Course Introduction

Welcome

Prerequisite

- Should have completed the Go tour https://tour.golang.org/welcome/1
- Should understand interfaces, struts and receiver functions
- Object oriented programming principles

Project

- Banking application
- Open new account for customer
- Making a Deposit or Withdrawal Transaction
- Role based access control (RBAC)

Objective

- Dev environment and the right tools
- Hexagonal Architecture
- Dependency inversion in Go and working with stubs
- JWT Tokens
- Banking-Auth microservice based on OAuth standard
- Unit test for routes and other components using mocks and state based tests

Getting Started

Dev prerequisite tools

Install Go

https://golang.org/doc/install

Project Code

https://github.com/ashishjuyal/banking

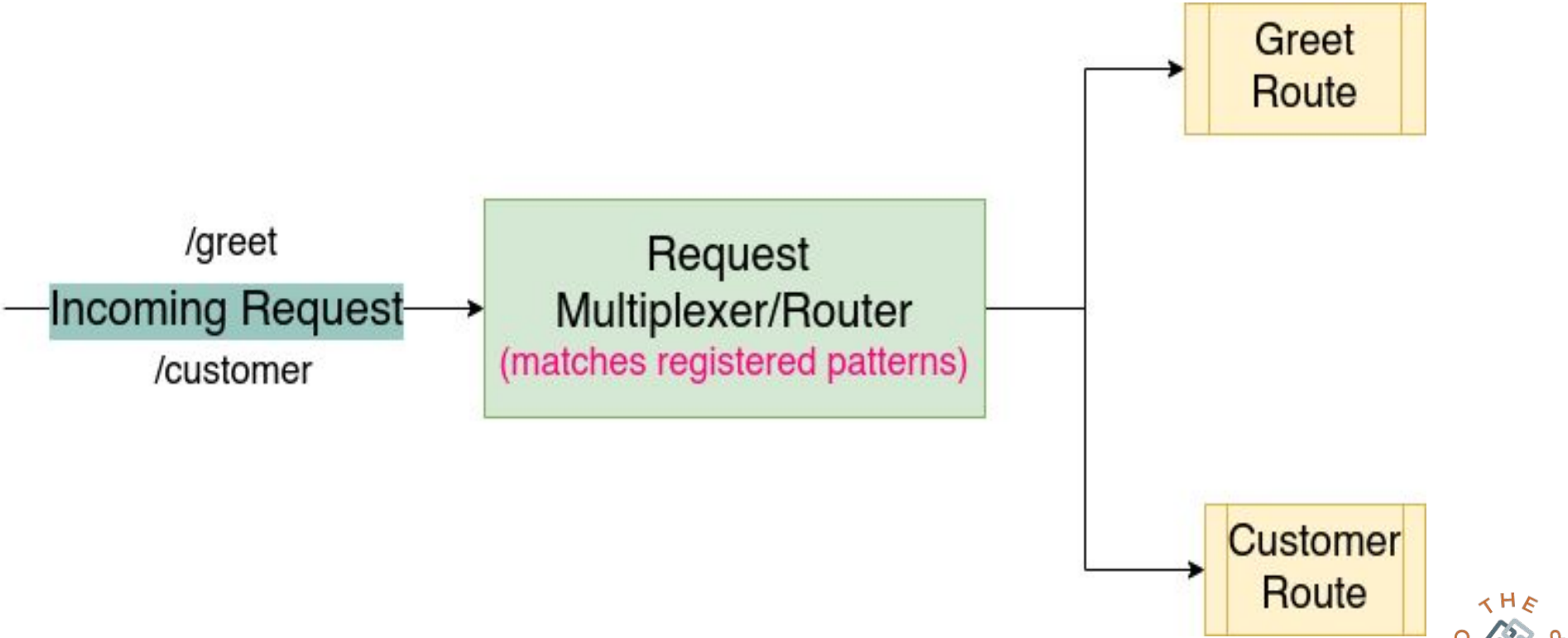
https://github.com/ashishjuyal/banking-auth

Hello World

The fun begins here...

Objective

- Mechanism of HTTP web server
- Handler Functions and Request Multiplexer (Router)
- Request and Response Headers
- Marshaling data structures to JSON and XML representations


/greet
Incoming Request
/customer
Request
Multiplexer/Router
(matches registered patterns)
Greet
Route
Customer
Route

{JSON}

Router basics: Sending response as JSON

Objective

- ✓ Mechanism of HTTP web server
- ✓ Handler Functions and Request Multiplexer (Router)
- Marshaling data structures to JSON representation
- Response Header
- Marshaling data structures to XML representation
- Request Headers

<XML>

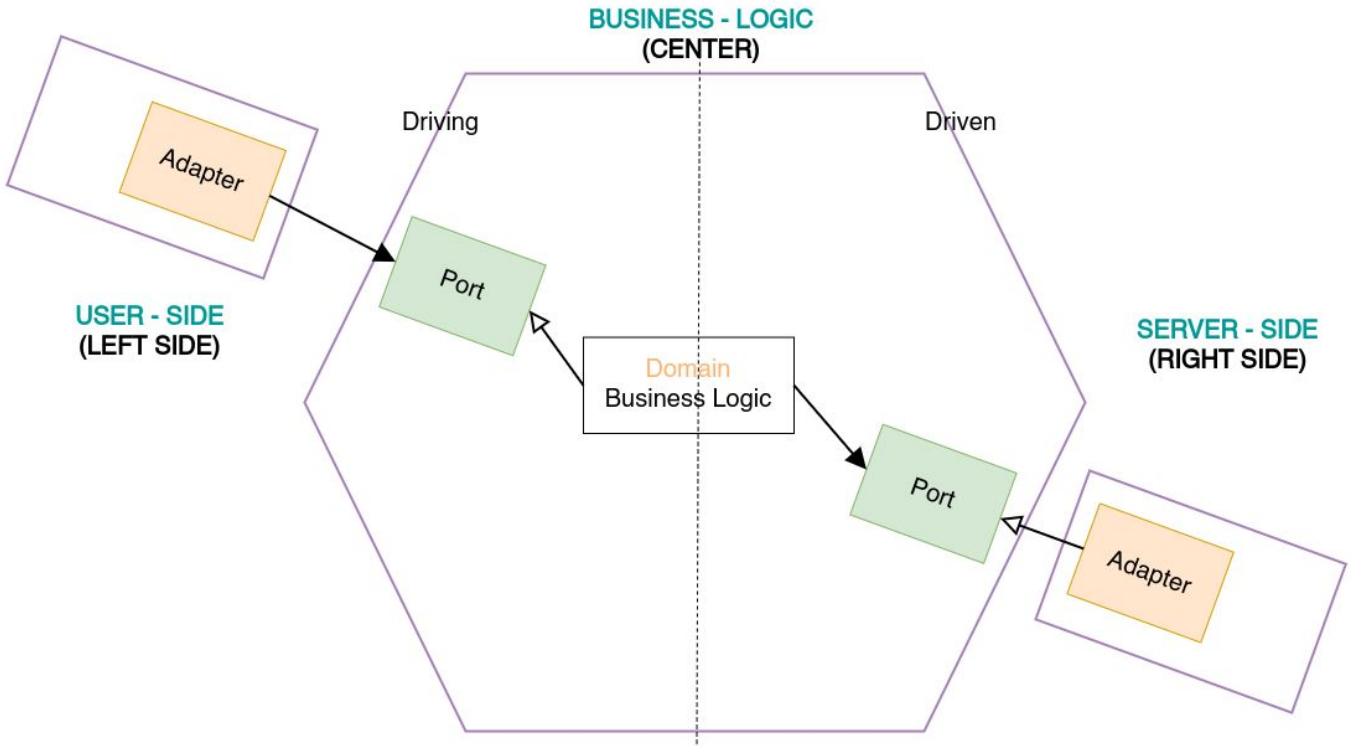
Router basics: Sending response as XML

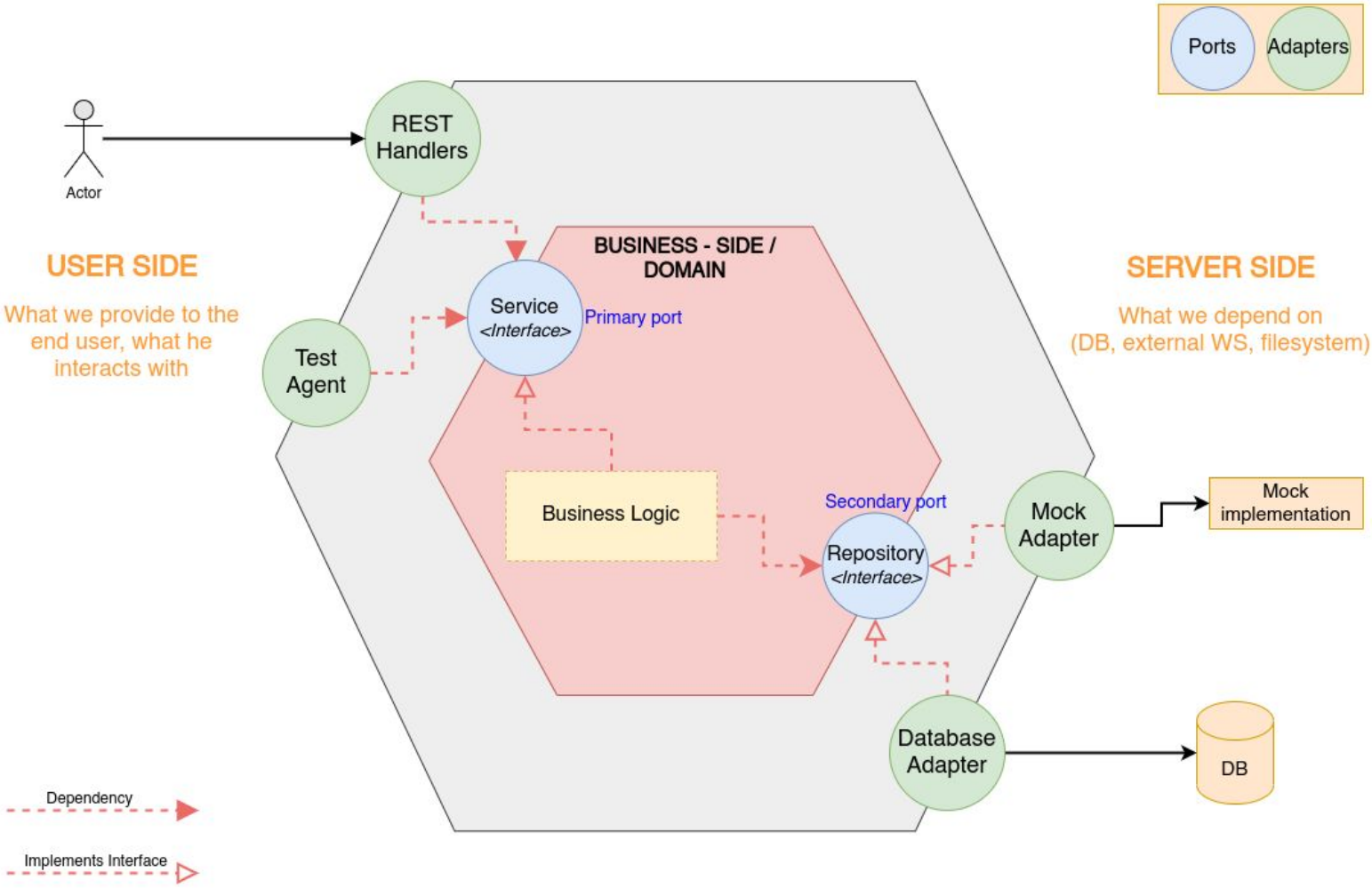
Objective

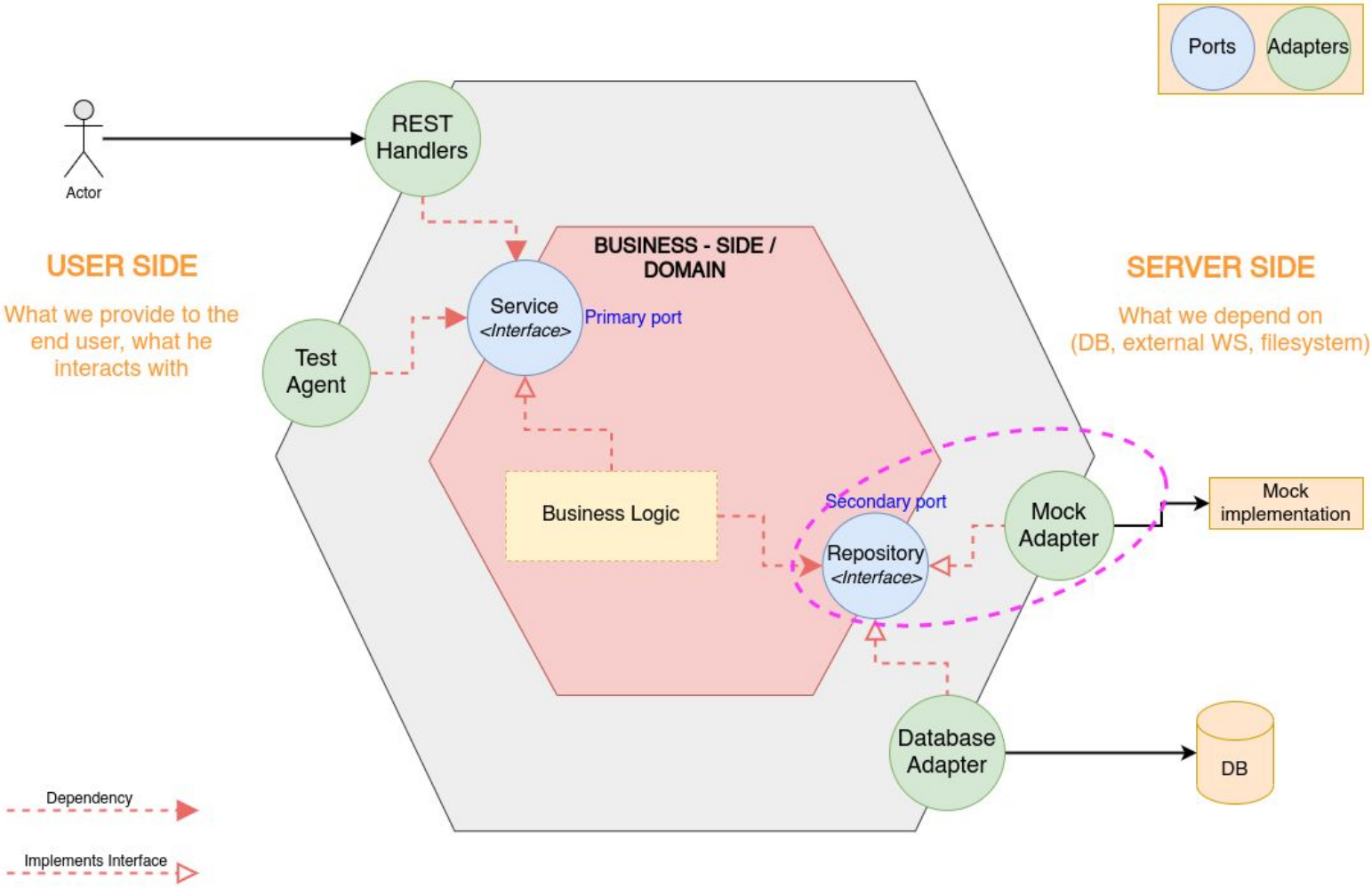
- ✓ Mechanism of HTTP web server
- ✓ Handler Functions and Request Multiplexer (Router)
- ✓ Marshaling data structures to JSON representation
- ✓ Response Header
- Marshaling data structures to XML representation
- Request Headers

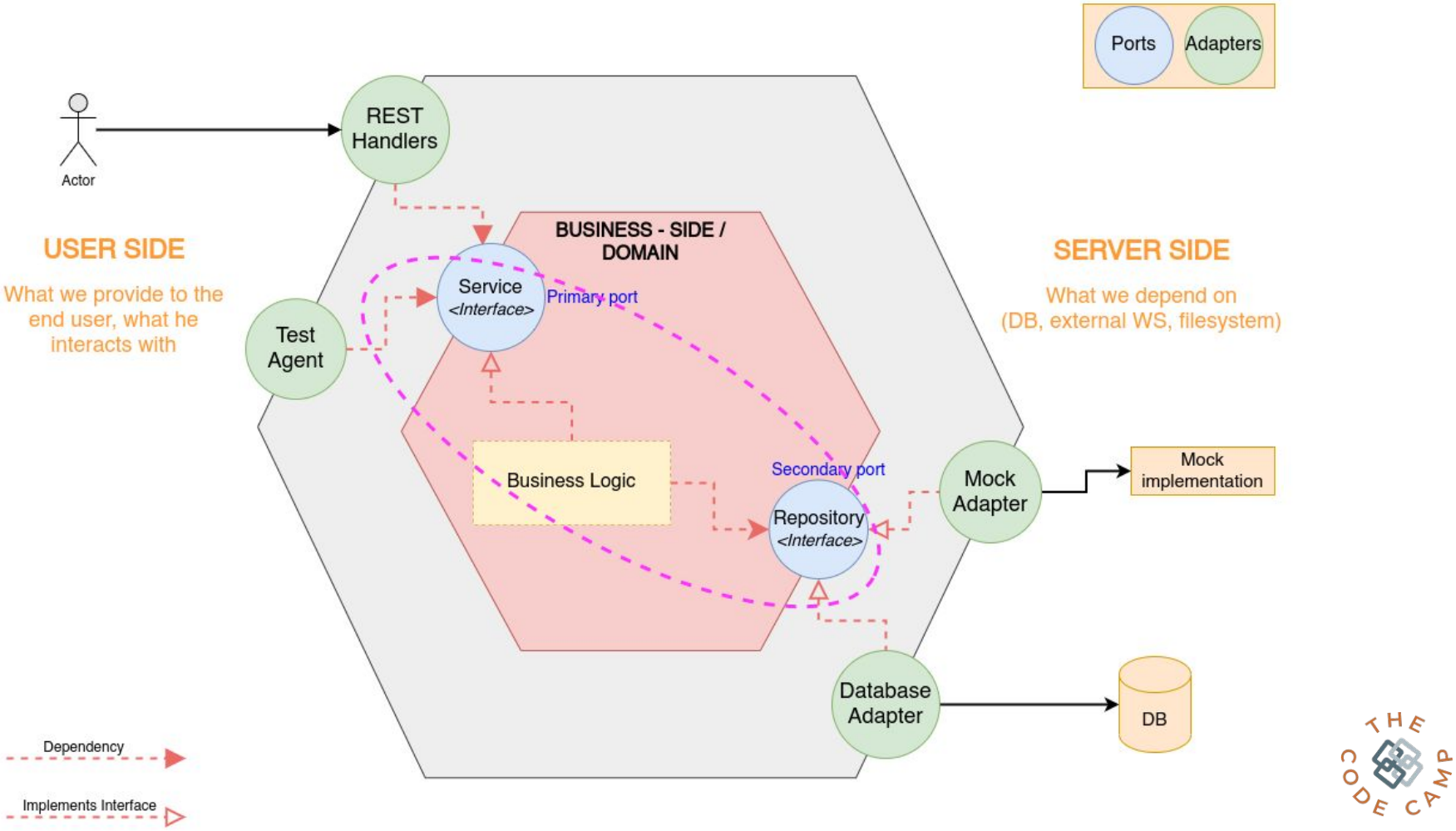
Hexagonal Architecture

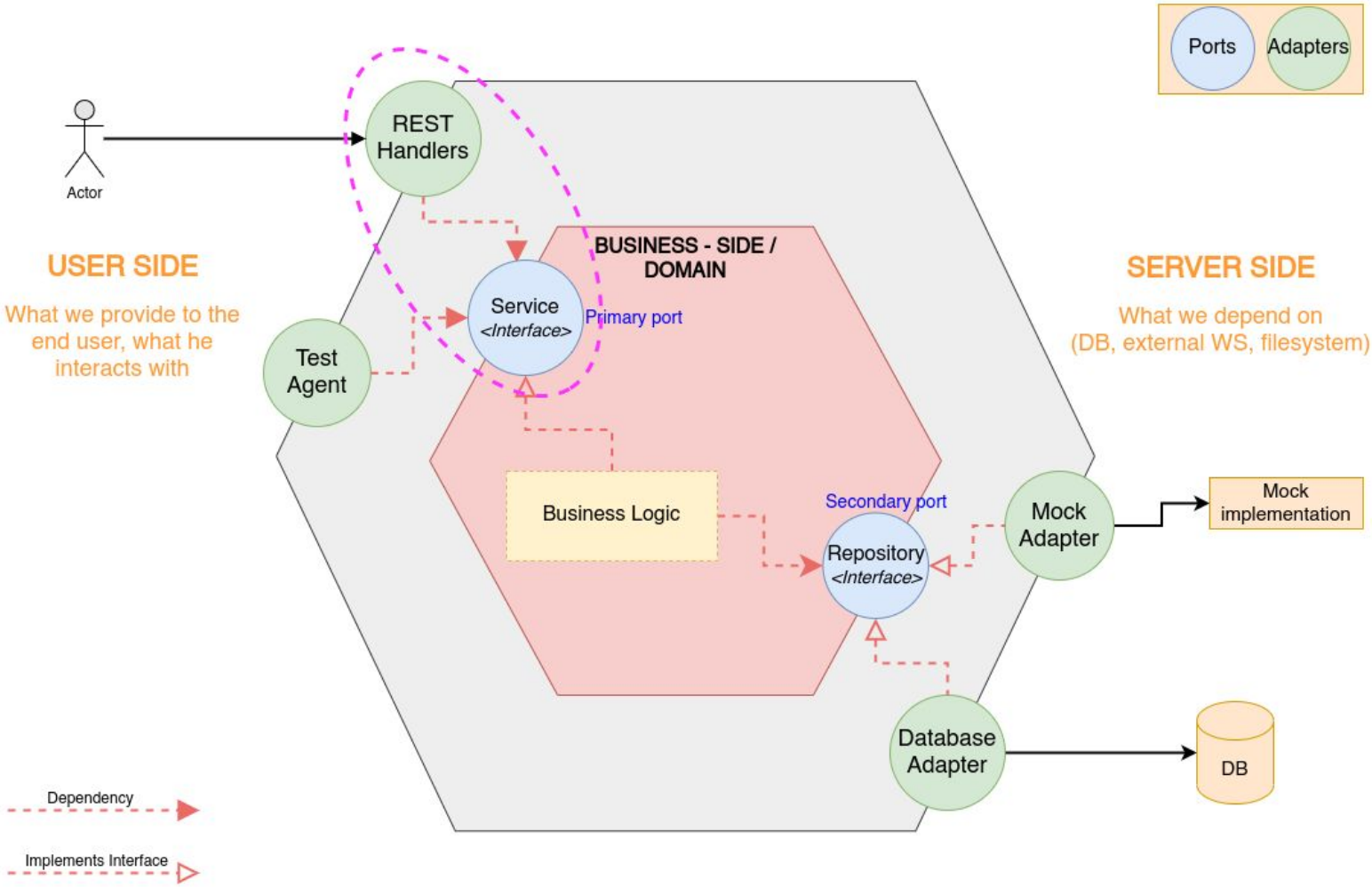
Ports & Adapter

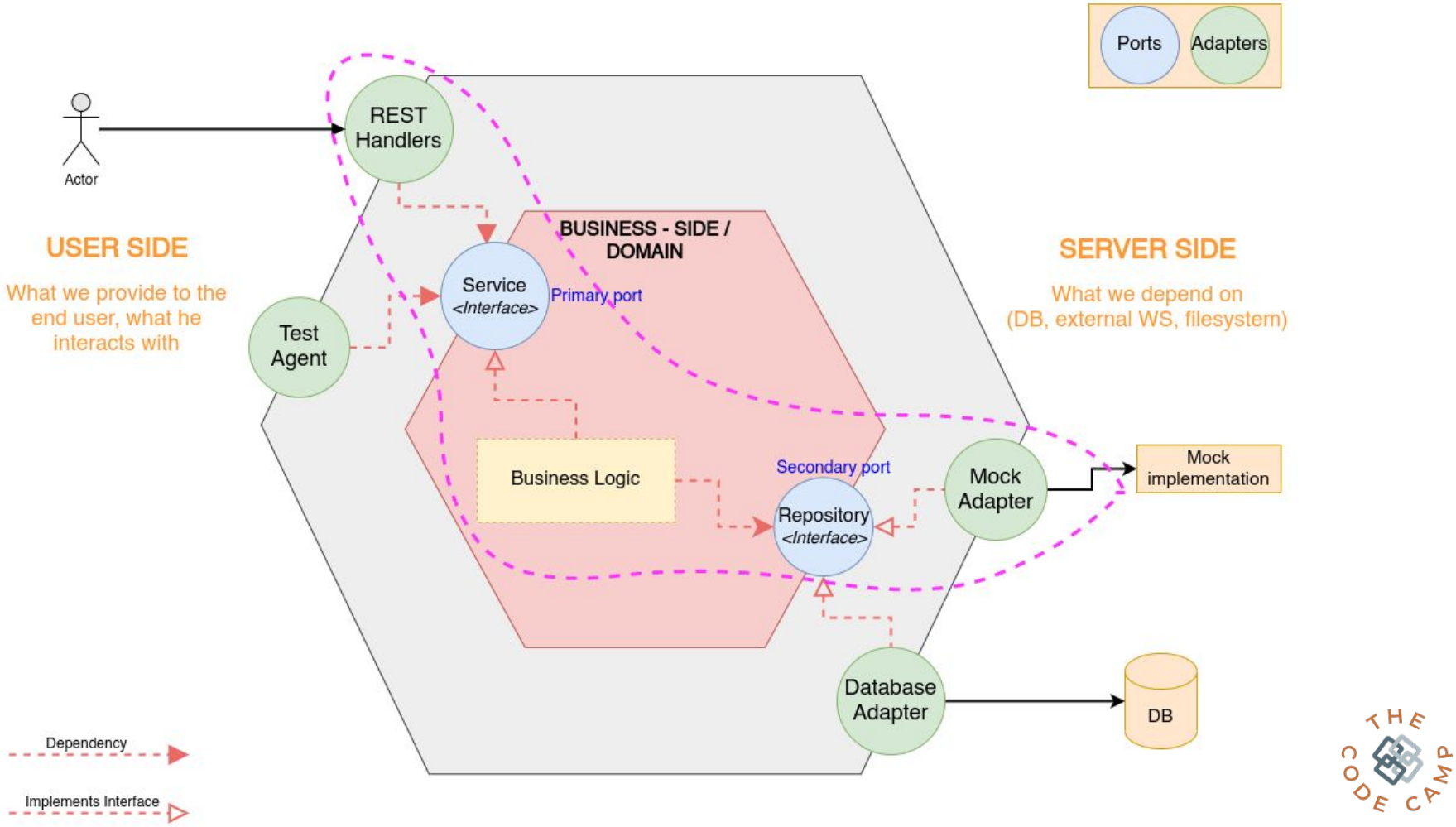

BUSINESS - LOGIC
(CENTER)
Driving
Driven
Adapter
Port
USER - SIDE
(LEFT SIDE)
Domain
Business Logic
SERVER - SIDE
(RIGHT SIDE)
Port
Adapter

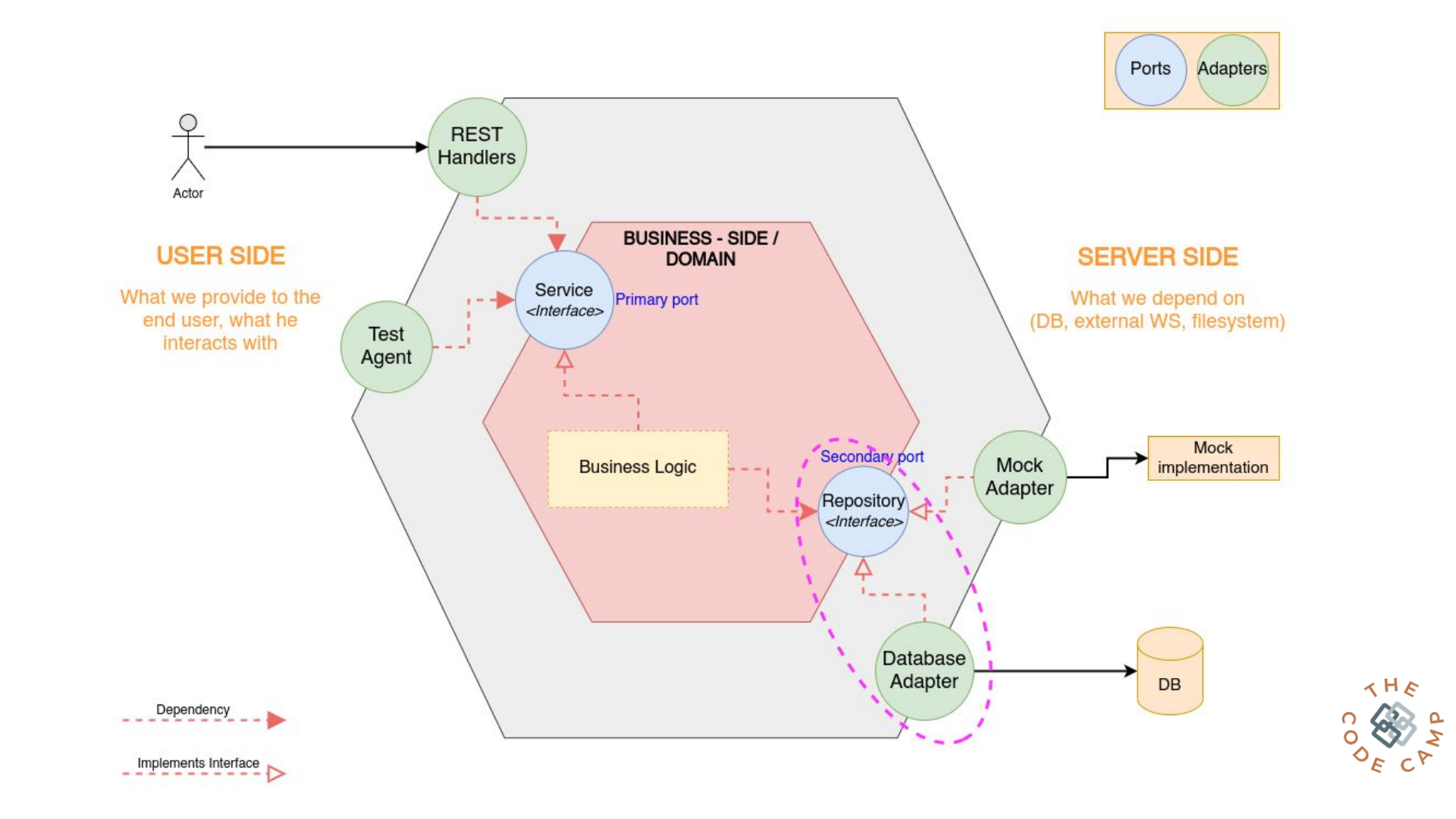

Ports
Adapters
Actor
REST Handlers
USER SIDE
What we provide to the end user, what he interacts with
BUSINESS - SIDE / DOMAIN
Service <Interface>
Primary port
Test Agent
Business Logic
Secondary port
Repository <Interface>
Mock Adapter
Mock implementation
SERVER SIDE
What we depend on (DB, external WS, filesystem)
Database Adapter
DB
Dependency
Implements Interface


Ports
Adapters
Actor
REST Handlers
USER SIDE
What we provide to the end user, what he interacts with
BUSINESS - SIDE / DOMAIN
Service <Interface>
Primary port
Test Agent
Business Logic
Secondary port
Repository <Interface>
Mock Adapter
Mock implementation
SERVER SIDE
What we depend on (DB, external WS, filesystem)
Database Adapter
DB
Dependency
Implements Interface


Ports
Adapters
Actor
REST Handlers
USER SIDE
What we provide to the end user, what he interacts with
BUSINESS - SIDE / DOMAIN
Service <Interface>
Primary port
Test Agent
Business Logic
Secondary port
Repository <Interface>
Mock Adapter
Mock implementation
SERVER SIDE
What we depend on (DB, external WS, filesystem)
Database Adapter
DB
Dependency
Implements Interface


Ports
Adapters
Actor
REST Handlers
USER SIDE
What we provide to the end user, what he interacts with
BUSINESS - SIDE / DOMAIN
Service <Interface>
Primary port
Test Agent
Business Logic
Secondary port
Repository <Interface>
Mock Adapter
Mock implementation
SERVER SIDE
What we depend on (DB, external WS, filesystem)
Database Adapter
DB
Dependency
Implements Interface


Ports
Adapters
Actor
REST Handlers
USER SIDE
What we provide to the end user, what he interacts with
BUSINESS - SIDE / DOMAIN
Service <Interface>
Primary port
Test Agent
Business Logic
Secondary port
Repository <Interface>
Mock Adapter
Mock implementation
SERVER SIDE
What we depend on (DB, external WS, filesystem)
Database Adapter
DB
Dependency
Implements Interface


Ports
Adapters
Actor
REST Handlers
USER SIDE
What we provide to the end user, what he interacts with
BUSINESS - SIDE / DOMAIN
Service <Interface>
Primary port
Test Agent
Business Logic
Secondary port
Repository <Interface>
Mock Adapter
Mock implementation
SERVER SIDE
What we depend on (DB, external WS, filesystem)
Database Adapter
DB
Dependency
Implements Interface

Error Handling

Sending error in response

Refactoring

Error infrastructure & handlers

Assignment 1: Fix GetAllCustomers API

Acceptance Criteria

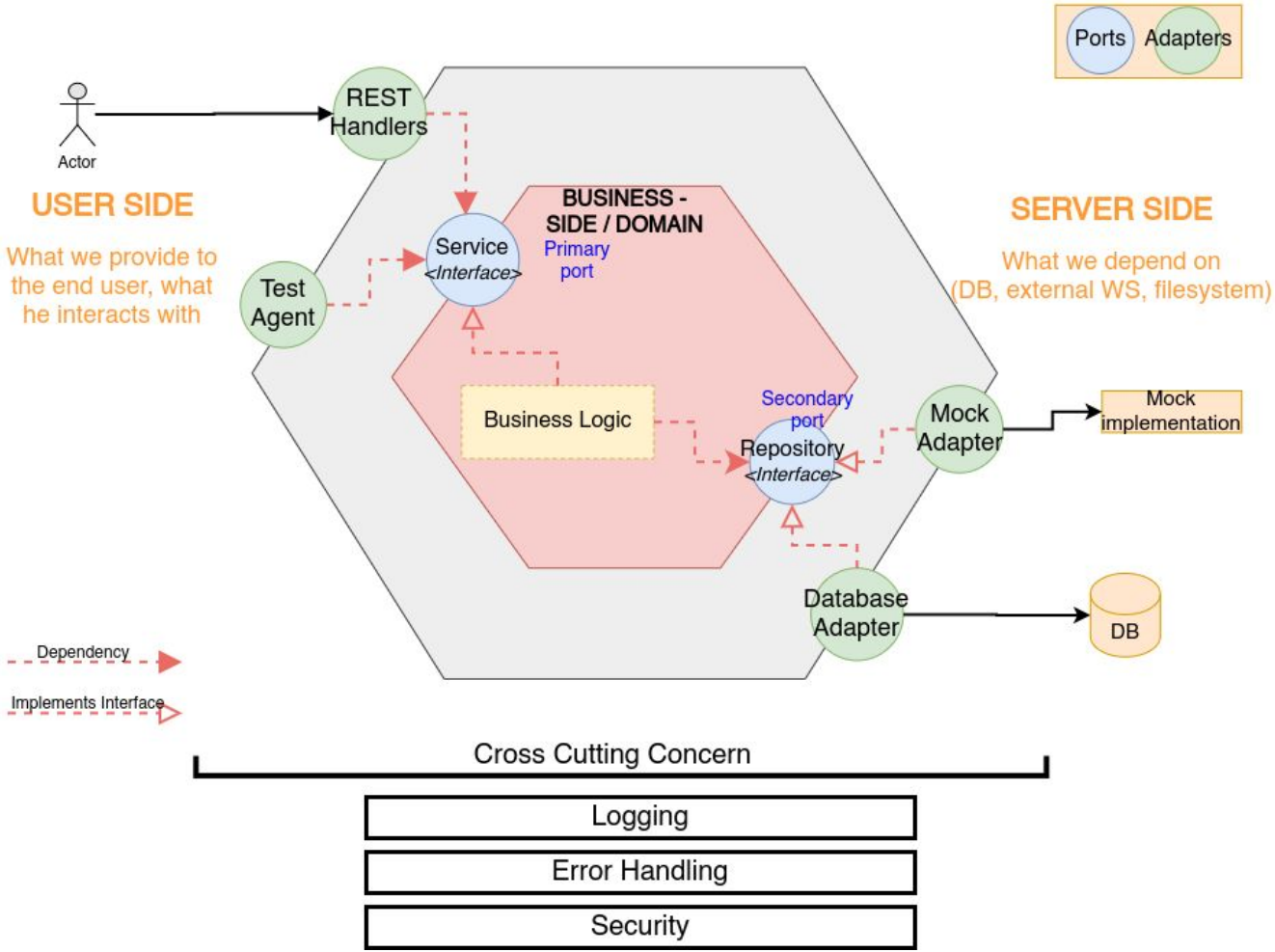
- API should only return JSON response
- API should return HTTP status code 500, in case of some unexpected error from server-side
- API should return HTTP status code 200 and the data in case of successfully retrieving customers from the server-side

Assignment 2: Enhance GetAllCustomers API

Acceptance Criteria

The API should provide an option to fetch the customers by status. For. e.g.

- `/customers?status=active` should return all active customers
- `/customers?status=inactive` should return all inactive customers
- `/customers` should return all customers

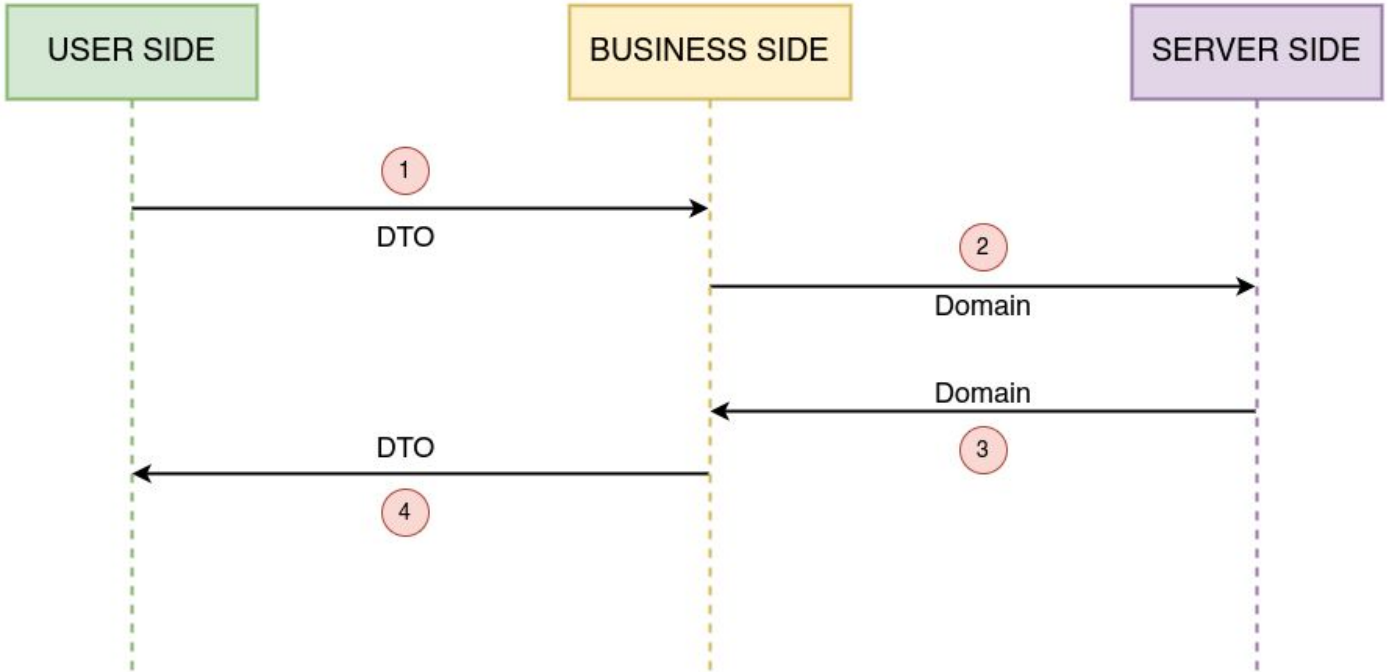

Ports
Adapters
Actor
REST Handlers
USER SIDE
What we provide to the end user, what he interacts with
Test Agent
Service
<Interface>
BUSINESS - SIDE / DOMAIN
Primary port
Business Logic
Secondary port
Repository
<Interface>
SERVER SIDE
What we depend on (DB, external WS, filesystem)
Mock Adapter
Mock implementation
Database Adapter
DB
Dependency
Implements Interface
Cross Cutting Concern
Logging
Error Handling
Security

Sqlx

General purpose extension to database/sql

DTO

Data Transfer Object

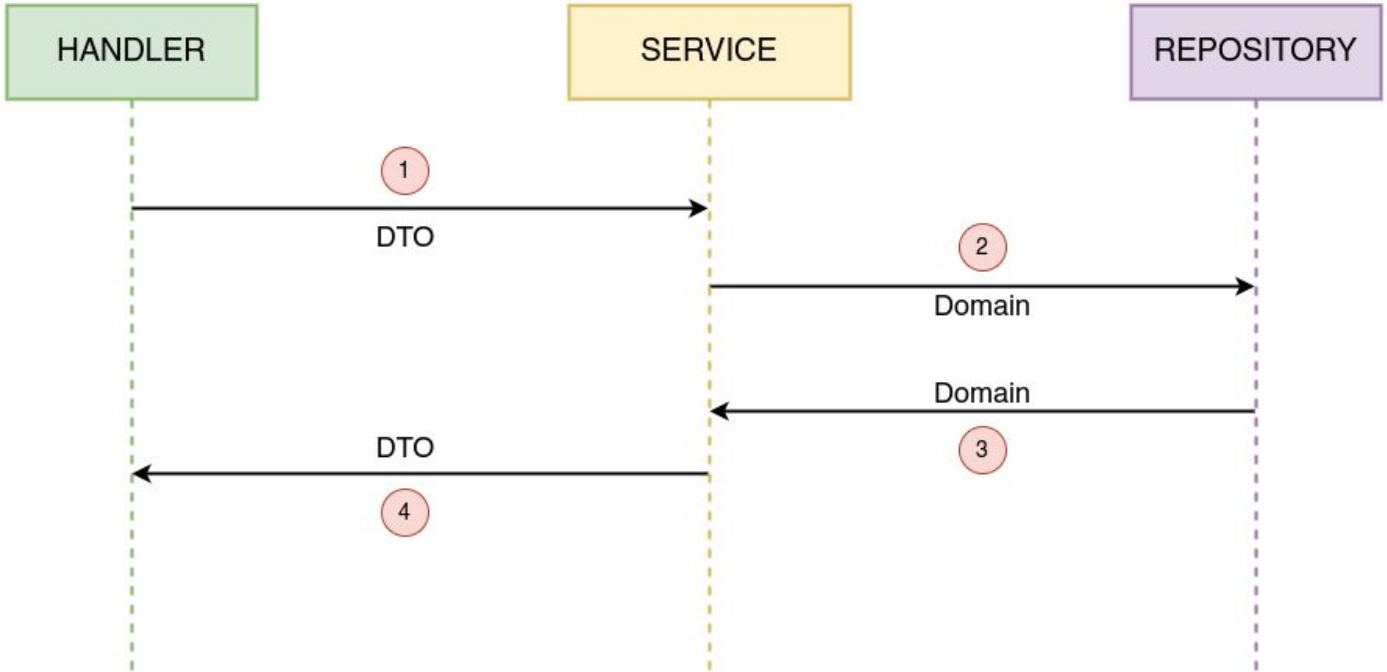

USER SIDE
BUSINESS SIDE
SERVER SIDE
1
DTO
2
Domain
Domain
3
DTO
4

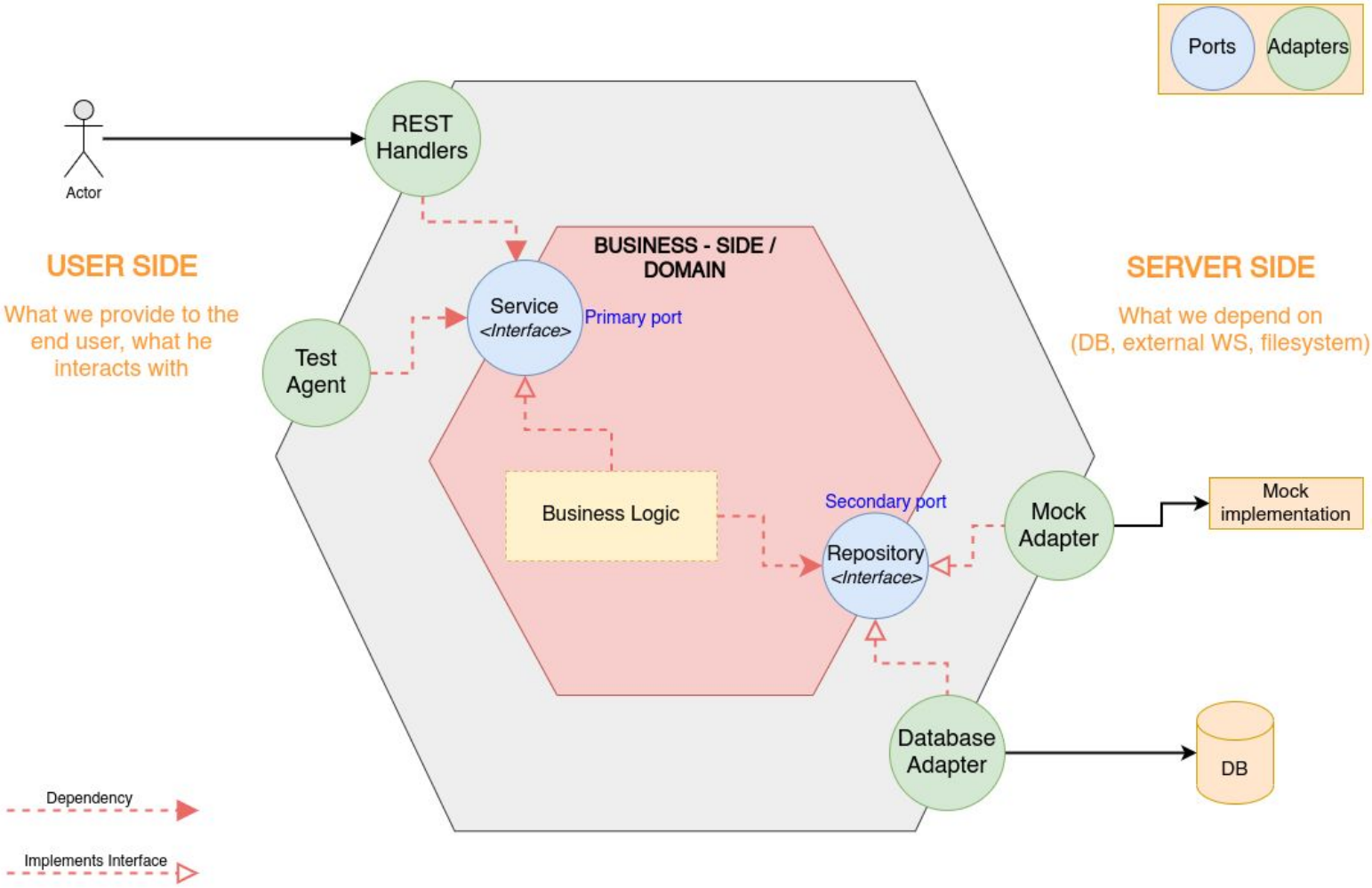
Application Configuration

Using environment variables

New Bank Account

Part 1: Business and Server side


HANDLER
SERVICE
REPOSITORY
1
DTO
2
Domain
Domain
3
DTO
4


Ports
Adapters
Actor
REST Handlers
USER SIDE
What we provide to the end user, what he interacts with
BUSINESS - SIDE / DOMAIN
Service <Interface>
Primary port
Test Agent
Business Logic
Secondary port
Repository <Interface>
Mock Adapter
Mock implementation
SERVER SIDE
What we depend on (DB, external WS, filesystem)
Database Adapter
DB
Dependency
Implements Interface

New Bank Account

Part 2: User and business side

Assignment 3: Make a transaction in bank account

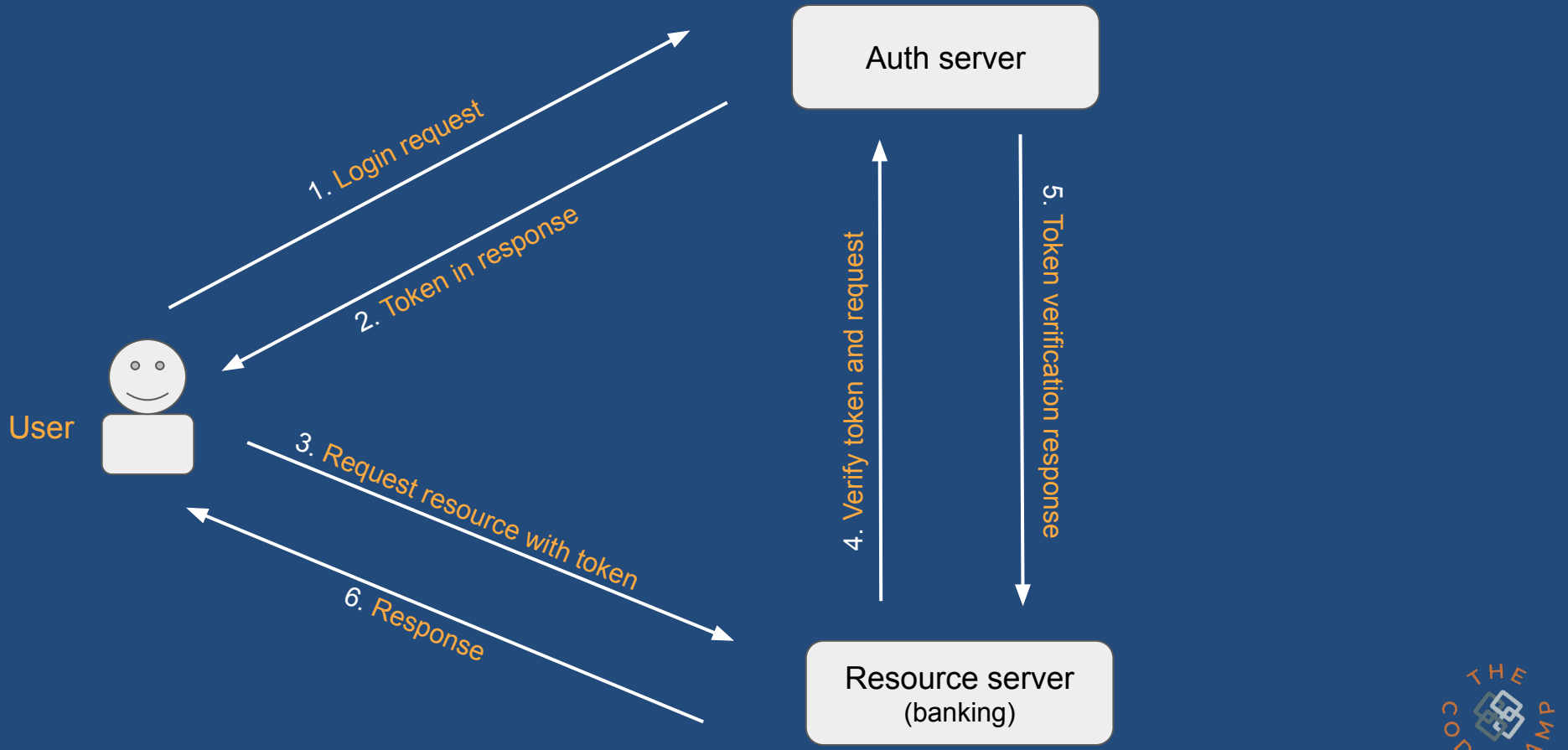
Acceptance Criteria

Write an API to create a new transaction for an existing customer

- transaction can only be "withdrawal" or "deposit"
- amount cannot be negative
- withdrawal amount should be available in the account
- successful transaction, should return the updated balance with transaction id response
- error handling should be done for bad request, validation and unexpected errors from the server side and should return the appropriate http status code with message

Securing Application

Authentication & Authorization, JWT Tokens and RBAC


Auth server
1. Login request
2. Token in response
User
3. Request resource with token
6. Response
4. Verify token and request
5. Token verification response
Resource server
(banking)

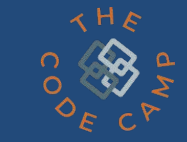
JWT

Json Web Token


Header
Signature
aaaaaaaaaaa.bbbbbbbbbb.ccccccccccc
Payload

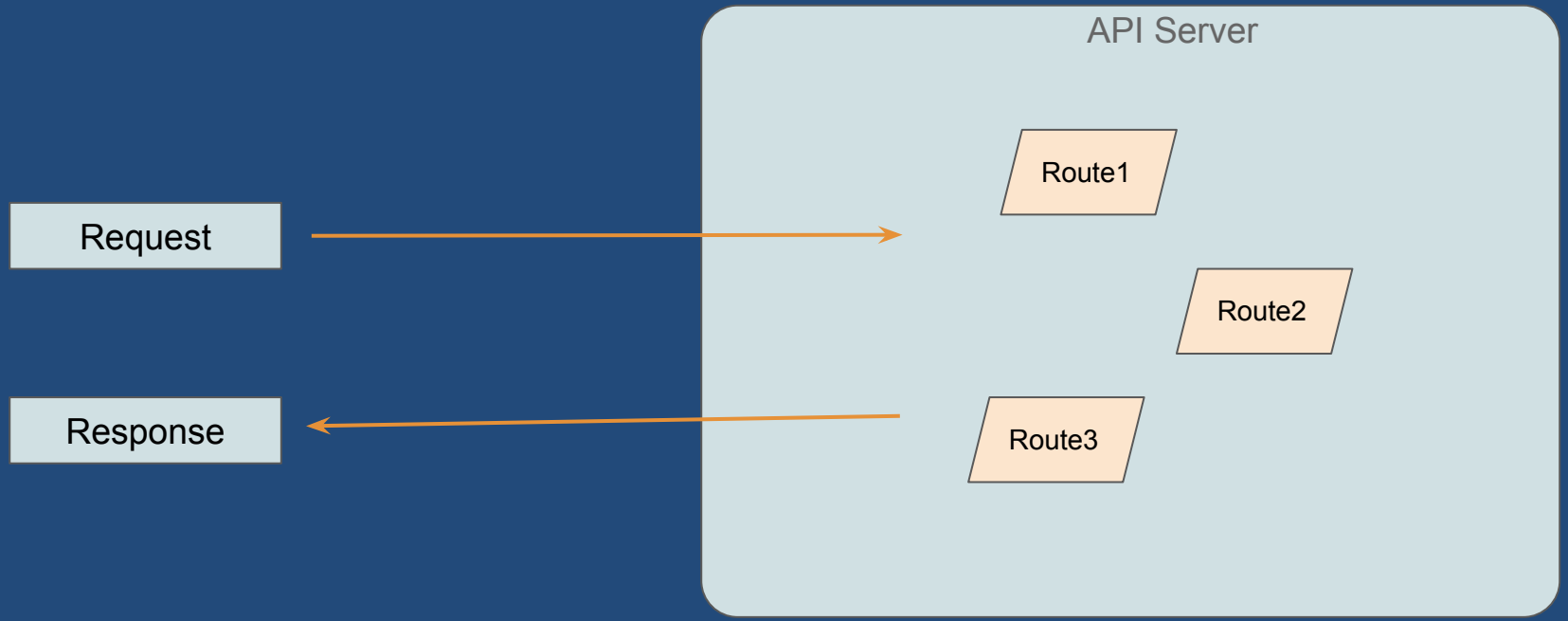
Auth Server

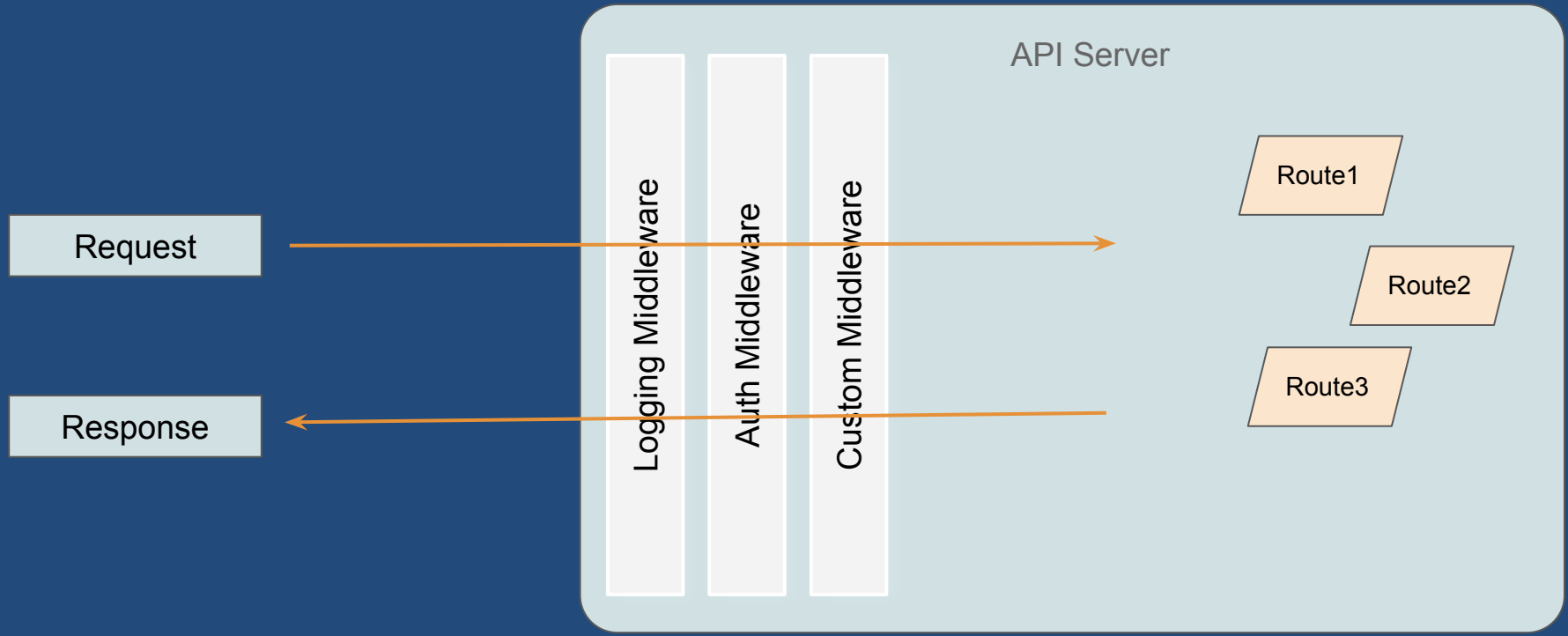
Login API



Auth Server

Verify Authorization API


API Server
Route1
Route2
Route3
Request
Response


API Server
Request
Response
Logging Middleware
Auth Middleware
Custom Middleware
Route1
Route2
Route3

Unit Testing

State based test

Unit Testing

Testing routes: using mocks

Unit Testing

Testing services: using mocks

Advanced Topics

Introduction

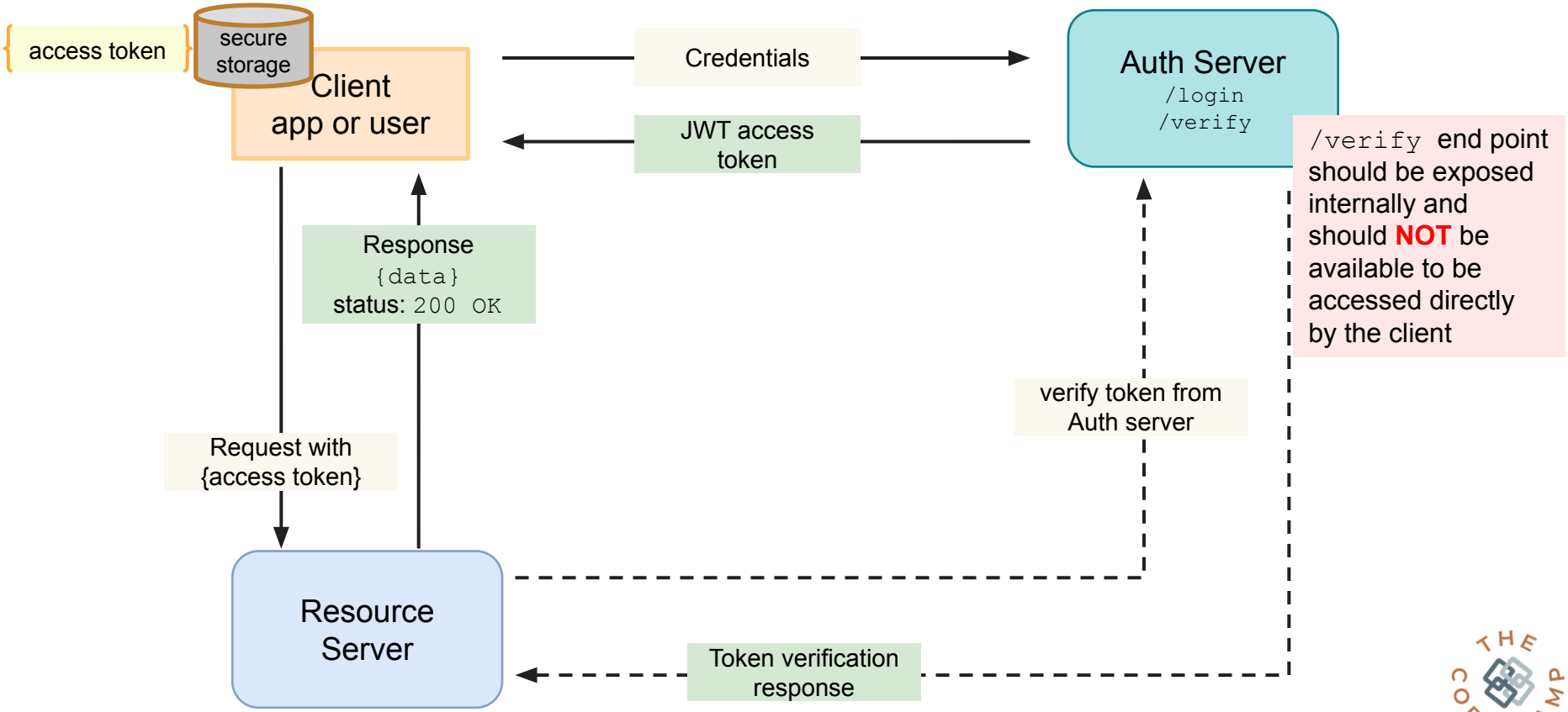
Topics

- AppError and logger as go-module [naming it as banking-lib]
- Integrate banking-lib with banking and banking-auth
- Versioning and publish a go module (banking-lib)
- Generate refresh token
- New access token from refresh token

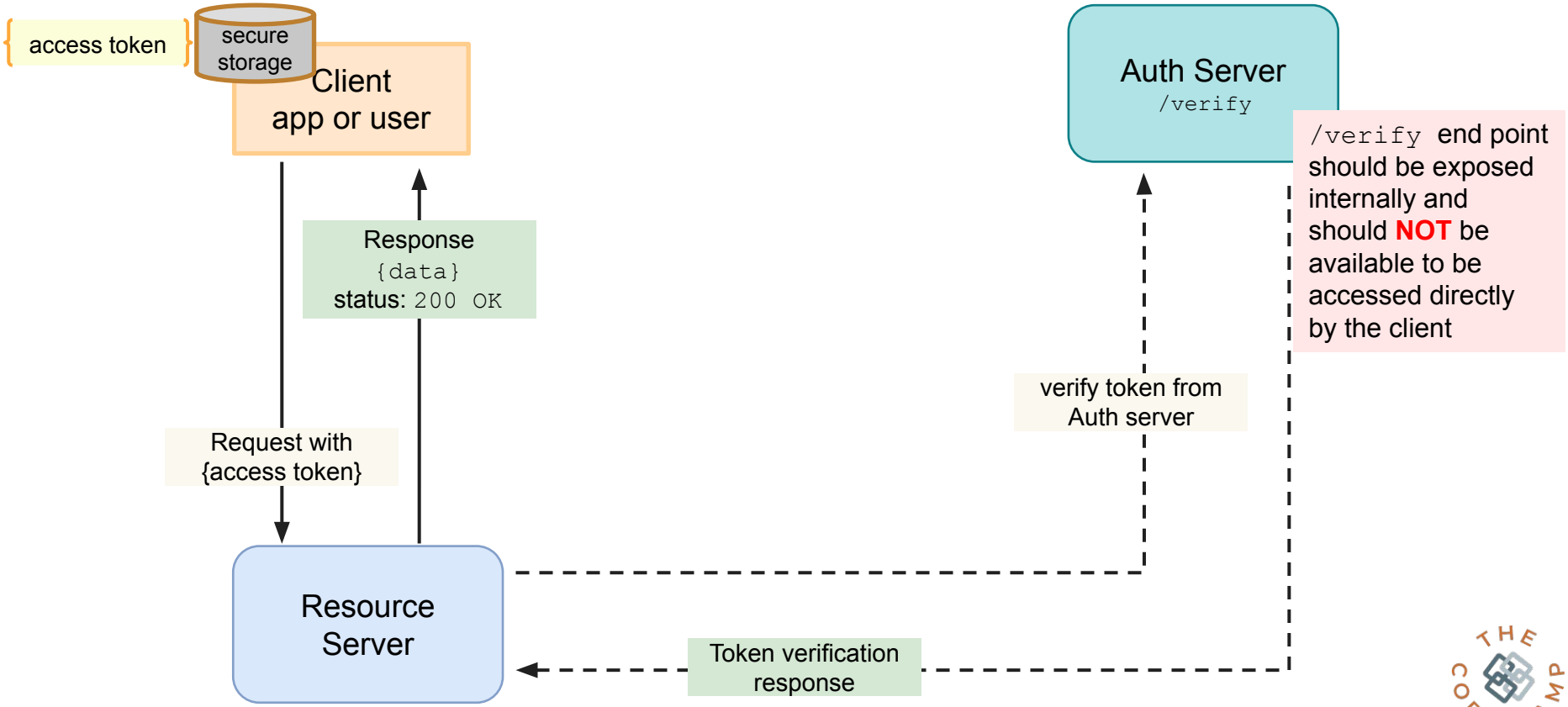
Refresh Token

Introduction

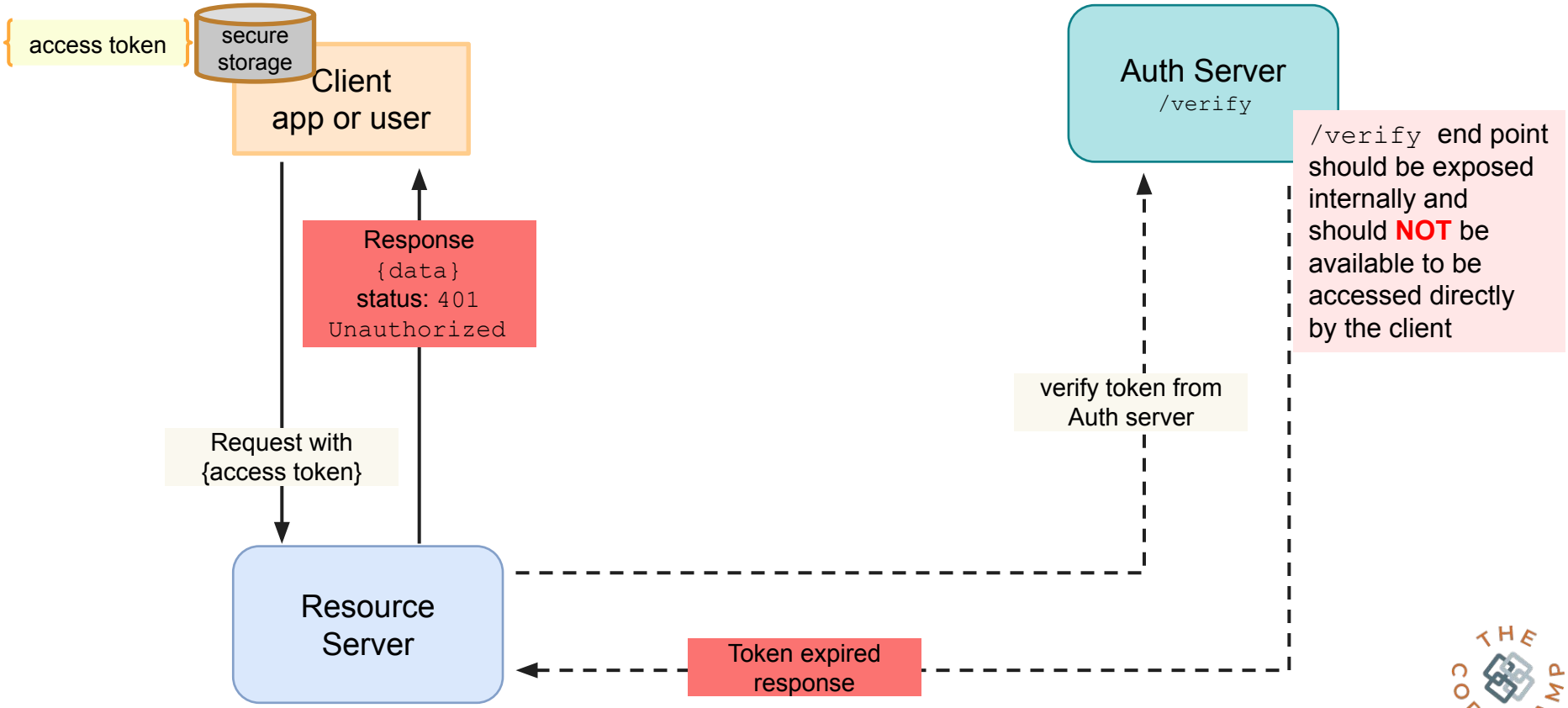
Login: access token



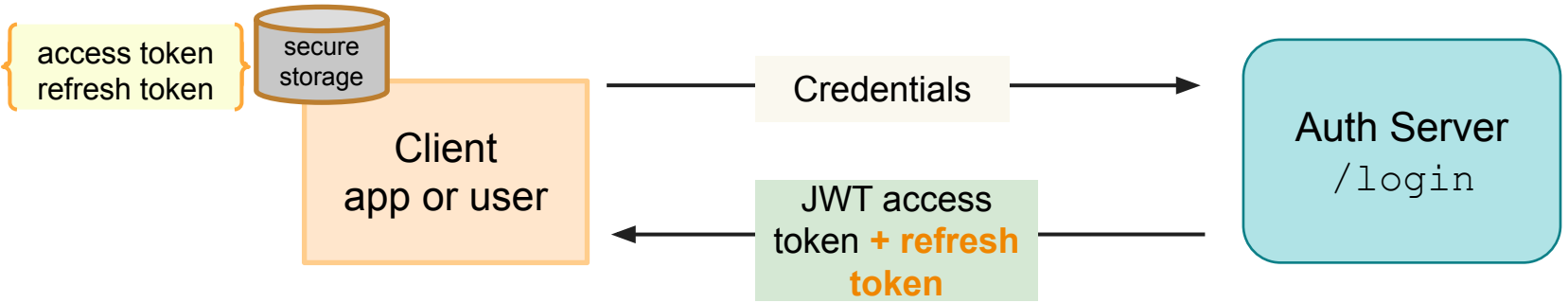
Continued…


access token
secure storage
Client app or user
Auth Server
/verify
/verify end point should be exposed internally and should NOT be available to be accessed directly by the client
Response
{data}
status: 200 OK
verify token from Auth server
Request with {access token}
Resource Server
Token verification response

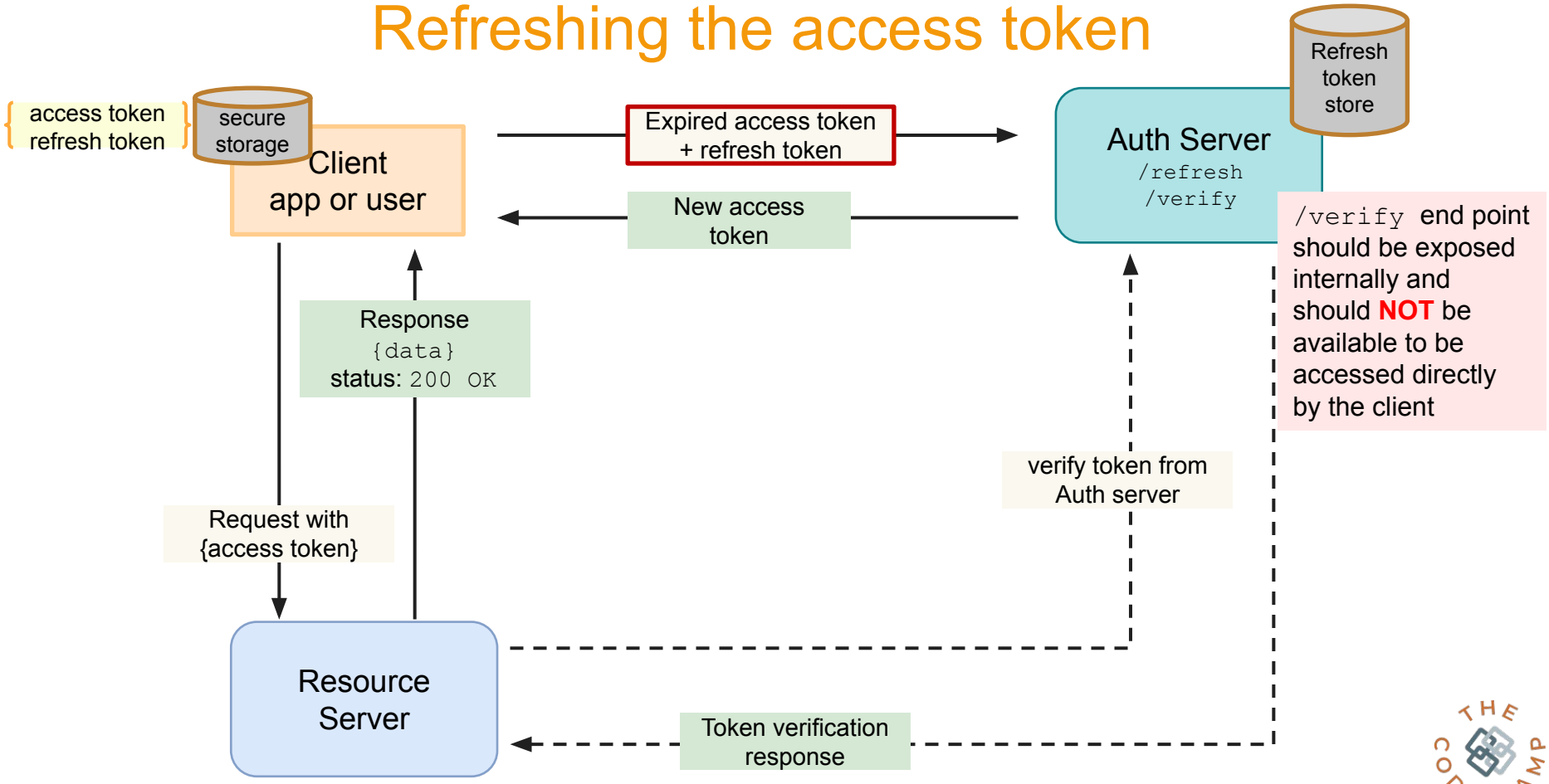
Expired token flow



Login: access token + refresh token



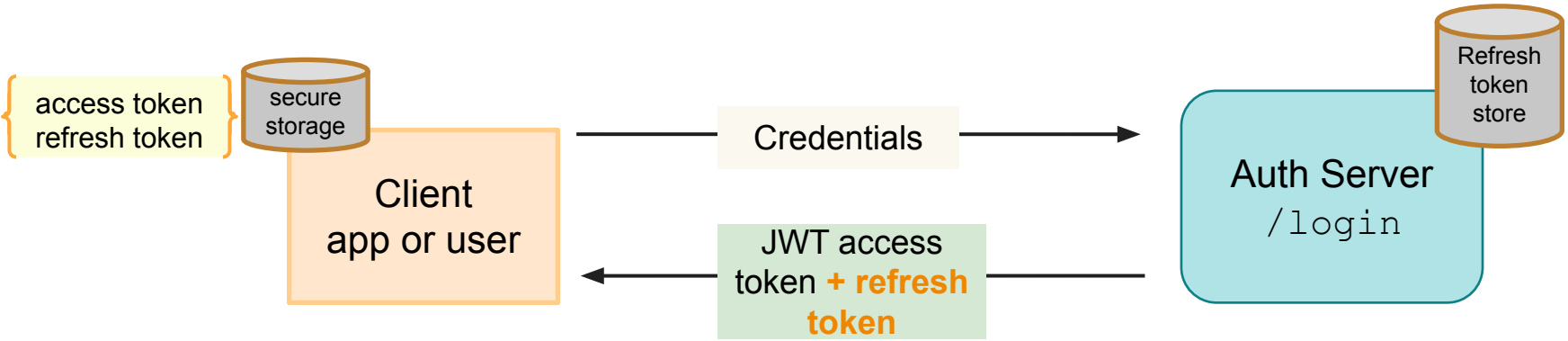
Refreshing the access token



Refresh Token

Generating refresh token: Part 1

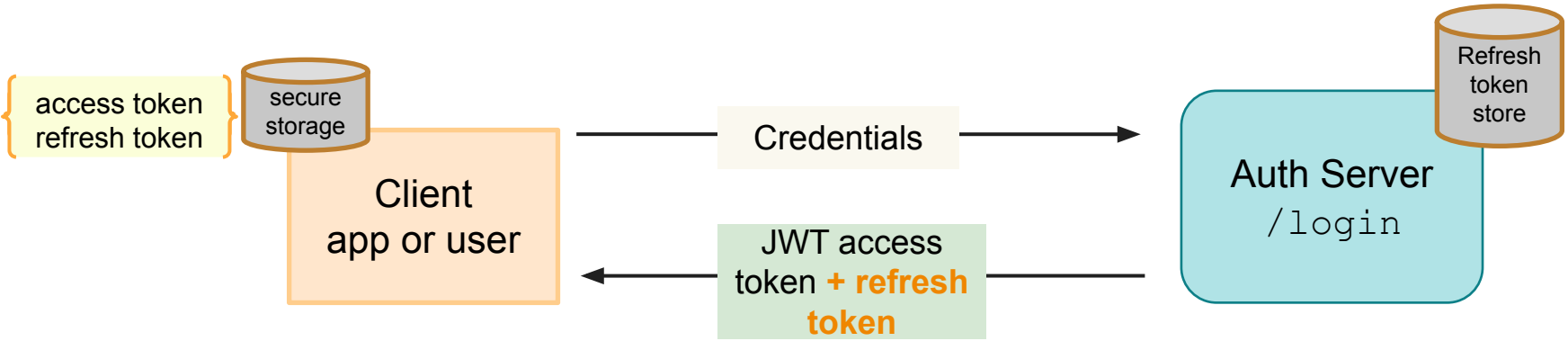
Login: access token + refresh token



Refresh Token

Generating refresh token: Part 2

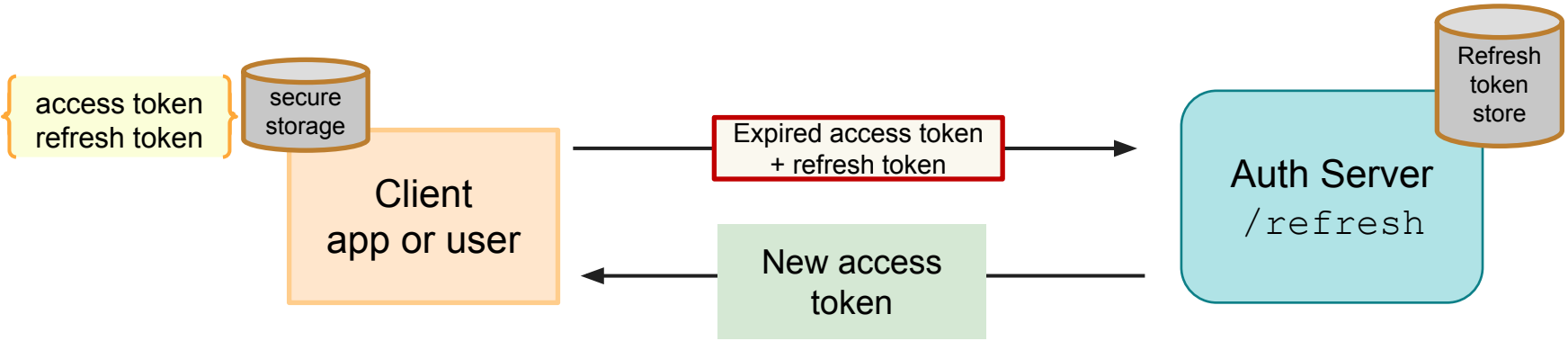
Login: access token + refresh token



Refresh

Refreshing access token

Refreshing access token



Validations

- ~~access token should be valid and should be expired~~
- ~~Valid = should be signed by us~~
- refresh token should exist in the store
- refresh token should be valid and should not be expired