ITI0205: Veebirakendused

14. REST API ehitamine ExpressJS raamistikule

Martin Verrev
martin.verrev@taltech.ee

{REST:API}

Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the internet. They allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

Roy Fielding

Aastal 2000 - doktoritöö

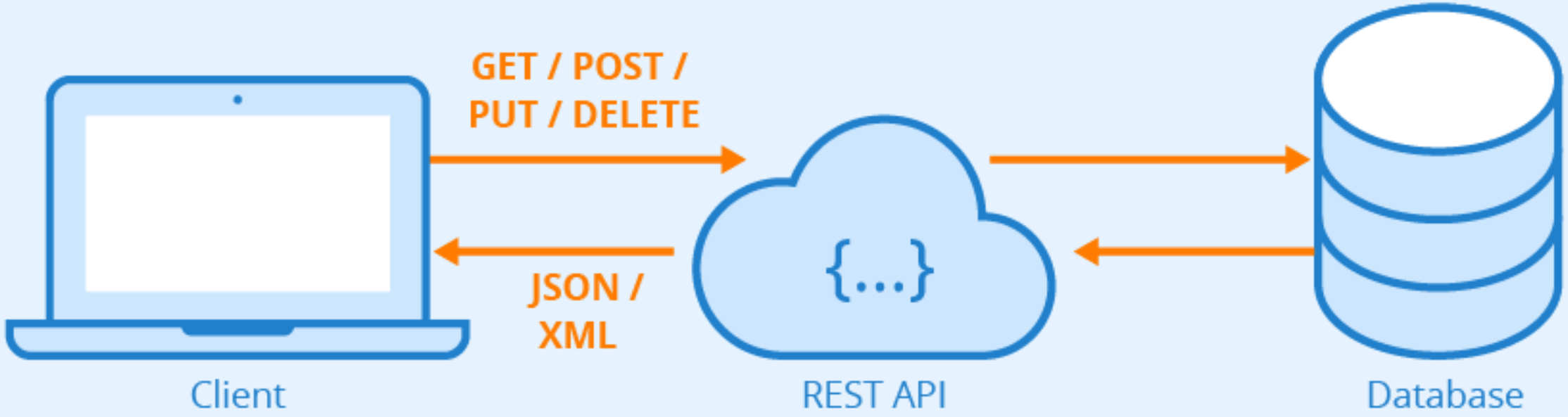
"Architectural Styles and the Design of Network-based Software Architectures"

The six principles of REST

- **Client-server architecture**: Separation of problems. Dividing UI from data storage improves the portability of that interface across multiple platforms. Components can be developed independently from each other.
- **Statelessness**: Communication between client and server always contains all the information needed to execute the request. There is no session state on the server.
- **Caching**: Any component can cache all resources to improve performance.

The six principles of REST

- **Uniform interface**: All components of a API have to follow the same rules to communicate with each other. This makes it easier to understand interactions between the various components of a system.
- **Layered system**: Individual components cannot see beyond the level they interact with. Therefore, components can be easily exchanged or expanded independently of each other.
- **Code-on-demand**: Additional code can be downloaded to extend client functionality. However, this is optional as the client may not be able to download or execute this code.


GET / POST /
PUT / DELETE
JSON /
XML
{...}
Client
REST API
Database

Express.js, or simply **Express**, is a back end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js

https://expressjs.com/

Installeerimine

Otse NPM paketihaldurist:

```
$ npm i express
```

Kasutades Express Generator'it

```
$ npm install -g express-generator
$ express
```

Express Server

Ruutimine

```
app.METHOD(PATH, HANDLER)
```

Näide:

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

Pöördumine rakendusest serveri poole

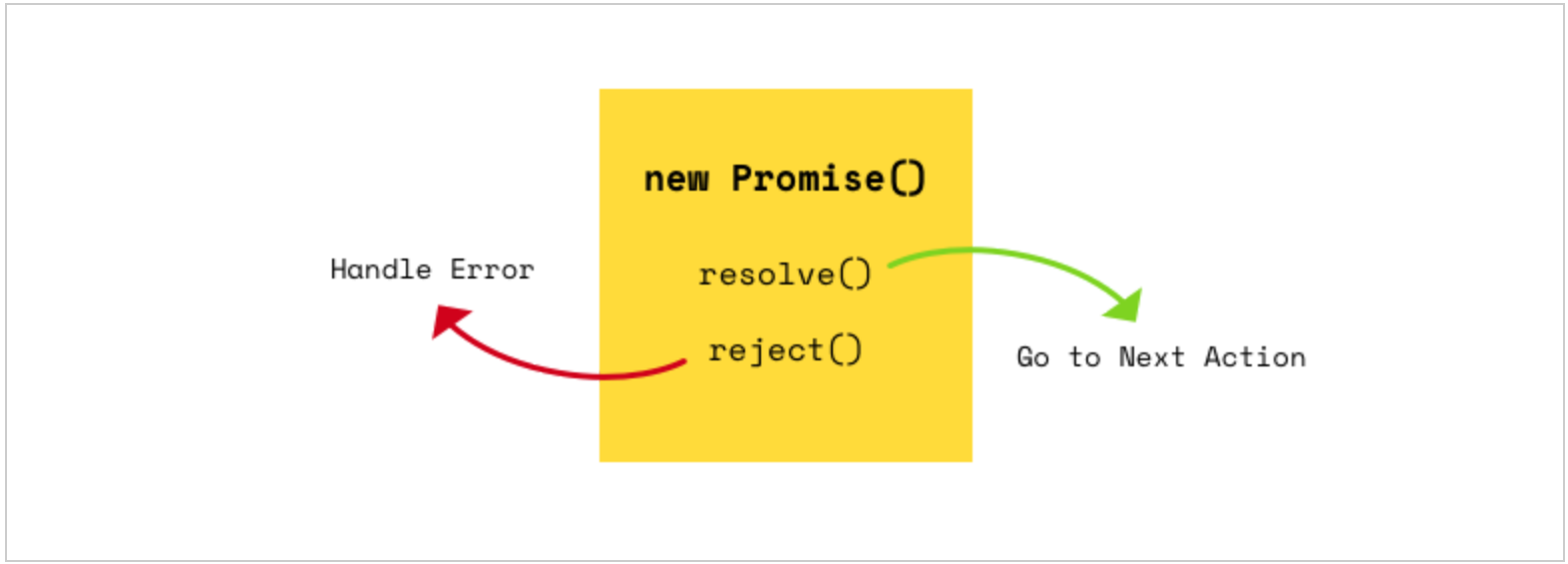
`fetch()` allows you to make network requests similar to XMLHttpRequest (XHR). The main difference is that the Fetch API uses Promises, which enables a simpler and cleaner API, avoiding callback hell and having to remember the complex API of XMLHttpRequest.

```
    fetch(url)
        .then(response => response.json())
        .then(data => {
            //Do something
        })
        .catch(err => console.error(err));
```

Promises

A **Promise** is an object representing the eventual completion or failure of an asynchronous operation. Essentially, a promise is a returned object to which you attach callbacks, instead of passing callbacks into a function.

```
const promise = doSomething();
promise.then(successCallback, failureCallback);
```



Katsetame :)

Tänan!

Viieid

- What is REST: https://restfulapi.net/
- Understanding And Using REST APIs: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/
- Node Filesystem Module: https://nodejs.dev/learn/the-nodejs-fs-module
- Fetch API: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- A simple guide to ES6 promises: https://codeburst.io/a-simple-guide-to-es6-promises-d71bacd2e13a