CS 4530: Fundamentals of Software Engineering

Module 10.3 Building REST APIs

Adeel Bhutta, Jan Vitek and Mitch Wand

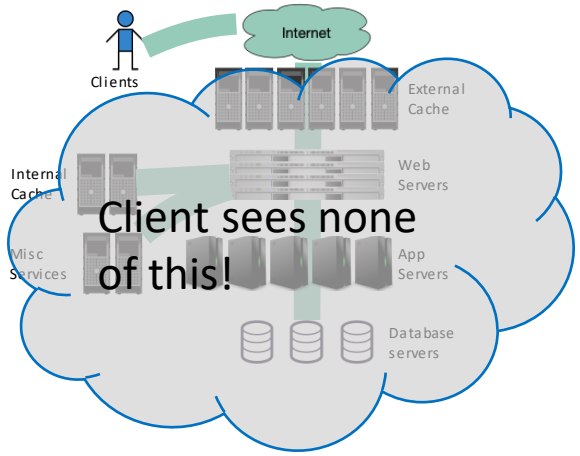
Khoury College of Computer Sciences

REST: Representational State Transfer

- A design principle for http requests
- Commonly used for APIs

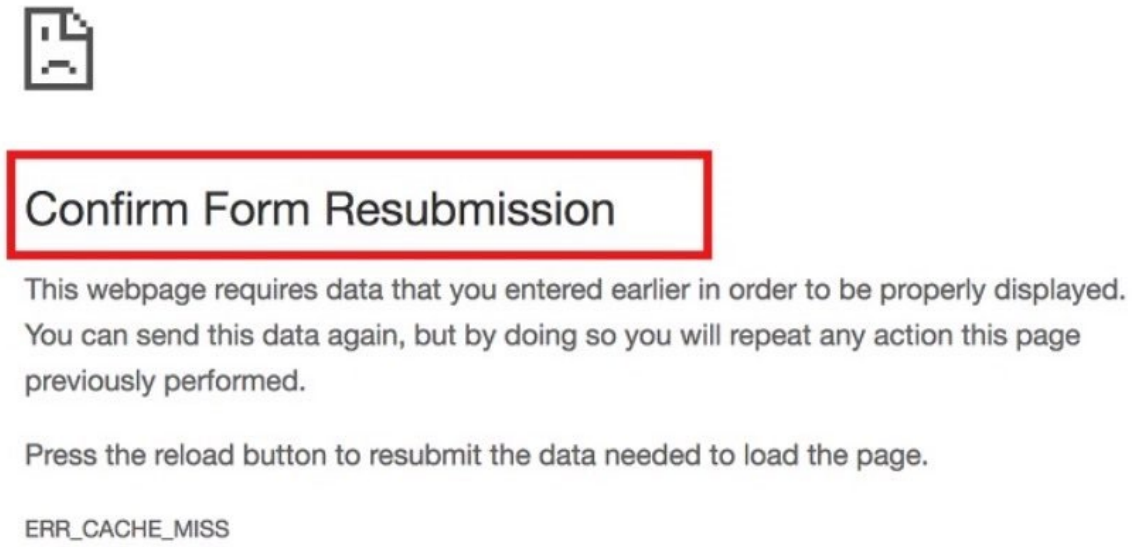
REST Principles

- Single Server - As far as the client knows, there's just one
- Stateless - Each request contains enough information that a different server could process it (if there were multiple…)
- Uniform Cacheability - Each request is identified as cacheable or not.
- Uniform Interface - Standard way to specify interface



“Not cacheable” means that it must be executed exactly once per user request.

- For example, POST is typically not cacheable



Uniform Interface: Nouns are represented as URIs

- In a RESTful system, the server is visualized as a store of resources (nouns), each of which has some data associated with it.
- A URI represents such a resource.

Examples

- Examples:
  - `/cities/losangeles`
  - `/transcripts/00345/graduate` (student 00345 has several transcripts in the system; this is the graduate one)
- Anti-examples:
  - `/getCity/losangeles`
  - `/getCitybyID/50654`
  - `/Cities.php?id=50654`

We prefer plural nouns for toplevel resources, as you see here.

Useful heuristic: if you were keeping this data in a bunch of files, what would the directory structure look like?
But you don't have to actually keep the data in that way.

Uniform Interface: Verbs are represented as http methods

- In REST, there are four things you can do with a resource
- POST: requests the server to create a resource
  - there are several ways in which the value for the new resource can be transmitted (more in a minute)
- GET: requests the server to respond with a representation of the resource
- PUT: requests the server to replace the value of the resource by the given value
- DELETE: requests the server to delete the resource

Path parameters specify portions of the path to the resource

For example, your REST protocol might allow a path like

`/transcripts/00345/graduate`

In a REST protocol, this API might be described as

`/transcripts/:studentid/graduate`

`:studentid` is a path parameter, which is replaced by the value of the parameter

Query parameters allow named parameters

Example:

`/transcripts/graduate?lastname=covey&firstname=avery`

These are typically used to specify more flexible queries, or to embed information about the sender's state, eg

https://calendar.google.com/calendar/u/0/r/month/2023/2/1?tab=mc&pli=1

This URI combines path parameters for the month and date, and query parameters for the format (tab and pli).

You can also put parameters in the body.

- You can put additional parameters or information in the body, using any coding that you like.

Example interface #1: a todo-list manager

- Resource: /todos
  - GET /todos - get list all of my todo items
  - POST /todos - create a new todo item (data in body; returns ID number of the new item)
- Resource: /todos/:todoItemID
  - :todoItemID is a path parameter
  - GET /todos/:todoItemID - fetch a single item by id
  - PUT /todos/:todoItemID - update a single item (new data in body)
  - DELETE /todos/:todoItemID - delete a single item

Example interface #2: the transcript database

```
POST /transcripts
 -- adds a new student to the database,
 -- returns an ID for this student.
 -- requires a body parameter 'name', url-encoded (eg name=avery)
 -- Multiple students may have the same name.
GET  /transcripts/:ID
 -- returns transcript for student with given ID.  Fails if no such student
DELETE /transcripts/:ID
 -- deletes transcript for student with the given ID, fails if no such student
POST /transcripts/:studentID/:courseNumber
 -- adds an entry in this student's transcript with given name and course.
 -- Requires a body parameter 'grade'.
 -- Fails if there is already an entry for this course in the student's transcript
GET  /transcripts/:studentID/:courseNumber
 -- returns the student's grade in the specified course.
 -- Fails if student or course is missing.
GET  /studentids?name=string
 -- returns list of IDs for student with the given name
```

Remember the heuristic: if you were keeping this data in a bunch of files, what would the directory structure look like?

Didn't seem to fit the model, sorry

It would be better to have a machine-readable specification

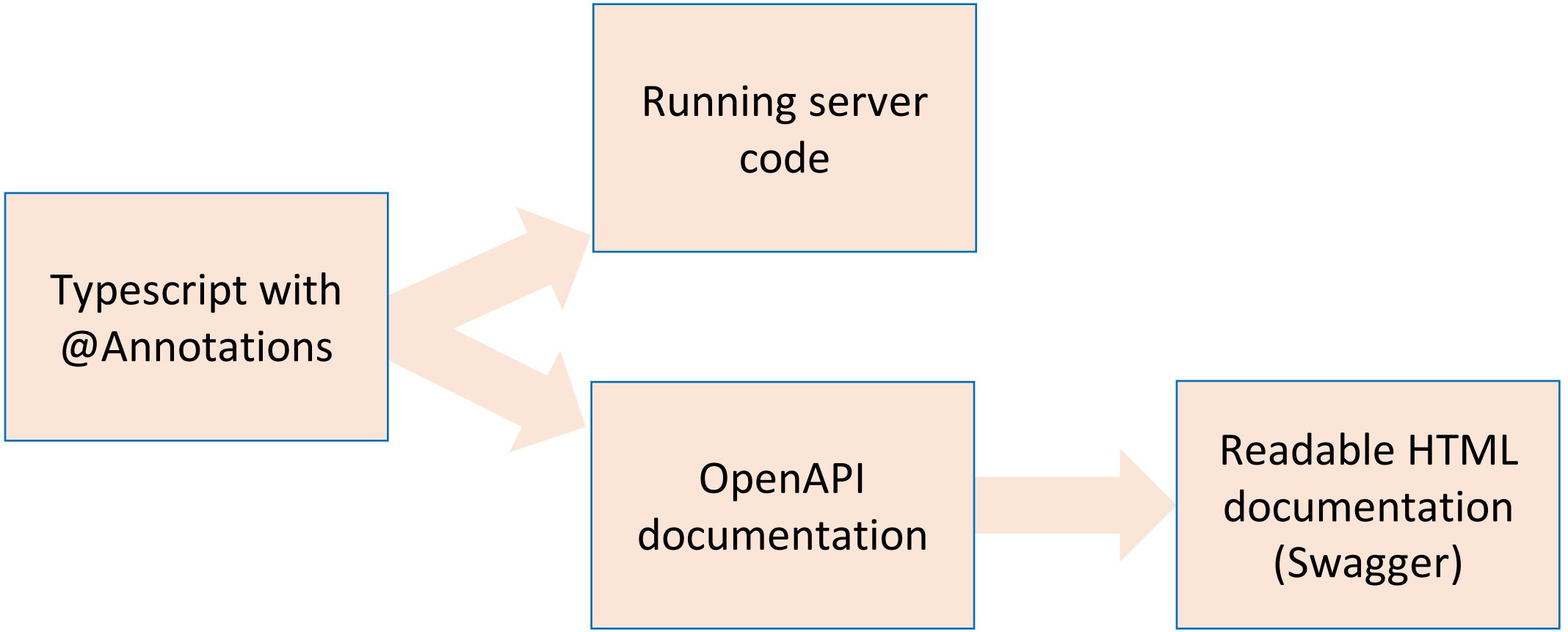
- The specification of the transcript API on the last slide is RESTful, but is not machine-readable
- A machine-readable specification is useful for:
  - Automatically generating client and server boilerplate, documentation, examples
  - Tracking how an API evolves over time
  - Ensuring that there are no misunderstandings

OpenAPI is a machine-readable specification language for REST

- Written in YAML
- Not really convenient for human use
- Better: use a tool!

```
/towns/{townID}/viewingArea:
post:
  operationId: CreateViewingArea
responses:
  '204':
description: No content
'400':
description: Invalid values specified
content:
  application/json:
schema:
  $ref: '#/components/schemas/InvalidParametersError'
description: Creates a viewing area in a given town
tags:
  - towns
security: []
parameters:
  - description: ID of the town in which to create the new viewing area
in: path
name: townID
required: true
schema:
  type: string
  - description: |-
  session token of the player making the request, must
match the session token returned when the player joined the town
in: header
name: X-Session-Token
required: true
schema:
  type: string
requestBody:
  description: The new viewing area to create
required: true
content:
  application/json:
schema:
  $ref: '#/components/schemas/ViewingArea'
description: The new viewing area to create
```

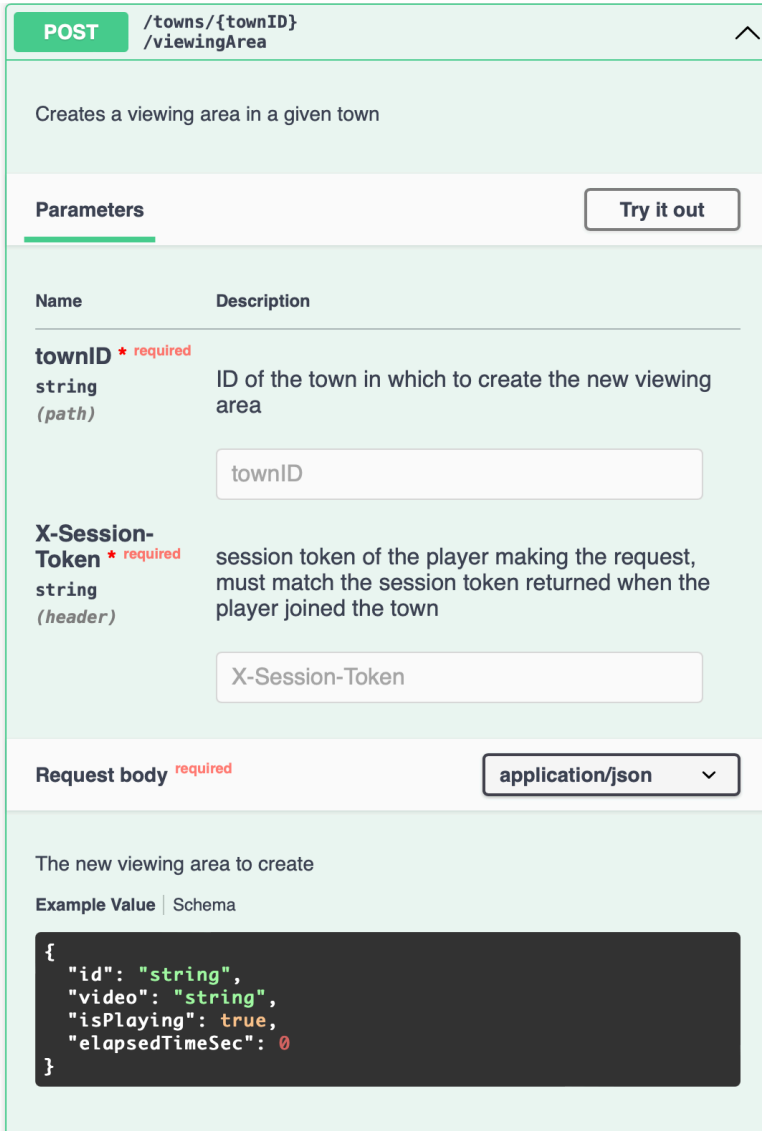
TSOA uses TS annotations to generate all the needed pieces



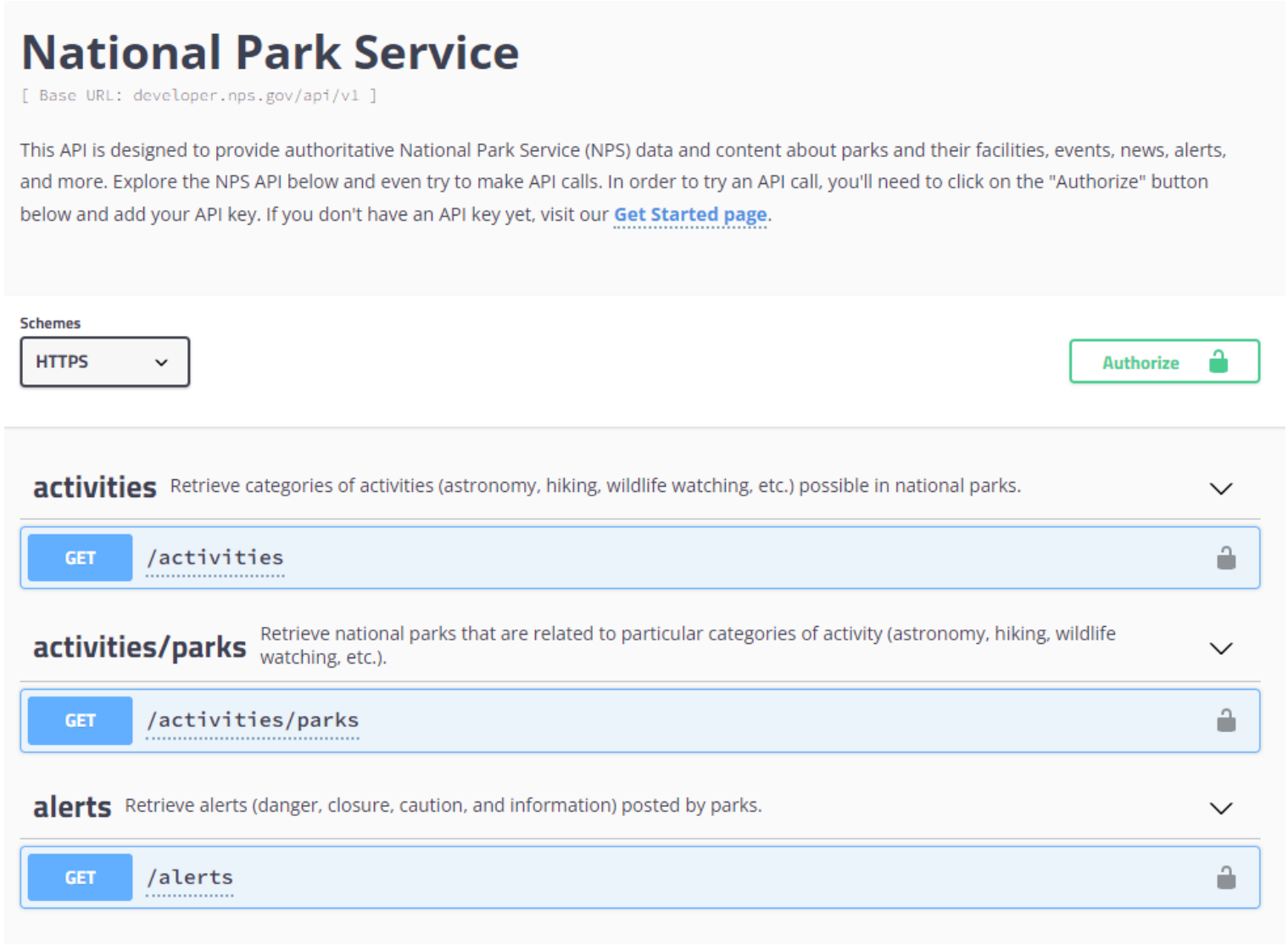
Sample annotated typescript

```typescript
@Route('towns')
export class TownsController extends Controller {



/**
 * Creates a viewing area in a given town
 *
 * @param townID ID of the town in which to create the new viewing area
 * @param sessionToken session token of the player making the request, must
 *        match the session token returned when the player joined the town
 * @param requestBody The new viewing area to create
 *
 * @throws InvalidParametersError if the session token is not valid, or if the
 *          viewing area could not be created
 */
@Post('{townID}/viewingArea')
@Response<InvalidParametersError>(400, 'Invalid values specified')
public async createViewingArea(
  @Path() townID: string,
  @Header('X-Session-Token') sessionToken: string,
  @Body() requestBody: ViewingArea,
){ /** method body goes here */ }
```

Sample generated HTML (“Swagger”)



Swagger in the wild



Resources

- What is a REST API?

Converting JavaScript Errors to HTTP Errors

- Under the hood, we use the popular express web server for NodeJS
- Express uses an internal pipeline architecture for processing requests
- We wrote this code snippet.
- It runs after the controller, inspects any error that might be thrown, and returns an HTTP error of 400, 422 or 500
- Example: if you say
  - a 400 error will be thrown.

```
throw new InvalidParametersError('Some message’)
```

  - The @Response is only for documentation

```
//server.ts
app.use(
  (
    err: unknown, _req: Express.Request, res: Express.Response,
    next: Express.NextFunction,
  ): Express.Response | void => {
    if (err instanceof ValidateError) {
      return res.status(422).json({
        message: 'Validation Failed',
        details: err?.fields,
      });
    }
    if(err instanceof InvalidParametersError){
      return res.status(400).json({
        message: 'Invalid parameters',
        details: err?.message
      })
    }
    if (err instanceof Error) {
      console.trace(err);
      return res.status(500).json({
        message: 'Internal Server Error',
      });
    }

    return next();
  },
)
```

Activity: Build the Transcript REST API

```
@Route('transcripts')
export class TranscriptsController extends
Controller {

    @Get()
    public getAll() {
        return db.getAll();
    }
```

Open API Specification