenshot shows two code files in a code editor. On the left is `SpeedChanger.jsx`, which contains a functional component that uses `observer` from `mobx-react-lite` to track `gameSpeed` and `setGameSpeed`. It includes a button to change game speed between 1 and 3. On the right is `SpeedChanger.component.scss`, which defines a CSS class `.speed-changer` for the container, styling for the button's state transitions, and styles for the icons within the button.

```
src > Ui > widgets > SpeedChanger > SpeedChanger.jsx ...
1 import { observer } from "mobx-react-lite"; 5.6k (gzipped: 2.1k)
2
3 // hooks
4 import { useGameStore } from "../../store/GameStoreProvider";
5
6 import style from "./SpeedChanger.component.scss";
7
8 const SpeedChanger = observer(() => {
9   const { gameSpeed, setGameSpeed } = useGameStore();
10
11   function changeGameSpeed() {
12     setGameSpeed(gameSpeed === 3 ? 1 : gameSpeed + 1);
13   }
14
15   return (
16     <div data-style={style} className="speed-changer">
17       <i className="fas fa-gauge-simple"></i>
18       <button onClick={changeGameSpeed}>
19         <i className="fas fa-play active"></i>
20         <i className={ fas fa-play ${gameSpeed} >= 2 ? "active" : "" }></i>
21         <i className={ fas fa-play ${gameSpeed} >= 3 ? "active" : "" }></i>
22       </button>
23     </div>
24   );
25 });
26
27 export default SpeedChanger;
28

src > Ui > widgets > SpeedChanger > SpeedChanger.component.scss > .speed-changer
1 .speed-changer {
2   display: flex;
3   align-items: center;
4   background-color: var(--on-surface);
5   color: var(--on-surface);
6   border-radius: 10px;
7   font-size: 1.5rem;
8   box-shadow: -2px 2px 2px #093d5e77;
9
10  & > i {
11    margin-inline: 1rem 0;
12  }
13}
14
15 .speed-changer button {
16   all: unset;
17   pointer-events: all;
18   cursor: pointer;
19
20   height: 100%;
21   display: flex;
22   align-items: center;
23   gap: 0.2rem;
24   padding-inline: 1rem;
25
26   i {
27     color: var(--surface-lighter);
28     transition: color 200ms;
29   }
30
31   i.active {
32     color: var(--on-surface);
33   }
34
35   &:is(:hover, :focus-visible) i:not(.active) {
36     color: var(--surface-lightest);
37   }
38
39   &:is(:hover, :focus-visible):not(:has(i:not(.active))) i {
40     color: var(--on-surface-lightest);
41   }
42 }
43
```

TipRenderer

Prosty komponent sprawdzający ścieżkę url i wyświetlający odpowiedni komponent podpowiedzi.

The screenshot shows the `TipRenderer.jsx` file. It checks if the current URL matches a build track or route creation endpoint. If so, it returns a `TrackDrawTips` or `RouteDrawTips` component respectively, wrapped in a panel with a collapsed state.

```
function TipRenderer() {
  const buildingTrackTip = !!useMatch("/game/tracks/build/*");
  const creatingRouteTip = useMatch("/game/routes/create");

  return (
    <div className={`${panel ${buildingTrackTip || creatingRouteTip ? "" : "collapsed"}`}>
      {buildingTrackTip && <TrackDrawTips />}
      {creatingRouteTip && <RouteDrawTips />}
    </div>
  );
}
```

Dla spójnego wyglądu obu komponentów podpowiedzi, stworzony jest uniwersalny komponent `TipRow`:

The screenshot shows the `TipRow.jsx` file and its corresponding CSS `TipRow.component.scss`. The component takes an icon and a label as props and renders them inside a list item. The CSS defines styles for the list item and its child elements.

```
src > Ui > common > TipRow > TipRow.jsx ...
1 import PropTypes from "prop-types"; 1.4k (gzipped: 778)
2 import styles from "./TipRow.component.scss";
3
4 function TipRow({ icon, label }) {
5   return (
6     <li data-style={styles}>
7       {icon}
8       <p>{label}</p>
9     </li>
10  );
11}
12
13 TipRow.propTypes = {
14   icon: PropTypes.node,
15   label: PropTypes.string
16 };
17
18 export default TipRow;
19

src > Ui > common > TipRow > TipRow.component.scss > li
1 li {
2   display: flex;
3   font-size: 1.125rem;
4   font-weight: 500;
5
6   img {
7     height: 1lh;
8   }
9
10  p {
11    margin: 0;
12    margin-inline-start: 1rem;
13  }
14
15
```

TrackDrawTips – podpowiedzi dotyczące budowy torów

```
Uijsx TrackDrawTips.jsx ...
src > UI > tips > TrackDrawTips.jsx ...
1 import { observer } from "mobx-react-lite"; 5.6k (gzipped: 2.1k)
2
3 // components
4 import TipRow from "../common/TipRow/TipRow";
5
6 // hooks
7 import { useGameStore } from "../../store/GameStoreProvider";
8
9 // assets
10 import stationLeftClick from "../../assets/icons/station-left-click.svg";
11 import trackLeftClick from "../../assets/icons/track-left-click.svg";
12 import rightClick from "../../assets/icons/computer-mouse-right-click.svg";
13
14 const TrackDrawTips = observer(() => {
15   const { trackStore } = useGameStore();
16
17   return (
18     <>
19       <h3>Sterowanie</h3>
20       <ul>
21         <TipRow
22           icon={<img src={stationLeftClick} alt="lewy przycisk myszy" />}
23           label={`Kliknij stację, aby ${trackStore.buildingTrack ? "zakończyć" : "rozpocząć"} tor`}
24         />
25         {!trackStore.buildingTrack &&
26           <TipRow
27             icon={<img src={trackLeftClick} alt="lewry przycisk myszy" />}
28             label="Kliknij tor, aby ulepszyć"
29           />
30         }
31         {trackStore.buildingTrack &&
32           <TipRow
33             icon={<img src={rightClick} alt="prawy przycisk myszy" />} label="Anuluj"
34           />
35         }
36       </ul>
37     </>
38   );
39
40   export default TrackDrawTips;
41
```

Kod jest bardzo prosty. Używa pola *buildingTrack* z klasy *TrackStore* aby wyświetlić podpowiedzi zależnie od obecnego stanu.

RouteDrawTips – podpowiedzi dotyczące tworzenia trasy

```
Uijsx RouteDrawTips.jsx ...
src > UI > tips > RouteDrawTips.jsx ...
1 import TipRow from "../common/TipRow/TipRow";
2
3 // assets
4 import stationLeftClick from "../../assets/icons/station-left-click.svg";
5 import stationRightClick from "../../assets/icons/station-right-click.svg";
6
7 const RouteDrawTips = () => {
8   return (
9     <>
10       <h3>Sterowanie</h3>
11       <ul>
12         <TipRow
13           icon={<img src={stationLeftClick} alt="lewy przycisk myszy, obok pinezka stacji" />}
14           label={`Kliknij stację lewym przyciskiem, aby dodać przystanek`}
15         />
16         <TipRow
17           icon={<img src={stationRightClick} alt="prawy przycisk myszy, obok pinezka stacji" />}
18           label={`Kliknij stację prawym przyciskiem, aby usunąć przystanek`}
19         />
20       </ul>
21     </>
22   );
23
24   export default RouteDrawTips;
25
```

Tu jeszcze prościej – podpowiedzi w tym przypadku nie są zależne od stanu.

QuitDialog

Najlepszym sposobem na implementację wyskakującego okienka takiego jak to było wykorzystanie elementu html `<dialog>`. To okienko ma 2 zawartości: wybór sposobu zapisu i ostrzeżenie o nadpisaniu zapisu w przeglądarce. Całość jest zdefiniowana w jednym komponencie:

```
⌘ QuitDialog.jsx ✘
src > Ui > widgets > QDialog > ⌘ QDialog.jsx > ...
28 const QDialog = ({ dialogRef, onClose }) => {
  52   return (
    53     <Dialog heading={confirmation ? "Czy nadpisać?" : "Chcesz wyjść?"} dialogRef={dialogRef}>
    54       {confirmation && (
    55         <div className="quit-dialog-buttons">
    56           <button className="save-quit-btn green" onClick={handleSaveAndQuit}>
    57             Zapisz i wyjdź
    58           </button>
    59           <button className="green" onClick={handleSaveToFile}>
    60             <i className="fas fa-download"></i>
    61           </button>
    62           <button className="red" onClick={handleQuitWithoutSave}>
    63             Wyjdź bez zapisywania
    64           </button>
    65           <button className="orange" onClick={onClose}>
    66             Anuluj
    67           </button>
    68         </div>
    69       )
    70       {confirmation && (
    71         <div className="confirm-dialog-buttons">
    72           <button
    73             className="green"
    74             onClick={() => {
    75               handleSaveToFile();
    76               navigate("/");
    77             }}
    78           >
    79             Pobierz i wyjdź
    80           </button>
    81           <button
    82             className="red"
    83             onClick={() => {
    84               gameStore.saveLocalStorage();
    85               navigate("/");
    86             }}
    87           >
    88             Nadpisz i wyjdź
    89           </button>
    90           <button
    91             className="orange"
    92             onClick={() => {
    93               setTimeout(() => setConfirmation(false), 1000);
    94               onClose();
    95             }}
    96           >
    97             Anuluj
    98           </button>
    99         </div>
    100      )
    101    </Dialog>
  
```

Logika jest bardzo prosta – ogranicza się do wywołania odpowiednich funkcji zapisu oraz przekierowywania do strony głównej. Zajmuje się także pokazywaniem ostrzeżenia o nadpisaniu. Wyświetlaniem i chowaniem całego dialogu zajmuje się komponent *Ui*.

```
⌘ QDialog.jsx ✘
src > Ui > widgets > QDialog > ⌘ QDialog.jsx > ...
27
28 const QDialog = ({ dialogRef, onClose }) => {
  29   const navigate = useNavigate();
  30   const gameStore = useGameStore();
  31   const [confirmation, setConfirmation] = useState(false);
  32
  33   const handleSaveAndQuit = () => {
  34     const hasSave = PersistenceService.loadFromLocalStorage() != null;
  35     if (hasSave) {
  36       setConfirmation(true);
  37       return;
  38     }
  39
  40     gameStore.saveLocalStorage();
  41     navigate("/");
  42   };
  43
  44   const handleSaveToFile = () => {
  45     gameStore.downloadSave();
  46   };
  47
  48   const handleQuitWithoutSave = () => {
  49     navigate("/");
  50   };
  51
  52   return (
  
```

Mapa

Gra będzie się odbywała na terenie województwa śląskiego, dlatego głównym jej elementem jest mapa. Do jej stworzenia użyję 2 bibliotek: *leaflet* oraz *react-leaflet*. Do wyświetlenia mapy mógłbym użyć kafelków np. z open-street-map, lecz tak dokładne szczegóły są zbędne w tym projekcie. Dlatego użyję formatu *geojson* by narysować jedynie kontury (przy okazji mam większą kontrolę nad stylami). Dane konturów pobrałem z rządowego geoportalu i przechowuję w pliku *slaskie.json*.

```
⑧ Map.jsx •
src > Game > Map > ⑧ Map.jsx > ...
1 import { MapContainer, GeoJSON, useMap, Pane } from "react-leaflet"; 4k (gzipped: 1.6k)
2 import mapGeoJson from "../../assets/data/slaskie.json";
3 import StationsController from "./Controllers/StationsController";
4 import TracksController from "./Controllers/TracksController";
5 import TrainController from "./Controllers/TrainController";
6
7 const Map = () => {
8   return (
9     <MapContainer center={[51.7, 19]} minZoom={8} zoom={9} maxZoom={14} className="map" zoomControl={false}>
10    <MapController />
11    <StationsController />
12    <TracksController />
13    <TrainController />
14    <Pane name="popup" style={{ zIndex: 700 }} />
15  </MapContainer>
16);
17};
18
19 const MapController = () => {
20   const map = useMap();
21
22   function onBaseLayerAdded(e) {
23     const layerBounds = Object.values(e.target._layers)[0].getBounds();
24     map.fitBounds(layerBounds);
25   }
26
27   return (
28     <>
29       <GeoJSON
30         data={mapGeoJson}
31         eventHandlers={{ add: onBaseLayerAdded }}
32         style={({
33           stroke: false,
34           color: "#35A453",
35           fill: true,
36           fillColor: "#35A453",
37           fillOpacity: 1,
38           interactive: false
39         })}
40       />
41     </>
42   );
43 };
44
45 export default Map;
46
```

Wewnątrz komponentu *Map* deklaruję jedynie mapę, warstwę *popup* (tak by dymki mogły wyświetlać się nad innymi elementami) oraz komponenty trzech „kontrolerów”. Pozwoli to na odizolowanie kodu związanego z poszczególnymi częściami gry od siebie.

MapController odpowiada wyłącznie za renderowanie warstwy zawierającej granice województwa i za śródkowanie mapy po wyłączeniu gry.

Kod pozostałych kontrolerów omówię później gdyż jest mocno powiązany ze stanem aplikacji.

StationController – wyświetlanie stacji na mapie

```
StationsController.jsx X
src > Game > Map > Controllers > StationsController.jsx > [observer] StationMarker > observer() callback

19 const StationsController = observer(() => {
20   const { stationStore } = useGameStore();
21   const { snappedStation, setSnappedStation, setShowedPopup } = stationStore;
22
23   function onMouseOver({ latlng }, station) {
24     const distance = latlng.distanceTo(latlng(station.coordinates));
25     if (!snappedStation.station || distance < snappedStation.distance) {
26       setSnappedStation({ station: station, distance });
27     }
28   }
29
30   function onMouseOut(station) {
31     if (snappedStation.station === station) {
32       setSnappedStation({ station: null, distance: null });
33     }
34   }
35
36   useMapEvent("click", () => {
37     setShowedPopup(null);
38   });
39
40   return (
41     <>
42       <Pane name="station-markers">
43         {stationStore.stations.map(station => (
44           <StationMarker station={station} key={station.name} />
45         ))}
46       </Pane>
47       {stationStore.enableSnapping && (
48         <Pane name="station-snap-areas">
49           {stationStore.stations.map(station => (
50             <StationSnapArea
51               station={station}
52               key={station.name}
53               onMouseOver={onMouseOver}
54               onMouseOut={onMouseOut}
55             />
56           ))}
57         </Pane>
58       )}
59     </>
60   );
61 );
62
63 StationsController.propTypes = {};
```

StationSnapArea

```
function StationSnapArea({ station, onMouseOver, onMouseOut }) {
181   const zoom = useMapZoom();
182   const radius = useMemo(() => -75 * (zoom / 2) * station.size + 1000 * station.size, [zoom, station.size]);
183
184   if (zoom <= 8 && station.size === 3) return null;
185   if (zoom <= 10 && station.size === 2) return null;
186   if (zoom <= 11 && station.size === 1) return null;
187
188   return (
189     <Circle
190       center={station.coordinates}
191       radius={radius}
192       pathOptions={{ fillOpacity: 0, opacity: 0 }}
193       eventHandlers={[
194         { mouseover: e => onMouseOver(e, station),
195          mouseout: () => onMouseOut(station)
196        }
197      ]}
198    />
199  );
200}
201
202 StationSnapArea.propTypes = {
203   station: PropTypes.instanceOf(Station).isRequired,
204   onMouseOver: PropTypes.func,
205   onMouseOut: PropTypes.func
206 };
```

Wyświetlaniem stacji zajmuje się komponent *StationController*. Uzyskuje on dostęp do *stationsStore* i na odpowiednich warstwach renderuje oznaczenia stacji (*StationMarker*) oraz obręby przyciągania (*StationSnapArea*). Dla tego drugiego, obsługuje również zdarzenia:

- *onMouseOver* – po najechaniu na obręb przyciągania, stacja jest ustawiana jako przyciągnięta (wykorzystany jest setter *setSnappedStation* z klasy *StationStore*)
- *onMouseOut* – jeśli obecna stacja była oznaczona jako przyciągnięta to po wyjechaniu myszy z obrębu taka adnotacja zostaje usunięta.

Dodatkowym zdarzeniem obsługiwany z poziomu tego komponentu jest kliknięcie na dowolne miejsce mapy, które powoduje schowanie popup'u danej stacji (więcej o popup'ach nieco dalej).

Jest to komponent definiujący obszar przyciągania torów do stacji. W rzeczywistości jest to niewidoczny komponent *Circle* z biblioteki *react-leaflet*. Jego promień zależy od stopnia przybliżenia mapy oraz wielkości stacji. Komponenty dotyczące mniejszych stacji zwracają *null* (czyli nie są renderowane) przy mniejszych przybliżeniach.

StationMarker

```
64 const StationMarker = observer({ station }) => {
65   const map = useMap();
66   const zoom = useMapZoom();
67
68   const { stationStore, showError } = useGameStore();
69   const { snappedStation, showedPopup, setShowedPopup } = stationStore;
70   const hover = snappedStation?.station?.name === station.name;
71
72   const drawingRoute = useMatch("/game/routes/create");
73   const buildingTrack = useMatch("/game/tracks/build/*");
74
75   const { routeId } = useParams();
76   const { routeStore } = useGameStore();
77
78   > function onClick(e) { ... }
79
80   > function onRightClick() { ... }
81
82   let z = 22 * Math.pow((zoom - 5) / 10, 0.5);
83   let showName = zoom >= 12;
84   let sizeName = "smallest";
85
86   > switch (station.size) { ... }
87
88   > return ( ... );
89
90 });
91
```

Jak widać powyżej, opisywany komponent obsługuje również zdarzenia kliknięć:

- *onClick* – Kliknięcie lewym przyciskiem myszy w pinezkę powoduje różne akcje w zależności od wybranego trybu:
 - *tryb normalny* – pokazuje lub chowa popup;
 - *budowanie torów* – zaczyna budowę toru od tej stacji lub ją na niej kończy;
 - *tworzenie trasy* – dodaje przystanek.
- *onRightClick* – W trybie rysowania trasy, kliknięcie prawym przyciskiem powoduje usunięcie przystanku z obecnie tworzonej/edytowanej trasy.

Tutaj szczegóły implementacji:

```
78   > function onClick(e) {
79     if (!buildingTrack && !drawingRoute && routeId == null)
80       setShowedPopup(showedPopup === station.name ? null : station.name);
81
82     if (buildingTrack && !stationStore.enableSnapping) {
83       stationStore.setSnappedStation({ station: station, distance: 0 });
84       setTimeout(() => map.fire("click", e), 0); // This event is used by TrackDrawController
85       return;
86     }
87
88     let error;
89     if (drawingRoute) error = routeStore.addToCurrentRoute(station);
90     if (routeId != null) {
91       const route = routeStore.routes.find(r => r.id === +routeId);
92       error = route.addStation(station.name, stationStore.stationsMap);
93     }
94     error && showError(error);
95   }
96
97
98   > function onRightClick() {
99     if (drawingRoute) routeStore.removeFromCurrentRoute(station);
100    if (routeId != null) {
101      const route = routeStore.routes.find(r => r.id === +routeId);
102      route.removeStation(station.name);
103    }
104  }
```

Komponent wyświetlający pinezki/ikonę reprezentującą stację na mapie. Styl ikony jest różny w zależności od jej wielkości. Mniejsze stacje są ukrywane przy mniejszych stopniach przybliżenia mapy. Ogólny wygląd komponentu jest widoczny obok. Poniżej zaś, instrukcja *switch* odpowiedzialna za styl i chowanie mniejszych pinezek:

```
105   let z = 22 * Math.pow((zoom - 5) / 10, 0.5);
106   let showName = zoom >= 12;
107   let sizeName = "smallest";
108
109   > switch (station.size) {
110     case 1:
111       if (zoom <= 11) return null;
112       z = zoom * (zoom / 10);
113       showName = zoom >= 13;
114       break;
115     case 2:
116       sizeName = "small";
117       if (zoom <= 10) return null;
118       showName = zoom >= 12;
119       break;
120     case 3:
121       sizeName = "medium";
122       if (zoom <= 8) return null;
123       z = zoom * (zoom / 5.2);
124       showName = zoom >= 10;
125       break;
126     case 4:
127       sizeName = "big";
128       z = zoom * (zoom / 4.5);
129       showName = true;
130       break;
131     case 5:
132       sizeName = "biggest";
133       z = zoom * (zoom / 3);
134       showName = true;
135       break;
136     }
137
138   > return ( ... );
139
140 });
141
```

Elementy zwracane z tego komponentu wyglądają następująco:

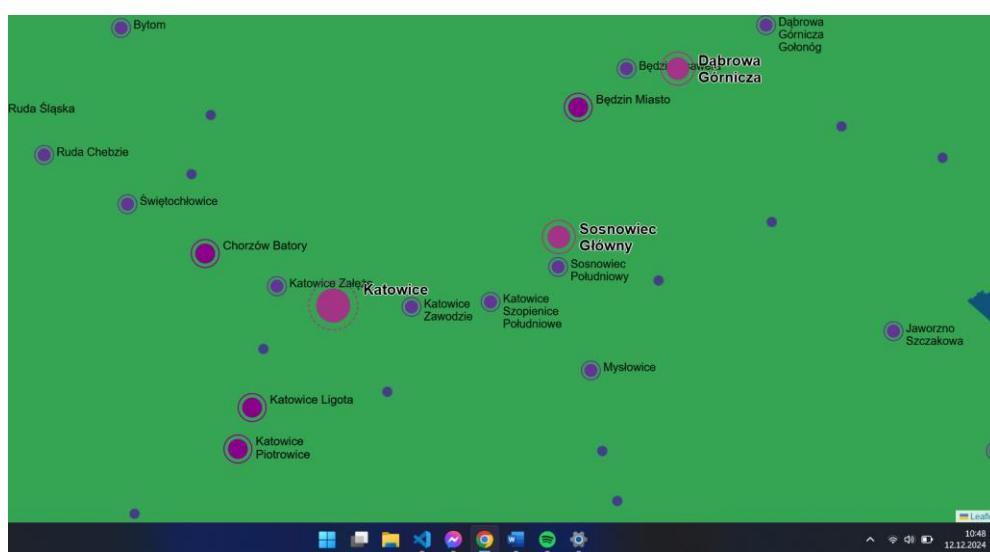
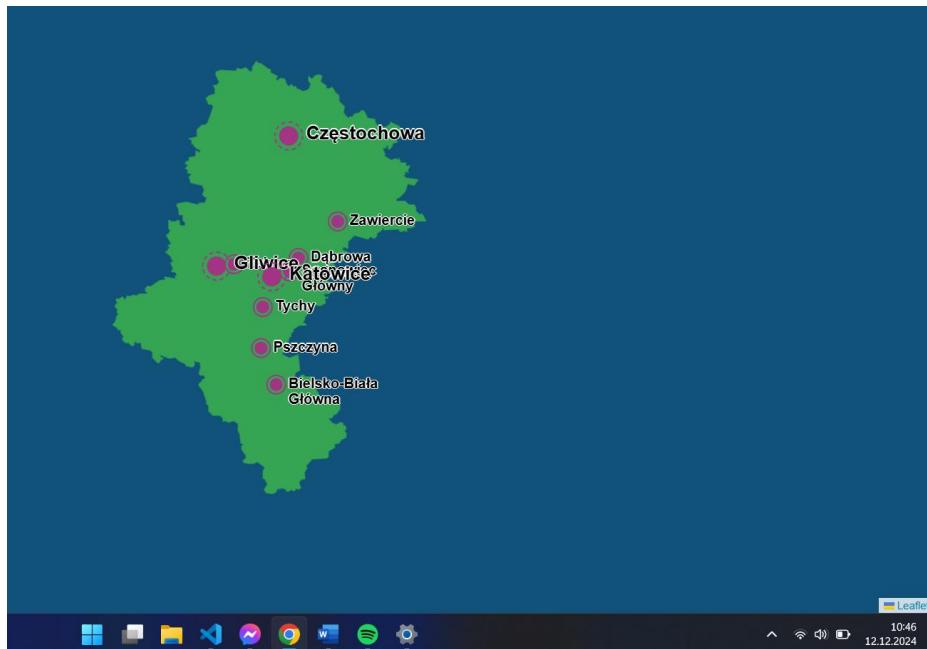
```
139 return (
140   <>
141   <Marker
142     position={station.coordinates}
143     zIndexOffset={station.size >= 4 ? 500 * station.size : 0}
144     icon={
145       <div className={'station-pin ${isHovered && 'hover'} ${sizeName} ${sizeName === 4 && 'big'}'>
146         <div className='pin' style={{ width: z + 'px', outlineOffset: z / 5 + 'px' }}></div>
147         {showName && (
148           <span style={{ left: z + 2 + z / 3 + 'px', top: z / -5 + 'px', fontSize: `clamp(1rem, ${z * 0.2 * station.size}px, 1.3rem)` }}>
149             {station.name}
150           </span>
151         )}
152       </div>
153     }
154     eventHandlers={{
155       click: onClick,
156       contextmenu: onRightClick
157     }}
158   </Marker>
159   {drawingRoute && !buildingTrack && showedPopup === station.name && (
160     <Marker
161       position={station.coordinates}
162       icon={<StationPopup station={station} />}
163       pane="popup"
164       bubblingMouseEvents={false}
165     </Marker>
166   )}
167 )
168 );
169 });
170 });
171 });
172 );
173 );
174 );
175 );
```

Pierwszy `<Marker/>` to ikona na mapie. Do jej wyświetlania nie używam zdjęć, lecz jsx (i oczywiście stylów css).

Umożliwia to biblioteka `react-leaflet-component-marker`.

Drugi marker używa tej samej techniki, lecz renderuje w swojej ikonie komponent `<StationPopup/>`.

Efekt jest następujący (w zależności od przybliżenia):



TrackController – wyświetlanie i budowanie torów

```
TracksController.jsx
src > Game > Map > Controllers > TracksController.jsx > TracksController > observer() callback > getTrackColors
12 const TracksController = observer(() => {
47   return (
48     <>
49       {renderDrawController && <TrackDrawController />}
50       {trackStore.tracks.map((track, i) => {
51         const C = track.getComponent();
52         const colors = getTrackColors(track);
53
54         const props = {
55           start: latLng(track.startStation.coordinates),
56           end: latLng(track.endStation.coordinates),
57           color: colors,
58           enableHover: renderDrawController && track.width !== trackWidth,
59           onClick: () => {
60             if (!renderDrawController || track.width === trackWidth) return;
61             const error = track.updateWidth(trackWidth);
62             if (error) showError(error);
63           },
64           setOptions: track.width > 1 ? setTrackOptions.bind(track) : undefined
65         };
66
67         if (renderWithActions) {
68           return (
69             <TrackWithActions
70               actions={
71                 <TrackDeleteAction
72                   onClick={() => trackStore.deleteTrack(track.id)}
73                   disabled={track.hasRoute}
74                 />
75                 }
76                 key={i}
77               >
78               <C {...props} />
79             </TrackWithActions>
80           );
81         }
82       })
83     );
84   });
85   return <C key={i} {...props} />;
86 );
87 );
88
89 export default TracksController;
99
```

Ten komponent odpowiada za wyświetlanie zbudowanych już torów. Istnieją 3 rodzaje torów. Każdy z nich jest reprezentowany przez osobny komponent, ale wszystkie z nich przyjmują te same props. Za dobranie odpowiedniego komponentu odpowiada metoda `getComponent()` wywołana na instancji każdego toru.

Jeśli użytkownik jest w ścieżce `/game/tracks`, ale nie wybrał rodzaju toru to nad torami pojawiają się ikony koszy. Ich kliknięcie powoduje usunięcie danego toru. Przyciski te są dezaktywowane jeśli przez dany fragment toru przechodzi trasa.

Za dobieranie kolorów odpowiada funkcja `getTrackColors()` zdefiniowana następująco:

```
19
20   const renderWithRoutes = useMatch("/game/routes/*");
21   const renderWithDraftRoute = useMatch("/game/routes/create");
22
23   function getTrackColors(track) {
24     if (!renderWithRoutes && !routeStore.highlightedRoute) {
25       return "#2572dd";
26     }
27
28     return track.lanes.map(l => {
29       if (!renderWithRoutes && l?.id !== routeStore.highlightedRoute.id) return "#2572dd";
30       if (!renderWithRoutes && l?.id === routeStore.highlightedRoute.id) return routeStore.highlightedRoute.color;
31
32       if (l == null) return "#2572dd";
33       if (l.draft && !renderWithDraftRoute) return "#2572dd";
34       return renderWithDraftRoute && !l.draft ? l.color + "a4" : l.color;
35     });
36   }
37 }
```

W tym komponencie zdefiniowana jest jeszcze funkcja `setTrackOptions`, która zajmuje się przypisywaniem współrzędnych geograficznych do instancji torów na podstawie obliczeń wykonanych w poszczególnych komponentach wyświetlających. Implementacja tej funkcji wygląda tak:

```
function setTrackOptions(info) {
  const track = this;
  if (track.width === 2) track.setLatlngs([info.leftTrackPoints, info.rightTrackPoints]);
  else if (track.width === 3) {
    const centerTrackPoints = [track.startStation.coordinates, track.endStation.coordinates].map(c => latLng(c));
    track.setLatlngs([info.leftTrackPoints, centerTrackPoints, info.rightTrackPoints]);
  }
}
```

TrackWithActions

```
24
25 function TrackWithActions({ children, actions }) {
26   const { start, end } = children.props;
27   const offset = latLng((start.lat + end.lat) / 2, (start.lng + end.lng) / 2);
28
29   return (
30     <LayerGroup>
31       {children}
32       <Marker position={offset} icon={actions} />
33     </LayerGroup>
34   );
35 }
36
37 TrackWithActions.propTypes = {
38   children: PropTypes.element.isRequired,
39   actions: PropTypes.element.isRequired
40 };
41
42 function TrackDeleteAction({ onClick, disabled = false }) {
43   function handleClick(e) {
44     e.stopPropagation();
45     onClick(e);
46   }
47
48   return (
49     <div className="track-action" data-style={tractActionStyles}>
50       <IconButton onClick={handleClick} disabled={disabled}>
51         | <i className="fas fa-trash"></i>
52       </IconButton>
53     </div>
54   );
55 }
56
57 TrackDeleteAction.propTypes = {
58   onClick: PropTypes.func.isRequired,
59   disabled: PropTypes.bool
60 };
61
```

Komponent stworzony był z myślą o możliwości wyświetlania wielu przycisków nad środkiem toru, ale w praktyce używany jest tylko do usuwania torów. Obok widoczna jest implementacja tego komponentu oraz z nim związanego *TrackDeleteAction*.

Komponenty poszczególnych torów

Szerokość linii wyznaczających tory jest zależna od stopnia przybliżenia mapy i za jej wyliczanie odpowiada hook *useTrackStyle*, który jest widoczny obok.

```
15
16 function useTrackStyle() {
17   const zoom = useMapZoom();
18
19   return {
20     fill: false,
21     weight: Math.round(zoom / 3) * (zoom > 12 ? Math.pow(1.2, zoom - 12) : 1)
22   };
23 }
```

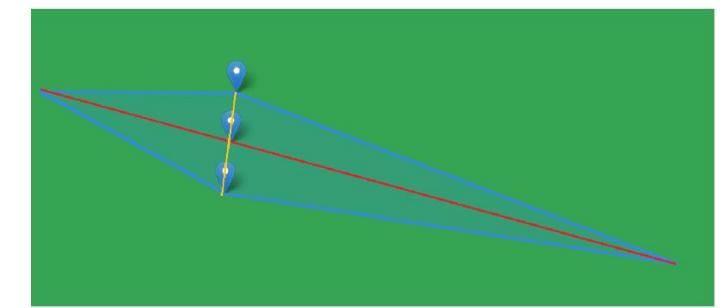
Pojedyńczy tor

```
62
63 function SingleTrack({ start, end, color, onClick, enableHover = false }) {
64   if (!Array.isArray(color)) color = [color];
65   if (color.length !== 1) console.warn("SingleTrack: color prop should be a string or an array with a single string");
66
67   const style = useTrackStyle();
68   const [hover, setHover] = useState(false);
69   const adjustedColors = useMemo(() => {
70     if (!hover) return color;
71     return color.map(c =>
72       c.replace(/[0-9a-f]{2}/gi, x => {
73         const num = parseInt(x, 16);
74         return Math.min(255, num + 20)
75           .toString(16)
76           .padStart(2, "0");
77       })
78     ), [hover, color]);
79
80   function onPolylineClicked() {
81     if (!enableHover) return;
82     onClick();
83   }
84
85   function hoverStart() {
86     if (!enableHover) return;
87     setHover(true);
88   }
89
90   function hoverEnd() {
91     setHover(false);
92   }
93
94   return (
95     <Polyline
96       positions={[start, end]}
97       pathOptions={{ ...style, color: adjustedColors[0] }}
98       eventHandlers={{ mouseover: hoverStart, mouseout: hoverEnd, click: onPolylineClicked }}
99     />
00   );
01 }
```

Najprostszy z komponentów wyświetlających tory. Tak naprawdę jest to jedynie linia bezpośrednio łącząca 2 stacje ze sobą. Jako *prop color* przyjmuję tylko jeden kolor, który jest automatycznie dostosowywany na jaśniejszy po najechaniu (jeśli *enableHover = true*).

Podwójny tor

Tory podwójne są reprezentowane przez 2 równoległe linie, które niedaleko stacji docelowych zbiegają się razem. To obliczanie współrzędnych punktów było największym wyzwaniem. Kod je obliczający korzysta ze skomplikowanej trygonometrii, więc nie będę go tu przytaczał – funkcję `calculateKiteVertices` należy traktować jako czarne pudełko. Argumenty przyjmowane przez tę funkcję to: punkt startowy, punkt końcowy, odległość między równoległą częścią toru oraz odległość od stacji, w której linie zaczynają się zbiegać. Podczas projektowania tego kodu było wiele eksperymentów – przy jednej z pierwszych implementacji był problem z kątem, który jest widoczny na załączonym obrazku.



Najważniejsza część implementacji widoczna jest tutaj:

```
  TracksController.jsx  Tracks.jsx  utils.js
src > Game > Map > Tracks.jsx > ...
111  function DoubleTrack({ start, end, color, onClick, separation = 1, enableHover = false, setOptions }){
112    const trackPoints = useMemo(() => {
113      const distance = start.distanceTo(end);
114      const tipLength = distance < 10000 ? distance * 0.2 : distance * 0.05;
115
116      const offsetX = distance - tipLength * 2;
117      const offsetY = Math.max(40 * zoom + (240 - 12 * -40) * separation, 80);
118      const { topLeft, topRight, bottomLeft, bottomRight } = calculateKiteVertices(start, end, offsetX, offsetY);
119      return [start, bottomRight, topRight, end, topLeft, bottomLeft];
120    }, [start, end, zoom, separation]);
121
122    const leftTrackPoints = useMemo(() => [trackPoints[0], trackPoints[1], trackPoints[2], trackPoints[3]], [trackPoints]);
123    const rightTrackPoints = useMemo(() => [trackPoints[0], trackPoints[5], trackPoints[4], trackPoints[3]], [trackPoints]);
124    const hoverDetectorPositions = useMemo(
125      () => [trackPoints[0], trackPoints[1], trackPoints[2], trackPoints[3], trackPoints[4], trackPoints[5]],
126      [trackPoints]
127    );
128
129    useEffect(() => {
130      setOptions && setOptions({ hoverDetectorPositions, leftTrackPoints, rightTrackPoints });
131    }, [hoverDetectorPositions, leftTrackPoints, rightTrackPoints, setOptions]);
132
133    return (
134      <>
135        {enableHover && (
136          <DoubleTrackHoverDetector
137            positions={hoverDetectorPositions}
138            style={style}
139            colors={rememberedColors}
140            setAdjustedColors={setAdjustedColors}
141            onClick={onClick}
142          />
143        )}
144        <Polyline
145          positions={leftTrackPoints}
146          pathOptions={{ ...style, color: (enableHover ? adjustedColors : rememberedColors)[0] }}
147          interactive={false}
148        />
149        <Polyline
150          positions={rightTrackPoints}
151          pathOptions={{ ...style, color: (enableHover ? adjustedColors : rememberedColors)[1] }}
152          interactive={false}
153        />
154      </>
155    );
156  };
157
```

```

 177 function DoubleTrackHoverDetector({ positions, style, colors, onClick, setAdjustedColors }) {
 178   const [hover, setHover] = useState(false);
 179
 180   function hoverStart() {
 181     setHover(true);
 182   }
 183
 184   function hoverEnd() {
 185     setHover(false);
 186   }
 187
 188   useEffect(() => {
 189     if (!hover) {
 190       setAdjustedColors(colors);
 191       return;
 192     }
 193
 194     const adjusted = colors.map(c => {
 195       const isHex = c.startsWith("#");
 196       if (isHex) {
 197         return c.replace(/\b[a-f]{2}\b/gi, x => {
 198           const num = parseInt(x, 16);
 199           return Math.min(255, num + 20)
200             .toString(16)
201             .padStart(2, "0");
202         });
203       }
204       return c;
205     });
206     setAdjustedColors(adjusted);
207   }, [hover, colors, setAdjustedColors]);
208
209   return (
210     <Polygon
211       positions={positions}
212       pathOptions={{ fillColor: "#f000", color: "#0f00", weight: style.weight + 1 }}
213       eventHandlers={{ mouseover: hoverStart, mouseout: hoverEnd, click: onClick }}
214     />
215   );
216 }

```

Ponieważ tor jest wyświetlany za pomocą pojedynczych linii nie ma prostego sposobu na wykrywanie czy kurSOR siĘ znajduje pomiędzy nimi. Dlatego stworzony zostaŁ komponent *DoubleTrackHoverDetector*. Używa on *Polygon* z *react-leaflet*. Odpowiada też za dostosowywanie kolorów po najechaniu myszą.

Potrójny tor

Tor potrójny jest połączeniem toru pojedynczego i podwójnego.

```

173 function TripleTrack({ start, end, color, onClick, enableHover = false, setOptions }){
174   const rememberedColors = useMemo(() => (!Array.isArray(color) ? [color, color, color] : color), [color]);
175   const [adjustedColors, setAdjustedColors] = useState(rememberedColors);
176   const [_options, _setOptions] = useState(null);
177   const style = useTrackStyle();
178
179   useEffect(() => {
180     if (!_options) return;
181     setOptions && setOptions({ leftTrackPoints: _options.leftTrackPoints, rightTrackPoints: _options.rightTrackPoints });
182   }, [_options, setOptions]);
183
184   return (
185     <>
186       {enableHover && _options && (
187         <DoubleTrackHoverDetector
188           positions={_options.hoverDetectorPositions}
189           style={style}
190           colors={rememberedColors}
191           setAdjustedColors={setAdjustedColors}
192           onClick={onClick}
193         />
194       )}
195       <SingleTrack start={start} end={end} color={(enableHover ? adjustedColors : rememberedColors)[2]} />
196       <DoubleTrack
197         start={start}
198         end={end}
199         color={enableHover ? adjustedColors : rememberedColors}
200         separation={1.5}
201         onClick={onClick}
202         setOptions={_setOptions}
203       />
204     </>
205   );
206 }

```

TrackDrawController

Jest to komponent zarządzający rysowaniem torów. Wyświetla je podczas rysowania śledząc pozycję kurSORA (jest ona aktualizowana co 4-tą „klatkę” dla lepszej wydajności). Oto fragmenty kodu za to odpowiadające:

```
120
121     if (!startStation || !endPoint) return null;
122     return (
123       <>
124         <Component
125           start={latLng(startStation.coordinates)}
126           end={endPoint}
127           color={isForbidden ? "#ec3220" : snappedStation.station ? "#2572dd" : "#da8220"}
128         />
129         <Marker position={endPoint} icon=<TrackCostIndicator cost={cost} /> interactive={false} />
130       </>
131     );
132   );
133
134   export default TrackDrawController;
135 
```

```
const mouseMoveCounter = useRef(0);
const handleMouseMove = ({ latlng }) => {
  if (mouseMoveCounter.current % 4 === 0 && startStation) {
    setSelectedEndPoint(latlng);
  }

  mouseMoveCounter.current++;
};

}; 
```

Kolizje

Dodatkowo zaimplementowane zostały obliczenia kolizji (sprawdzane jest czy tor nie przechodzi zbyt blisko innej stacji) – wtedy tor zmienia się na czerwony. Mimo to, można dalej zbudować taki tor – doszczętnie do wniosku, że w skrajnych sytuacjach obecna implementacja kolizji uniemożliwiały łączenia ze sobą blisko położonych stacji.

```
const checkCollision = () => {
  if (!startStation || !endPoint) {
    setIsForbidden(false);
    return;
  }

  const collisionRadiusMeters = 200 + 350 * trackWidth;
  const hasCollision = stationStore.stations.some(station => {
    if (station === startStation || snappedStation.station?.name === station.name) return false;
    const point = pointNearestCircle(latlng(startStation.coordinates), endPoint, latlng(station.coordinates));
    return point.distanceTo(latlng(station.coordinates)) < collisionRadiusMeters;
  });

  setIsForbidden(hasCollision);
};

}; 
```

Obsługa kliknięcia

Jeśli nie wybrano stacji początkowej to jest ona ustawiana. W przeciwnym wypadku kliknięcie zakańcza tor na klikniętej stacji oraz ustawia na niej stację początkową następnego toru. W momencie zakończenia toru jest pobierana także opłata zależna od długości oraz „szerokości” toru. Wcześniej jest oczywiście dokonywana walidacja i wyświetlane ewentualne błędy. Kliknięcia przełączają też tryb przyciągania.

```
const handleClick = () => {
  if (!snappedStation.station) return;
  if (startStation === null) {
    setStartStation(snappedStation.station);
    setBuildingTrack(true);

    setSnappedStation({ station: null, distance: null }); // Need to reset manually
    toggleSnapping(true);
  } else if (snappedStation.station.name !== startStation.name) {
    acceptTrack();
  }
};

}; 
```

```

function rejectTrack() {
  if (startStation == null) {
    navigate("/game/tracks");
  }

  setBuildingTrack(false);

  toggleSnapping(false);
  setSnappedStation({ station: null, distance: null }); // Need to reset manually

  setStartStation(null);
  setSelectedEndPoint(null);
}

function acceptTrack() {
  const exists = trackStore.trackExists(startStation.name, snappedStation.station.name);
  if (exists) {
    rejectTrack();
    showError("Tor pomiędzy wybranymi stacjami już istnieje");
    return;
  }

  const paySuccess = subtractMoney(cost);
  if (!paySuccess) return;

  const track = new Track(trackWidth, startStation, snappedStation.station);
  setStartStation(snappedStation.station);
  addTrack(track);
}

```

Wszystkie zdarzenia są
nastąpiły za pomocą hook'ów
`useMapEvent`. Wygląda to
następnie:

```

useMapEvent("click", handleClick);
useMapEvent("mousemove", handleMouseMove);
useMapEvent("contextmenu", rejectTrack);

```

TrainController – wyświetlanie pociągów

Ten komponent jedynie wyświetla komponenty `<TrainMarker>` dla wszystkich pociągów z przypisaną trasą.

```

10
11  const TrainController = observer(() => {
12    const { trainStore } = useGameStore();
13    const trainsToRender = trainStore.trains.filter(train => train.route !== null);
14
15    return trainsToRender.map(train => {
16      return <TrainMarker key={train.id} train={train} />;
17    });
18  });
19

```

TrainMarker

Do wyświetlania pociągu na mapie bezpośrednio użyty jest komponent `MovingMarker` autorskiej implementacji. Opisywany aktualnie komponent zajmuje się dostosowywaniem ścieżki do kierunku oraz zdarzeniami (które „przekazuje” do metod klasy `Train`) i restartowaniem animacji po ukończeniu pojedynczego przebiegu trasy. Logika restartująca uwzględnia odliczanie czasu postoju.

```

// Handle restarting the animation (and waiting)
useEffect(() => {
  if (isEnd > 0) {
    const normalInterval = route.routeInterval * 1000;
    const timePassed = Date.now() - isEnd;
    const remainingTime = normalInterval - timePassed;

    timeoutRef.current = setTimeout(() => {
      setIsEnd(0);
      markerRef?.resetAnimation();
    }, remainingTime / gameSpeed);
  }

  return () => clearTimeout(timeoutRef.current);
}, [gameSpeed, isEnd, markerRef, route.routeInterval]);

```

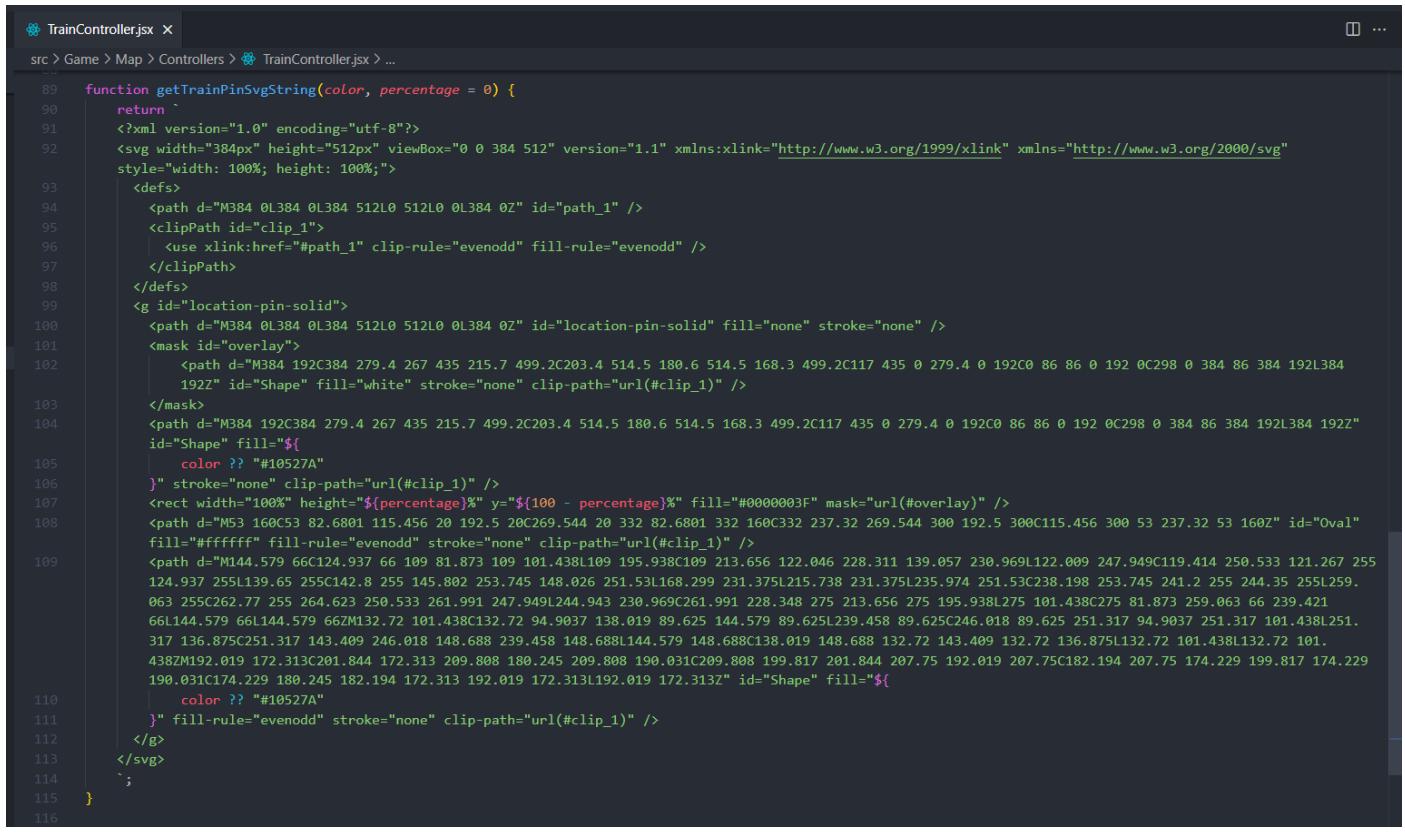
```

68
69  if (!adjustedPath) return null;
70  return (
71    <MovingMarker
72      key={train.id}
73      path={adjustedPath}
74      startPosition={train._currentLatlng}
75      ref={setMarkerRef}
76      speed={train.speed}
77      simulationSpeed={gameSpeed}
78      icon={icon}
79      eventHandlers={{
80        onStart: train.onRouteStart,
81        onEnd,
82        onStop: train.onStopReached,
83        onStopEnd: train.onStopDeparting
84      }}
85    />
86  );
87}
88

```

Ikona

Wyświetlana na mapie ikona jest w formacie *svg*. Nie jest jednak wczytywana jako obrazek tylko jako tekst używając dobrodziejstwa klasy *DivIcon* z biblioteki *leaflet*. Dzięki takiemu podejściu kolor ikony dynamicznie dopasowywuje się do koloru przypisanej trasy. Dodatkowo zaimplementowana jest maska, która ciemniejszym kolorem zaznacza część ikony równą procentowi wypełnienia pociągu.



```
src > Game > Map > Controllers > TrainController.jsx > ...
89  function getTrainPinSvgString(color, percentage = 0) {
90    return `
91      <?xml version="1.0" encoding="utf-8"?>
92      <svg width="384px" height="512px" viewBox="0 0 384 512" version="1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg"
93        style="width: 100%; height: 100%;"
94      </defs>
95      <path d="M384 0L384 0L384 512L0 512L0 0L384 0Z" id="path_1" />
96      <clipPath id="clip_1">
97        <use xlink:href="#path_1" clip-rule="evenodd" fill-rule="evenodd" />
98      </clipPath>
99    </defs>
100   <g id="location-pin-solid">
101     <path d="M384 0L384 0L384 512L0 512L0 0L384 0Z" id="location-pin-solid" fill="none" stroke="none" />
102     <mask id="overlay">
103       <path d="M384 192C384 279.4 267 435 215.7 499.2C203.4 514.5 180.6 514.5 168.3 499.2C117 435 0 279.4 0 192C0 86 86 0 192 0C298 0 384 86 384 192L384
104       <path d="M384 192C384 279.4 267 435 215.7 499.2C203.4 514.5 180.6 514.5 168.3 499.2C117 435 0 279.4 0 192C0 86 86 0 192 0C298 0 384 86 384 192L384 192Z" id="Shape" fill="white" stroke="none" clip-path="url(#clip_1)" />
105     </mask>
106     <path d="M144.579 66C124.937 66 109 81.873 109 101.438L109 195.938C109 213.656 122.046 228.311 139.057 230.969L122.009 247.949C119.414 250.533 121.267 255
107       124.937 255L139.65 255C142.8 255 145.802 253.745 148.026 251.53L168.299 231.375L215.738 231.375L235.974 251.53C238.198 253.745 241.2 255 244.35 255L259.
108       063 255C262.77 255 264.623 250.533 261.991 247.949L244.943 230.969C261.991 228.348 275 213.656 275 195.938L275 101.438C275 81.873 259.063 66 239.421
109       66L144.579 66L144.579 66ZM132.72 94.9037 138.019 89.625 144.579 89.625L239.458 89.625C246.018 89.625 251.317 94.9037 251.317 101.438L251.
110       317 136.875C251.317 143.409 246.018 148.688 239.458 148.688L144.579 148.688C138.019 148.688 132.72 143.409 132.72 136.875L132.72 101.438L132.72 101.
111       438ZM192.019 172.313C201.844 172.313 209.808 180.245 209.808 190.031C209.808 199.817 201.844 207.75 192.019 207.75C182.194 207.75 174.229 199.817 174.229
112       190.031C174.229 180.245 182.194 172.313 192.019 172.313L192.019 172.313Z" id="Shape" fill="${color ?? "#10527A"}"
113     }" stroke="none" clip-path="url(#clip_1)" />
114   </g>
115 </svg>
116 `;
117 }
```

```
const icon = useMemo(() => {
  return new DivIcon({
    html: getTrainPinSvgString(route.color, (train.passengers.length / train.seats) * 100),
    iconSize: [38, 51],
    iconAnchor: [38 / 2, 51]
  });
}, [route.color, train.passengers.length, train.seats]);
```

MovingMarker – przesuwająca się pinezka

Jest to autorski komponent umożliwiający animowanie pinezki na mapie, kompatybilny z biblioteką *react-leaflet*. Ma szeroki zakres zastosowania, dlatego przyjmuje szeroką gamę *prop'sów*:

```
211 MovingMarker.propTypes = {
212   path: PropTypes.arrayOf(
213     PropTypes.shape({
214       latlng: PropTypes.instanceOf(LatLng),
215       stop: PropTypes.shape({ duration: PropTypes.number })
216     })
217   ).isRequired,
218   eventHandlers: PropTypes.shape({
219     onStart: PropTypes.func,
220     onEnd: PropTypes.func,
221     onStop: PropTypes.func,
222     onStopEnd: PropTypes.func
223   }),
224   ref: PropTypes.object,
225   speed: PropTypes.number, // Speed in kilometers per hour
226   simulationSpeed: PropTypes.number, // Simulation speed multiplier
227   icon: PropTypes.object,
228   startPosition: PropTypes.instanceOf(LatLng)
229 };
230
231 export default MovingMarker;
232
```

Animacja używa funkcji wbudowanej w przeglądarki *requestAnimationFrame*, która pozwala wywoływać funkcję animującą około 60 razy na sekundę.

Zmienne pomocnicze

Z powodu tak wysokiej częstotliwości odświeżania animacji, do implementacji zmiennych pomocniczych używam hook'a *useRef* zamiast *useState*. Ten drugi odświeżałyby komponent setki razy i wszystko by się zacinało. Oto opis tych zmiennych:

```
const requestRef = useRef();
const previousTimeRef = useRef();
const progressRef = useRef(0);
const currentSegmentRef = useRef(0); // Index of last passed path point
const stopTimeoutRef = useRef();
const isResumingRef = useRef(false);
const isEndedRef = useRef(false);
const hasStartedRef = useRef(false); // Add this line
const pathMetricsRef = useRef(calculatePathMetrics(path));
```

- *requestRef* – Przechowuje id zwrócone przez ostatnie wywołanie *requestAnimationFrame*. Jest ono używane do „posprzątania” animacji w *cancelAnimationFrame*.
- *previousTimeRef* – Miejsce przechowywania czasu pomiędzy klatkami. Jest to używane do różnych obliczeń wewnętrz klatki.
- *progressRef* – Łączny przebyty dystans w metrach od początku aktualnego segmentu.
- *currentSegmentRef* – Indeks obecnego segmentu w tablicy *path*.
- *stopTimeoutRef* – Przechowuje id zwrócone przez wywołanie funkcji *setTimeout* odpowiadającej za odjazd z przystanku. Jet to konieczne ponieważ szybkość gry może ulec zmianie a wtedy konieczne jest ponowne obliczenie pozostałego czasu postoju.
- *isResumingRef* – Zmienna ma wartość *true* podczas 1 klatki po odjeździe z przystanku. Pozwala to w odpowiednim momencie wywołać zdarzenie *onStopEnd*.
- *isEndedRef* – Zmienna ma wartość *true* gdy animacja dobiegła końca. Jest to mechanizm zabazpieczający.
- *hasStartedRef* – Pozwala na wywołanie fragmentu kodu tylko zaraz po pierwszej inicjalizacji komponentu (pierwszym rozpoczęciu animacji).
- *pathMetricsRef* – Przechowuje obliczone długości każdego segmentu ścieżki. Dzięki temu lokalizacja pinezki obliczana jest bardziej dokładnie – nie musi bazować na czasie.

Ścieżka

Jak można zobaczyć po przytoczonych wcześniej typach, ścieżka używana wewnętrz tego komponentu odbiega od tej, na której bazują trasy. Ta ścieżka zbudowana jest z tablicy obiektów zawierających współrzędne geograficzne punktów oraz opcjonalnie informacje o przystanku. Ten rodzaj ścieżki na potrzeby tego projektu jest pozyskiwany za pomocą gettera *fullPath* z klasy *Route*.

```

34  class Route {
105    /**
106     * @returns {Array<latlng: import("leaflet").Latlng, stop: {duration: number, name: string} | undefined, debug: string>}|null
107    */
108    get fullPath() {
109      const path = this.path.flatMap((segment, i) => {
110        if (!segment.track.latlngs) return null;
111        let segmentLatlngs = segment.track.latlngs[segment.trackIndex];
112        if (segment.track.startStation.name === segment.to) segmentLatlngs = segmentLatlngs.toReversed();
113
114        return segmentLatlngs
115          .map((latlng, j) => {
116            const hasStop = j === segmentLatlngs.length - 1 && this.stations.includes(segment.to);
117            const stop = hasStop ? { duration: this.stopDuration * 1000, name: segment.to } : undefined;
118            return { latlng, stop, debug: `${segment.from} -> ${segment.to}` };
119          })
120          .slice(i === 0 ? 0 : 1);
121    });
122
123    if (path.includes(null)) return null;
124    return path;
125  }
126

```

Następnie, sam komponent *MovingMarker* oblicza odległości między jej poszczególnymi punktami:

```

10
11 // Calculate cumulative distances for each point
12 const calculatePathMetrics = path => {
13   const distances = [0]; // First point starts at 0
14   let totalLength = 0;
15
16   for (let i = 0; i < path.length - 1; i++) {
17     totalLength += path[i].latlng.distanceTo(path[i + 1].latlng);
18     distances.push(totalLength);
19   }
20
21   return { totalLength, distances };
22 };

```

Główna pętla animacji

```

src > Game > Map > MovingMarker > MovingMarker.jsx > MovingMarker > animate > useCallback() callback
48  const MovingMarker = ({ path, speed = 50, simulationSpeed = 1, eventHandlers = {}, startPosition, ref, ...props }) => {
50
51    const animate = useCallback(
52      time => {
53
54        const deltaTime = (time - previousTimeRef.current) / 1000;
55        previousTimeRef.current = time;
56
57        // Update total distance traveled
58        // Convert speed from km/h to m/s: (km/h) / 3.6 = m/s
59        progressRef.current += deltaTime * ((speed * simulationSpeed) / 3.6);
60
61        // Find current segment based on total distance
62        let newSegment = currentSegmentRef.current;
63        while (newSegment < path.length - 1 && progressRef.current > pathMetricsRef.current.distances[newSegment + 1]) {
64          newSegment++;
65        }
66
67        const didJustUpdateSegment = newSegment !== currentSegmentRef.current;
68        currentSegmentRef.current = newSegment;
69
70        if (currentSegmentRef.current >= path.length - 1) {
71          setPosition(path[path.length - 1].latlng);
72          stopAnimation();
73          return;
74        }
75
76        // Handle station stops
77        if (path[currentSegmentRef.current].stop && didJustUpdateSegment) {
78          typeof onStop === "function" && onStop(path[currentSegmentRef.current].stop);
79          setIsPaused(Date.now());
80        }
81
82        // Calculate fraction within current segment
83        const segmentStartDist = pathMetricsRef.current.distances[currentSegmentRef.current];
84        const segmentEndDist = pathMetricsRef.current.distances[currentSegmentRef.current + 1];
85        const segmentLength = segmentEndDist - segmentStartDist;
86        const segmentProgress = progressRef.current - segmentStartDist;
87        const fraction = segmentProgress / segmentLength;
88
89        const currentPos = interpolatePosition(
90          path[currentSegmentRef.current].latlng,
91          path[currentSegmentRef.current + 1].latlng,
92          Math.min(fraction, 1)
93        );
94        setPosition([currentPos.lat, currentPos.lng]);
95        requestRef.current = requestAnimationFrame(animate);
96      },

```

Najważniejszą częścią funkcji *animate* jest interpolacja pozycji, czyli obliczanie kolejnych punktów ścieżki. Implementacja metody interpolującej na następnej stronie:

```
6 // Linear interpolation between two points
7 const interpolatePosition = (start, end, fraction) => {
8   return new LatLng(start.lat + (end.lat - start.lat) * fraction, start.lng + (end.lng - start.lng) * fraction);
9 };
```

Wznawianie animacji po wczytaniu

By po wczytaniu gry, wznowić animację pociągu od odpowiedniego punktu, trzeba ten punkt obliczyć na podstawie ostatniej znanej pozycji.

```
24 const findClosestPassedPointAndProgress = (path, pathMetrics, startPosition) => {
25   if (!startPosition) return { index: 0, additionalProgress: 0 };
26
27   let minDistance = Infinity;
28   let closestIndex = 0;
29
30   // Find closest point
31   for (let i = 0; i < path.length; i++) {
32     const distance = path[i].latlng.distanceTo(startPosition);
33     if (distance < minDistance) {
34       minDistance = distance;
35       closestIndex = i;
36     }
37   }
38
39   // Get the previous point as our reference
40   const startIndex = Math.max(0, closestIndex - 1);
41
42   // Calculate additional progress from the previous point to startPosition
43   const additionalProgress = pathMetrics.distances[startIndex] + path[startIndex].latlng.distanceTo(startPosition);
44
45   return { index: startIndex, additionalProgress };
46 };
```

Rozpoczynanie animacji

Poniższy kod wykonywany jest tylko raz na samym początku cyklu życia komponentu. Rozpoczyna animację – opcjonalnie od wskazanego punktu.

```
120
159   // Handle animation start and clean-up
160   useEffect(() => {
161     if (!requestRef.current) {
162       if (!hasStartedRef.current) {
163         if (startPosition) {
164           const { index, additionalProgress } = findClosestPassedPointAndProgress(
165             path,
166             pathMetricsRef.current,
167             startPosition
168           );
169           currentSegmentRef.current = index;
170           progressRef.current = additionalProgress;
171         }
172
173         hasStartedRef.current = true;
174         typeof onStart === "function" && onStart();
175       }
176
177       requestRef.current = requestAnimationFrame/animate);
178     }
179
180     return cleanupAnimation;
181   }, [animate, cleanupAnimation, onStart, startPosition, path]);
```

Obsługa przystanków

Konieczne jest wzięcie pod uwagę możliwości zmiany prędkości symulacji podczas odbywania przystanku.

```
188 // Handle Pause / unpause
189 useEffect(() => {
190   if (isPaused > 0) {
191     stopTimeoutRef.current && clearTimeout(stopTimeoutRef.current);
192
193     const elapsedTime = Date.now() - isPaused;
194     const originalDuration = path[currentSegmentRef.current].stop.duration;
195     const remainingTime = Math.max(0, (originalDuration - elapsedTime * simulationSpeed) / simulationSpeed);
196
197     stopTimeoutRef.current = setTimeout(() => {
198       setIsPaused(false);
199       isResumingRef.current = true;
200     }, remainingTime);
201   }
202
203   return () => stopTimeoutRef.current && clearTimeout(stopTimeoutRef.current);
204 }, [simulationSpeed, path, isPaused]);
```

Możliwość resetu animacji

W react'cie istnieje tylko jedna możliwość przekazywania metod lub wartości do komponentu-rodzica. Jest poprzez `ref` i hook `useImperativeHandle`. Taka też możliwość została tutaj wykorzystana:

```
105
106   useImperativeHandle(ref, () => ({ resetAnimation, position }), [position, resetAnimation]);
107
```

Sama funkcja resetująca wygląda tak:

```
144
145   const resetAnimation = useCallback(() => {
146     cleanupAnimation();
147     progressRef.current = 0;
148     currentSegmentRef.current = 0;
149     previousTimeRef.current = undefined;
150     isResumingRef.current = false;
151     isEndedRef.current = false;
152
153     setIsPaused(false);
154     setPosition(path[0].latlong);
155     requestRef.current = requestAnimationFrame/animate);
156     typeof onStart === "function" && onStart();
157   }, [animate, cleanupAnimation, onStart, path]);
```

Główne menu

TracksMenu

```
src > Ui > pages > TracksMenu.jsx > TracksMenu > observer() callback
12 const TracksMenu = observer(() => {
13   const { "*": splat } = useParams();
14   const trackWidth = +splat;
15 
16   function getLinkPath(newTrackWidth) {
17     if (trackWidth === newTrackWidth) return "/game/tracks";
18     return `/game/tracks/build/${newTrackWidth}`;
19   }
20 
21   return (
22     <div className="tracks-menu" data-style={style}>
23       <h2>Buduj tory</h2>
24       <ul>
25         <li className={trackWidth === 1 ? "active" : ""}>
26           <Link to={getLinkPath(1)}>
27             <div className="img-container">
28               <img src={singleTrackIcon} alt="Ikona pojedyńczego toru" />
29             </div>
30             <div className="label">Pojedyńczy tor</div>
31             <div className="cost">
32               <img src={moneyImg} alt="Ikona zielonych pieniędzy" /> {Track.prices[1]} / km
33             </div>
34           </Link>
35         </li>
36         <li className={trackWidth === 2 ? "active" : ""}>
37           <Link to={getLinkPath(2)}>
38             <div className="img-container">
39               <img src={singleTrackIcon} alt="Ikona pojedyńczego toru" />
40               <img src={singleTrackIcon} alt="Ikona pojedyńczego toru" />
41             </div>
42             <div className="label">Podwójny tor</div>
43             <div className="cost">
44               <img src={moneyImg} alt="Ikona zielonych pieniędzy" /> {Track.prices[2]} / km
45             </div>
46           </Link>
47         </li>
48         <li className={trackWidth === 3 ? "active" : ""}>
49           <Link to={getLinkPath(3)}>
50             <div className="img-container">
51               <img src={singleTrackIcon} alt="Ikona pojedyńczego toru" />
52               <img src={singleTrackIcon} alt="Ikona pojedyńczego toru" />
53               <img src={singleTrackIcon} alt="Ikona pojedyńczego toru" />
54             </div>
55             <div className="label">Potrójny tor</div>
56             <div className="cost">
57               <img src={moneyImg} alt="Ikona zielonych pieniędzy" /> {Track.prices[3]} / km
58             </div>
59           </Link>
60         </li>
61       </ul>
62     </div>
63   )
64 }
```

Efekt ze stylami jest taki (screen wykonany po wybraniu podwójnego toru):



Jest to najprostsze ze wszystkich menu gry.

Wyświetla jedynie listę z 3 rodzajami torów do wyboru.

Po kliknięciu, wybrany tor się podświetla, a gra przechodzi w tryb budowania.

Komponent nie przechowuje żadnego swojego stanu – wszystko jest na podstawie adresu url.

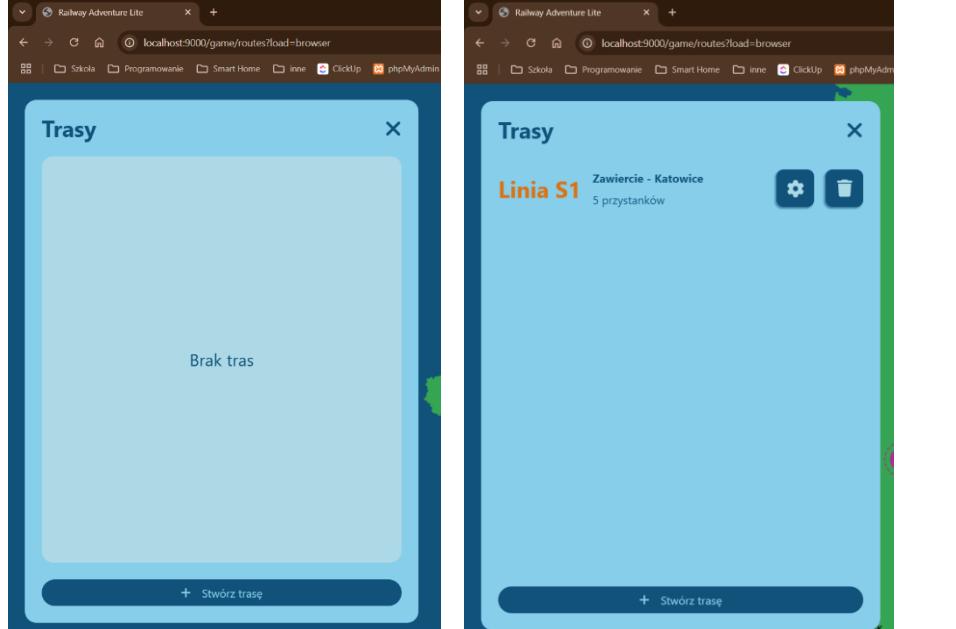
RoutesMenu

Zakładka do zarządzania trasami jest bardziej skomplikowana. Posiada 2 widoki w zależności od ścieżki url:

- /game/routes
- /game/routes/create lub /game/routes/details/id

Przegląd tras

```
RoutesMenu.jsx
src > Ui > pages > RoutesMenu > RoutesMenu.jsx > ...
14
15 const RoutesMenu = observer(() => {
16   const { routeStore } = useGameStore();
17
18   return (
19     <>
20       <h2>Trasy</h2>
21       <div className="routes-menu" data-style={style}>
22         {routeStore.routes.length > 0 && (
23           <Scrollbars>
24             <ul className="routes-list">
25               {routeStore.routes.map(route => (
26                 <RouteTile route={route} key={route.id}>
27                   <IconLinkButton to={"/game/routes/details/" + route.id} inverted>
28                     <i className="fas fa-cog"></i>
29                   </IconLinkButton>
30                   <IconButton onClick={() => routeStore.deleteRoute(route.id)} inverted>
31                     <i className="fas fa-trash"></i>
32                   </IconButton>
33                 </RouteTile>
34               )));
35             </ul>
36             </Scrollbars>
37         )
38         {routeStore.routes.length === 0 && <div className="no-routes">Brak tras</div>}
39         <ElevatedLinkButton to="create">
40           <i className="fas fa-plus"></i> Stwórz trasę
41         </ElevatedLinkButton>
42       </div>
43     </>
44   );
45 );
46
47 const RouteTile = observer(({ route, children }) => {
48   return (
49     <li data-style={tileStyle} className="route-tile">
50       <p className="name" style={{ color: route.color }}>
51         {route.name}
52       </p>
53       <p className="route">
54         {route.stations[0]} - {route.stations.at(-1)}
55       </p>
56       <div className="stations">{route.stations.length} przystanków</div>
57       <div className="buttons">{children}</div>
58     </li>
59   );
60 );
61
62 export { RouteTile };
63 export default RoutesMenu;
```



W podstawowej ścieżce użytkownik może zobaczyć wszystkie swoje trasy. Wyświetlane są tylko podstawowe parametry: nazwa, kolor, stacja początkowa i końcowa oraz liczba przystanków. Z tego poziomu można usunąć trasy lub przejść do ich szczegółów. Jeśli nie ma tras to wyświetlany jest adekwatny komunikat.

Tworzenie/szczegóły trasy

Do wyświetlania szczegółów oraz tworzenia trasy wykorzystywany jest ten sam komponent. Nie było potrzeby powtarzania kodu, a dzięki takiemu rozwiązaniu widok szczegółów umożliwia także edycję (która na dodatek zapisuje się automatycznie).

```
7 const RouteDetails = observer(() => {
8   const { routeStore, stationStore, showError } = useGameStore();
9   const navigate = useNavigate();
10
11   const { routeId } = useParams();
12   const route = useMemo(() => {
13     if (routeId == null) {
14       if (!routeStore.currentRoute) routeStore.initCurrentRoute();
15       return routeStore.currentRoute;
16     }
17
18     return routeStore.routes.find(r => r.id === +routeId);
19   }, [routeId, routeStore]);
20
21   useEffect(() => () => routeStore.discardCurrentRoute(), [routeStore, routeStore.discardCurrentRoute]);
22 })
```

Trasa jest tworzona jeśli nie jest podana w adresie url (czyli ścieżka to `/create`). Jeżeli nowa trasa jest tworzona a użytkownik opuści menu to wprowadzone zmiany są odrzucone (wywołanie metody `discardCurrentRoute` wewnętrz `cleanup function` w hook'u `useEffect`).

Pozostałe funkcje zdefiniowane w tym komponencie to `toggleStop` oraz `accept`. Ta pierwsza sprawdza czy stacja jest dodana jako przystanek i zmienia to oraz obsługuje błędy. Druga zaś, zapisuje tworzoną trasę i przekierowywuje do przeglądu.

Interfejs wyświetlany przez ten komponent jest złożony i składa się z 3 części:

- Pole na nazwę oraz kolor trasy:

```
60
61   if (!route) return null;
62   return (
63     <>
64       <Link to="/game/routes" className="back-button" style={{ float: "left" }}>
65         | <i className="fas fa-arrow-left">/</i>
66       </Link>
67       <h2>(routeId ? "Szczegóły trasy" : "Stwórz trasę")</h2>
68       <div className="route-details-menu" data-style={style}>
69         <Scrollbar className="scroll-container">
70           <form>
71             <input type="text" value={route.name} onChange={({ target: t }) => route.setName(t.value)} />
72             <div className="color-input-wrapper">
73               <input type="color" value={route.color} onChange={({ target: t }) => route.setColor(t.value)} />
74             </div>
75           </form>
76         <div className="stops-list-wrapper" >
```

- Lista przystanków (wraz z dekoracyjną strzałką po lewej stronie):

```
<div className="stops-list-wrapper">
  <h3>Trasa</h3>
  {route.stations.length > 0 && (
    <>
      <div className="arrow-container">
        <div className="tip"></div>
        <div className="middle"></div>
        <div className="bottom"></div>
      </div>
      <ul className="stops-list">
        {route.path.map((segment, i) => {
          const isStop = route.stations.includes(segment.from);
          return (
            <li className={isStop ? "" : "shadow"} key={segment.from + segment.to}>
              <div className="decoration"></div>
              <p>{segment.from}</p>
              {i > 0 && (
                <button onClick={() => toggleStop(segment.from)}>
                  <i className={`${isStop ? "fas" : "fan"} fa-stop-circle`}>/<i>
                  {isStop ? "\u00A0Usuń przystanek" : "Dodaj przystanek"}
                </button>
              )}

              {i === 0 && (
                <button onClick={() => toggleStop(segment.from)}>
                  <i className="fas fa-trash">/<i>
                  Usuń
                </button>
              )}
            </li>
          );
        })
      {route.stations.length > 0 && (
        <li>
          <div className="decoration"></div>
          <p>{route.stations.at(-1)}</p>
          <button onClick={() => toggleStop(route.stations.at(-1))}>
            <i className="fas fa-trash">/<i>
            Usuń
          </button>
        </li>
      )}
    </ul>
  </>
)}
```

Oczywiście lista wyświetlana jest tylko wtedy gdy została wybrana co najmniej 1 stacja. W przeciwnym wypadku w tym miejscu jest wyświetlany odpowiedni komunikat:

```
77   <div className="empty">
78     Kliknij na stację początkową,
79     następnie wybierz przystanki pośrednie
80   </div>
81 }
```

- Ustawienia parametrów trasy takich jak cena oraz czas postoju. W tym celu wykorzystane są autorskie komponenty `<Slider/>` oraz `<InfoRow/>`, o których więcej w dalszej części dokumentacji. Na koniec komponentu jest jeszcze wyświetlany przycisk do zapisywania trasy (jeśli tworzona jest nowa trasa – w przypadku edycji on nie występuje).

```

  1 const RouteDetails = observer(() => {
  2   <div>
  3     <h3>Ustawienia</h3>
  4     <Slider min={0.1} max={0.7} step={0.05} value={route.pricePerKm} onChange={route.setPricePerKm}>
  5       <i className="fas fa-money-bills"></i> Cena za kilometr
  6     </Slider>
  7     <Slider>
  8       min={30}
  9       max={5 * 60}
 10      step={30}
 11      value={route.stopDuration}
 12      renderedValue={route.stopDuration / 60 + " min"}
 13      onChange={route.setStopDuration}
 14    >
 15      <i className="fas fa-clock"></i> Czas postoju na stacjach pośrednich
 16    </Slider>
 17    <Slider>
 18      min={2 * 60}
 19      max={20 * 60}
 20      step={60}
 21      value={route.routeInterval}
 22      renderedValue={route.routeInterval / 60 + " min"}
 23      onChange={route.setRouteInterval}
 24    >
 25      <i className="fas fa-clock"></i> Czas postoju na stacjach końcowych
 26    </Slider>
 27  </div>
 28  <div>
 29    <h3>Informacje</h3>
 30    <InfoRow value={route.stations.length}>
 31      <i className="fas fa-location-dot"></i> Przystanki
 32    </InfoRow>
 33    <InfoRow value={(route.distance / 1000).toFixed(1) + " km"}>
 34      <i className="fas fa-ruler-horizontal"></i> Długość
 35    </InfoRow>
 36  </div>
 37  <Scrollbars>
 38    {route.draft && (
 39      <ElevatedButton onClick={accept} disabled={route.stations.length < 2}>
 40        <i className="fas fa-check"></i> Stwórz trasę
 41      </ElevatedButton>
 42    )}
 43  </div>
 44);
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
  
```

TrainsMenu

Tutaj podobnie jak w przypadku tras mamy więcej niż jedną „zakładkę”. Są to następujące ścieżki:

- `/game/trains` – przegląd wszystkich pociągów;
- `/game/trains/buy/locomotive` lub `/game/trains/buy/emu` – zakładka do kupowania pociągów (określonego typu);
- `/game/trains/id/assign-route` – menu, gdzie można przypisać trasę.

Przegląd pociągów

```

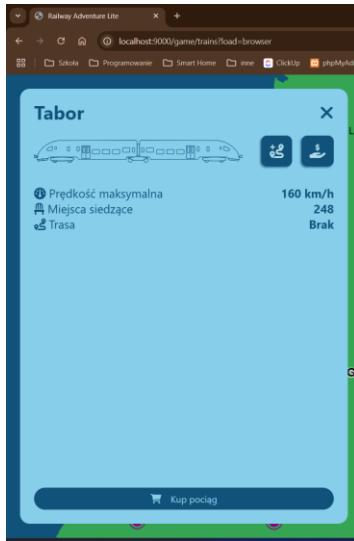
  1 import style from "./TrainsMenu.component.scss";
  2
  3 const TrainsMenu = observer(() => {
  4   const { trainStore } = useGameStore();
  5
  6   return (
  7     <>
  8       <h2>Tabor</h2>
  9       <div>
 10         <div>
 11           {trainStore.trains.length > 0 && (
 12             <Scrollbars>
 13               <ul>
 14                 {trainStore.trains.map(train => (
 15                   <TrainTile key={train.id} train={train} />
 16                 ))}
 17               </ul>
 18             </Scrollbars>
 19           )
 20         </div>
 21         {trainStore.trains.length === 0 && (
 22           <div>
 23             <p>Nie masz jeszcze żadnych pociągów</p>
 24           </div>
 25         )
 26         <ElevatedLinkButton to="buy/locomotive">
 27           <i className="fas fa-cart-shopping"></i> Kup pociąg
 28         </ElevatedLinkButton>
 29       </div>
 30     </>
 31   );
 32 }
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
  
```

Widok ten prezentuje listę wszystkich pociągów zakupionych przez użytkownika wraz z ich danymi. Komponent `TrainMenu` wyświetla listę, a poszczególne elementy `TrainTile`. Ten drugi można podzielić na 2 części: obrazek wraz z przyciskami oraz poniżej dane techniczne.

```
RouteDetails.jsx TrainsMenu.jsx M X
src > Ui > pages > TrainsMenu > TrainsMenu.jsx > [o] TrainTile > observer() callback
49 const TrainTile = observer(({ train }) => {
50
51     return (
52         <li>
53             <img src={train.type === "unit" ? emojiIcon : locomotiveIcon} alt="" />
54             {!train.route && (
55                 <IconLinkButton className="add-route-btn" to={`/game/trains/${train.id}/assign-route`} inverted>
56                     <img src={routeAddIcon} alt="Trasa z plusikiem" />
57                 </IconLinkButton>
58             )}
59             {train.route && (
60                 <IconButton onClick={() => train.assignRoute(null)} className="remove-route-btn" inverted>
61                     <img src={routeRemoveIcon} alt="Przekreślona trasa" />
62                 </IconButton>
63             )}
64             {train.type === "carriage" && (
65                 <IconLinkButton className="carriage-config-btn" to={`/game/trains/${train.id}`} inverted>
66                     <i className="fas fa-cog"></i>
67                 </IconLinkButton>
68             )}
69             <IconButton inverted onClick={() => trainStore.sellTrain(train.id)}>
70                 <i className="fas fa-hand-holding-dollar"></i>
71             </IconButton>
72             <div className="data">
73                 <p>{train.speed}</p>
74             </div>
75         </li>
76     );
77 });
78
79 export default TrainsMenu;
```

```
RouteDetails.jsx TrainsMenu.jsx M X
src > Ui > pages > TrainsMenu > TrainsMenu.jsx > [o] TrainTile > observer() callback
49 const TrainTile = observer(({ train }) => {
50
51     </IconButton>
52     <div className="data">
53         <p>{train.speed}</p>
54         <span>
55             <i className="fas fa-gauge"></i> Prędkość maksymalna{" "}
56         </span>
57         <b>{train.speed}</b> km/h
58     </div>
59     {train.type === "carriage" && (
60         <p>{train.seats}</p>
61         <span>
62             <i className="fas fa-weight-hanging"></i> Liczba wagonów{" "}
63         </span>
64         <b>{train.carriages.length} / {train.maxCarriages}</b>
65     )}
66     </p>
67     <p>{train.seats}</p>
68     <span>
69         <i className="fas fa-chair"></i> Miejsca siedzące{" "}
70     </span>
71     <b>{train.seats}</b>
72     </p>
73     {train.route && (
74         <p>{train.passengers}</p>
75         <span>
76             <i className="fas fa-users"></i> Pasażerowie{" "}
77         </span>
78         <b>{train.passengers.length}</b>
79     )}
80     </p>
81     <p>{train.route}</p>
82     <span>
83         <i className="fas fa-route"></i> Trasa{" "}
84     </span>
85     <b>{train.route ? train.route.stations[0] + " - " + train.route.stations.at(-1) : "Brak"}</b>
86     </p>
87     </div>
88     </li>
89 );
90
91 );
92
93 export default TrainsMenu;
```

Na następnej stronie widoczny jest efekt powyższego kodu w przeglądarce.



Kupowanie pociągów

```
const BuyTrain = () => {
  const { trainStore, money, subtractMoney } = useGameStore();
  const gameStore = useGameStore();

  const [speed, setSpeed] = useState(160);
  const [object, setObject] = useState({ cost: 0 });

  const locomotive = useMatch("/game/trains/buy/locomotive");
  const emu = useMatch("/game/trains/buy/emu");
  const navigate = useNavigate();

  function buy() {
    const paySuccess = subtractMoney(object.cost);
    if (!paySuccess) return;

    if (locomotive) {
      trainStore.addTrain(new CarriageTrain(object, gameStore));
    } else if (emu) {
      trainStore.addTrain(new UnitTrain(object, gameStore));
    }
  }

  navigate("/game/trains");
}
```

Zwracany jsx jest taki:

```
RouteDetails.jsx   TrainsMenujsx M   BuyTrainjsx X
src > Ui > pages > TrainsMenu > BuyTrainjsx > ...
24  const BuyTrain = () => {
48    return (
49      <>
50        <Link to="/game/trains" className="back-button" style={{ float: "left" }}>
51          <i className="fas fa-arrow-left"></i>
52        </Link>
53        <h2>Kup pociąg</h2>
54        <div className="buy-train-menu" data-style={style}>
55          <Scrollbars>
56            <div className="inner-wrapper">
57              <nav className="tabs">
58                <ul>
59                  <li>
60                    <NavLink to="/game/trains/buy/locomotive">Lokomotywy</NavLink>
61                  </li>
62                  <li>
63                    <NavLink to="/game/trains/buy/emu">Zespół trakcyjny</NavLink>
64                  </li>
65                </ul>
66              </nav>
67              <img src={locomotive ? locomotiveIcon : emuIcon} alt="" />
68              <Slider
69                min={100}
70                max={220}
71                step={20}
72                value={speed}
73                renderedValue={`${speed + " km/h"}`}
74                onChange={v => setSpeed(v)}
75              >
76                <i className="fas fa-gauge"></i> Prędkość
77              </Slider>
78              {locomotive && <BuyLocomotive speed={speed} setObject={setObject} />}
79              {emu && <BuyEmu speed={speed} setObject={setObject} />}
80            </div>
81          <InfoRow
82            value={
83              <span className={object.cost > money ? "red" : undefined}>
84                {object.cost.toLocaleString("en-US").replace(/\,/g, " ")}.replace(" ", " ")
85              </span>
86            >
87              <i className="fas fa-money-bills"></i> Koszt
88            </InfoRow>
89          </div>
90        <Scrollbars>
91          <ElevatedButton onClick={buy} disabled={object.cost > money}>
92            <i className="fas fa-cart-shopping"></i> Kup
93          </ElevatedButton>
94        </div>
95      </>
96    );
97  );
```

Tak jak było opisane wcześniej w tym dokumencie, istnieją 2 rodzaje pociągów. Dlatego po wejściu w menu kupowania są 2 zakładki. W obecnym stanie nie są zaimplementowane wagony, więc o ile kupowanie lokomotyw działa to jest pozbawione sensu.

Nadrzędny komponent *BuyTrain* przechowuje stan wspólny dla obu typów pociągów. Jest też funkcja do kupowania pociągów, która obsługuje błędy.

Tutaj ponownie zastosowane zostały komponenty *Slider* i *InfoRow*. Do wyświetlania specyficznych opcji każdego rodzaju pociągu są użyte komponenty *BuyLocomotive* oraz *BuyEmu*. O nich na następnych stronach.

BuyLocomotive

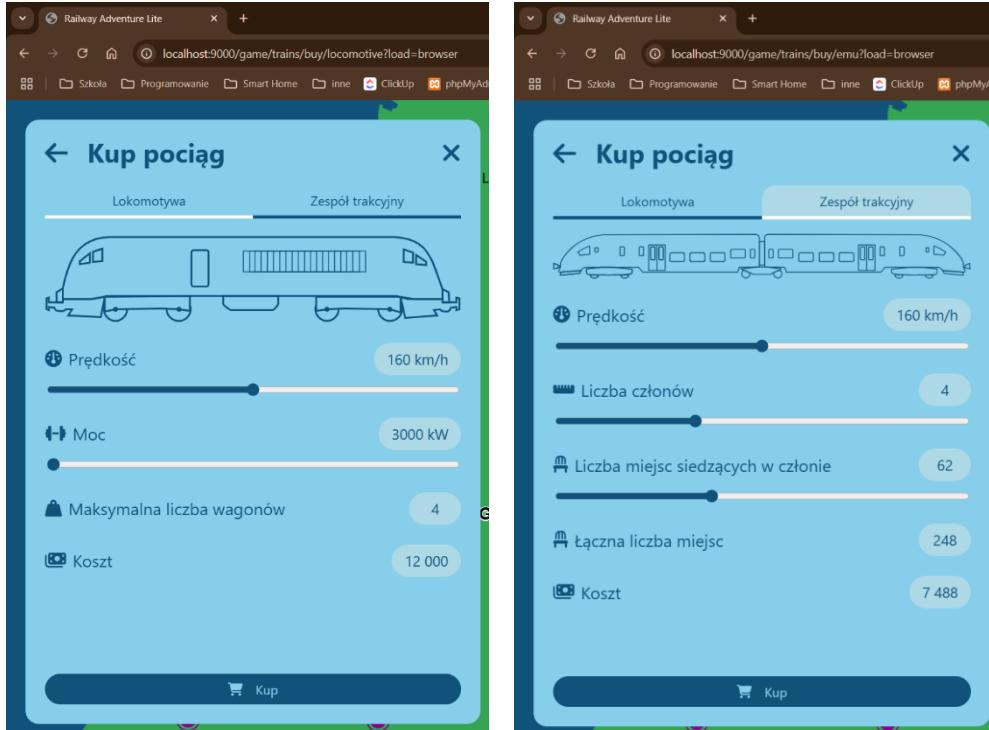
```
RouteDetails.jsx TrainsMenu.jsx M BuyTrain.jsx X
src > Ui > pages > TrainsMenu > BuyTrain.jsx > BuyTrain
99
100 const BuyLocomotive = ({ speed, setObject }) => {
101   const [strength, setStrength] = useState(3000);
102   const maxCarriages = Math.round(strength / (speed * 5));
103
104   useEffect(() => {
105     setObject({
106       speed,
107       strength,
108       maxCarriages,
109       cost: (strength * 30 * (speed / 120)) / 10
110     });
111   }, [strength, speed, setObject, maxCarriages]);
112
113   return (
114     <>
115       <Slider
116         min={3000}
117         max={9000}
118         step={1000}
119         value={strength}
120         renderedValue={strength + " kW"}
121         onChange={v => setStrength(v)}
122       >
123         <i className="fas fa-dumbbell"></i> moc
124       </Slider>
125       <InfoRow value={maxCarriages}>
126         <i className="fas fa-weight-hanging"></i> maksymalna liczba wagonów
127       </InfoRow>
128     </>
129   );
130 };

```

BuyEmu

```
RouteDetails.jsx TrainsMenu.jsx M BuyTrain.jsx X
src > Ui > pages > TrainsMenu > BuyTrain.jsx > BuyTrain
136
137 const BuyEmu = ({ speed, setObject }) => {
138   const [segments, setSegments] = useState(4);
139   const [seats, setSeats] = useState(62);
140
141   useEffect(() => {
142     const locomotivePrice = 600;
143     const segmentsPrice = (segments - 2) * 180;
144     const seatingPrice = segments * seats * 15;
145
146     setObject({
147       speed,
148       segments,
149       seats: seats * segments,
150       cost: (locomotivePrice + segmentsPrice + seatingPrice) * (speed / 100)
151     });
152   }, [segments, seats, speed, setObject]);
153
154   return (
155     <>
156       <Slider min={2} max={8} step={1} value={segments} onChange={v => setSegments(v)}>
157         <i className="fas fa-ruler-horizontal"></i> liczba członów
158       </Slider>
159       <Slider min={50} max={82} step={4} value={seats} onChange={v => setSeats(v)}>
160         <i className="fas fa-chair"></i> liczba miejsc siedzących w członie
161       </Slider>
162
163       <InfoRow value={seats * segments}>
164         <i className="fas fa-chair"></i> łączna liczba miejsc
165       </InfoRow>
166     </>
167   );
168 };
169
170 BuyEmu.propTypes = {
171   speed: PropTypes.number,
172   setObject: PropTypes.func
173 };
174
175 export default BuyTrain;
176
```

Efekt w przeglądarce



Przypisywanie tras

```

AssignRoute.jsx
src > Ui > pages > TrainsMenu > AssignRoute.jsx > ...
17 const AssignRoute = observer(() => {
18   const { routeStore, trainStore } = useGameStore();
19   const [selectedRoute, setSelectedRoute] = useState(null);
20   const { trainId } = useParams();
21   const navigate = useNavigate();
22
23   function assignRoute() {
24     trainStore.getTrainById(+trainId).assignRoute(selectedRoute);
25     navigate("/game/trains");
26   }
27
28   useEffect(() => {
29     if (!trainStore.getTrainById(+trainId)) navigate("/game/trains");
30   }, [navigate, trainId, trainStore]);
31
32   useEffect(() => {
33     routeStore.setHighlightedRoute(selectedRoute);
34   }, [routeStore, selectedRoute]);
35
36   useEffect(() => () => routeStore.setHighlightedRoute(null), [routeStore]);
37
38   return (
39     <>
40       <Link to="/game/trains" className="back-button" style={{ float: "left" }}>
41         <i className="fas fa-arrow-left"></i>
42       </Link>
43       <h2>Przypisz trasę</h2>
44       <div className="assign-route-menu" data-style={style}>
45         <Scrollbars>
46           <ul>
47             {routeStore.routes.map(route => (
48               <RouteTile key={route.id} route={route}>
49                 <IconButton
50                   onClick={() => setSelectedRoute(route)}
51                   active={selectedRoute?.id === route.id}
52                   inverted
53                 >
54                   <i className="fas fa-check"></i>
55                 </IconButton>
56               </RouteTile>
57             )));
58           </ul>
59         <Scrollbars>
60           <ElevatedButton onClick={assignRoute} disabled={!selectedRoute}>
61             <i className="fas fa-check"></i> Przypisz trasę
62           </ElevatedButton>
63         </div>
64       </>
65     );
66   );
67 }

```

Przypisania trasy można dokonać po kliknięciu odpowiedniego przycisku przy wybranym pociągu w widoku przeglądu. Wtedy pojawia się lista wszystkich tras. Trasę można wybrać klikając przycisk koło niej – wtedy podświetli się na mapie. W ten sposób użytkownik może się upewnić czy na pewno przypisuje pociąg do dobrej trasy. Faktyczne przypisanie następuje dopiero po kliknięciu przycisku potwierdzającego na dole okienka.

