codecentric

Lightweight and Repeatable Integration Testing of Keycloak Extensions

Photo: Markus Spiske, Pexels

Dr. Florian Rademacher, florian.rademacher@codecentric.de

What exactly is CRM?

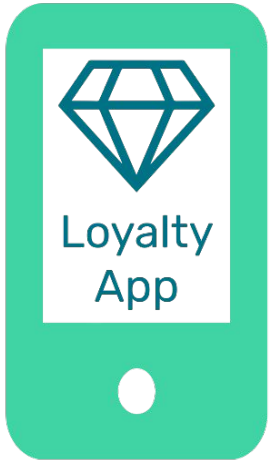
Project Context

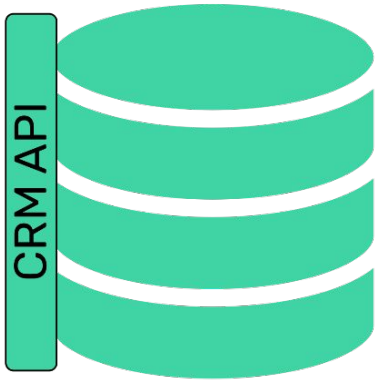
- Customer Relationship Management (CRM)[1]
  - Process in which a business administers its interactions with customers
  - Goal: Achieve optimal customer orientation
  - CRM systems as management tools
    - Provide consolidated customer view(s)
    - Integration and automation of sales, marketing, and customer support
    - Example use case: Management of loyalty points

[1] Wikipedia: https://en.wikipedia.org/wiki/Customer_relationship_management

Project Context

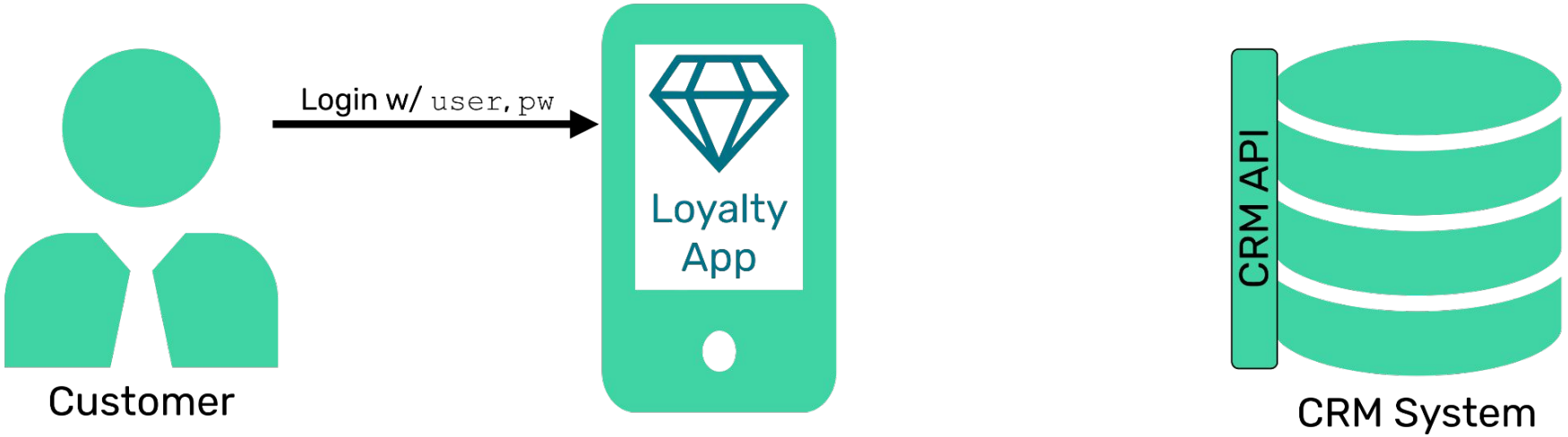
Customer



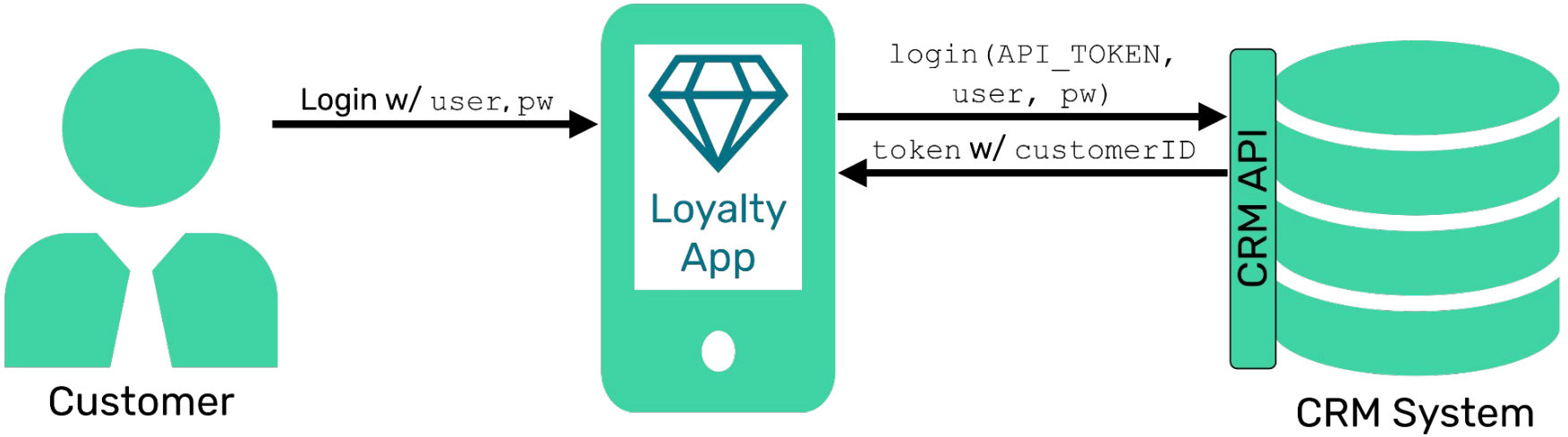


CRM System

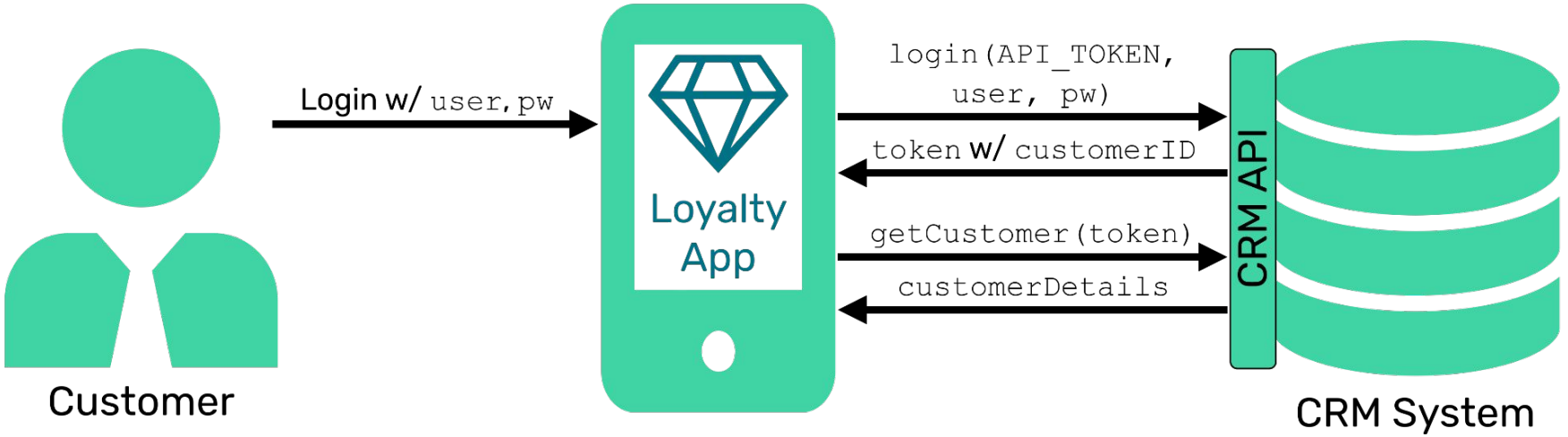
Project Context



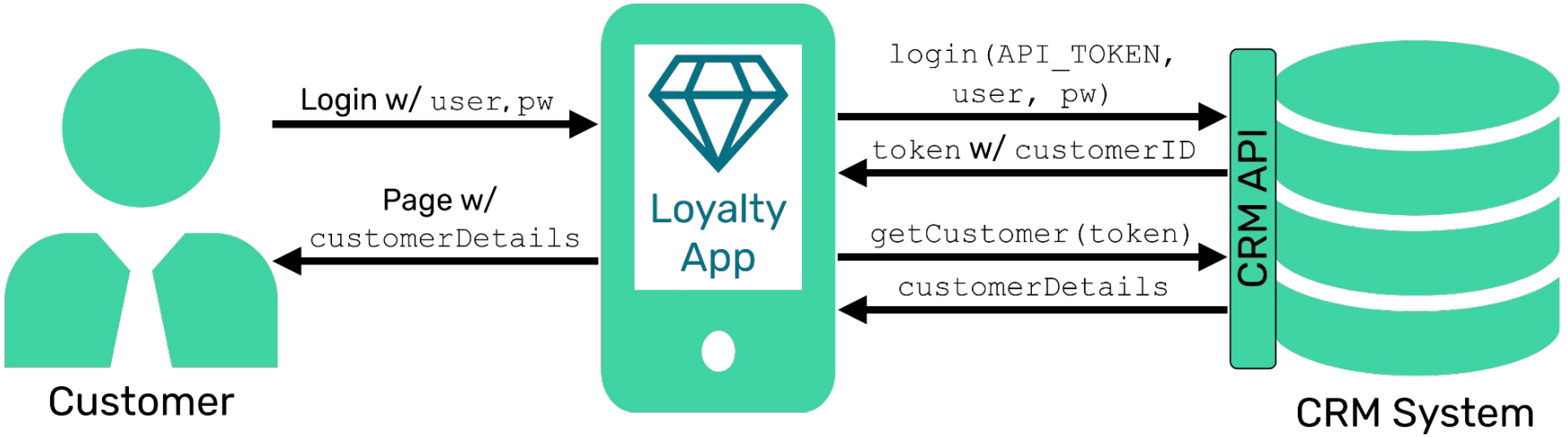
Project Context



Project Context



Project Context



Project Context

- Retailer requests Keycloak integration
  - Enable single sign-on between Loyalty App and webshop
  - Leverage industry standards for authentication and authorization
  - Harness out-of-the-box features when needed
    - Federation of support users
    - Social login
    - Two-factor authentication

Project Context

- Retailer requests Keycloak integration
  - Enable single sign-on between Loyalty App and webshop
  - Leverage industry standards for authentication and authorization
  - Harness out-of-the-box features when needed
    - Federation of support users
    - Social login
    - Two-factor authentication

Project Context

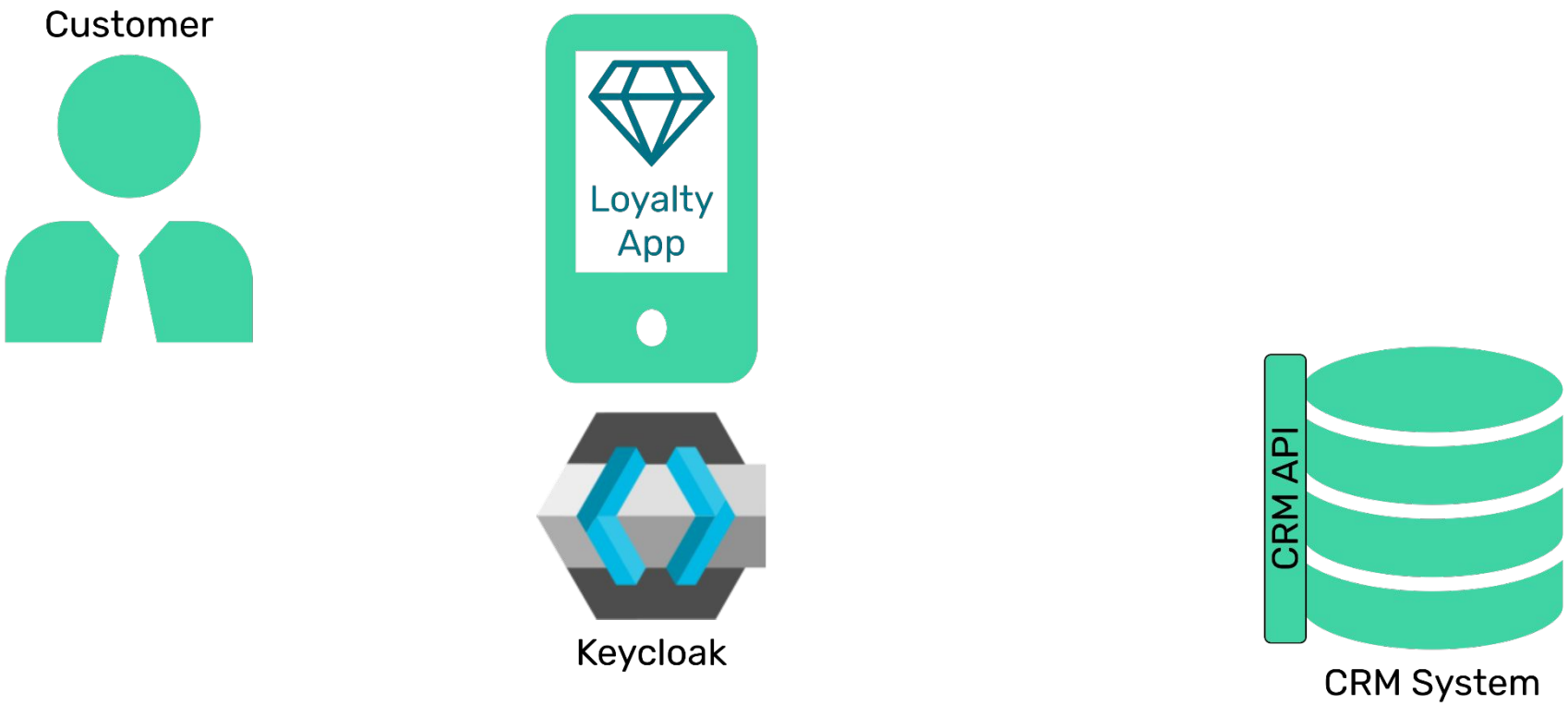
- Retailer requests Keycloak integration
  - Enable single sign-on between Loyalty App and webshop
  - Leverage industry standards for authentication and authorization
  - Harness out-of-the-box features when needed
    - Federation of support users
    - Social login
    - Two-factor authentication

Dr. Florian Rademacher, florian.rademacher@codecentric.de

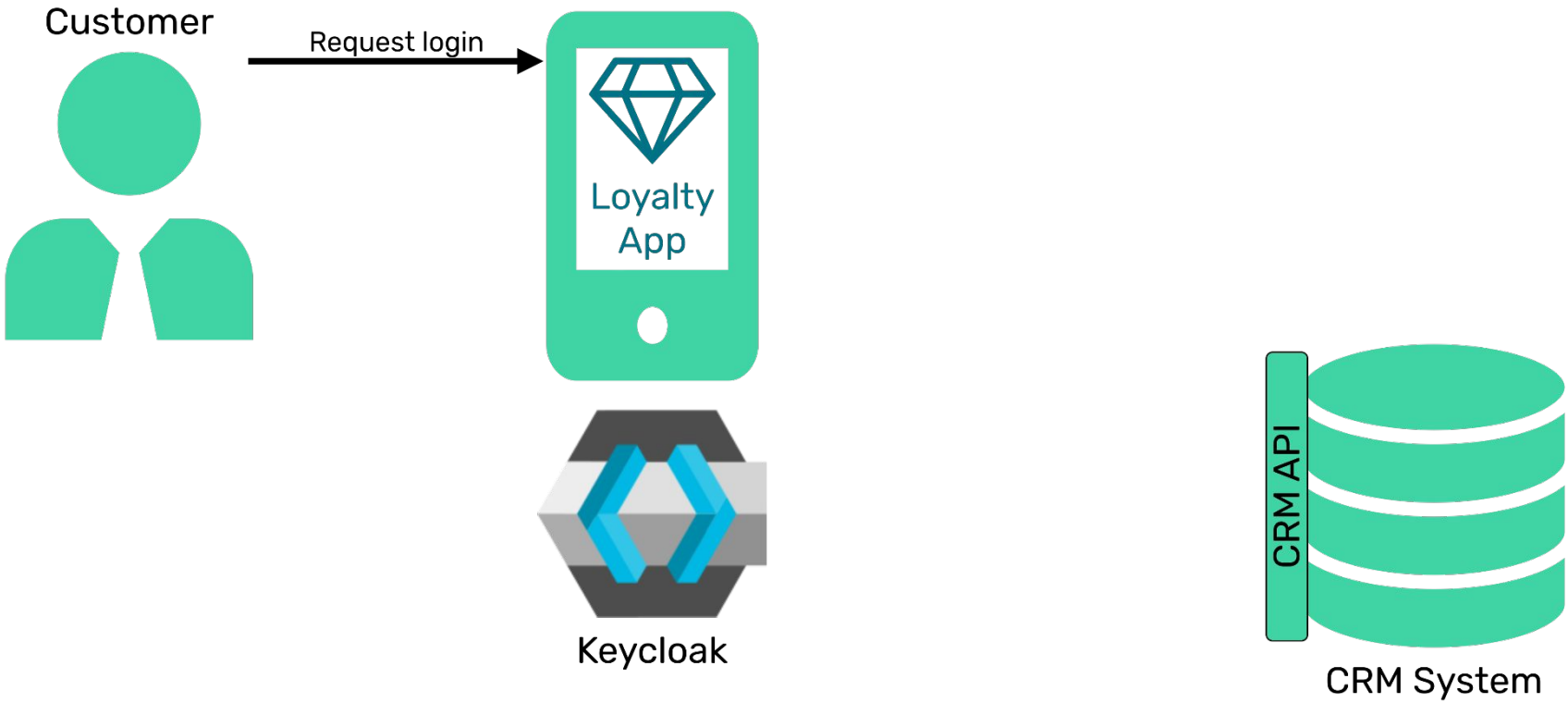
Project Context

- Retailer requests Keycloak integration
  - Enable single sign-on between Loyalty App and webshop
  - Leverage industry standards for authentication and authorization
  - Harness out-of-the-box features when needed
    - Federation of support users
    - Social login
    - Two-factor authentication

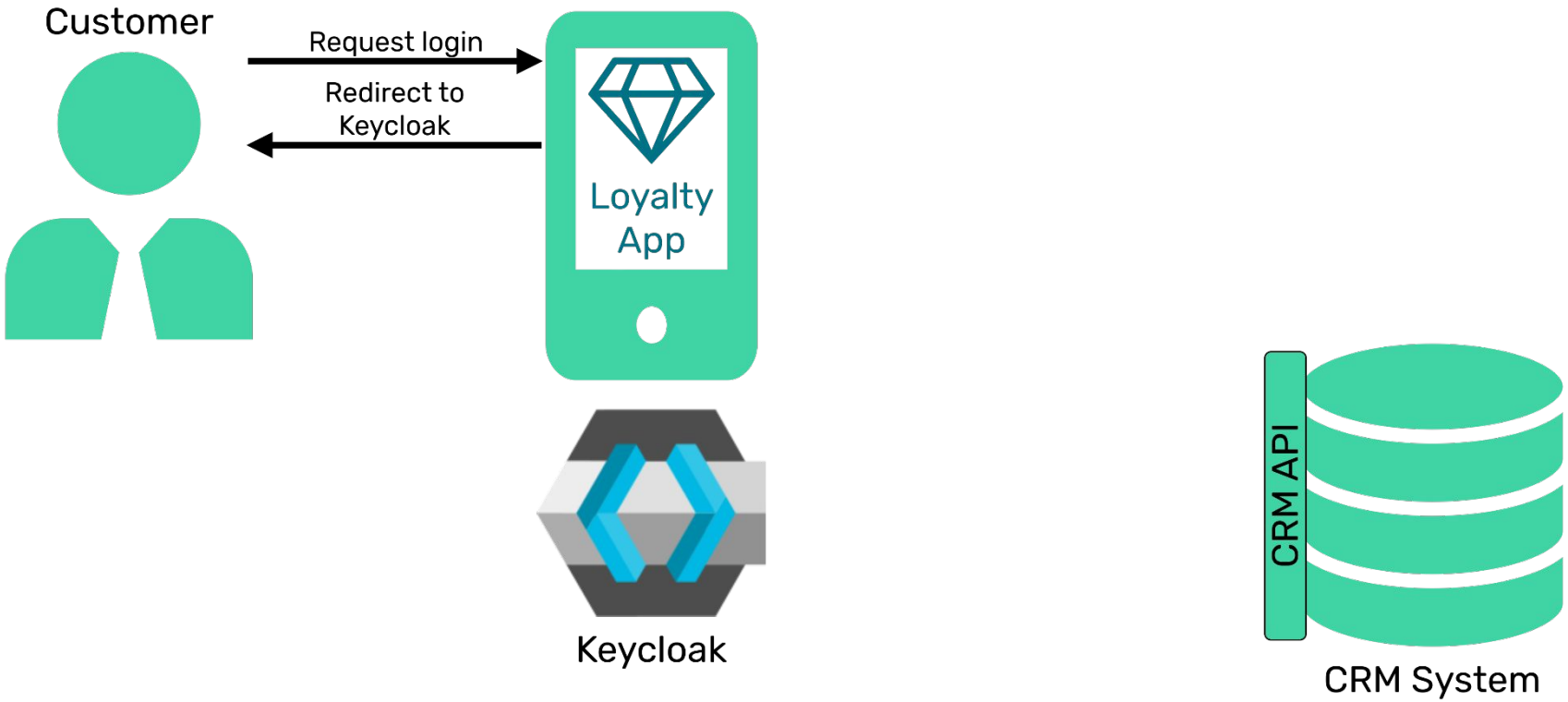
Project Context



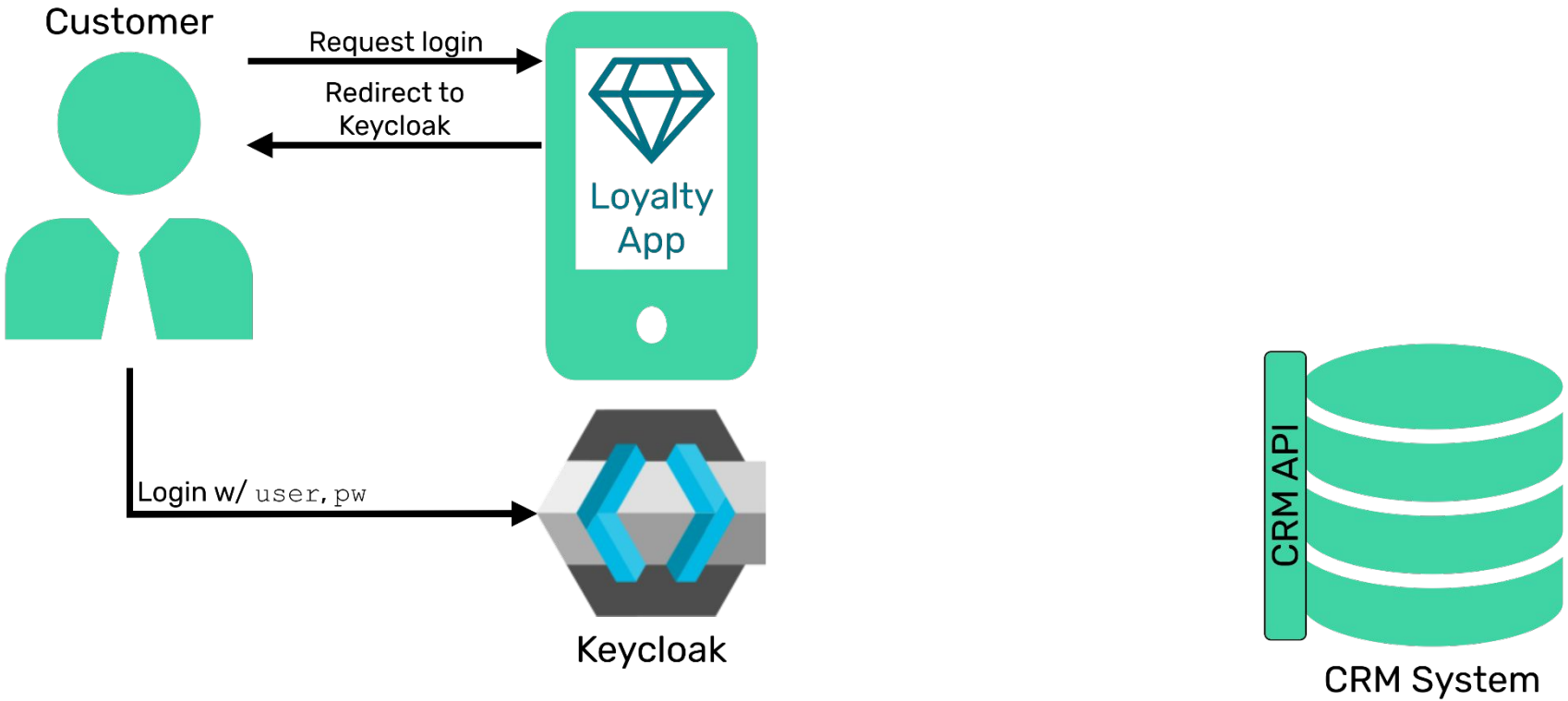
Project Context



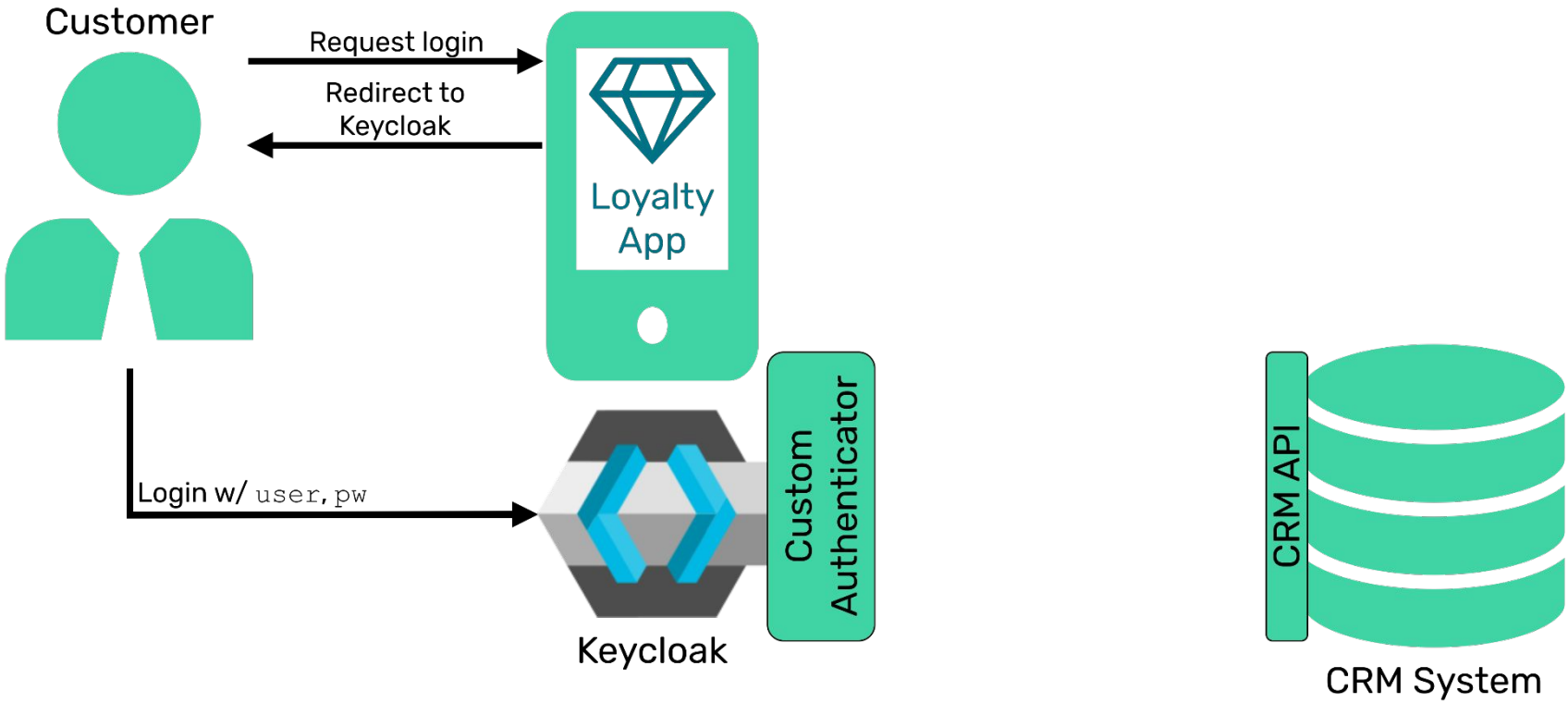
Project Context



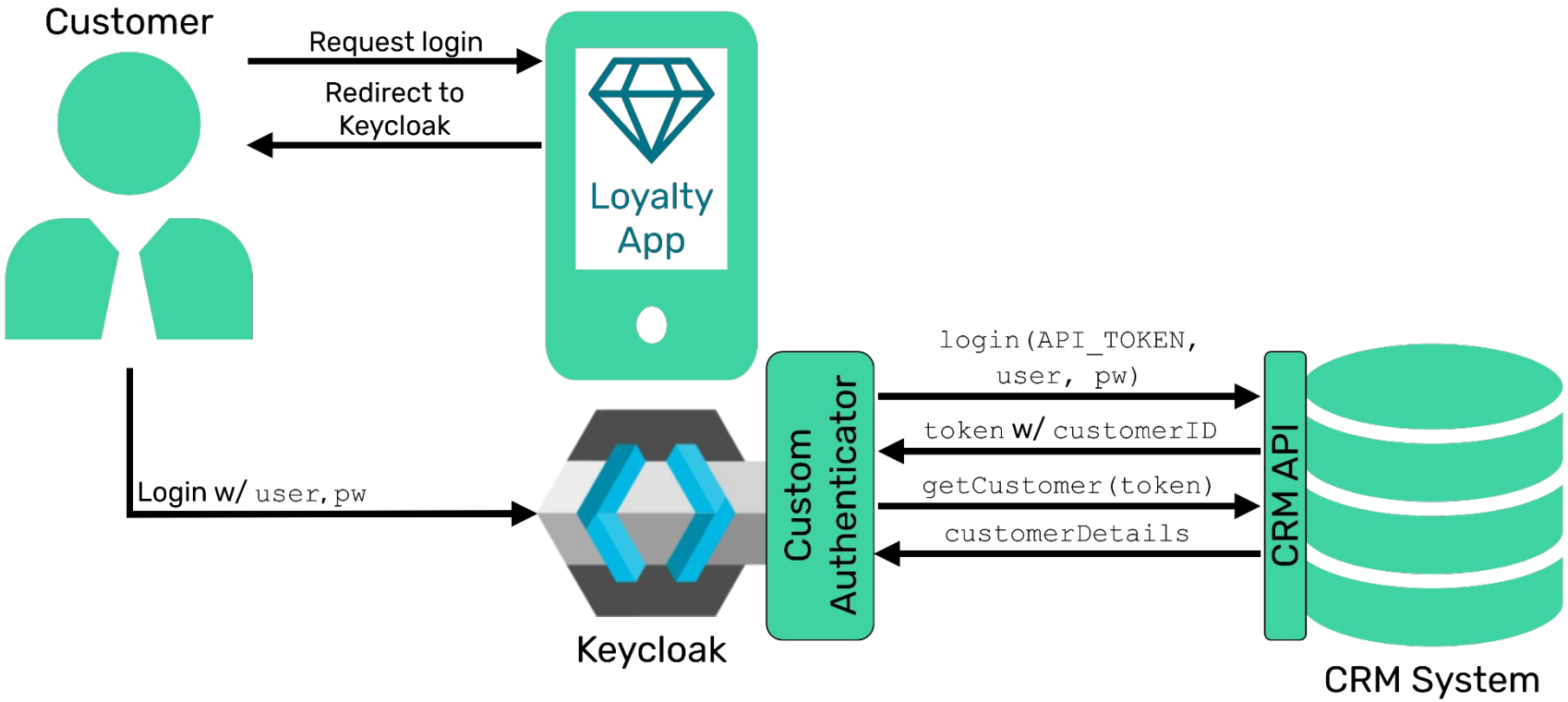
Project Context



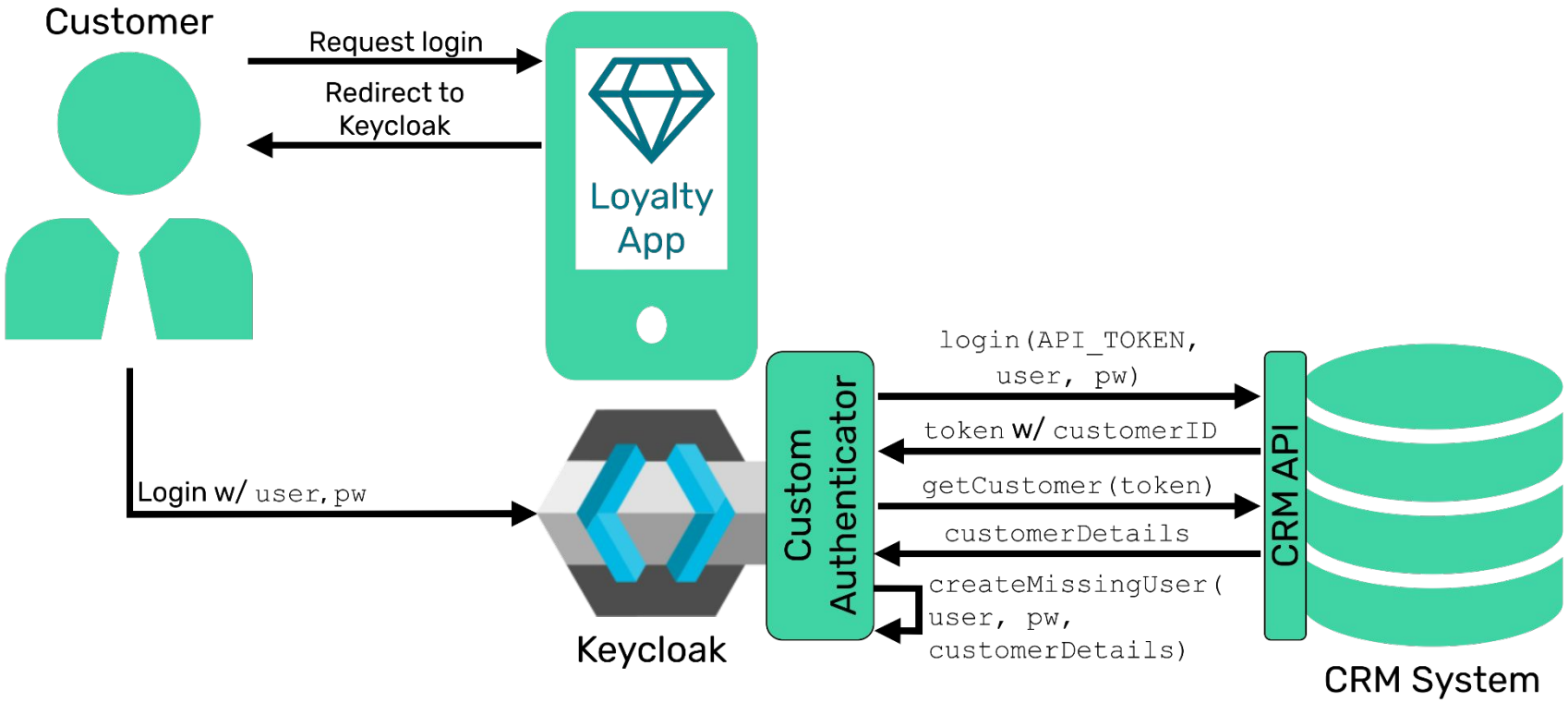
Project Context



Project Context

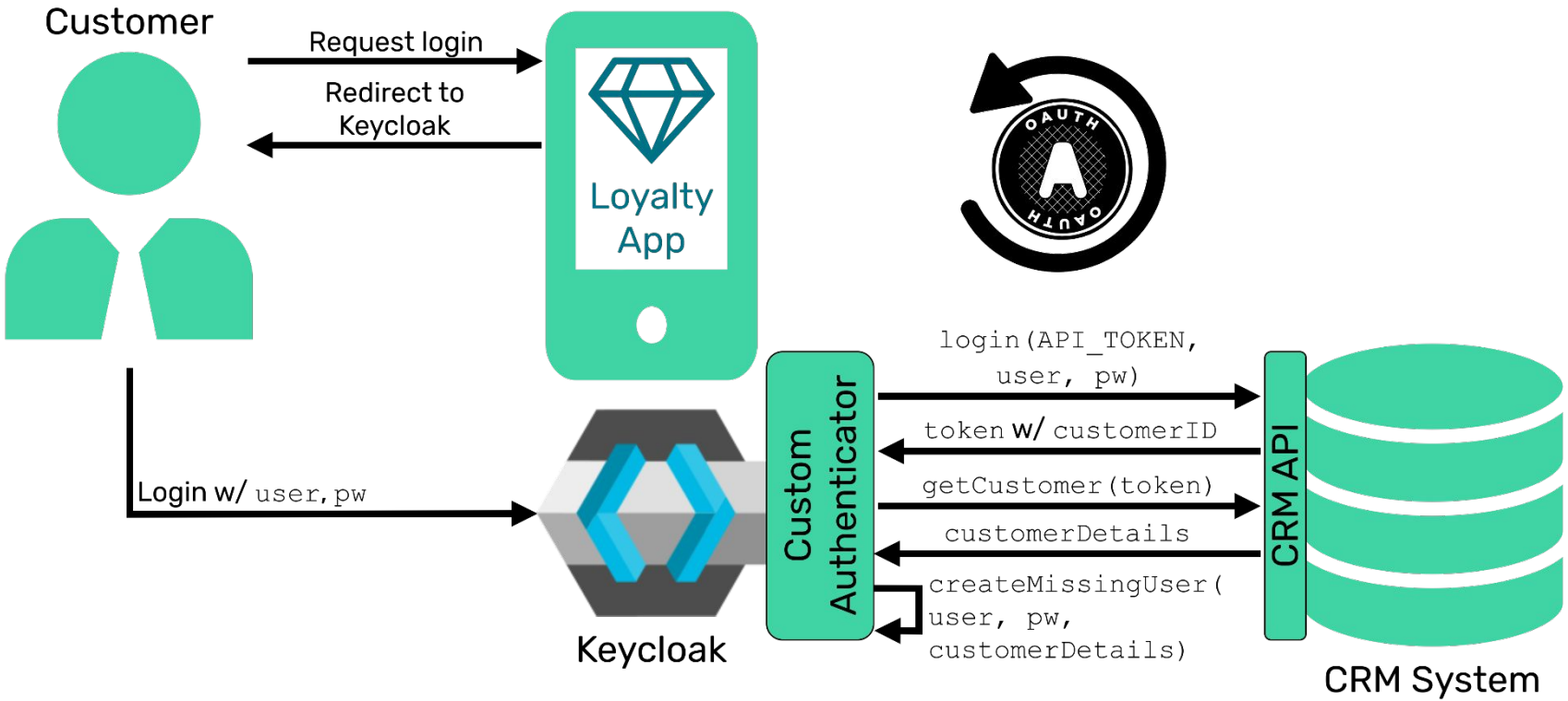


Project Context

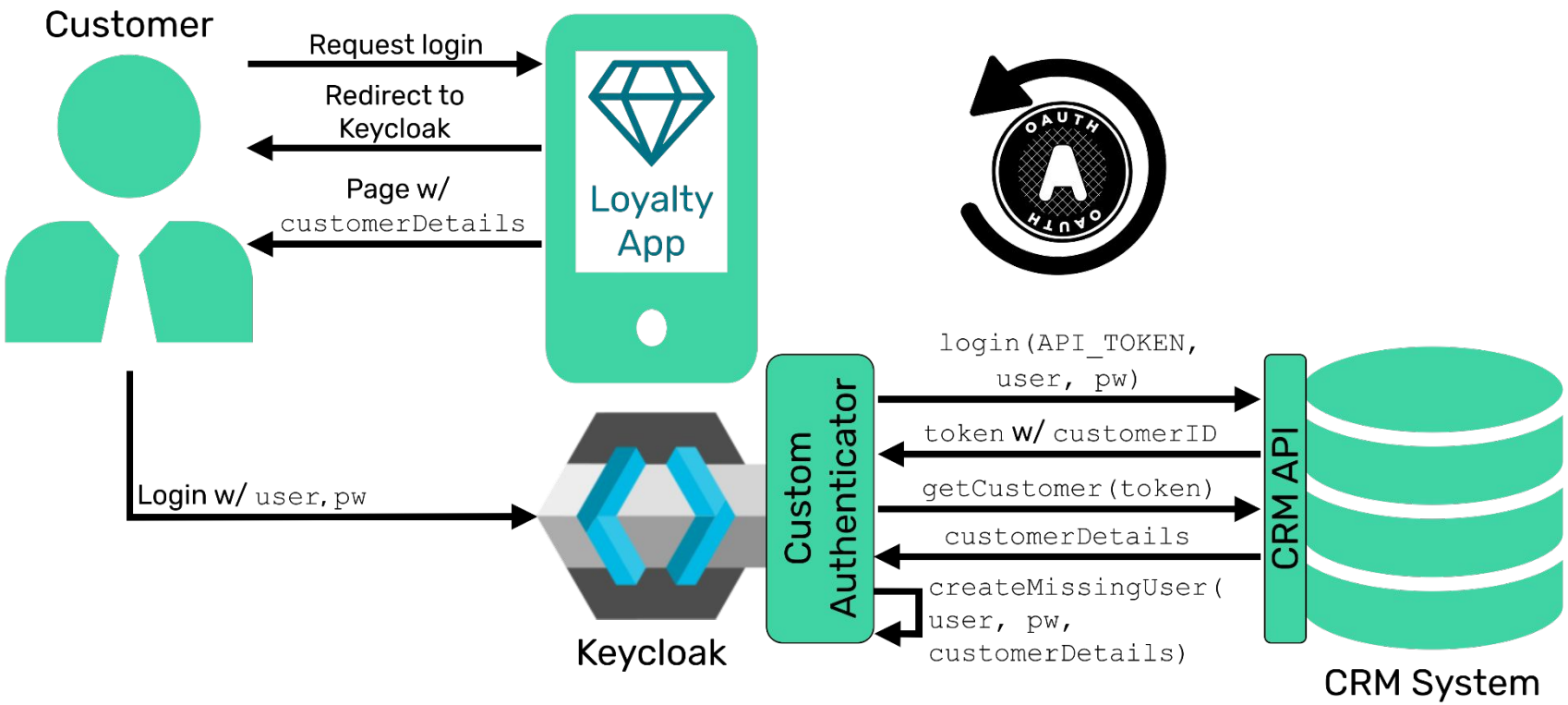


Dr. Florian Rademacher, florian.rademacher@codecentric.de

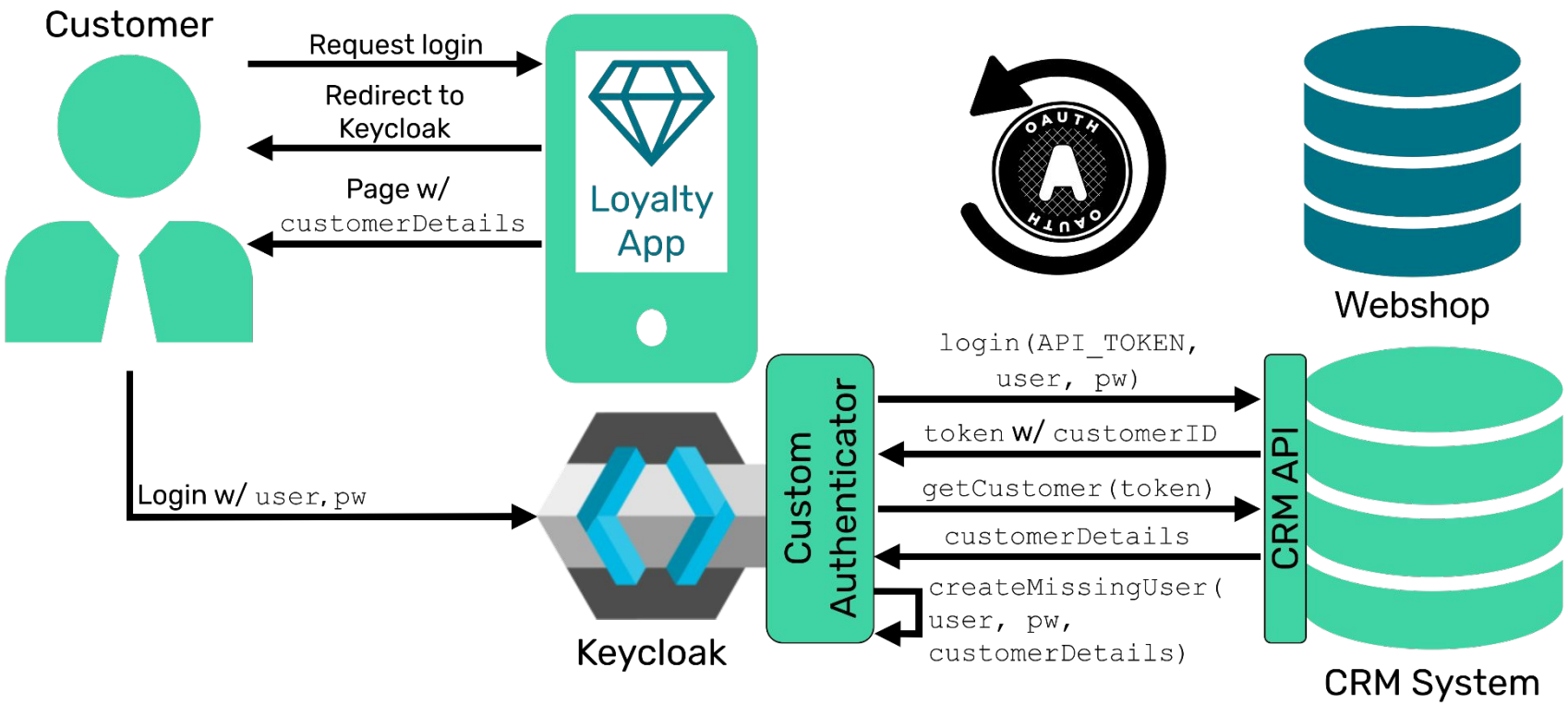
Project Context



Project Context

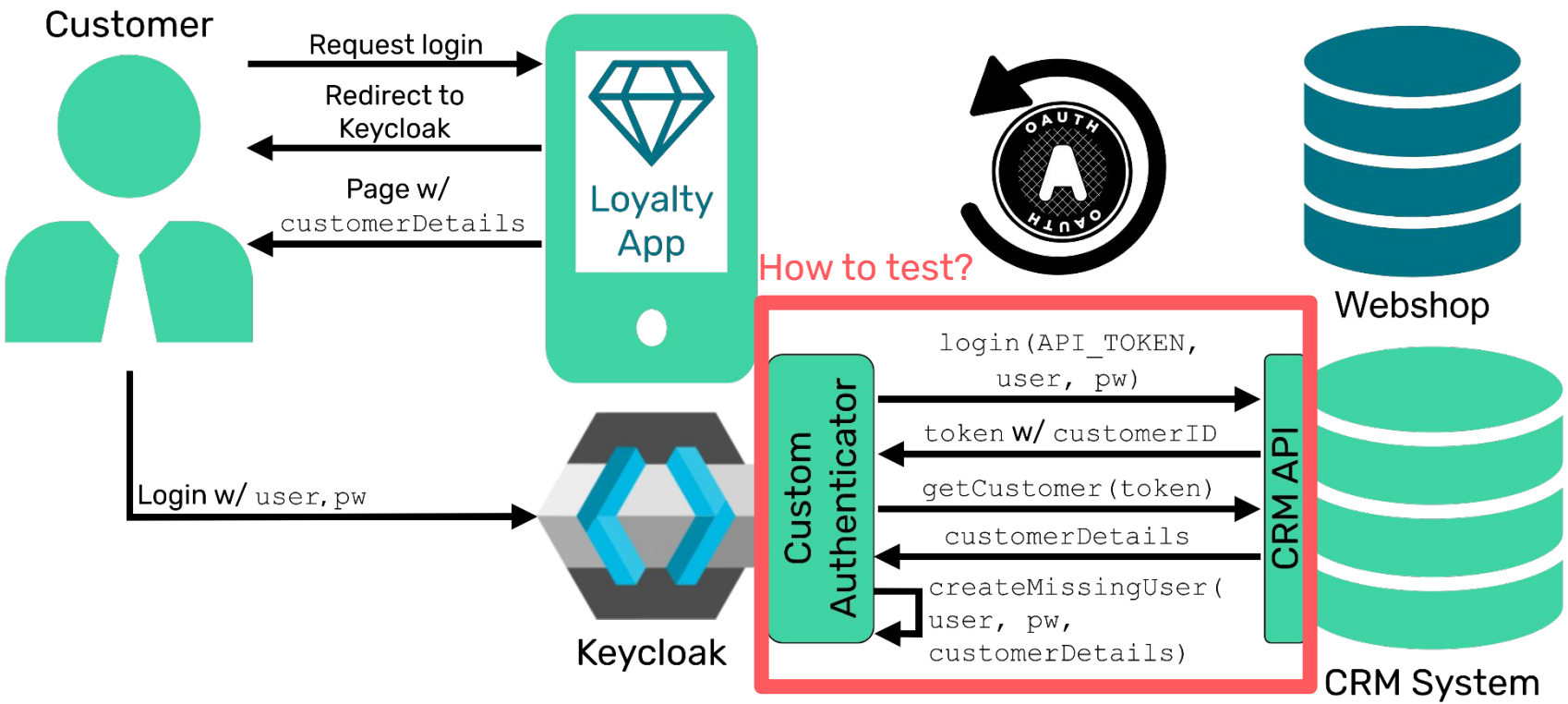


Project Context

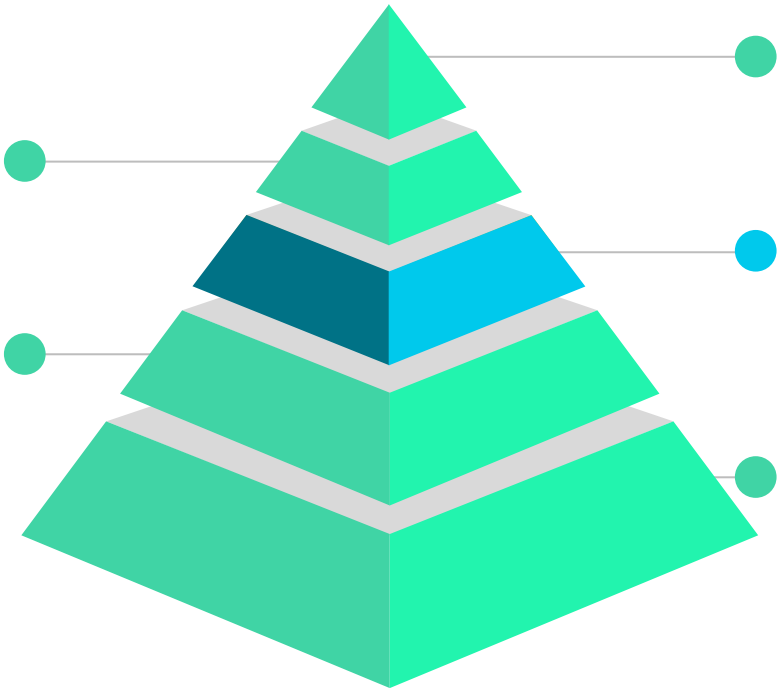


Dr. Florian Rademacher, florian.rademacher@codecentric.de

Project Context



Excursus: The Practical Test Pyramid[1]



Manual Exploratory Tests

Automated End-to-End-Tests

Automated Integration Tests

Test integration of application with application-external services.

Automated Component Tests

Automated Unit Tests

[1]Ham Vocke, The Practical Test Pyramid: https://martinfowler.com/articles/practical-test-pyramid.html

Project Context: Testing Challenges

Project Context: Testing Challenges

- CRM system is developed and operated by another party, without us being able to influence it in any way
  - **Availability of CRM test environment is not guaranteed**
    = Tests are flaky
  - **Stability of our test data is not guaranteed**
    = Tests are flaky *and inconsistent*
  - **Local test setup and boot of CRM system is not supported**
    = Tests are flaky and inconsistent *and slow*
- **Solution: Implement a suitable mock of the CRM system**
  - CRM system is complex and lacks documentation
  - Project scope and budget are constrained

Project Context: Testing Challenges

- CRM system is developed and operated by another party, without us being able to influence it in any way
  - **Availability of CRM test environment is not guaranteed**
    = Tests are flaky
  - **Stability of our test data is not guaranteed**
    = Tests are flaky *and inconsistent*
  - **Local test setup and boot of CRM system is not supported**
    = Tests are flaky and inconsistent *and slow*
- **Solution: Implement a suitable mock of the CRM system**
  - CRM system is complex and lacks documentation
  - Project scope and budget are constrained

Project Context: Testing Challenges

- CRM system is developed and operated by another party, without us being able to influence it in any way
  - **Availability of CRM test environment is not guaranteed**
    = Tests are flaky
  - **Stability of our test data is not guaranteed**
    = Tests are flaky *and inconsistent*
  - **Local test setup and boot of CRM system is not supported**
    = Tests are flaky and inconsistent *and slow*
- **Solution: Implement a suitable mock of the CRM system**
  - CRM system is complex and lacks documentation
  - Project scope and budget are constrained

Project Context: Testing Challenges

- CRM system is developed and operated by another party, without us being able to influence it in any way
  - **Availability of CRM test environment is not guaranteed**
    = Tests are flaky
  - **Stability of our test data is not guaranteed**
    = Tests are flaky *and inconsistent*
  - **Local test setup and boot of CRM system is not supported**
    = Tests are flaky and inconsistent *and slow*
- **Solution: Implement a suitable mock of the CRM system**
  - CRM system is complex and lacks documentation
  - Project scope and budget are constrained

Project Context: Testing Challenges

- CRM system is developed and operated by another party, without us being able to influence it in any way
  - **Availability of CRM test environment is not guaranteed**
    = Tests are flaky
  - **Stability of our test data is not guaranteed**
    = Tests are flaky *and inconsistent*
  - **Local test setup and boot of CRM system is not supported**
    = Tests are flaky and inconsistent *and slow*
- **Solution: Implement a suitable mock of the CRM system**
  - CRM system is complex and lacks documentation
  - Project scope and budget are constrained

Project Context: Testing Challenges

- CRM system is developed and operated by another party, without us being able to influence it in any way
  - **Availability of CRM test environment is not guaranteed**
    = Tests are flaky
  - **Stability of our test data is not guaranteed**
    = Tests are flaky *and inconsistent*
  - **Local test setup and boot of CRM system is not supported**
    = Tests are flaky and inconsistent *and slow*
- **Solution: Implement a suitable mock of the CRM system**
  - CRM system is complex and lacks documentation
  - Project scope and budget are constrained

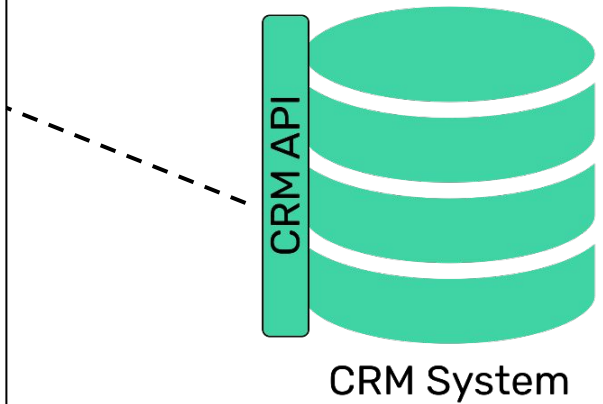
Project Context: Testing Challenges

Project Context: Testing Challenges

Project Context: Testing Challenges

crm-api.yaml

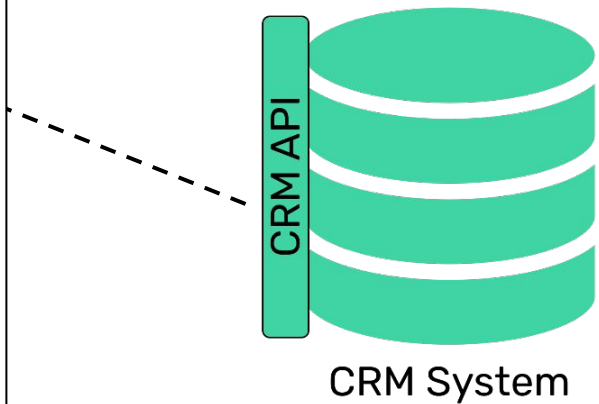
```yaml
openapi: 3.1.1
paths:
 /login:
   post:
     operationId: login_post
     requestBody:
       content:
         application/json:
           schema:
             $ref: "#/components/schemas/LoginRequest"
       required: true
     responses:
       "200":
         description: OK
         content:
           application/json:
             schema:
               $ref: "#/components/schemas/LoginResponse"
       "401":
         description: Unauthorized
     security:
       - api_token: []
...
```



Project Context: Testing Challenges

crm-api.yaml

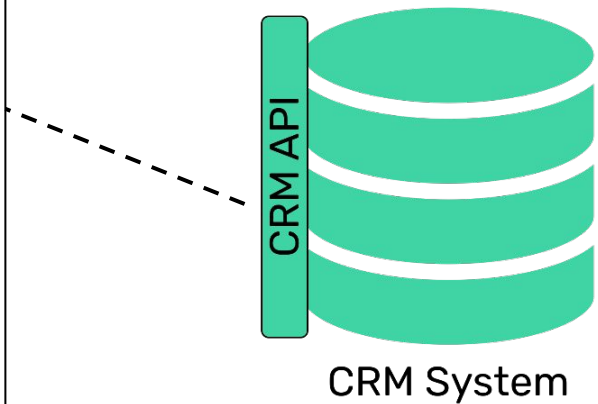
```yaml
openapi: 3.1.1
paths:
 /login:
   post:
     operationId: login_post
     requestBody:
       content:
         application/json:
           schema:
             $ref: "#/components/schemas/LoginRequest"
       required: true
     responses:
       "200":
         description: OK
         content:
           application/json:
             schema:
               $ref: "#/components/schemas/LoginResponse"
       "401":
         description: Unauthorized
     security:
       - api_token: []
...
```



Project Context: Testing Challenges

crm-api.yaml

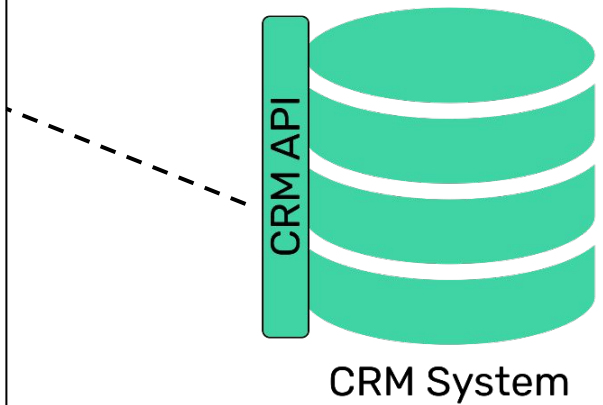
```yaml
openapi: 3.1.1
paths:
 /login:
   post:
     operationId: login_post
     requestBody:
       content:
         application/json:
           schema:
             $ref: "#/components/schemas/LoginRequest"
       required: true
     responses:
       "200":
         description: OK
         content:
           application/json:
             schema:
               $ref: "#/components/schemas/LoginResponse"
       "401":
         description: Unauthorized
     security:
       - api_token: []
...
```



Project Context: Testing Challenges

crm-api.yaml

```yaml
openapi: 3.1.1
paths:
 /login:
   post:
     operationId: login_post
     requestBody:
       content:
         application/json:
           schema:
             $ref: "#/components/schemas/LoginRequest"
       required: true
     responses:
       "200":
         description: OK
         content:
           application/json:
             schema:
               $ref: "#/components/schemas/LoginResponse"
       "401":
         description: Unauthorized
     security:
       - api_token: []
...
```



Excursus: Automated Integration Testing

- Core techniques of automated integration testing
  - **Contracts** to derive effective mock implementations
    - **Agreement between API consumer and provider** on input and output parameters, errors, protocols, and data formats
    - Mock implementations that adhere to successfully concluded contracts can **mimic the behavior of external services as needed**
  - **Configuration management** to provide stable and consistent test environments and data
  - **Virtualization** to improve test repeatability and enable local testing

Excursus: Automated Integration Testing

- Core techniques of automated integration testing
  - **Contracts** to derive effective mock implementations
    - **Agreement between API consumer and provider** on input and output parameters, errors, protocols, and data formats
    - Mock implementations that adhere to successfully concluded contracts can **mimic the behavior of external services as needed**
  - **Configuration management** to provide stable and consistent test environments and data
  - **Virtualization** to improve test repeatability and enable local testing

Excursus: Automated Integration Testing

- Core techniques of automated integration testing
  - **Contracts** to derive effective mock implementations
    - **Agreement between API consumer and provider** on input and output parameters, errors, protocols, and data formats
    - Mock implementations that adhere to successfully concluded contracts can **mimic the behavior of external services as needed**
  - **Configuration management** to provide stable and consistent test environments and data
  - **Virtualization** to improve test repeatability and enable local testing

Excursus: Automated Integration Testing

- Core techniques of automated integration testing
  - **Contracts** to derive effective mock implementations
    - **Agreement between API consumer and provider** on input and output parameters, errors, protocols, and data formats
    - Mock implementations that adhere to successfully concluded contracts can **mimic the behavior of external services as needed**
  - **Configuration management** to provide stable and consistent test environments and data
  - **Virtualization** to improve test repeatability and enable local testing

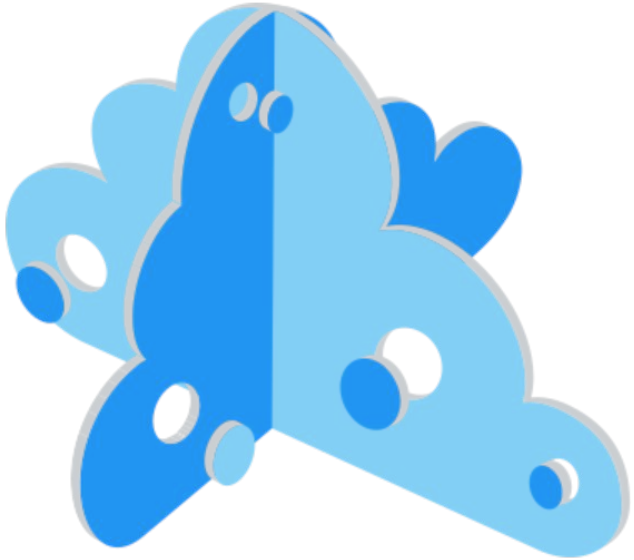
Project Context: Testing Challenges

- CRM system is developed and operated by another party, without us being able to influence it in any way
  - **Availability of test environment is not guaranteed**
    = Tests are flaky
  - **Stability of test data is not guaranteed**
    = Tests are flaky *and inconsistent*
  - **Local test setup and boot of CRM system is not supported**
    = Tests are flaky and inconsistent *and slow*
- **Solution: Implement a suitable mock of the CRM system**
  - CRM system is complex and lacks documentation
  - Project scope and budget are constrained

Project Context: Testing Challenges

- CRM system is developed and operated by another party, without us being able to influence it in any way
  - **Availability of test environment is not guaranteed**
    = Tests are flaky
  - **Stability of test data is not guaranteed**
    = Tests are flaky *and inconsistent*
  - **Local test setup and boot of CRM system is not supported**
    = Tests are flaky and inconsistent *and slow*
- **Solution: Implement a suitable mock of the CRM system**
  - CRM system is complex and lacks documentation
  - Project scope and budget are constrained

How to overcome this limitation?



MICROCKS

API Mocking & Testing

Microcks

- **Open source multi-protocol tool for API mocking**, simulation, and testing
- **Automated mock derivation and provisioning** from API definitions, e.g., in OpenAPI but also AsyncAPI, GraphQL, SOAP…
- **Low-effort features** for subsequent mock refinement
  - Specification-based mock requests, responses, and behavior
  - Parameter constraints
  - Dispatchers for flexible mock routing
  - Dynamic mock content
  - Stateful mocks
- **Testcontainers support**
- https://microcks.io

Microcks

- **Open source multi-protocol tool for API mocking**, simulation, and testing
- **Automated mock derivation and provisioning** from API definitions, e.g., in OpenAPI but also AsyncAPI, GraphQL, SOAP…
- **Low-effort features** for subsequent mock refinement
  - Specification-based mock requests, responses, and behavior
  - Parameter constraints
  - Dispatchers for flexible mock routing
  - Dynamic mock content
  - Stateful mocks
- **Testcontainers support**
- https://microcks.io

Microcks

- **Open source multi-protocol tool for API mocking**, simulation, and testing
- **Automated mock derivation and provisioning** from API definitions, e.g., in OpenAPI but also AsyncAPI, GraphQL, SOAP…
- **Low-effort features** for subsequent mock refinement
  - Specification-based mock requests, responses, and behavior
  - Parameter constraints
  - Dispatchers for flexible mock routing
  - Dynamic mock content
  - Stateful mocks
- **Testcontainers support**
- https://microcks.io

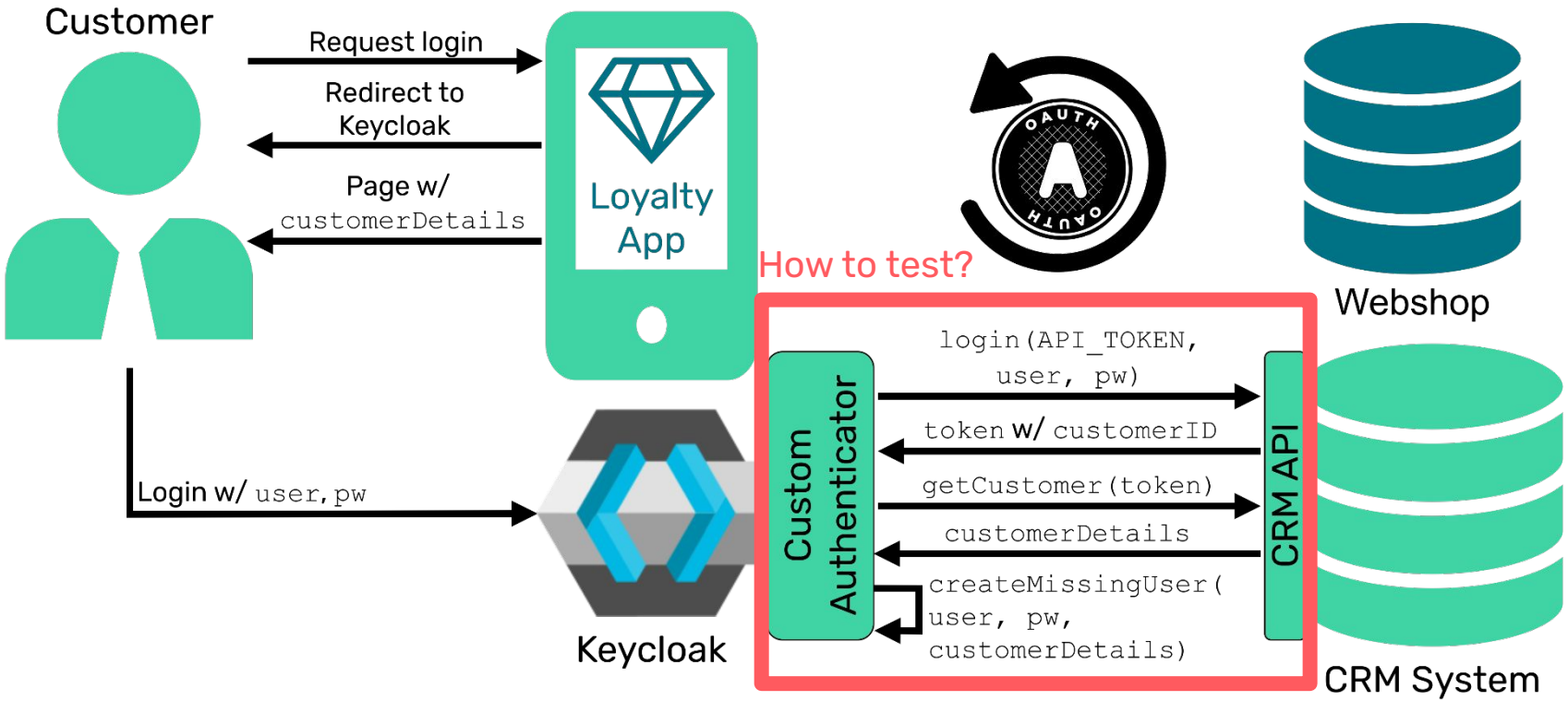
Microcks

- **Open source multi-protocol tool for API mocking**, simulation, and testing
- **Automated mock derivation and provisioning** from API definitions, e.g., in OpenAPI but also AsyncAPI, GraphQL, SOAP…
- **Low-effort features** for subsequent mock refinement
  - Specification-based mock requests, responses, and behavior
  - Parameter constraints
  - Dispatchers for flexible mock routing
  - Dynamic mock content
  - Stateful mocks
- **Testcontainers support**
- https://microcks.io

Microcks

- **Open source multi-protocol tool for API mocking**, simulation, and testing
- **Automated mock derivation and provisioning** from API definitions, e.g., in OpenAPI but also AsyncAPI, GraphQL, SOAP…
- **Low-effort features** for subsequent mock refinement
  - Specification-based mock requests, responses, and behavior
  - Parameter constraints
  - Dispatchers for flexible mock routing
  - Dynamic mock content
  - Stateful mocks
- **Testcontainers support**
- https://microcks.io

Keycloak Integration Testing with Microcks



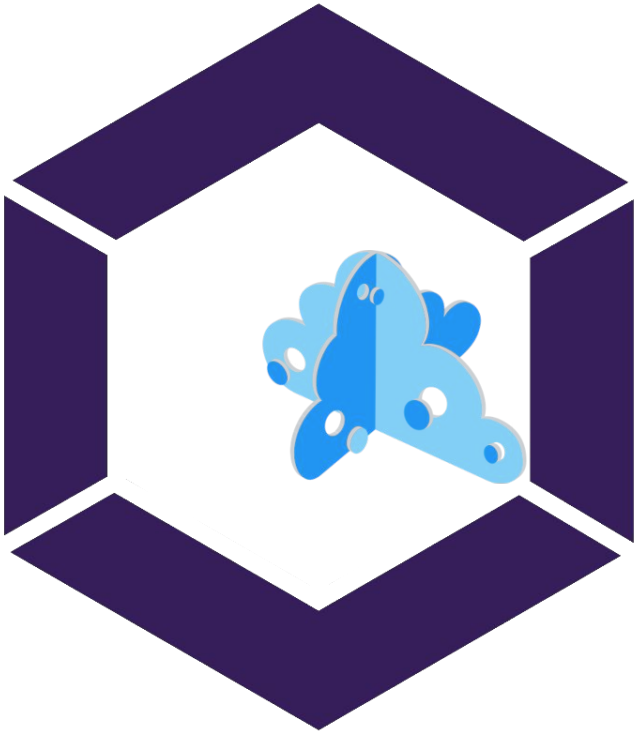
Dr. Florian Rademacher, florian.rademacher@codecentric.de

Keycloak Integration Testing with Microcks: Test Environment

Keycloak Testcontainer with Custom Authenticator

Keycloak Integration Testing with Microcks: Test Environment

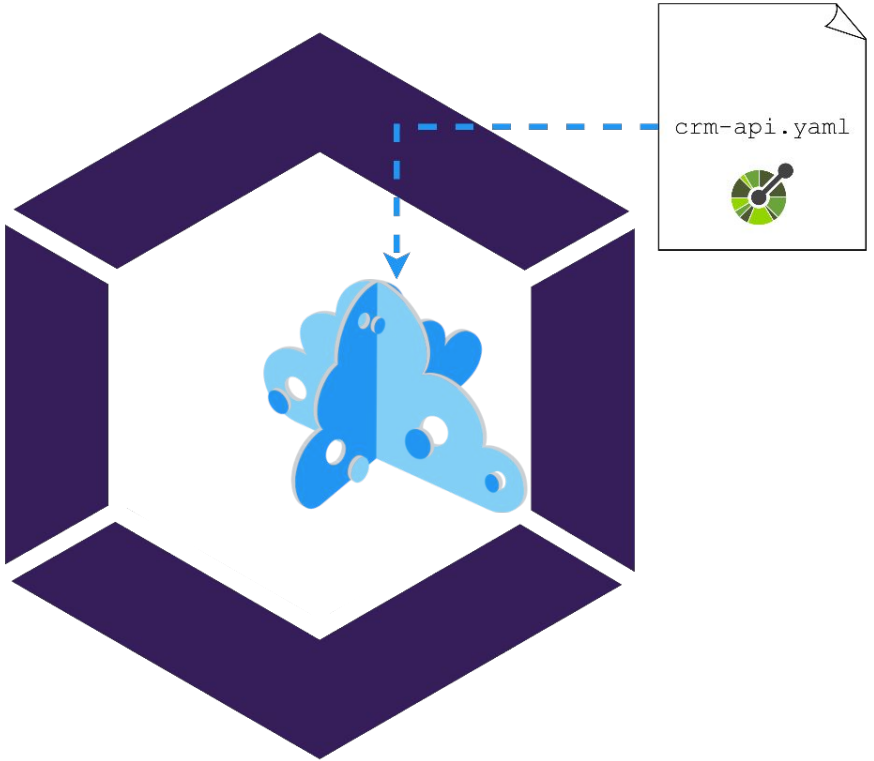
Keycloak Testcontainer with Custom Authenticator



CRM System Mock Powered By Microcks Testcontainer

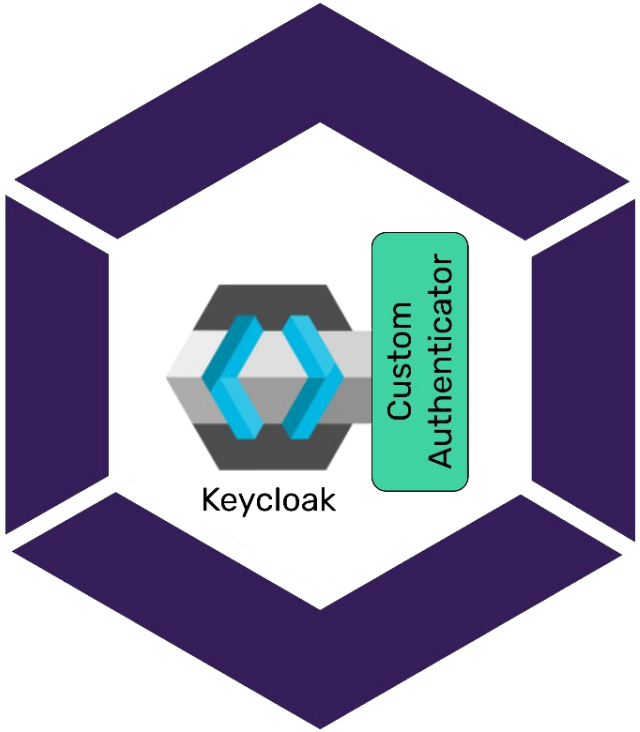
Keycloak Integration Testing with Microcks: Test Environment

Keycloak Testcontainer with Custom Authenticator

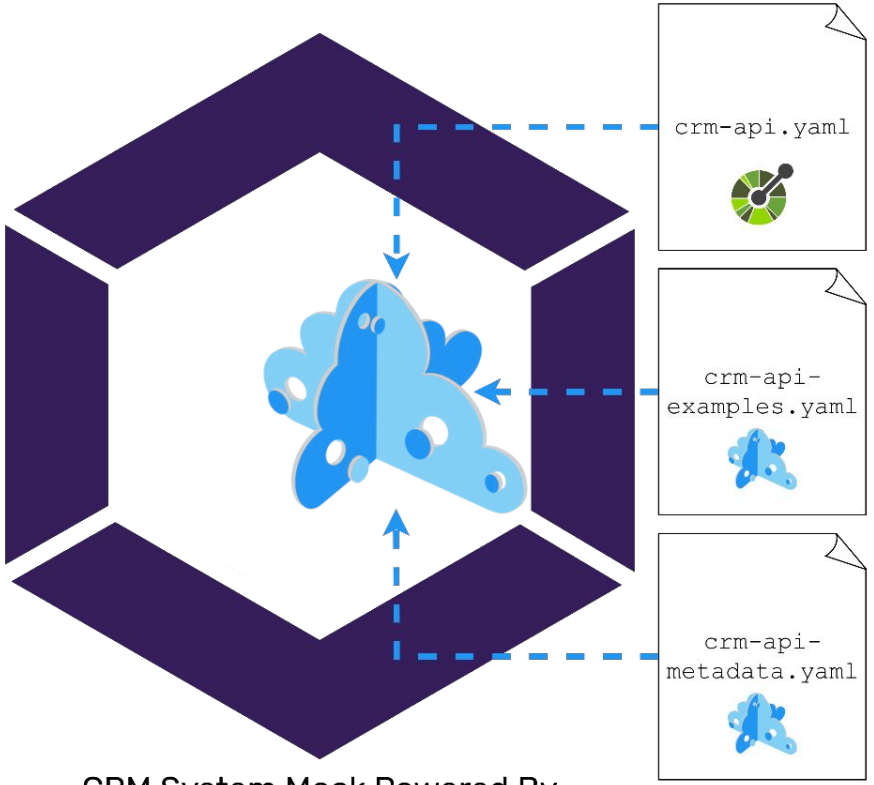


CRM System Mock Powered By Microcks Testcontainer

Keycloak Integration Testing with Microcks: Test Environment

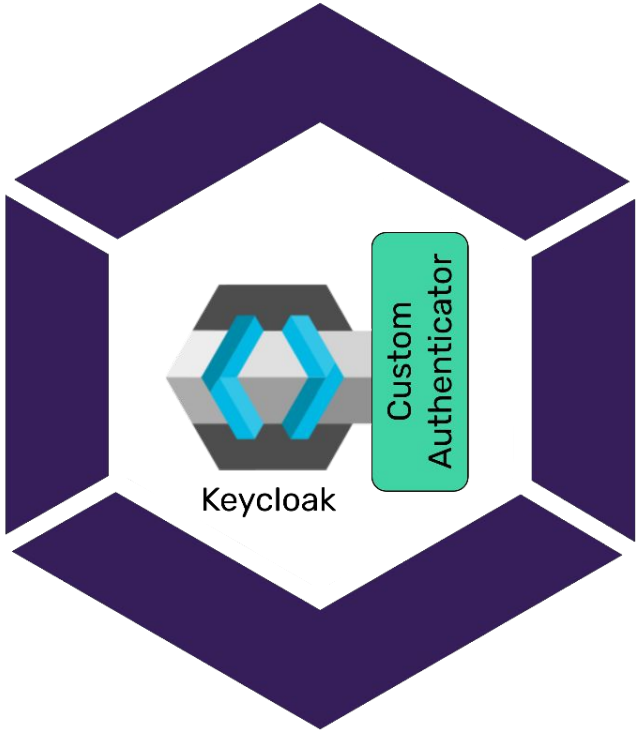


Keycloak Testcontainer with Custom Authenticator

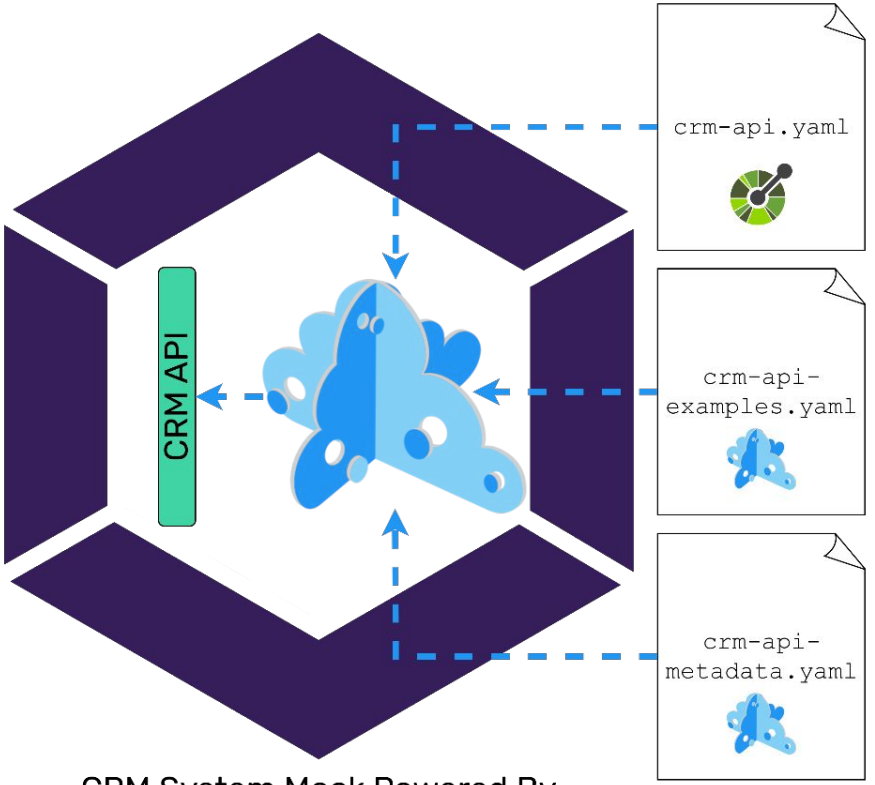


CRM System Mock Powered By Microcks Testcontainer

Keycloak Integration Testing with Microcks: Test Environment

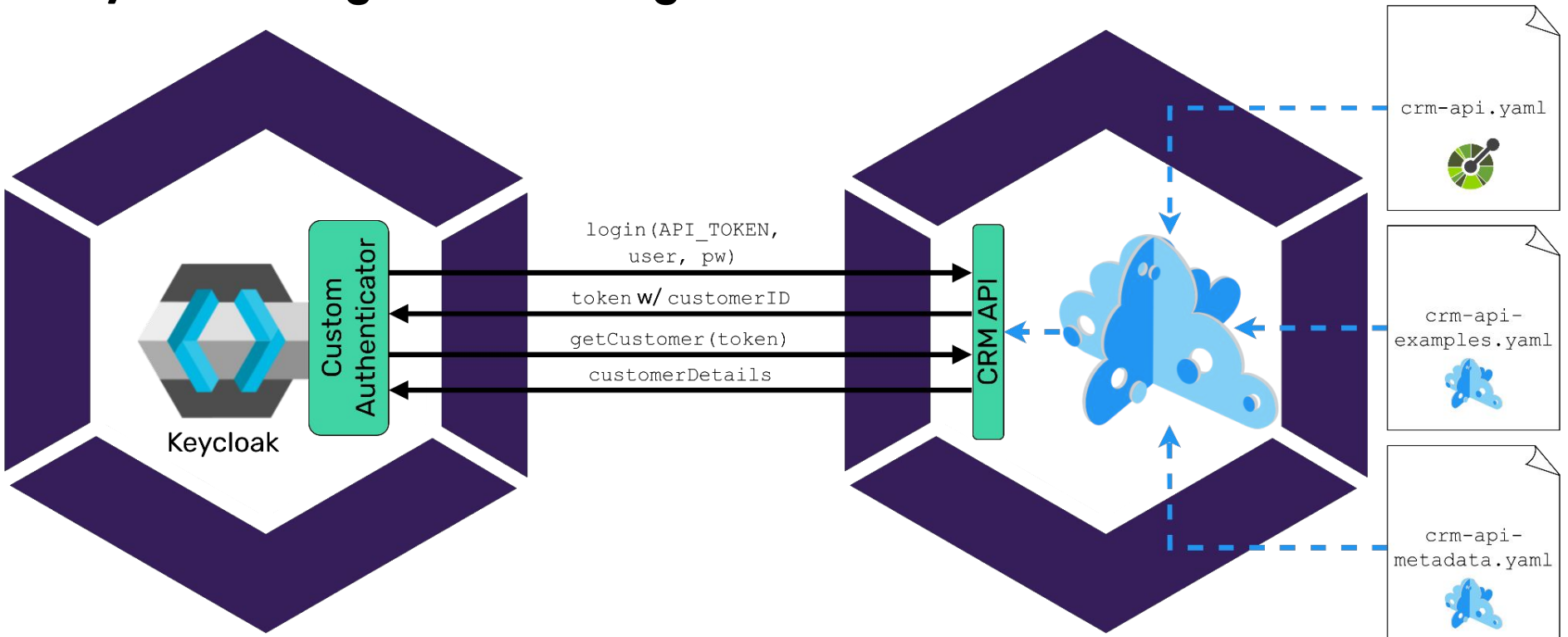


Keycloak Testcontainer with Custom Authenticator



CRM System Mock Powered By Microcks Testcontainer

Keycloak Integration Testing with Microcks: Test Environment



Keycloak Testcontainer with Custom Authenticator

CRM System Mock Powered By Microcks Testcontainer

Keycloak Integration Testing with Microcks: Microcks Setup

- **OpenAPI definition of CRM API:** `crm-api.yaml`
  - Derivation of basic contract information
- **Microcks API Examples:** `crm-api-examples.yaml`
  - Non-intrusive definition of mock requests and responses
- **Microcks API Metadata:** `crm-api-metadata.yaml`
  - Specification of mock behavior

Keycloak Integration Testing with Microcks: Microcks Setup

- **OpenAPI definition of CRM API:** `crm-api.yaml`
  - Derivation of basic contract information

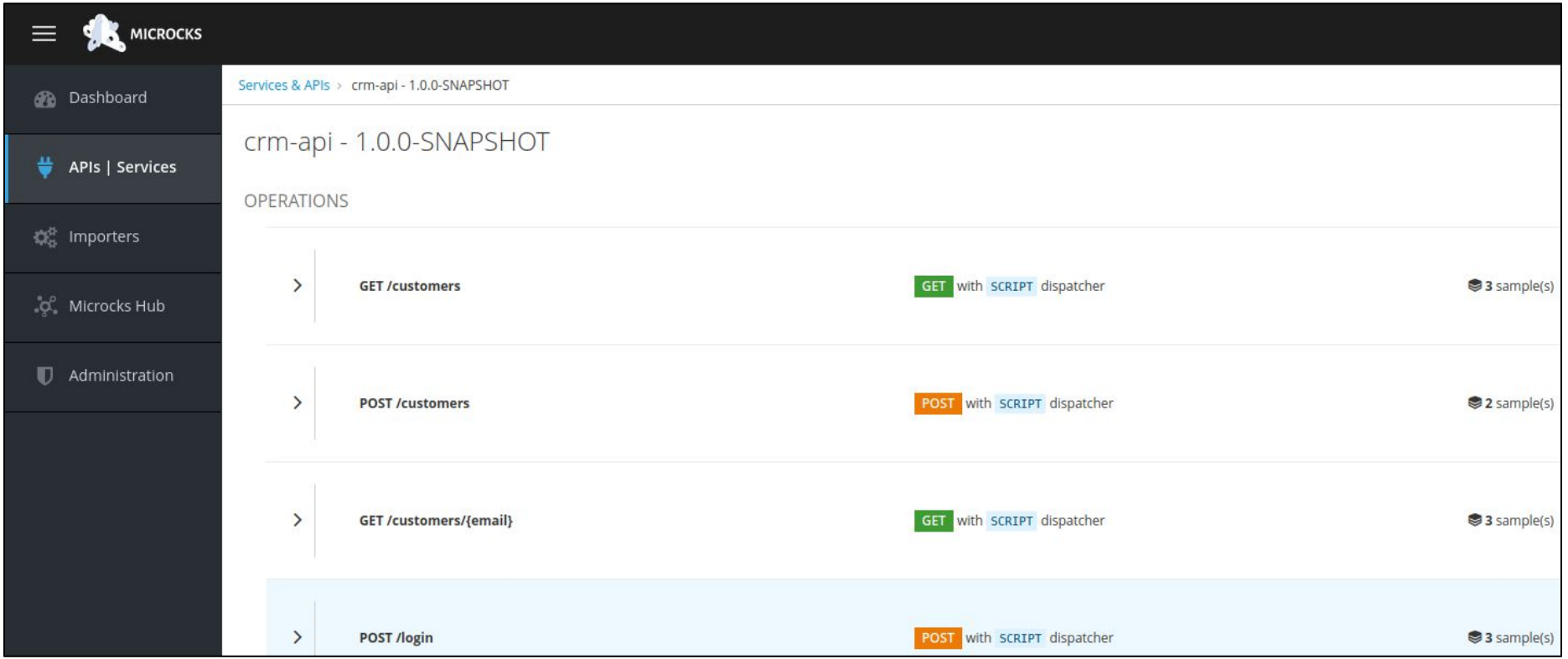
- **Microcks API Examples:** `crm-api-examples.yaml`
  - Non-intrusive definition of mock requests and responses

- **Microcks API Metadata:** `crm-api-metadata.yaml`
  - Specification of mock behavior

Keycloak Integration Testing with Microcks: Microcks Setup

- **OpenAPI definition of CRM API:** `crm-api.yaml`
  - Derivation of basic contract information
- **Microcks API Examples:** `crm-api-examples.yaml`
  - Non-intrusive definition of mock requests and responses
- **Microcks API Metadata:** `crm-api-metadata.yaml`
  - Specification of mock behavior

Keycloak Integration Testing with Microcks: Microcks Setup



Landing page of the Microcks Testcontainer for the CRM System Mock

Keycloak Integration Testing with Microcks: Common Use Cases

- `login(API_TOKEN, user, pw)`: Test Credential Definition

```yaml
apiVersion: mocks.microcks.io/v1alpha1
kind: APIExamples
operations:
 "POST /login":
   existing_customer:
     request:
       mediaType: application/json
       body:
         email: "john.doe@example.com"
         password: "securePassword1!"
     response:
       status: "200"
       mediaType: application/json
       body:
         login_token: "eyJhbGciOiJIUzI1NiJ9..."
   wrong_credentials:
     response:
       status: "401"
```

crm-api-examples.yaml

Microcks API Examples to define mock requests and responses

Keycloak Integration Testing with Microcks: Common Use Cases

- `login(API_TOKEN, user, pw)`: Test Credential Definition

```yaml
apiVersion: mocks.microcks.io/v1alpha1
kind: APIExamples
operations:
 "POST /login":
   existing_customer:
     request:
       mediaType: application/json
       body:
         email: "john.doe@example.com"
         password: "securePassword1!"
     response:
       status: "200"
       mediaType: application/json
       body:
         login_token: "eyJhbGciOiJIUzI1NiJ9..."
   wrong_credentials:
     response:
       status: "401"
```

crm-api-examples.yaml

Microcks API Examples to define mock requests and responses

Keycloak Integration Testing with Microcks: Common Use Cases

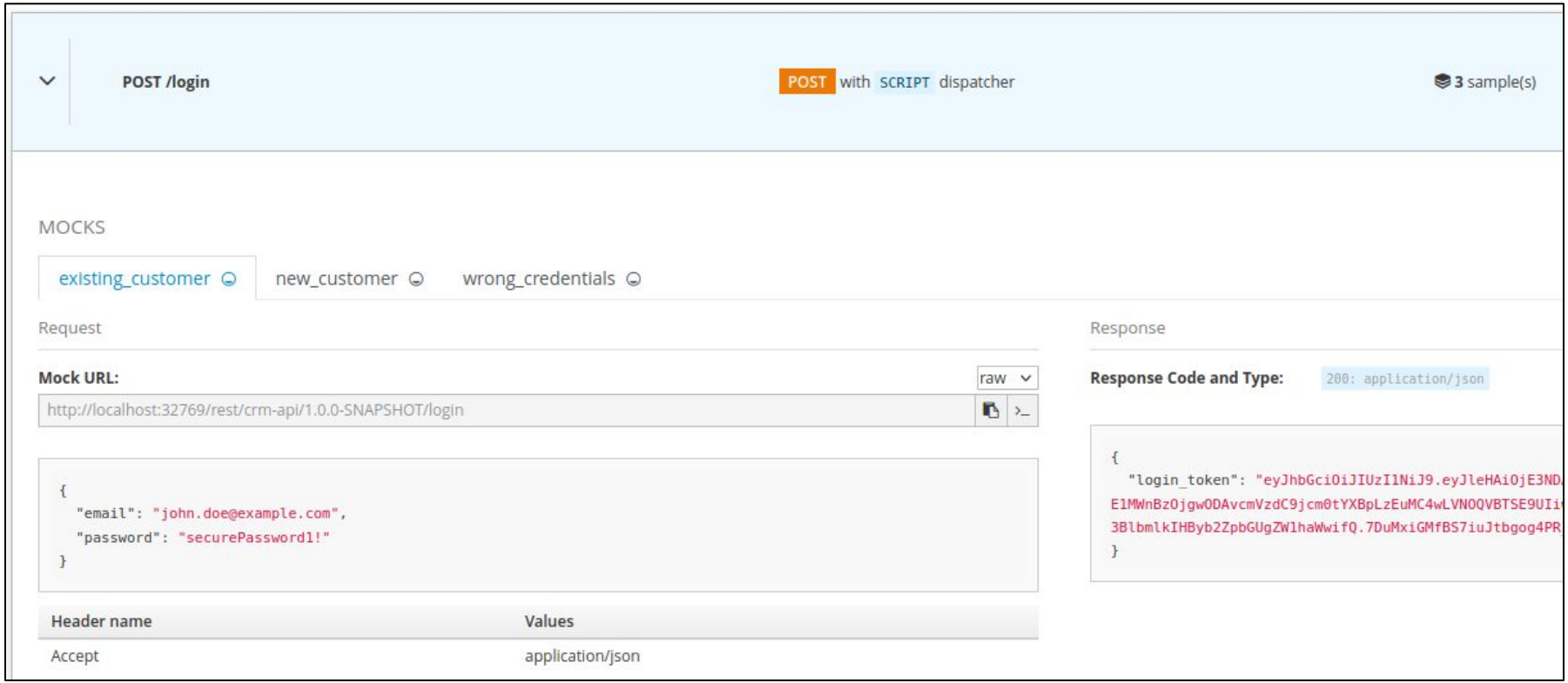
- `login(API_TOKEN, user, pw)`: Test Credential Definition

crm-api-examples.yaml

```yaml
apiVersion: mocks.microcks.io/v1alpha1
kind: APIExamples
operations:
 "POST /login":
   existing_customer:
     request:
       mediaType: application/json
       body:
         email: "john.doe@example.com"
         password: "securePassword1!"
     response:
       status: "200"
       mediaType: application/json
       body:
         login_token: "eyJhbGciOiJIUzI1NiJ9..."
   wrong_credentials:
     response:
       status: "401"
```

Microcks API Examples to define mock requests and responses

Keycloak Integration Testing with Microcks: Common Use Cases



Details of the mocked `login()` endpoint in the Microcks Testcontainer for the CRM System Mock

Keycloak Integration Testing with Microcks: Common Use Cases

- `login(API_TOKEN, user, pw)`: Access Restriction with API Token

crm-api-metadata.yaml

```yaml
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
 "POST /login":
   parameterConstraints:
     - name: Authorization
       in: header
       required: true
       recopy: false
       mustMatchRegexp: "^Bearer\\s\\QeyJhbGciOiJIUzI1NiIs...\\E$"
```

Microcks Parameter Constraints with regexes to define fixed Bearer tokens

Keycloak Integration Testing with Microcks: Common Use Cases

- `login(API_TOKEN, user, pw)`: Credential Validation

```
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
 "POST /login":
   ...
   dispatcher: SCRIPT
   dispatcherRules: |-
     def request = new groovy.json.JsonSlurper()
       .parseText(mockRequest.requestContent)
     if (request.email == "john.doe@example.com" &&
         request.password == "securePassword1!")
       return "existing_customer"

     return "wrong_credentials"
```

crm-api-metadata.yaml

Microcks SCRIPT Dispatchers with JSON parsing support to select mock responses from request data

Keycloak Integration Testing with Microcks: Common Use Cases

- `login(API_TOKEN, user, pw)`: Creder

```
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
 "POST /login":
   ...
   dispatcher: SCRIPT
   dispatcherRules: |-
     def request = new groovy.json.JsonSlurper()
       .parseText(mockRequest.requestContent)
     if (request.email == "john.doe@example.com" &&
         request.password == "securePassword1!")
       return "existing_customer"

     return "wrong_credentials"
```

crm-api-examples.y

```
kind: APIExamples
operations:
 "POST /login":
   existing_customer:
     request:
       mediaType: application/json
       body:
         email: "john.doe@example.com"
         password: "securePassword1!"
     response:
       status: "200"
       mediaType: application/json
       body:
         login_token: "eyJhb..."
   wrong_credentials:
     response:
       status: "401"
```

Keycloak Integration Testing with Microcks: Common Use Cases

- login(API_TOKEN, user, pw): Creder

```
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
 "POST /login":
   ...
   dispatcher: SCRIPT
   dispatcherRules: |-
     def request = new groovy.json.JsonSlurper()
       .parseText(mockRequest.requestContent)
     if (request.email == "john.doe@example.com" &&
         request.password == "securePassword1!")
       return "existing_customer"

     return "wrong_credentials"
```

crm-api-examples.y

```
kind: APIExamples
operations:
 "POST /login":
   existing_customer:
     request:
       mediaType: application/json
       body:
         email: "john.doe@example.com"
         password: "securePassword1!"
     response:
       status: "200"
       mediaType: application/json
       body:
         login_token: "eyJhb..."
   wrong_credentials:
     response:
       status: "401"
```

Keycloak Integration Testing with Microcks: Common Use Cases

- login(API_TOKEN, user, pw): Creder

```
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
 "POST /login":
   ...
   dispatcher: SCRIPT
   dispatcherRules: |-
     def request = new groovy.json.JsonSlurper()
       .parseText(mockRequest.requestContent)
     if (request.email == "john.doe@example.com" &&
         request.password == "securePassword1!")
       return "existing_customer"

     return "wrong_credentials"
```

crm-api-examples.y

```
kind: APIExamples
operations:
 "POST /login":
   existing_customer:
     request:
       mediaType: application/json
       body:
         email: "john.doe@example.com"
         password: "securePassword1!"
     response:
       status: "200"
       mediaType: application/json
       body:
         login_token: "eyJhb..."
   wrong_credentials:
     response:
       status: "401"
```

Keycloak Integration Testing with Microcks: Common Use Cases

- login(API_TOKEN, user, pw): Creder

```
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
 "POST /login":
   ...
   dispatcher: SCRIPT
   dispatcherRules: |-
     def request = new groovy.json.JsonSlurper()
       .parseText(mockRequest.requestContent)
     if (request.email == "john.doe@example.com" &&
         request.password == "securePassword1!")
       return "existing_customer"

     return "wrong_credentials"
```

crm-api-examples.y

```
kind: APIExamples
operations:
 "POST /login":
   existing_customer:
     request:
       mediaType: application/json
       body:
         email: "john.doe@example.com"
         password: "securePassword1!"
     response:
       status: "200"
       mediaType: application/json
       body:
         login_token: "eyJhb..."
   wrong_credentials:
     response:
       status: "401"
```

Keycloak Integration Testing with Microcks: Common Use Cases

- `getCustomer(token)`: Token-Based Data Retrieval

crm-api-metadata.yaml

```
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
  "GET /customers":
    dispatcher: SCRIPT
    dispatcherRules: |-
      def headers = mockRequest.getRequestHeaders()
      if (headers.hasValues("Authorization")) {
        def authHeader = headers.get("Authorization", "null")
        switch(authHeader) {
          case "Bearer eyJhbGciOiJIUzI1NiJ9...":
            return "existing_customer"
        }
      }
      return "unauthorized_or_unknown_customer"
```

Microcks SCRIPT Dispatchers with access to request headers to select mock responses from request data

Keycloak Integration Testing with Microcks: Common Use Cases

- `getCustomer(token)`: Dynamic Customer Detail Data

crm-api-examples.yaml

```yaml
apiVersion: mocks.microcks.io/v1alpha1
kind: APIExamples
operations:
  "GET /customers":
    existing_customer:
      request:
      response:
        status: "200"
        mediaType: application/json
        body:
          firstname: "John"
          lastname: "Doe"
          address: "{{ randomStreetName() }} {{ randomInt(1, 500) }},
            {{ randomInt(1000, 65635) }} {{ randomCity() }},
            {{ randomCountry() }}"
```

Microcks Mock Templates and Function Expressions to specify dynamic mock content

Keycloak Integration Testing with Microcks: Common Use Cases

- `createCustomer(details)`: Dynamic Customer Registration

crm-api-metadata.yaml

Microcks Stateful Mocks to share data between mock requests

```
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
  "POST /customers":
    ...
    dispatcher: SCRIPT
    dispatcherRules: |-
      ...

      def customer = new groovy.json.JsonSlurper().parseText(mockRequest.requestContent)
      store.put(customer.email, "")
      ...
      return "new_customer"
```

Keycloak Integration Testing with Microcks: Common Use Cases

- `existsCustomer(email)`: Check for Customer Registration

crm-api-metadata.yaml

```
apiVersion: mocks.microcks.io/v1alpha1
kind: APIMetadata
operations:
  "GET /customers/{email}":
    ...
    dispatcher: SCRIPT
    dispatcherRules: |-
      def requestedEmail = mockRequest.getRequest().getRequestURI().tokenize("/").last()
      if (store.get(requestedEmail) != null)
        return "customer_exists"

      return "unknown_customer"
```

Microcks Stateful Mocks to share data between mock requests

Keycloak Integration Testing: Lessons Learned

- Keycloak Testcontainers Module[1] provides tools to **greatly facilitate the management of stable and consistent test environments** (e.g., `withRealmImportFile()`)
  - **Out-of-the-box support by Keycloak's new Test Framework** would be nice, e.g., to inject OAuth and Admin Clients in test classes
  - Demo code comprises a **proof-of-concept implementation of a corresponding Test Framework extension**

[1] Niko Köbler et al.: https://github.com/dasniko/testcontainers-keycloak

Keycloak Integration Testing: Lessons Learned

- Keycloak Testcontainers Module[1] provides tools to **greatly facilitate the management of stable and consistent test environments** (e.g., `withRealmImportFile()`)
  - **Out-of-the-box support by Keycloak's new Test Framework** would be nice, e.g., to inject OAuth and Admin Clients in test classes
  - Demo code comprises a **proof-of-concept implementation of a corresponding Test Framework extension**

[1] Niko Köbler et al.: https://github.com/dasniko/testcontainers-keycloak

Keycloak Integration Testing: Lessons Learned

- Microcks and its Testcontainers Module[1] **greatly facilitate contract-based integration testing**
  - Microcks comes with **many low-effort features** for mock provisioning and refinement
  - However, **access to correct API definitions is a must**, and **features for testing in IAM scenarios are yet missing or hard to retrofit** (e.g., native OIDC flow support or JWT generation)

[1] https://github.com/microcks/microcks-testcontainers-java

Keycloak Integration Testing: Lessons Learned

- Microcks and its Testcontainers Module[1] **greatly facilitate contract-based integration testing**
  - Microcks comes with **many low-effort features** for mock provisioning and refinement
  - However, **access to correct API definitions is a must**, and **features for testing in IAM scenarios are yet missing or hard to retrofit** (e.g., native OIDC flow support or JWT generation)

[1] https://github.com/microcks/microcks-testcontainers-java

Auxiliary Material

Demo Code

Blog Series on Microcks

www.codecentric.de