1 Technology Overview

REST: Representational State Transfer

Features
REST compliant web services allow requesting systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations.

Practically: each object (for example each database record) has its own URL and each action on the object a specific method or a specific child URL.

- List people with GET at http://example.com/people
- Add new person with POST at http://example.com/people
- Get person info with GET at http://example.com/people/42
- Update person info with POST at http://example.com/people/42
- Delete person info with DELETE at http://example.com/people/42

REST: Motivation

Motivation
Strike balance between need for explicit interfaces and need for loose coupling.

- Standard communication protocol (HTTP)
  - Already defines CRUD operations
  - Provides security and reliability
  - Is easy to deploy across internet
- Encourages separating model from view
- Supports independent implementation technology between client and server

CRUD

Create to create an object
Read to query object attributes
Update to update object attributes
Delete to delete an object

- The recommended minimum set of operations
- Corresponds reasonably well to HTTP methods
- Anything beyond CRUD is not considered pure REST

REST: Data Representation

Data exchange format is application specific but there are obvious choices
- XML because of existing library support
- JSON because of JavaScript in the browser
- YAML because it is the cool version of JSON

```json
{
  "name": "Jane Doe",
  "email": "jane.doe@example.com",
  "url": ["http://example.com/~jane.doe",
           "http://example.com/people/jane.doe"],
  "address": {
    "street1": "Our Street One",
    "street2": "Street Line Two",
    "city": "The City",
    "postal": "12345"
  },
  "room": 123
}
```

REST: Data Representation

Using links to make API more self contained is often encouraged
- Links to express relationships
- Links to explore the API

```json
{
  "name": "Jane Doe",
  "email": "jane.doe@example.com",
  "address": {
    "street1": "Our Street One",
    "street2": "Street Line Two",
    "city": { "href": "/cities/123" }
  },
  "links": {
    "self": { "href": "/users/123" },
    "connections": { "href": "/users/123/connections" }
  }
}
```

OpenAPI: API Development for REST

Interface Description
Paths to identify data model classes
Actions to operate on class instances
Attributes with types to describe class instances
Security defines access rules
Comments provide human readable description

- Code generation
  - Client libraries

¹Debates on pure REST vs pragmatic REST can get quite heated...
2 Assignment Details

Assignment

Inventory Application
Keeps track of *users* and *assets*. Basic user related operations are already defined. Define similar operations for assets and implement everything.

- Interface
  - Elementary CRUD operations for assets
  - One to many relationship between users and assets
- Server
  - Python implementation using Flask, or
  - Java implementation using Spring
- Client
  - TypeScript implementation using Angular, or
  - R and bash helper scripts

Assignment Interface: Prologue

openapi: 3.0.0

info:
  description: Inventory database
  version: 1.0.0
  title: Inventory
  termsOfService: ''
  license:
    name: Apache 2.0
    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'

servers:
  - url: 'http://localhost:8080/v1'

Assignment Interface: Defining Users

components:
  schemas:
    UserBase:
      type: object
      properties:
        id:
          type: integer
        firstname:
          type: string
          description: First name
        lastname:
          type: string
          description: Last name
    User:
      allOf:
        - $ref: '#/components/schemas/UserBase'
        - type: object
          properties:
            mail:
              type: string
              description: Mail
Assignment Interface: Listing Users

paths:
/users:
  get:
    summary: List all users
    operationId: readUsers
    x-openapi-router-controller: controllers.users
    responses:
      '200':
        description: Success
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/UserBase'

Assignment Interface: Querying User Data

paths:
'/users/{user_id}':
  get:
    summary: Query user
    operationId: readUser
    x-openapi-router-controller: controllers.users
    parameters:
      - in: path
        name: user_id
        description: User identifier
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: Success
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'

Assignment Interface: Updating User Data

paths:
'/users/{user_id}':
  put:
    summary: Update user
    operationId: updateUser
    x-openapi-router-controller: controllers.users
    parameters:
      - in: path
        name: user_id
        description: User identifier
        required: true
        schema:
          type: integer
    requestBody:
      $ref: '#/components/requestBodies/User'
    responses:
      '405':
        description: Invalid input
Assignment Details

Interface
Extend with operations and definitions related to assets.
- Same operations as already exist for users
- Additionally querying assets per user

Server
Pick one and extend it with asset related operations.

Client
Pick one and extend it as suggested.
- Angular: All asset operations and per user listing
- bash: Population and per user asset listing
- R: Plot average asset cost per department

Submission

GitLab
Use your personal GitLab repository under https://gitlab.mff.cuni.cz/teaching/nswi163/2022

Requirements
- Use the assignment subdirectory.
- Write brief report in SOLUTION.md.
- Include build scripts with instructions.
- Do not commit binaries or temporary build artifacts.
- Tag your solution with task-06 and push the tag.