W07-1: API Specs, Mocks & Examples

Dr. Alex Adkins

CPSC 3720: Software Engineering

REST API Key Lessons

- The term endpoint is focused on the URL that is used to make a request.
  - The same endpoint can return different resources, depending on a query string.

- The term resource is focused on the data that is returned by a request.
  - The same resource can often be accessed by multiple and different endpoints.
  - A group of resources are often called collections (used in Postman)

REST API Key Lessons

- An endpoint should be a noun that makes sense from the perspective of the API consumer, not verbs

  Example:

GET /tickets	- Retrieves a list of tickets
GET /tickets/12	- Retrieves a specific ticket
POST /tickets	- Creates a new ticket
PUT /tickets/12	- Updates ticket #12
PATCH /tickets/12	- Partially updates ticket #12
DELETE /tickets/12	- Deletes ticket #12

- Keep the URL format consistent and always use a plural

REST API Key Lessons

- Nested or Sub-Resources
  - When do you we use them?

```
/posts/:postId/comments/:commentId
/users/:userName/articles/:articleId
```

OR

```
/comments/:commentId
/articles/:articleId
```

- Readability, meaning, maintenance should be considered
- https://www.moesif.com/blog/technical/api-design/REST-API-Design-Best-Practices-for-Sub-and-Nested-Resources/

REST API Key Lessons

- Different endpoints accessing the same resource:

```
/api/companies/5/employees/3
/api/v2/companies/5/employees/3
/api/employees/3
```

- If your endpoint returns a collection of resources, you should implement searching/filtering/sorting using query strings:

```
/api/companies
/api/companies?sort=name_asc
/api/companies?location=germany
/api/companies?search=siemens
```

REST API Key Lessons

- Use the right method for the right action:
  - GET: Transfer a current representation of the target resource.
  - POST: Perform resource-specific processing on the request payload.
  - PUT: Replace all current representations of the target resource with the request payload.
  - DELETE: Remove all current representations of the target resource.

- It is a POST, PUT or PATCH method with which we will/must send a request body.
- https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/PUT-vs-POST-Whats-the-difference

POST, PUT, PATCH Methods

When to use POST, PUT, and PATCH can be confusing:

- A POST request creates a new resource (adds to the collection of resources).
- A PUT request creates a resource or updates an existing resource.
  - The request body contains a complete representation of the resource and if it already exists, it is replaced, otherwise, a new resource is created. PUT requests are most frequently applied to individual items. A server (service) might support updates but not creation via PUT.
- A PATCH request performs a partial update to an existing resource.
  - The request body specifies a set of changes to apply to the resource. This can be more efficient than using PUT, because the client only sends the changes, not the entire representation of the resource.

REST API Services Example

If we are developing a “users” resource, we need to create 5 basic web services:

	HTTP Method	PATH	BODY	Successful Response
Fetch all users	GET	/users		List of user representations
Get a user	GET	/users/<userId>		User representation
Add a user	POST	/users	User representation	User representation
Edit a user	PUT	/users/<userId>	User representation	User representation
Delete a user	DELETE	/users/<userId>		

REST API Key Lessons

- Path params: used to identify a specific resource or resources (usually by id)
- Query params: used to sort/filter resources
- Car Resource Example:

```
GET /cars
GET /cars/:id
POST /cars
PUT /cars/:id
DELETE /cars/:id
```

- Want to filter cards by color in your GET requests?
  - Color is not a resource (but is a property of a resource)
  - So you could add a *query* parameter:

```
GET /cars?color=blue
```

REST API Key Lessons

- Think about Return/Error Codes; they may not always be the same, i.e.:

  2xx Successful response

  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No content
  - etc.

- Error code responses should be consistent:
  - Response status code: 403

    Response body

    {

    code: 5,

    message: "Error message"

    }

References

https://solidgeargroup.com/en/best-practices-rest-api/

https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

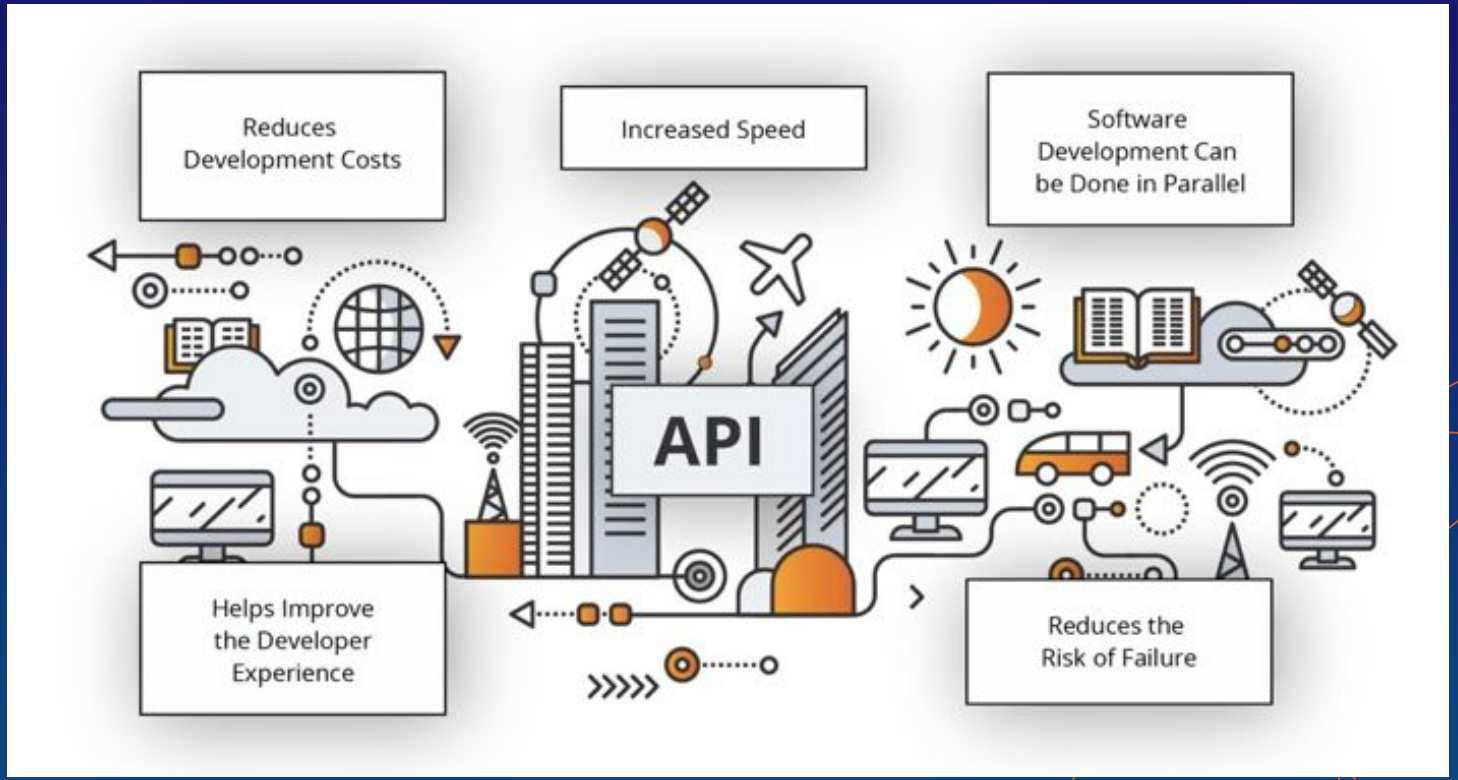
https://stackoverflow.com/a/50524312/9185662

https://www.moesif.com/blog/technical/api-design/REST-API-Design-Best-Practices-for-Sub-and-Nested-Resources/

https://medium.com/@fullsour/when-should-you-use-path-variable-and-query-parameter-a346790e8a6d

Why API First?

- New functionality should first be exposed as an API



Why API First?

- First step in API first: define your API
  - This is done with the OpenAPI specification
  - You can share this contract with API consumers
  - They can build around your contract even if it doesn't work yet!

- OpenAPI Specification is an open-source format for describing and documenting APIs
  - Formerly known as Swagger Specification

- The latest version of OpenAPI is 3.1
- OpenAPI definitions can be written in JSON or YAML
  - We will use YAML

Anatomy of the Open API

Main sections of an API defined with OpenAPI Specification:

- Meta information
- Path items (endpoints)
  - Parameters (path and query)
  - Request bodies (what you send in the request)
  - Responses (what you send back including status)
- Components (reusable items)
  - Schemas (data models)
  - Parameters
  - Responses
  - Other components

- Walkthrough artists tutorial here: https://support.smartbear.com/swaggerhub/docs/tutorials/openapi-3-tutorial.html

Create an API & Documentation Collection

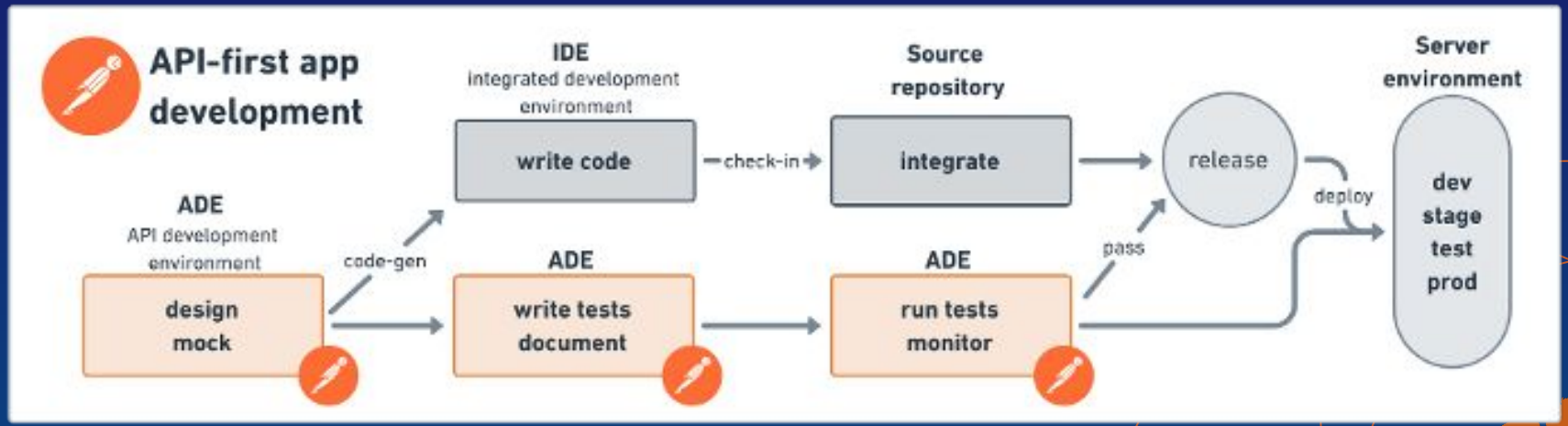
On your *personal* Postman workspace…

- Create an API
  - Activate the “APIs” Sidebar
  - “Create new API” from Boilerplate
  - Rename “New API” to “Spacecraft API”
  - Explore the YAML specification
- Create a Collection
  - Add Collection > Generate From Definition
  - Rename Collection to “Spacecraft Mock Definition”
  - Explore the generated Collection

Mocks & Examples

Mocks & Examples: Why do we need them?

- Supports API First through enabling continuous development
  - Across all stages of the agile development process, teams use mocks to decouple the development process, empowering people to work independently and in parallel.



Mocks and Examples: What are they?

- A **mock** is some code that substitutes for some real code and returns certain values given a certain input
- An **example** is the actual input and output you create for the mock to simulate real data

Mocks and Examples: Use Cases

1. Imagine an online store that uses a credit card service by calling an API. When testing that store, you might not want to send the credit card details to the credit card service, so you could instead create a mock server that you send these details to. This server could be set up so that it responds with a message that says it was processed, or maybe some different error states. Doing this allows you to test the application without needing to call the actual credit card service and perhaps rack up charges. It also allows you to try out some tests that would otherwise be difficult to do.

2. You have a UI team that wants to begin developing the front end of an application and would like to connect to your backend APIs as they build their front end in order to test their code. You can provide the UI team a mock server and examples so that they can build in parallel while you develop your API backend code.

Mocks & Examples: Additional Info

- When you send a request to the Mock server, the mock server sends back a response based on an Example with the same path and method.

- To make more useful examples use:
  - User-defined variables for your data
  - Random numbers/text provided by Postman
  - x-mock header values for control
  - Wildcard matching for undefined ids

Mocks & Examples: Additional Info

- Postman mock matching is picky!
- You cannot have multiple mock servers for the same collection – Postman will get confused! To start over, delete everything previous
- Edit the EXAMPLES and not the definition of the method and path
- Duplicate or Add Examples for more tests

checkin.cs.clemson.edu:

Create a Mock Server & Examples

On your *personal* Postman workspace…

- Create Mock Tests
  - Fill in the Examples Request & Response information
  - Save the Example
- Create a Mock Server
  - Click “Mock Collection” to Create a Mock Server
  - Save the mock server URL as a new ENV var
  - Edit the new ENV var to be “baseUrl”
  - Delete the “baseUrl” variable in the collection

checkin.cs.clemson.edu:

Create & Run Collection Tests

On your *personal* Postman workspace…

- Create a Collection called “Spacecraft Mock Test”
  - Create GET requests using previously defined Example call information
  - Use the Mock Server Environment for baseUrl
- Run your Test Collection with Postman Runner

CUSports Project: Sprint 1

- Sprint 1 (100 points)
- Due March 4
  - Some teams will demo on this day!

- You will be given Epics for Account & Notifications Services
  - Break down these Epics into User Stories & Tasks!
  - For each story you will need to create an API call in your Test Collection
- For each Service you will need to create:
  - API Specification in YAML/JSON
  - A Definition Collection in Postman
  - A Test Collection in Postman
    - This Test Collection should run with Postman's Runner tool!

- Details available on Canvas later tonight
  - Time on Thursday for questions/clarifications

Walkthrough of Creating API in Postman

See Canvas module for walkthrough link:

3720 Postman API Steps.pdf