Week 1 Server side

REST & Test

SAXION HOGESCHOOL

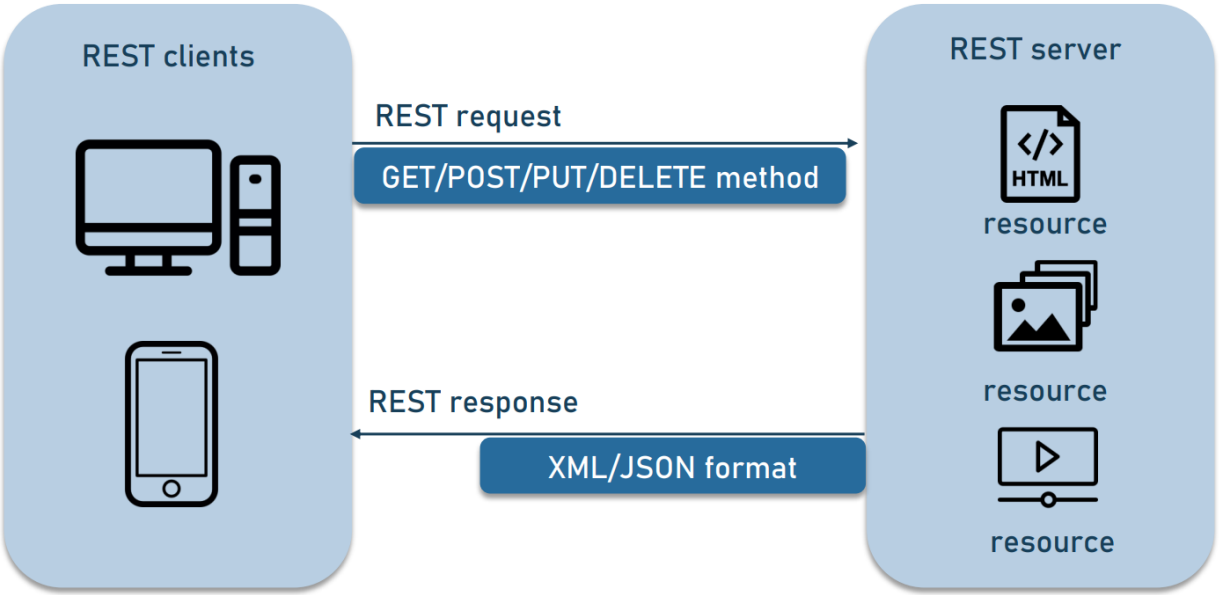
Contents

- Short recap of REST
- Rest specifications with OpenAPI/Swagger
- Automated testing

REST RECAP

What is REST

- REST = Representational State Transfer
- A software architecture style for web services
- Based on standard HTTP methods:
  - GET, POST, PUT, DELETE, etc.
- Stateless, scalable, and cacheable



REST uses: **Resources** HTTP verbs Status codes Headers

Resources

Grouped information you allow your users to interact with.

Examples: bikes, users, posts, etc…

Resource can be used through url's

i.e. `http://localhost:3000/users` → Endpoint

REST uses: **Resources** HTTP verbs Status codes Headers

Resources

Endpoint checklist

- Resource names are **plural** (`bikes` not `bike`, `users` not `user`)
- Endpoint to not imply actions (no terms like `filter`, `search`, `login`, `addBike`)
- Endpoints can be nested `/buildings/1/rooms/2`
- Filtering should be done using query parameters `/buildings?stories=4`

Proper REST is part of the grading rubrics and impacts your grade!

HTTP Verbs

Determine what is to be done with a resource.

GET	fetch a resource from the server
POST	Create a new resource
PUT/PATCH	Edit a resource on the server
DELETE	Remove a resource from the server

There are others, you can check them out at https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Status codes

The server sends back a code to indicate to the client what happened. They are grouped into five ranges:

`1xx`	Informational
`2xx`	Success
`3xx`	Redirection
`4xx`	Client error
`5xx`	Server error

More info at https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Headers

Headers provide information about the actual message being sent. Both server and client send headers.

Most often you will want the server to know the content you are sending (`Content-Type`) and what you want the server to return (`Accept`).

You are allowed to create custom headers so the applications are limitless.

More info at https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

SWAGGER / OPENAPI

```yaml
openapi: 3.1.0
info:
  title: Swagger API
  version: 1.0.0
servers:
  - url: https://api.swagger.io
paths:
  /resources:
    get:
      tags:
        - API Resources
      summary: Retrieve all resources based on filter criteria
      description: Fetches a list of all available resources.
      responses:
        '200':
          description: A
          content:
            application/j
              schema:
                type: arr
                items:
                  $ref: '
    post:
      tags:
        - API Resources
      summary: Create a n
      description: Create
      requestBody:
        required: true
        content:
          application/jso
            schema:
              $ref: '#/co
```

Servers

https://api.swagger.io

API Resources

GET	/resources	Retrieve all resources based on filter criteria
POST	/resources	Create a new resource
GET	/resources/{resourceId}	Retrieve a resource
PUT	/resources/{resourceId}	Update a resource
PATCH	/resources/{resourceId}	Partially update a resource
DELETE	/resources/{resourceId}	Delete a resource
OPTIONS	/resources/{resourceId}	Retrieve options
HEAD	/resources/{resourceId}	Check resource presence

SAXION HOGESCHOOL

Swagger / OpenAPI

- OpenAPI is a specification for describing REST APIs
- Swagger is a set of tools built around OpenAPI

- Used for:
  - API documentation
  - Client/server code generation
  - API testing and mocking

- There are multiple ways to describe a REST API, we will use the **JSDoc** option for this.
- We will mainly use Swagger during this course for API documentation!

- Your project(s) will have an endpoint **/api-docs** that will show a website with the full specification of your REST API.

Documenting your routes with JSDoc

- JSDoc is a documentation standard for JavaScript.
- It allows you to write structured comments directly in your code.
- You can combine JSDoc comments with Swagger/OpenAPI annotations using the **swagger-jsdoc** package.

- You will need to use the @openapi annotation in your JSDoc to let swagger know that you are describing a REST API.

Example: Documenting your routes with JSDoc

```
/**
 * @openapi
 * /users:
 *   get:
 *     summary: Retrieve a list of users
 *     description: Returns an array of user objects with `id` and `email`.
 *     tags:
 *       - Users
 *     responses:
 *       200:
 *         description: A list of users
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 type: object
 *                 properties:
 *                   id:
 *                     type: integer
 *                     example: 1
 *                   email:
 *                     type: string
 *                     format: email
 *                     example: "user@example.com"
 */
app.get('/users', (req, res) => {
  const users = [
    { id: 1, email: 'user1@example.com' },
    { id: 2, email: 'user2@example.com' },
  ];

  res.status(200).json(users);
});
```

TEST REST SPECIFICATIONS

Automated testing

- There are multiple tools available for testing your REST API, in this course we are using the combination of Vitest and Supertest.
  - Vitest is a fast unit test framework for vite and/or Node.js projects
  - Supertest is a library for test HTTP APIs by sending real requests to a server.

- Vitest provides the test runner, supertest handles the HTTP calls.

- You run your tests with **npm run test**

Example: Automated testing

```
import request from 'supertest';
import { describe, it, expect } from 'vitest';
import app from '../src/app'; // Import your express app

// describe = group with multiple tests, e.g. for one endpoint
describe('GET /users', () => {

  // it = a single test case within a group
  it('should return a list of users', async () => {
    const res = await request(app).get('/users');
    expect(res.status).toBe(200);
    expect(Array.isArray(res.body)).toBe(true);
    expect(res.body[0]).toHaveProperty('email');
  });
});
```

Questions?

Assignment:

Determine the REST specification