# 4. Creating your first API

**Introduction**

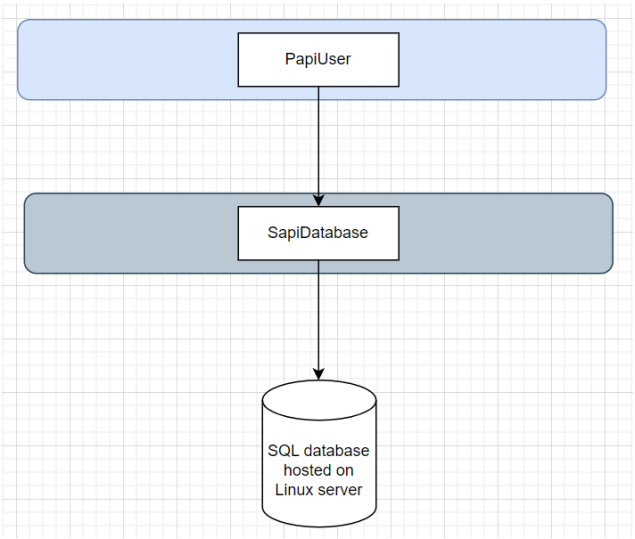
- Our internal education program is composed of five comprehensive parts that are designed to equip our employees with the necessary skills and knowledge to successfully handle client projects. The **first** part of the program covers the setup of a Linux server and the installation of SQL on it along with appropriate database. The **second** part is focused on creating a RAML specification and its implementation in Anypoint Studio (system API called SapiDatabase), while the **third** part focuses on implementing business logic in MuleSoft (process API called PapiUser). The **fourth** part of the program is centered on setting up MuleSoft runtime and deploying the application and the **final** part is dedicated to connecting nJams with the application to monitor its performance
- Program has been successful in the past and has been refined over time to ensure that our employees are adequately prepared for client projects. We expect our employees to be proactive in finding solutions to challenges that may arise during the project by utilizing resources such as Google, YouTube and other relevant resources
- Overall, our internal education program is geared towards empowering our employees with the skills and knowledge necessary to tackle complex client projects with confidence and professionalism

**List of technologies**

- MuleSoft (Design Center for RAML and Anypoint Studio for implementation)
- Linux server (to host SQL database)
- MUnit (to perform unit testing of MuleSoft applications)

**API-led connectivity overview**

- Note: In order to proceed with subsequent steps, it is essential to gain an understanding of MuleSoft's recommended architectural approach and its constituent elements
- API-led connectivity in MuleSoft involves creating a network of APIs that connect applications, data, and devices, each with its own specific purpose and set of capabilities. This network is typically composed of three layers: the system, process, and experience layers
- The **system** layer represents the core systems and applications that make up an organization's IT infrastructure, such as databases, legacy systems, and enterprise applications. This type of API is typically designed to provide access to data or operations that are specific to a particular system or application, such as a database, ERP system or legacy application
- The **process** layer sits between the system and experience layers, and is responsible for orchestrating the flow of data between different systems and applications. The APIs in this layer are typically designed to support specific business processes, such as order management or customer onboarding
- The **experience** layer represents the front-end applications and devices that interact with the backend systems and data. The APIs in this layer are typically designed to provide a user-friendly interface that exposes the functionality and data available in the system and process layers
- Upper box (very pale blue) represents single **Process API** that contacts **System API** located in lower box (Grayish blue). Additional information about **each API** are available in the coming chapters
- Note: Process and System API-s are using **HTTP request** to communicate with each other



**First Part**

- To begin the process of accessing the "http://controller/autodeploy/servers" URL, please open a web browser and enter the URL into the address bar. This will allow you to access the web-based interface for server deployment. and follow the instructions provided to create your server with your username. Upon successful completion of the server creation process, you will receive an email within 10 minutes containing the IP address of your server. Please note that the username for the Linux server is "root", and the password is "toor"
- Undertake the installation of MySQL on a Linux server that has been previously provisioned
- Enable remote connections by configuring the server to accept incoming connections from remote clients. This can be achieved by configuring the server's network settings, including the server's IP address and port number, as well as enabling remote connections through the server's configuration settings. Link to useful content
- Disable the firewall by modifying the server's firewall settings to allow incoming traffic on the appropriate ports. This can be achieved by creating a new firewall rule that allows incoming traffic on the specified port, or by disabling the firewall altogether
- Create a database table based on the schema provided below this text

| Field Name | Data Type | Example | Note |
|---|---|---|---|
| id | integer | 1 | • Automatically incremented and **unique** for each user record |
| email | string | kruno@net.hr | • The email address of the user that is **unique** for each user record |
| firstName | string | kruno | • The first name of the user |
| lastName | string | krainovic | • The last name of the user |
| manager_id | integer | 1 | • The **unique** identifier of a user's manager |
| street | string | Wall Street 5 | • The name of the street where user lives |
| city | string | New York | • The name of city where user lives |
| dob | date | 2008-11-11 | • The date of birth when user was born<br>• Follow **date** format YYYY-MM-DD |
| updated_at | datetime | 2008-11-11 13:23:44 | • The last time when user updated in the database<br>• Follow **datetime** format YYYY-MM-DD HH:MM:SS |
| initials | string | KK | • The first letter of fields “firstName” and “lastName” concatenated together<br>• Field should contain exactly 2 characters |

**Second Part**

- Note: In MuleSoft, RAML is used as a design tool to define the interface of RESTful APIs, including their resources, methods, parameters, headers, responses, and other details. RAML provides a standardized way to document APIs, making it easier for developers, testers and other stakeholders to understand and consume the APIs
- Create RAML for each endpoint defined in API reference guide chapter with having in mind few rules:
- RAML should be consisted of 3 separate folders (dataType, examples and root file)
- In dataType define data models that API is going to consume through the definition of data types
- In examples provide example for each defined data model
- In the root file, specify the endpoints and their corresponding requests and responses. Use the provided examples in the designated folder to guide the definition of each endpoint's structure and behavior
- Root file must be based on abstractions rather than concrete implementations
- Link to useful content
- Note: Query parameters are optional and used for filtering, sorting or paginating results based on non-primary keys, while URI parameters are mandatory and used to identify specific resources based on primary keys

**API reference guide**

- Note: All endpoints needs to be secured by **Client ID Enforcement Policy**
- **sapi-database (System API)**
- get by id: /users/1

| Request | Response | Note | Error handler |
|---|---|---|---|
| | {<br>"id":1,<br>"email":"kruno@net.hr",<br>"firstName":"kruno",<br>"lastName":"kra",<br>"manager_id":1,<br>"addrese":{<br>"street":"Vukovarska 120",<br>"city":"Vinkovci"<br>},<br>"dob": “08/08/1988”,<br>"updated_at":"2021-11-26 12:27:16",<br>"initials":"KK"<br>} | • Get user with URI parameter (ID)<br>• Status code: 200<br>• Client ID enforcement policy is mandatory | • If the user doesn't exist, status code is "404" and response {“message": "User Not Found"} will be shown |
| | | | |

- get by email: /users?email=kruno@net.hr

| Request | Response | Note | Error handler |
|---|---|---|---|
| | {<br>"id":1,<br>"email":"kruno@net.hr",<br>"firstName":"kruno",<br>"lastName":"kra",<br>"manager_id":1,<br>"addrese":{<br>"street":"Vukovarska 120",<br>"city":"Vinkovci"<br>},<br>"dob": “08/08/1988”,<br>"updated_at":"2021-11-26 12:27:16",<br>"initials":"KK"<br>} | • Get user with query parameter (email)<br>• Status code: 200<br>• Client ID enforcement policy is mandatory | • If the user doesn't exist, status code is "404" and response {“message": "User Not Found"} will be shown |
| | | | |

- get all: /users

| Request | Response | Note |
|---|---|---|
| | [{<br>"id":1,<br>"email":"kruno@net.hr",<br>"firstName":"kruno",<br>"lastName":"kra",<br>"manager_id":1,<br>"addrese":{<br>"street":"Vukovarska 120",<br>"city":"Vinkovci"<br>},<br>"dob": “08/08/1988”,<br>"updated_at":"2021-11-26 12:27:16",<br>"initials":"KK"<br>}] | • Get all users<br>• Status code: 200<br>• Client ID enforcement policy is mandatory |
| | | |

- delete by id: /users/1

| Request | Response | Note | Error handler |
|---|---|---|---|
| | | • Delete user with URI parameter (ID)<br>• Status code: 204<br>• Client ID enforcement policy is mandatory | • If the user doesn't exist, status code is "404" will be shown without response |
| | | | |

- post: /users

| Request | Response | Note | Error handler |
|---|---|---|---|
| {<br>"email":"kruno@net.hr",<br>"firstName":"kruno",<br>"lastName":"kra",<br>"manager_id":1,<br>"addrese":{<br>"street":"Vukovarska 120",<br>"city":"Vinkovci"<br>},<br>"dob": “08/08/1988”,<br>"updated_at":"2021-11-26 12:27:16",<br>"initials":"KK"<br>} | {<br>"message": "User Successfully Created"<br>} | • Create user with values defined in request<br>• Status code: 200<br>• Client ID enforcement policy is mandatory | • If the user already exist, status code is "409" and response {“message": "User Already Exist"} will be shown. |
| | | | |

- put: /users/1

| Request | Response | Note | Error handler |
|---|---|---|---|
| {<br>"email":"kruno@net.hr",<br>"firstName":"kruno",<br>"lastName":"kra",<br>"manager_id":1,<br>"addrese":{<br>"street":"Vukovarska 120",<br>"city":"Vinkovci"<br>},<br>"dob": “08/08/1988”,<br>"updated_at":"2021-11-26 12:27:16",<br>"initials":"KK"<br>} | {<br>"message": "User Successfully Updated"<br>} | • Update user with URI parameter<br>• Status code: 200<br>• Client ID enforcement policy is mandatory | • If the user doesn't exist, status code is "404" and response {“message": "User Not Found"} will be shown |
| | | | |

**Implementation**

- Create new Mule Project named SapiDatabase in Anypoint Studio
- Import RAML you have previously created to SapiDatabase. Link to instructions
- In the Mule Palette add Database module
- In src/main/mule folder select the sapidatabase.xml file → Global Elements → Create → Database Config → Connect to SQL database using appropriate credentials
- Based on the RAML specification and the endpoints defined in the API Reference Guide chapter, you are now fully equipped to proceed with the implementation process
- Note: It is recommended that reusable components are organized in separate Mule configuration files. This approach promotes modularity and facilitates maintenance, as changes to individual components can be isolated and tested without affecting the overall system. Link to **SOLID principles**

**Template cloning**

- Please begin by cloning our template project using the provided link. Once cloned, create property files to store the relevant database credentials for each environment and ensure their security. Next, transfer all necessary files from SapiDatabase into the cloned project
- Your mentor will then guide you through the project template, explaining how to properly use it, establish coding conventions, establish naming conventions and address other relevant considerations such as those related to the property files
- Note: Transfer all configurations from sapidatabase.xml file into global.xml file located in src/main/mule folder
- Implement an error handling mechanism

**MUnit testing**

- MUnit is a testing framework provided by MuleSoft to test Mule applications. It allows you to write unit tests for your Mule flows and connectors to ensure they are working as expected
- In src/test/munit folder in your Mule project → New → MUnit Test Suite. This will create a new MUnit test file
- Note: To optimize the organization and efficiency of MUnit test suites, it is recommended to partition them based on the expected outcome of the Mule flows being tested. This approach ensures that the success scenarios and error handling mechanisms of the application are thoroughly tested and evaluated in separate test suites
- Add test cases: In the MUnit test file, add one or more test cases. Each test case consists of a set of inputs and expected outputs. You can use various MUnit test components such as Mocks and Verifiers to define the inputs and outputs of the test case
- Run the test: To run the test, right-click on the MUnit test file and select "Run MUnit Suite". The test results will be displayed in the Mule console
- Analyze the test results: The test results will show whether each test case passed or failed. If a test case fails, you can analyze the error message to identify the issue and fix it
- Note: As a best practice for ensuring abstractions rather than concrete implementations, it is recommended to store assets utilized in MUnit components in two separate directories. These directories should be located in the 'src/test/resources' directory
- The “cmd” folder, which stands for common data model, serves as a repository for **mock** objects used for simulating various scenarios such as query parameters in an HTTP request or retrieving user from a database
- The “targetModel” folder serves as a repository for **validating** specific components of the Mule event, such as the value of relevant variables or payloads before, during or after the completion of the entire flow
- Example of calling the function “getUser” from “userResponse.dwl” file located in “targetModel” folder
- ```
  %dw 2.0
  import getUser from targetModel::userResponse
  output application/java

  getUser()
  ```

**Third part**

- Now when you are equipped with the first API according to the API-led connectivity approach, it is time to proceed with developing a process API. As you already know **starting point** in the development process is the creation of the RAML. **Second** step is to clone our template project to establish the structure of the entire project. The **third** step is the implementation process according to the guidelines and specifications provided in the API reference guide. The **fourth** step in ensuring the reliability and functionality of your implementation involves validating your Mule application through MUnit tests

**RAML creation**

- The established best practices that you are already familiar with remain valid and applicable

**Template setup**

- Now when you are familiar with template key components and its setup, open this link to setup it on your own

**Implementation**

- The established best practices that you are already familiar with remain valid and applicable

**MUnit testing**

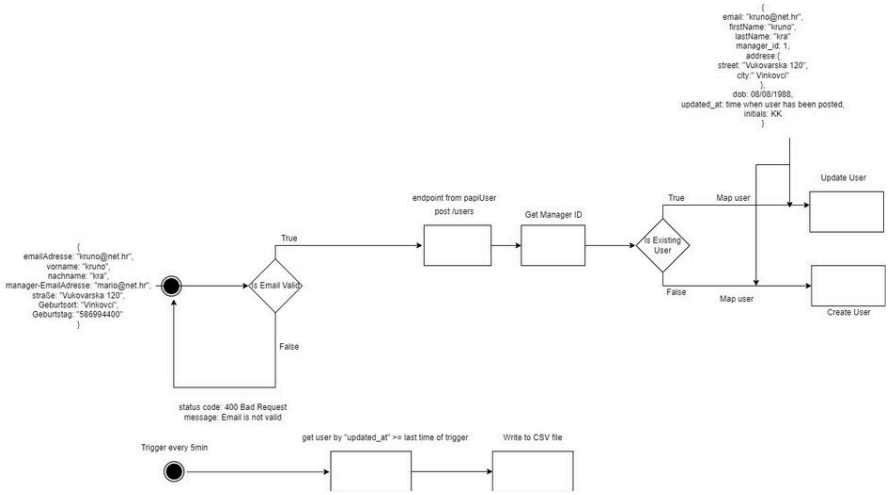
- The established best practices that you are already familiar with remain valid and applicable

**API reference guide**

- Note: Endpoint needs to be secured by **Client ID Enforcement Policy**
- **papi-users (Process API)**
- post: /user
  ```

| Request | Response | Note |
| --- | --- | --- |
| {<br>”emailAdresse”: "kruno@net.hr",<br>”vorname”: "kruno",<br>”nachname”: "kra",<br>”manager-EmailAdresse”:<br>"mario@net.hr",<br>”straße”: "Vukovarska 120",<br>"Geburtsort": "Vinkovci",<br>"Geburtstag": "586994400"<br>} | | • Status code: 201<br>• RAML needs to validate email structure<br>• Client ID enforcement policy is mandatory |
| | | |

- Visual representation of events in chronological order along with an endpoint that handles HTTP request




**Fourth part**

- Deploy the MuleSoft Runtime on your Linux server. Link to useful content
- Establish a connection between your MuleSoft Runtime server and CloudHub
- Modify the properties file in your application to replace the "localhost" configuration with the IP address of your Linux server. This will enable the application to connect to the MuleSoft Runtime server running on the Linux server
- Upload your applications to the MuleSoft Runtime server on the Linux server. Link to useful content

**Fifth part**

- Register for an account at https://customerportal.integrationmatters.com
- Follow the instructions on the website to download and install the Njams client and establish a connection
- Install the Njams client on both API servers
- Make a few requests on the PapiUser
- View all requests made by accessing the Njams client dashboard

Exercise made by: @Krunoslav Krainovic