MULESOFT ARCHITECTURE:
API Design

PRACTIC LEARNING

API Design Introduction

1. API Design Basics
2. API methods and resources
3. RAML design and best practices
4. API versioning
5. API Design Scenarios

API Design Overview

API Design Basics

- CRUD operations (Create, Read, Update, Delete)
- API Methods = HTTP verbs
  - POST, GET, PUT, PATCH, DELETE
    - Note: PATCH can be used for incremental updates, but is rarely used in practice because PUT can do the same. PUT can also be used to UPSERT
- API Resources = nouns
  - API resources are the domains in domain driven design (DDD)
  - Examples: Orders, Employees, Files, etc
- Design APIs using DDD around your resources (nouns) first, then your methods (verbs)

CRUD Method	HTTP Verb
Create	POST
Read	GET
Update	PUT, PATCH
Delete	DELETE

API Resources and Methods Examples

Description	Resource	HTTP Method
Create order	Order	POST
Get orders	Orders	GET
Get order	Order	GET
Update order	Order	PUT, PATCH
Delete order	Order	DELETE

RAML Design and Best Practices

Designing an API in RAML

Resources

- Always start the URI
- Can be in the middle or end of the URI also
- Examples
  - /orders
  - /orders/{orderID}/payments
    - Orders and payments are the resources

URI Parameters

- Follow a resource in the URI
- Used to target a specific resource by its primary identifier (think primary key in a database)
- Wrapped in curly braces
- **Caution**: be careful of URI parameter overlap with other resources
- Examples
  - /orders/{orderID}
    - orderID is the URI parameter and primary lookup value for the orders resource

Query Parameters

- Comes after the question mark (?) in the URI
- Used when searching for a resource based on attributes that are not the primary key
- Examples
  - /orders/{orderID}?customerName=Practic
    - customerName is the query parameter

RAML Reusability Basics

- Traits
  - Reusable RAML fragment that can be reused across methods
  - A great use case for this is standard error responses reused across all API endpoints
- Resource Types
  - Reusable RAML fragment that can be used across resources
  - Great for defining reusable descriptions and API endpoints across resources
- Security Schemes
  - Defined security scheme to put on each API method
- Types
  - Defined schema for a particular object type

RAML Best Practices

- Externalize traits, resourceTypes, types, etc into different files than your main RAML for reuse, readability, and cleanliness
- Strategize and define success and error response formats/schemas for enterprise-wide reuse
  - Success responses can follow the same schema
  - Consider using a reusable trait for error responses
- Get early feedback on API responses
- Strategize and define API security on each API layer using securitySchemes
- Include verbose descriptions on API endpoints, query parameters, URI parameters, etc

API Versioning

URI Parameter Versioning

- Include the version in the URI
    - Examples
        - /api/v1/orders
        - /api/v2/orders
- Benefits
    - Same API managing multiple versions reduces MuleSoft core usage and makes consumer version upgrades easier
- Pitfalls
    - More difficult to separate API version traffic

Concurrent Deployment

- Deploy a separate API for each version of the API
    - Examples
        - practic-orders-papi-v1.cloudhub.io/api/orders
        - practic-orders-papi-v2.cloudhub.io/api/orders
- Benefits
    - Completely decouples different versions of an API
- Pitfalls
    - Increases MuleSoft core usage and required to manage multiple APIs

Design Scenarios

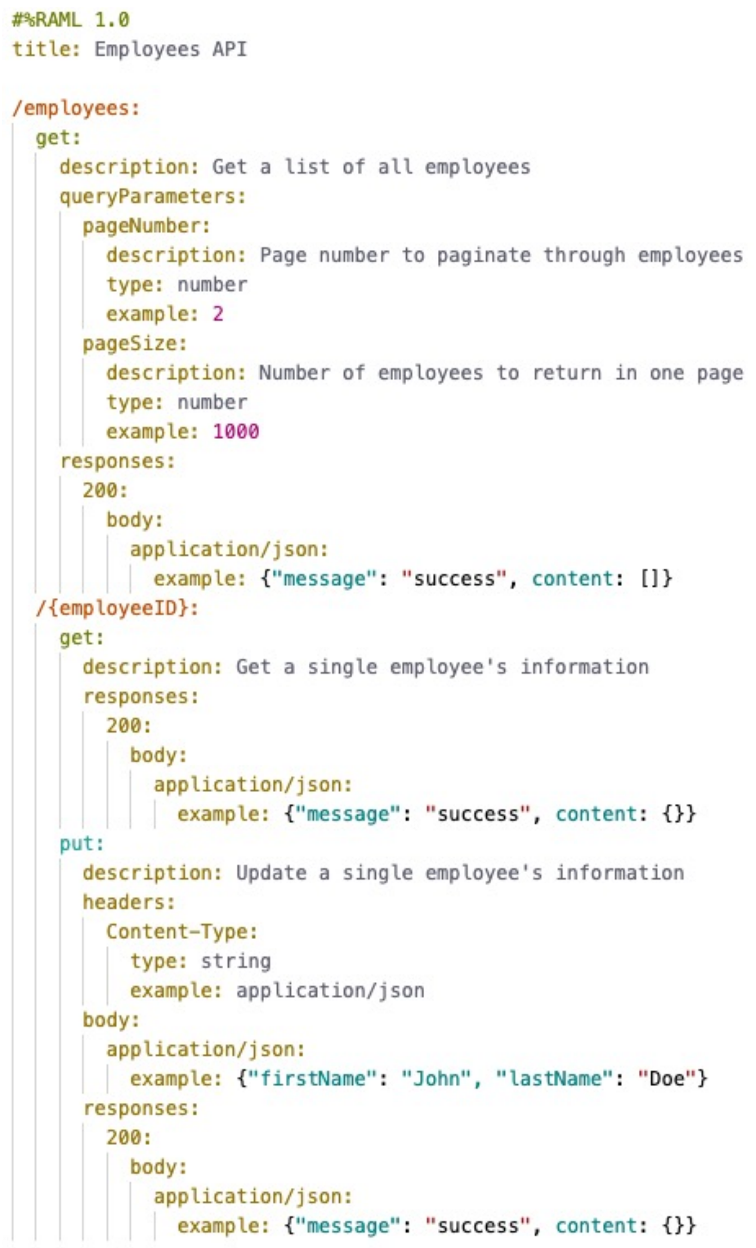
API Design Scenario #1

An organization has a need to get and update employee information, such as email address, first name, and last name, in WorkDay with the following operations:

- *Get a list of all employees*
- *Get a single employee’s information by employeeID*
- *Update an employee’s information by employeeID*

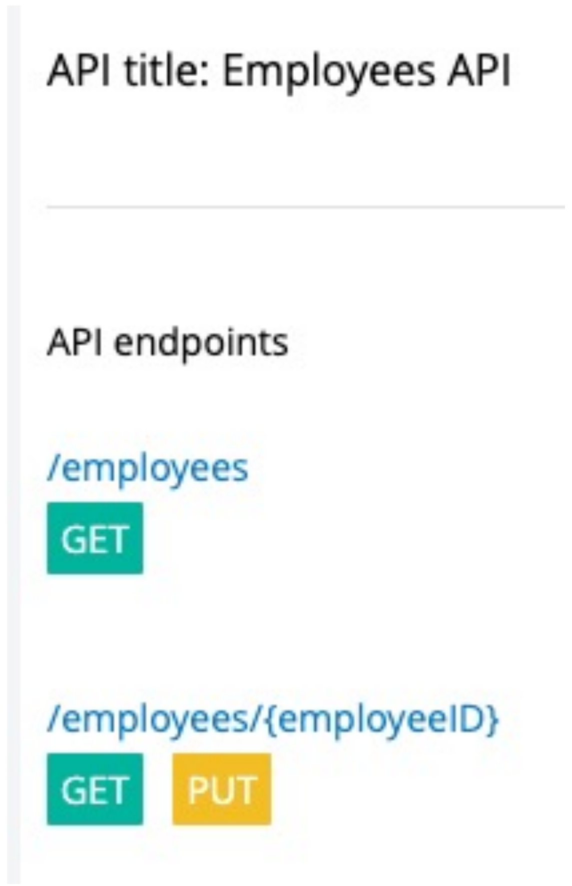
Design an API in RAML to meet the organization’s requirements.

API Design Scenario #1 Architecture



```
#%RAML 1.0
title: Employees API

/employees:
  get:
    description: Get a list of all employees
    queryParameters:
      pageNumber:
        description: Page number to paginate through employees
        type: number
        example: 2
      pageSize:
        description: Number of employees to return in one page
        type: number
        example: 1000
    responses:
      200:
        body:
          application/json:
            example: {"message": "success", content: []}
  /{employeeID}:
    get:
      description: Get a single employee's information
      responses:
        200:
          body:
            application/json:
              example: {"message": "success", content: {}}
    put:
      description: Update a single employee's information
      headers:
        Content-Type:
          type: string
          example: application/json
      body:
        application/json:
          example: {"firstName": "John", "lastName": "Doe"}
      responses:
        200:
          body:
            application/json:
              example: {"message": "success", content: {}}
```



API title: Employees API

API endpoints

/employees

GET

/employees/{employeeID}

GET PUT

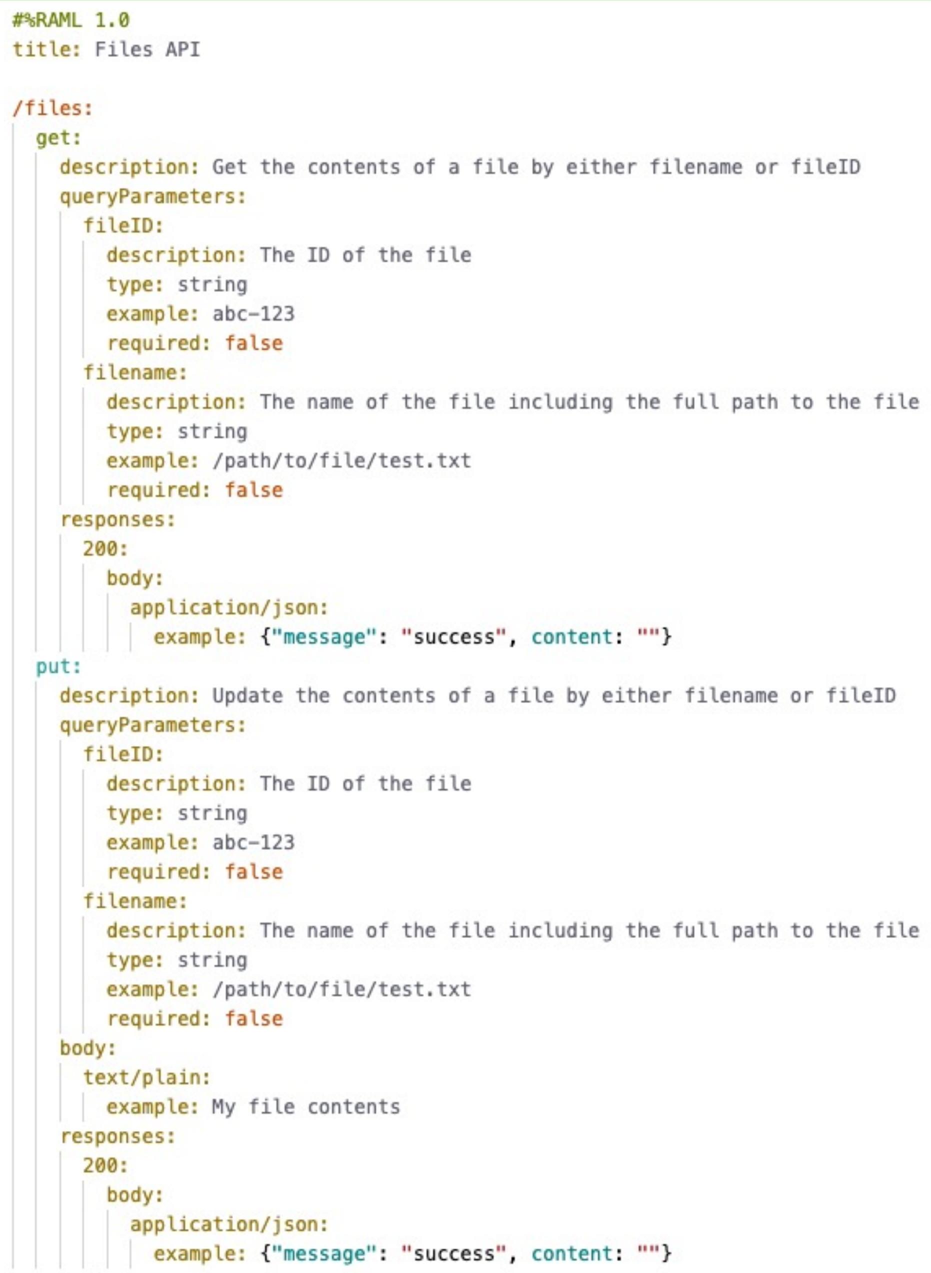
API Design Scenario #2

An organization has a need to get and update files stored in file storage with the following operations where filename and fileID are both primary lookup values:

- *Get file content by name*
- *Get file content by fileID*
- *Update file content by name*
- *Update file content by fileID*

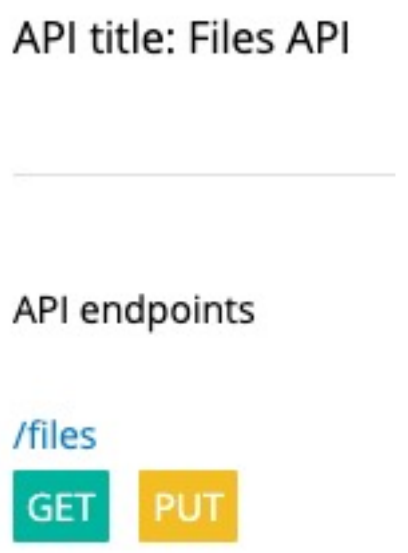
Design an API in RAML to meet the organization's requirements.

API Design Scenario #2 Architecture Option 1



```
#%RAML 1.0
title: Files API

/files:
  get:
    description: Get the contents of a file by either filename or fileID
    queryParameters:
      fileID:
        description: The ID of the file
        type: string
        example: abc-123
        required: false
      filename:
        description: The name of the file including the full path to the file
        type: string
        example: /path/to/file/test.txt
        required: false
    responses:
      200:
        body:
          application/json:
            example: {"message": "success", content: ""}
  put:
    description: Update the contents of a file by either filename or fileID
    queryParameters:
      fileID:
        description: The ID of the file
        type: string
        example: abc-123
        required: false
      filename:
        description: The name of the file including the full path to the file
        type: string
        example: /path/to/file/test.txt
        required: false
    body:
      text/plain:
        example: My file contents
    responses:
      200:
        body:
          application/json:
            example: {"message": "success", content: ""}
```

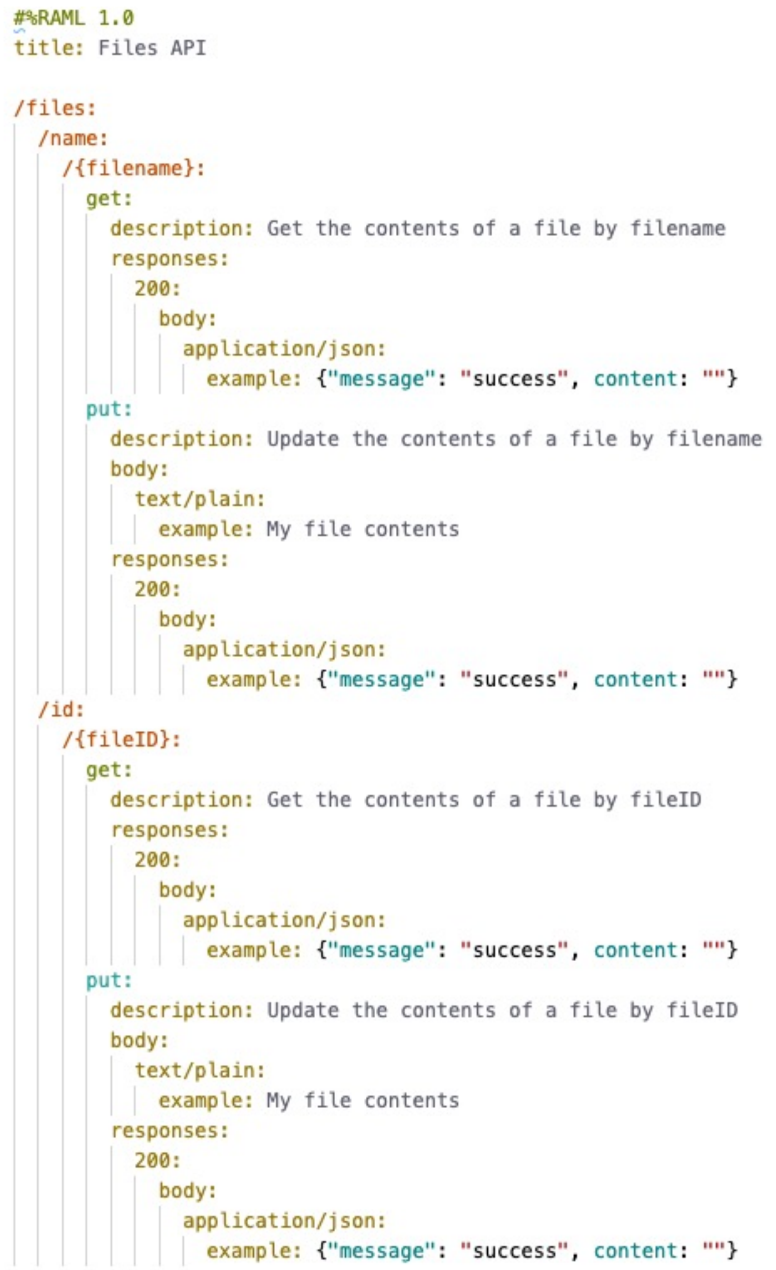


API title: Files API

API endpoints

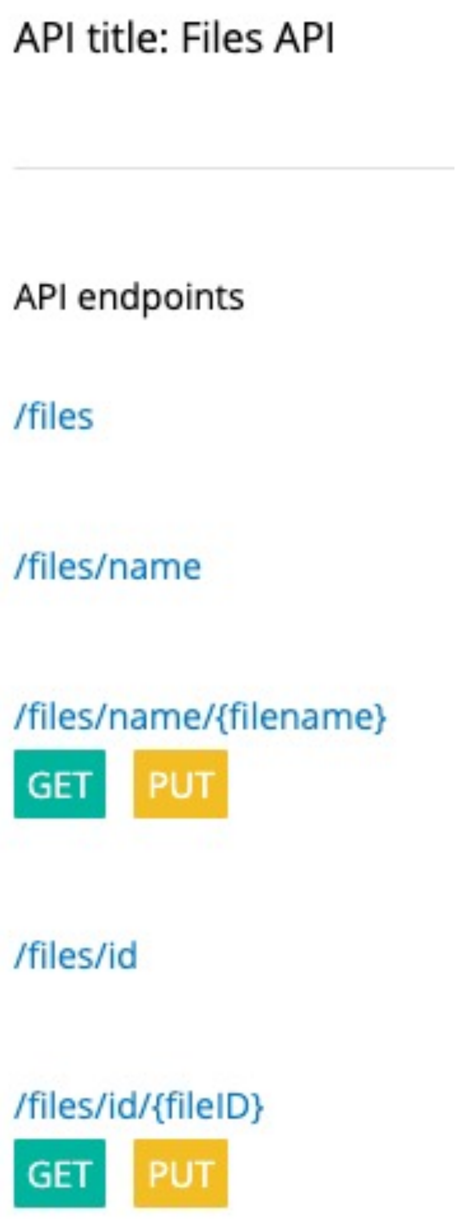
/files

API Design Scenario #2 Architecture Option 2



```
#%RAML 1.0
title: Files API

/files:
  /name:
    /{filename}:
      get:
        description: Get the contents of a file by filename
        responses:
          200:
            body:
              application/json:
                example: {"message": "success", content: ""}
      put:
        description: Update the contents of a file by filename
        body:
          text/plain:
            example: My file contents
        responses:
          200:
            body:
              application/json:
                example: {"message": "success", content: ""}
  /id:
    /{fileID}:
      get:
        description: Get the contents of a file by fileID
        responses:
          200:
            body:
              application/json:
                example: {"message": "success", content: ""}
      put:
        description: Update the contents of a file by fileID
        body:
          text/plain:
            example: My file contents
        responses:
          200:
            body:
              application/json:
                example: {"message": "success", content: ""}
```



API title: Files API

API endpoints

/files

/files/name

/files/name/{filename}

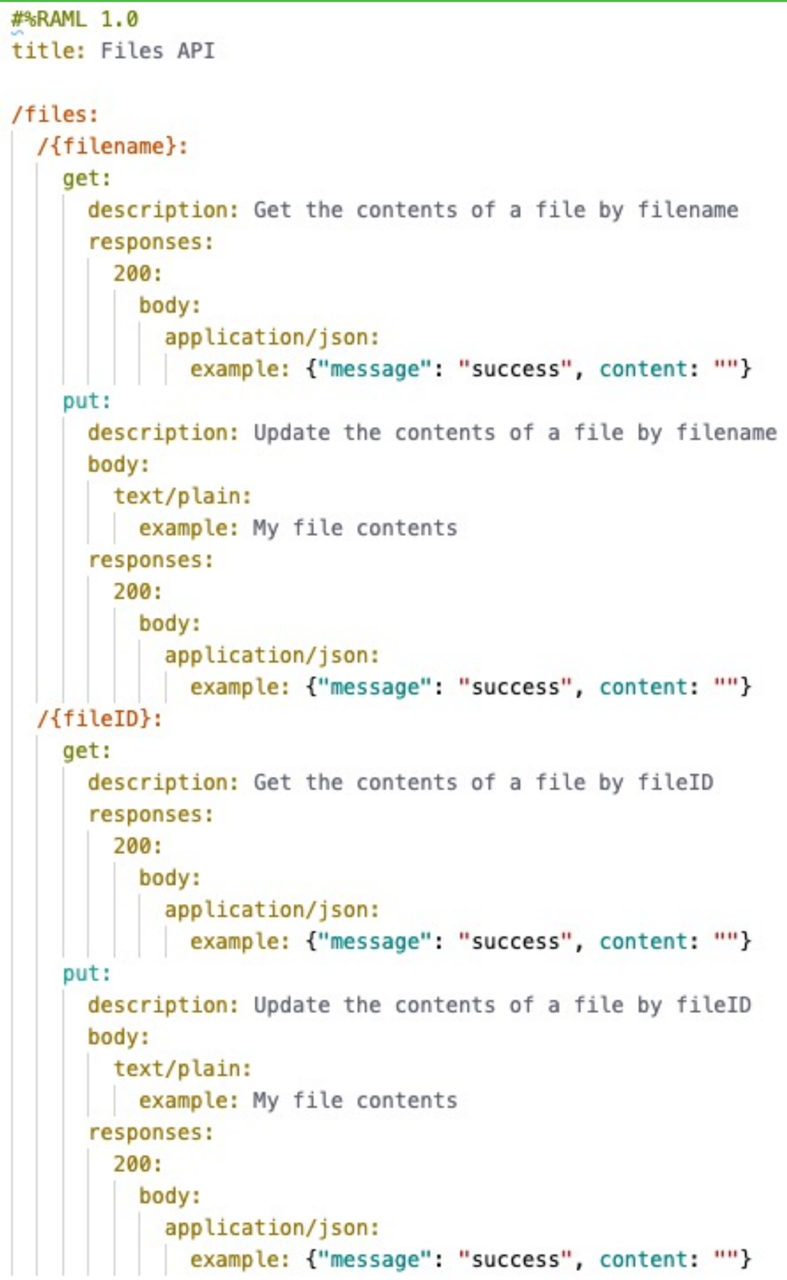
GET PUT

/files/id

/files/id/{fileID}

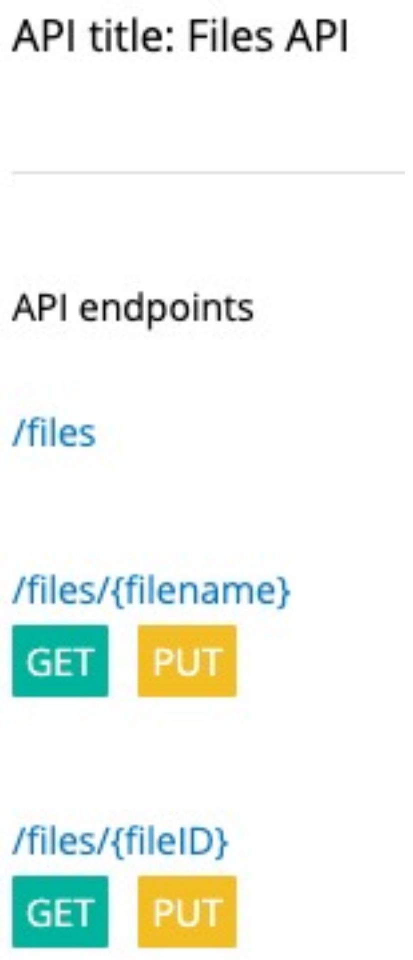
GET PUT

API Design Scenario #2 Architecture Common Mistake



```
#%RAML 1.0
title: Files API

/files:
  /{filename}:
    get:
      description: Get the contents of a file by filename
      responses:
        200:
          body:
            application/json:
              example: {"message": "success", content: ""}
    put:
      description: Update the contents of a file by filename
      body:
        text/plain:
          example: My file contents
      responses:
        200:
          body:
            application/json:
              example: {"message": "success", content: ""}
  /{fileID}:
    get:
      description: Get the contents of a file by fileID
      responses:
        200:
          body:
            application/json:
              example: {"message": "success", content: ""}
    put:
      description: Update the contents of a file by fileID
      body:
        text/plain:
          example: My file contents
      responses:
        200:
          body:
            application/json:
              example: {"message": "success", content: ""}
```



API title: Files API

API endpoints

/files

/files/{filename}

/files/{fileID}

API-Led Connectivity Summary

- CRUD operations relating to API methods
- API methods and resources
- RAML resources, URI parameters, and query parameters usage
- RAML best practices
- API versioning
- Designing APIs

Additional Reading

1. https://raml.org/developers/raml-100-tutorial
2. https://raml.org/developers/raml-200-tutorial
3. https://docs.mulesoft.com/exchange/to-change-raml-version
4. https://programmerspub.com/blog/general/raml-best-practices