8기 파이썬반 PJT - 9

Created	@2022년 11월 10일 오후 7:38
Last Edited Time	@2022년 11월 10일 오후 10:14
Type	
Status	
Created By	
Last Edited By	
Stakeholders	

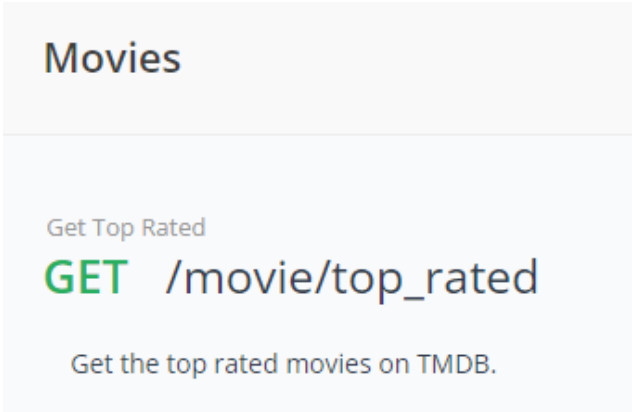
1. 세팅

우선, 제공된 템플릿을 압축해제한 후, `$ npm i axios` 명령어를 입력해 axios 를 포함한 필요한 패키지를 설치한다.

Q. `$ npm i` 를 먼저 하고 그 다음 `$ npm i axios` 를 해야 하지 않나요?

A. 상관없다. 최초 세팅 시 어떤 하나의 패키지를 설치하면, 나머지 `package.json` 에 기록된 모든 것들이 깔린다.

TMDB API 사이트에 접속해, Get Top Rated 사용법을 파악하자.



해당 URL에, TMDB 에서 발급받은 KEY만 제대로 넣어주면 잘 동작할 것이다.

우선, 루트 `/` 디렉터리에 `.env.local` 파일을 생성하고, 다음과 같이 작성한다.

```
VUE_APP_API_KEY=abcdasdfsadgafsasadf
VUE_APP_API_URL=https://api.themoviedb.org/3/movie/top_rated
```

Vue.js 에선 환경변수를 쓰기 위해, 다음과 같은 형태로, 반드시 환경변수 이름을 `VUE_APP` 으로 시작해야만 동작한다.

그리고 `App.vue` 를 다음과 같이 작성한다.

```
<template>
  <div id="app">
    <nav>
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </nav>
    <button v-on:click="test">api test</button>
    <router-view/>
  </div>
</template>

<script>
import axios from "axios";
export default {
  methods: {
    async test() {
      console.log(process.env);
      const API_KEY = process.env.VUE_APP_API_KEY;
      const API_URL = process.env.VUE_APP_API_URL;
      try {
        const response = await axios.get(API_URL, {
          params: {
            api_key: API_KEY,
            language: 'ko-KR',
          }
        })
        console.log(response.data);
      } catch (error) {
        console.log(error);
      }
    }
  }
}
</script>
```

버튼 하나 달고, axios 요청하는게 끝이다. 비동기 통신이기 때문에, `async` `await` 를 빼먹지 않도록 하자.

지정한 환경변수를 가져올 땐, `process.env` 객체에 접근한다.

단, TMDB API 사용 시엔, 다음 형태로, `params` 객체에 `api_key` 와 `language` 를 정해주어야한다.

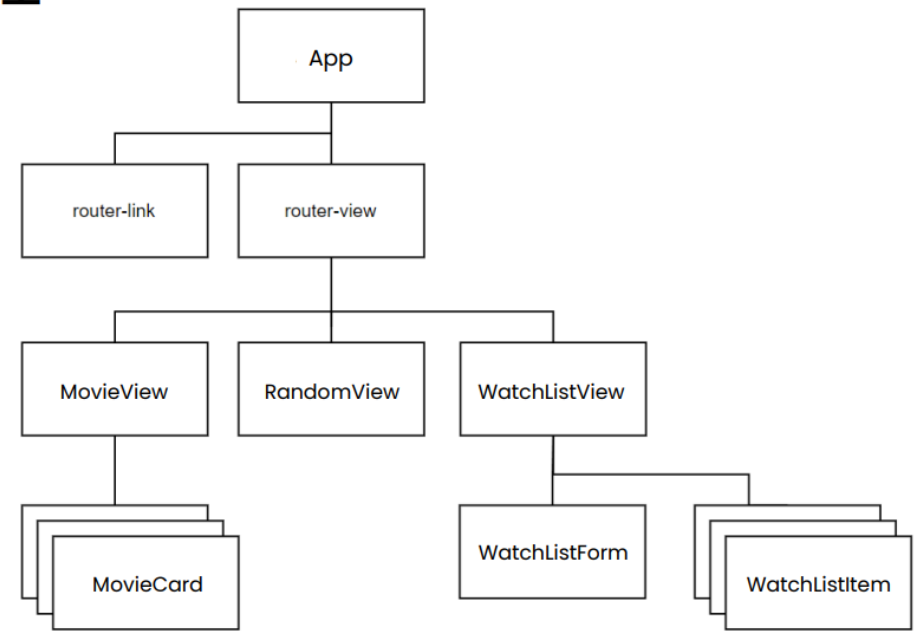
결과는 다음과 같다.

테스트가 끝났으므로, `App.vue` 는 원래 형태로 돌려놓도록한다.

```
<template>
  <div id="app">
    <nav>
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </nav>
    <router-view/>
  </div>
</template>
```

명세에 작성된대로 컴포넌트를 미리 만들어두자.

컴포넌트 구조



router view

Path	Component	Description
/movies	MovieView	전체 영화 목록 페이지
/random	RandomView	랜덤 영화 출력 페이지
/watch-list	WatchListView	영화 검색 출력 페이지

`App.vue` 는 다음과 같다. 각각의 경로로 향하는 링크를 달았다.

```
<template>
  <div id="app">
    <nav>
      <router-link to="/movies">Movies</router-link> |
      <router-link to="/random">Random</router-link> |
      <router-link to="/watch-list">Watch List</router-link> |
    </nav>
    <router-view/>
  </div>
</template>
```

`router/index.js` 는 다음과 같다.

```
import Vue from "vue";
import VueRouter from "vue-router";
import MovieView from "../views/MovieView.vue";
import RandomView from "../views/RandomView.vue";
import WatchListView from "../views/WatchListView.vue";

Vue.use(VueRouter);

const routes = [
  {
    path: "/movies",
    name: "movies",
    component: MovieView,
  },
  {
    path: "/random",
    name: "random",
    component: RandomView,
  },
  {
    path: "/watch-list",
    name: "watchlist",
    component: WatchListView,
  },
];

const router = new VueRouter({
  mode: "history",
  base: process.env.BASE_URL,
  routes,
});

export default router;
```

컴포넌트는 기본적으로 다음 형태를 가진다.

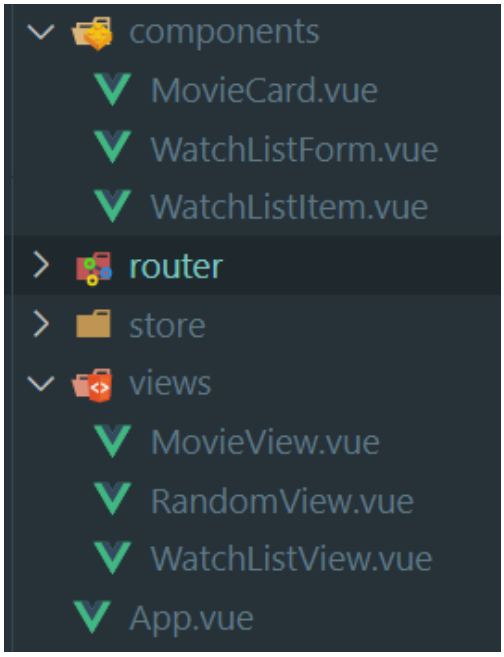
```
<template>
  <div>
    <h1>컴포넌트 이름</h1>
  </div>
</template>
```

만약, 자식 컴포넌트가 있다면? 다음과 같은 형태를 가진다. `MovieView` 라우트 컴포넌트의 예시다.

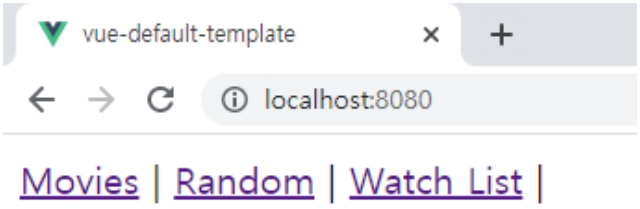
```
<template>
  <div>
    <h1>MovieView</h1>
    <MovieCard />
  </div>
</template>

<script>
import MovieCard from "../components/MovieCard.vue";
export default {
  components: { MovieCard },
```

```
};
</script>
```

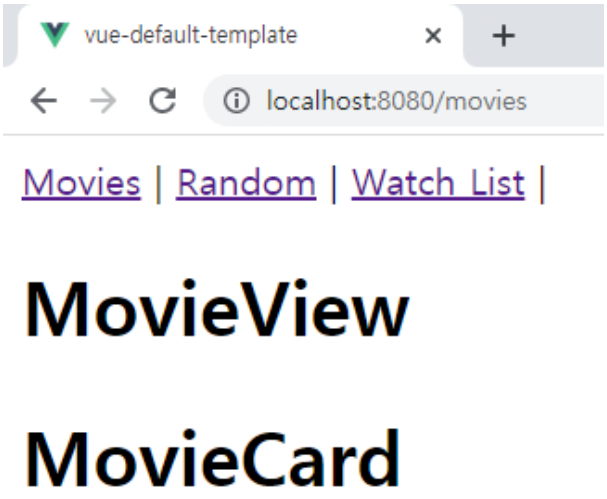
이를 기반으로 컴포넌트 트리를 구현해보니, 다음과 같았다.



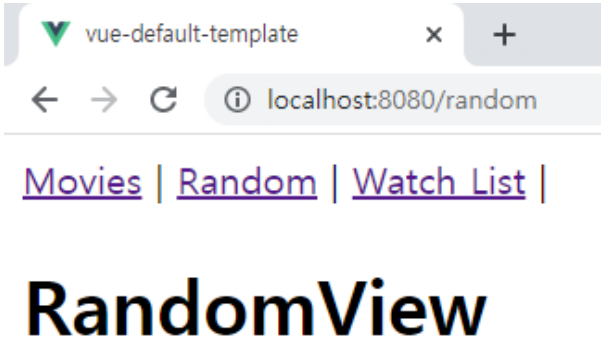
화면 테스트해보자. 최초 접속시엔 아래와 같다.



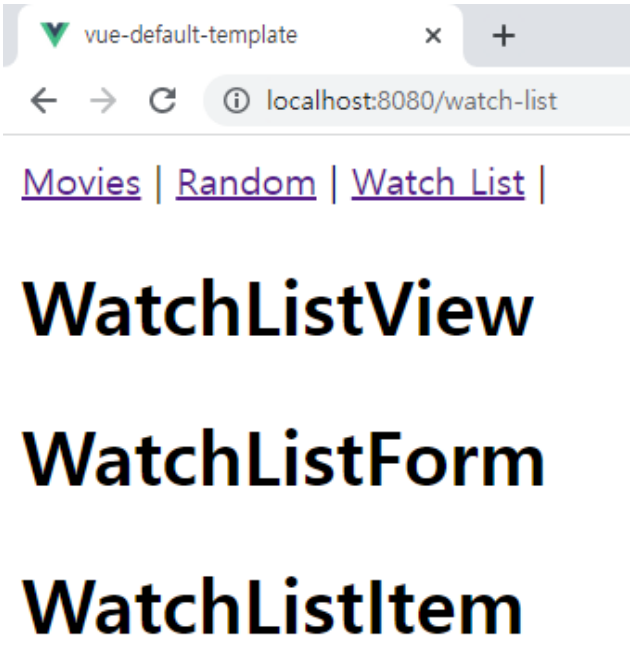
`/movies` 접속 시엔 아래와 같다. 자식 컴포넌트로 `MovieCard` 를 가진다.



`/random` 접속 시엔 아래와 같다.



`/watch-list` 접속 시엔 다음과 같다. 이 경우, 두 개의 자식 컴포넌트인 `WatchListForm` 과, `WatchListItem` 을 가진다.

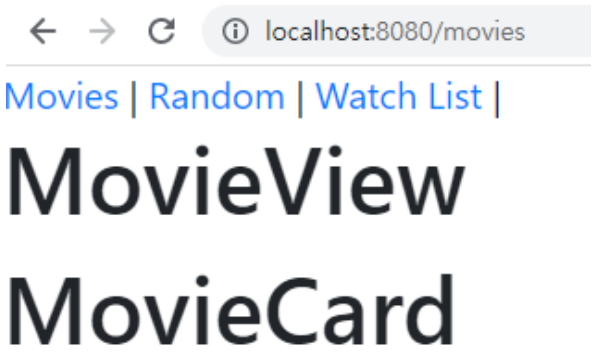


`bootstrap-vue` 를 사용할 예정이므로, 미리 설치해두자.

```
$ npm i bootstrap@4.6.1 bootstrap-vue
```

사용을 위해, `src/main.js` 에 다음을 추가하자.

```
import { BootstrapVue } from 'bootstrap-vue'
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'
Vue.use(BootstrapVue)
```



폰트가 변경된 것을 보아, `BootstrapVue` 가 잘 적용된 것이 확인된다.

2. `/movies`

최고 평점 영화를 출력하는 페이지다.

우선, `MovieView.vue` 는 다음과 같다.

```
<template>
  <div>
    <ul>
      <li v-for="movie in movies" v-bind:key="movie.id">
        <MovieCard
          v-bind:title="movie.title"
          v-bind:posterPath="movie.poster_path"
          v-bind:overview="movie.overview"
        />
      </li>
    </ul>
  </div>
</template>

<script>
import axios from "axios";
import MovieCard from "../components/MovieCard.vue";
export default {
  components: { MovieCard },
  data() {
    return {
      movies: [],
    };
  },
  async created() {
    const API_KEY = process.env.VUE_APP_API_KEY;
    const API_URL = process.env.VUE_APP_API_URL;
    try {
      const response = await axios.get(API_URL, {
        params: {
          api_key: API_KEY,
          language: "ko-KR",
        },
      });
      this.movies = response.data.results;
    } catch (error) {
      console.log(error);
    }
  },
};
</script>
```

핵심은, `v-for` 를 사용하되, 특정한 `props` 를 내려보낸다는 것이다. 단, 기준이 되는 건 항상 받아온 데이터이므로, 들어온 JSON 양식이 어떤 것인지, 나에게 무엇이 필요한지, 어떻게 접근해야 아는지 그 접근법에 대해 항상 고민해야 한다.

다음, `MovieCard.vue` 는 다음과 같다.

```
<template>
  <div>
    <div>{{ title }}</div>
    <img v-bind:src="movieImgURL" alt="movieImg" height="300" />
    <div>{{ overview }}</div>
  </div>
</template>

<script>
export default {
  props: ["title", "posterPath", "overview"],
  computed: {
    movieImgURL() {
      return `https://themoviedb.org/t/p/w600_and_h900_bestv2${this.posterPath}`;
    },
  },
};
</script>
```

`props` 로 받고, `<img src="">` 부분은 전처리가 필요하기 때문에, 화면에 붙이기 전 `computed` 를 사용했다.

해당 영화 이미지에 접근하려면, TMDB 사이트를 참고한 결과,

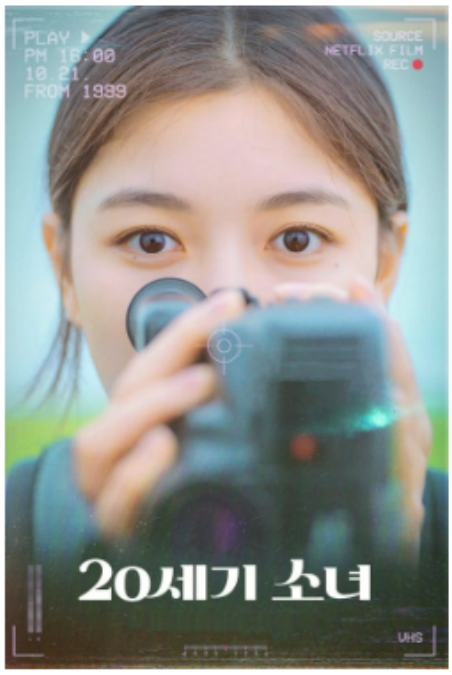
`https://themoviedb.org/t/p/w600_and_h900_bestv2`

이 경로에 접근하면 된다.

결과만 보자.

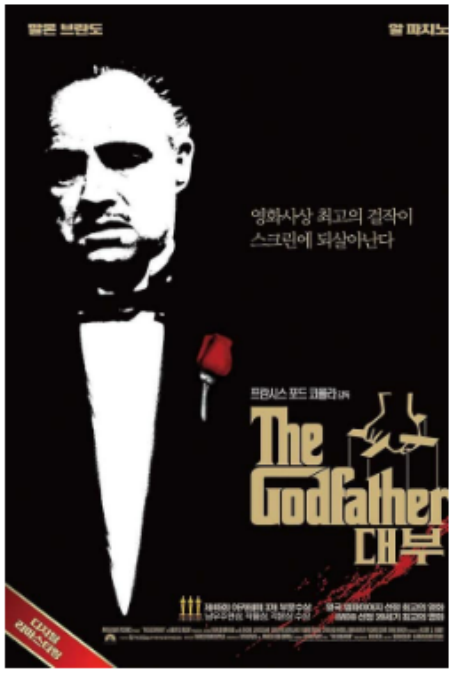
Movies | Random | Watch List |

- 20세기 소녀



보라의 둘도 없는 친구 연두는 심장 수술을 위해 미국에
다. 하지만 보라의 서투른 계획은 예상치 못한 방향으로

- 대부



시실리에서 이민온 뒤, 정치권까지 영향력을 미치는 거물
리아 패밀리와 손잡고 새로운 마약 사업을 제안한다. 돈

맘에 안 드는 게 있다면, 리스트에 찍힌 점과, 리스트의 들여쓰기다. 없애주자.

`App.vue` 로 가서, 전역 스타일링한다.

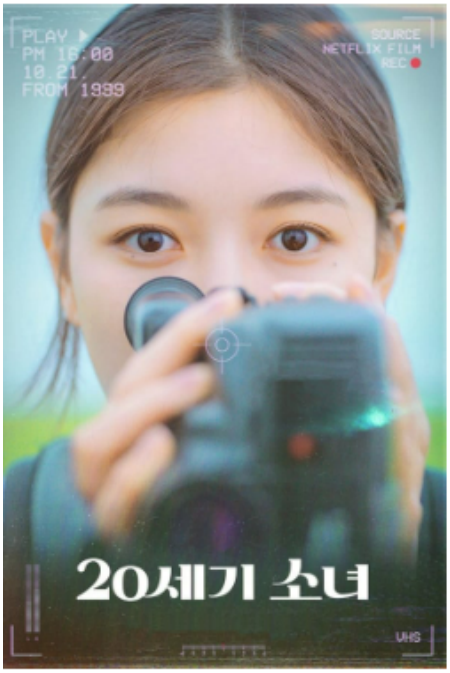
```
<style>
* {
  list-style: none;
  padding-left: 0;
}
</style>
```

전역 스타일링시엔 `scoped` 가 없어야 한다.

결과는 다음과 같다.

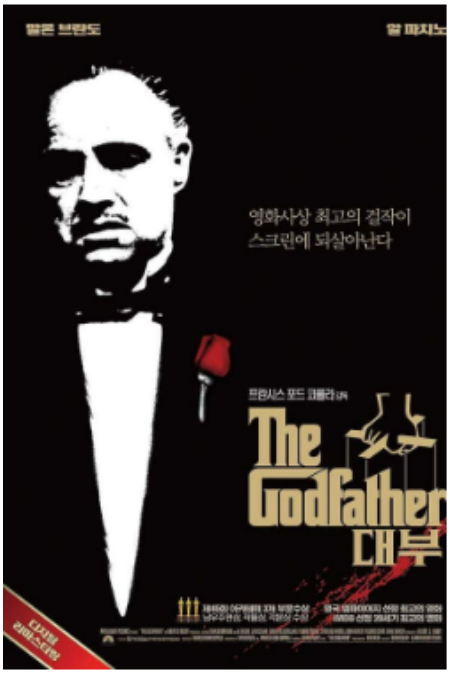
Movies | Random | Watch List |

20세기 소녀



보라의 둘도 없는 친구 연두는 심장 수술을 위해 미국에 가
지만 보라의 서투른 계획은 예상치 못한 방향으로 전개된다

대부



시실리에서 이민온 뒤, 정치권까지 영향력을 미치는 거물로
밀리와 손잡고 새로운 마약 사업을 제안한다. 돈 꼴레오네

3. `/random`

편리한 랜덤 사용을 위해 `lodash` 를 설치하자.

`$ npm i lodash`

```
<template>
  <div>
    <b-button variant="success" v-on:click="pickMovie">PICK</b-button>
    <div v-if="selectedMovie.title">
      <img v-bind:src="movieImgURL" alt="movieImg" height="300" />
      <div>{{ selectedMovie.title }}</div>
    </div>
  </div>
</template>

<script>
import axios from "axios"
import _ from "lodash"
export default {
  data() {
    return {
      movies: [],
      selectedMovie: {},
    }
  },
  computed: {
    movieImgURL() {
      return `https://themoviedb.org/t/p/w600_and_h900_bestv2${this.selectedMovie.poster_path}`;
    }
  },
  async created() {
    const API_KEY = process.env.VUE_APP_API_KEY;
    const API_URL = process.env.VUE_APP_API_URL;
    try {
      const response = await axios.get(API_URL, {
        params: {
          api_key: API_KEY,
          language: "ko-KR",
        },
      });
      this.movies = response.data.results;
    } catch (error) {
      console.log(error);
    }
  },
  methods: {
    pickMovie() {
      const n = _.random(this.movies.length - 1);
      this.selectedMovie = this.movies[n];
    }
  }
};
</script>
```

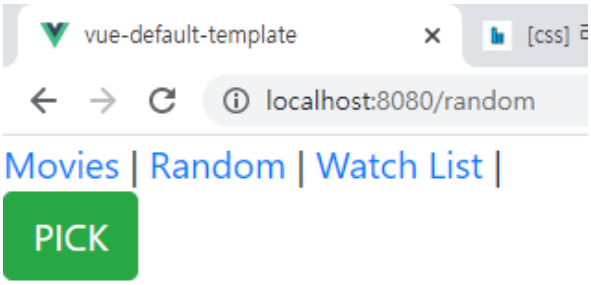
일단, BootstrapVue 중, 문서를 참고해 `<b-button>` 하나를 달았고, 클릭하면 `pickMovie` 메서드가 동작하도록 처리했다.

사용자가 버튼을 클릭해야만 영화가 나와야하는데, 이를 위해 `v-if` 를 사용했고, `selectedMovie.title` 이 존재할때만 화면이 보이도록 처리했다.

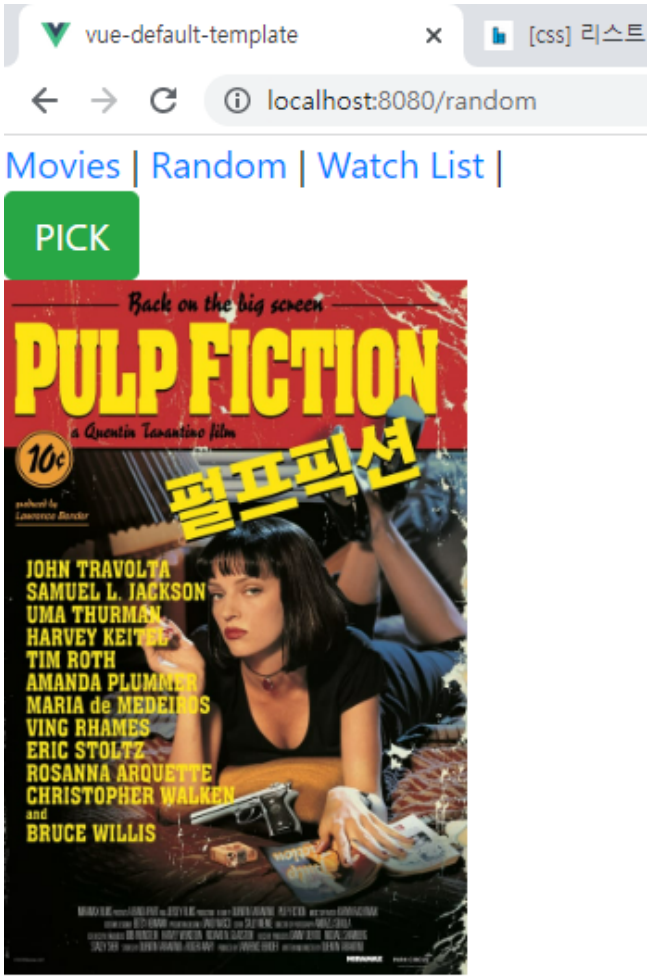
우선, `movies` 는 맨 처음 빈 배열 `[]` 이고, `selectedMovie` 는 빈 객체 `{}` 이다. 즉, 데이터를 받아온 후, 사용자가 `pickMovie` 를 호출하면, 20개의 영화 리스트 중 하나가 뽑히고, `selectedMovie` 에 담길 것이다.

`pickMovie` 메서드를 자세히 보면, lodash 의 `random` 메서드를 사용했고, 랜덤한 수 하나를 받아서 `movies` 중 하나를 `selectedMovie` 로 선택한다.

결과는 다음과 같다.



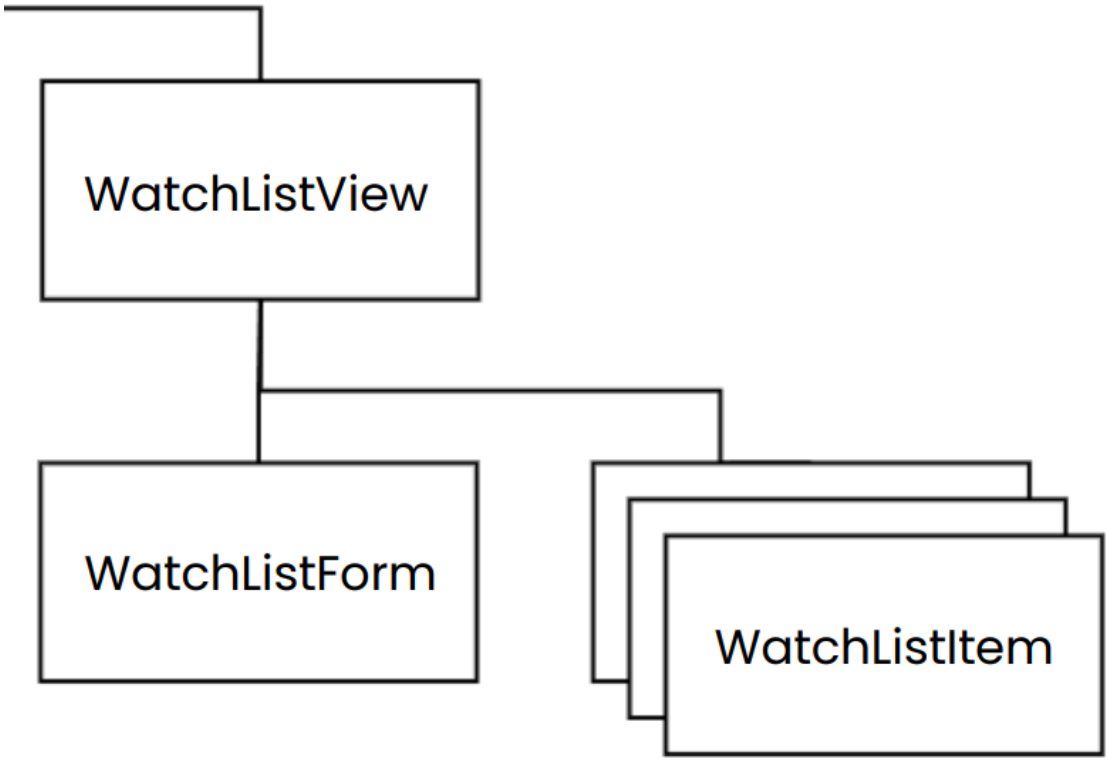
첫 화면이고, 여기서 버튼을 누르면?



펄프 픽션

랜덤한 영화가 등장한다. 버튼을 계속 눌러서 테스트해보자.

4. `/watch-list`



이건 좀 어렵다. 먼저, 자식 컴포넌트인 `WatchListForm.vue` 부터 보자.

```
<template>
  <div>
    <h1>보고싶은 영화</h1>
    <input type="text" v-model="userInput">
    <b-button v-on:click="addMovie">Add</b-button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      userInput: "",
    }
  },
  methods: {
    addMovie() {
      this.$emit("addMovie", this.userInput);
      this.userInput = "";
    }
  }
}
</script>
```

`<input>` 을 보면? 무조건이다. `v-model` 을 써야한다.

사용자가 입력을 하고, Add 버튼을 누르면 `addMovie` 가 작동되도록 했다.

그리고, 상위 컴포넌트에서 처리해야한다. 왜냐면, 해당 데이터를 형제 컴포넌트라 할 수 있는 `WatchListItem.vue` 로 보내야하기 때문이다. 따라서 `$emit` 을 사용했고, 사용자가 입력한 값은 지워주도록 처리했다.

다음, `WatchListItem.vue` 다.

```
<template>
  <div>
    <div v-on:click="deleteMovie">{{ movie }}</div>
  </div>
</template>

<script>
export default {
  props: ["movie"],
  methods: {
    deleteMovie() {
      this.$emit("deleteMovie", this.movie);
    }
  }
}
</script>
```

핵심은, 누를 때 지워진다는 것이다. 교안엔 취소선으로 되어있는데, 클릭하면 사라지게 하는 게 훨씬 난이도가 쉽다.

이건, 상위 컴포넌트에서 단일 영화 제목을 의미하는 movie 를 받는 `props` 가 하나 존재하며, `deleteMovie` 메서드 실행 시 역시 부모로 `$emit` 을 보낸다.

마지막으로, `WatchListView.vue` 이다.

```
<template>
  <div>
    <WatchListForm v-on:addMovie="addMovie"/>
    <ul>
      <li v-for="(movie, index) in movies" v-bind:key="index">
        <WatchListItem v-bind:movie="movie" v-on:deleteMovie="deleteMovie"/>
      </li>
    </ul>
  </div>
</template>

<script>
import WatchListForm from "../components/WatchListForm.vue";
import WatchListItem from "../components/WatchListItem.vue";
export default {
  components: {
    WatchListForm,
    WatchListItem,
  },
  data() {
    return {
```

```
      movies: [],
    }
  },
  methods: {
    addMovie(userInput) {
      this.movies.push(userInput);
    },
    deleteMovie(movie) {
      const index = this.movies.indexOf(movie);
      if (index > -1) {
        this.movies.splice(index, 1);
      }
    }
  }
};
</script>
```

반복을 실행해야 하므로 `v-for` 를 사용했다. 각각, `props` , `emit` 에 따라 `v-bind` 와 `v-on` 을 어디에 썼는지 잘 살펴보자.

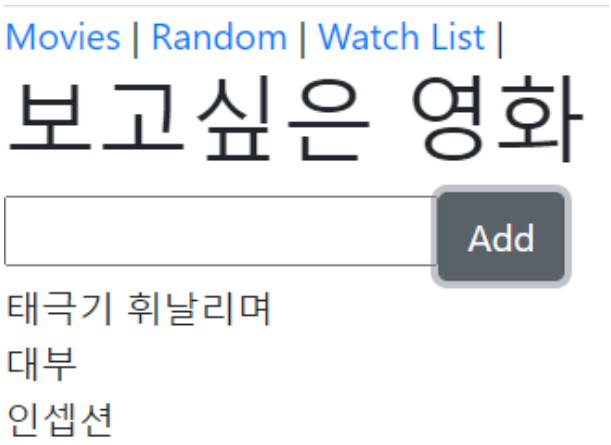
핵심은 상태 `movies` 이다. `addMovie` 메서드로 사용자 입력을 받아서 배열에 추가하고, `deleteMovie` 메서드로 사용자가 선택한 영화를 배열에서 삭제한다.

`deleteMovie` 구현은 짧지만 만만하지 않다.

일단, 사용자가 입력한 값이 몇 번째 인덱스에 있는지를 찾아야 한다. 이에, `indexOf` 메서드를 사용해서 인덱스를 받았다. `indexOf` 는 찾는 걸 실패하면 -1 을 리턴하는 특성이 있다. 즉, 찾기 성공했을 때에만, `splice` 메서드를 사용한다.

`splice` 메서드는 원본 배열에 새 값을 추가하거나 삭제할 때 사용한다. 지금 파라미터는, 현재 인덱스부터 시작해, 하나의 엘리먼트를 제거하라는 뜻이다. 즉, 사용자가 입력한 값 삭제가 된다.

결과를 확인해보자.



사용자가 입력한 값에 따라 추가되고, 해당 영화 이름을 클릭했을 때 삭제되는것을 확인할 수 있다.

도전과제

1. BootstrapVue 와 CSS 를 사용해 스타일링
2. 날씨에 따른 영화 추천 알고리즘 설계 (Open Weather API)