

Project Report for Fruits Vending

Practice Module for Certificate in Architecting Scalable Systems

Team 6

Members:

Kevin Chen Xiaojie (E0385016)

Zou Xuan (E0384799)

Kunal Jadhav (E0385000)

Nurdin Effendi (E0384876)

CONTENTS

1. Introduction 4

1.1 Background 4

1.2 Business Needs 4

1.3 Stakeholders 4

1.4 Project Scope 4

1.4.1 Functionality in scope 4

1.4.2 Functionality out of scope 4

2. Solution Overview 5

2.1 Logical View 5

2.1.1 Key Architectural Decisions 5

2.1.2 Subsystems 5

2.1.3 Platform Design 5

2.1.4 Tiers and Layers 5

2.2 Physical View 5

2.2.1 Technology and Services 6

2.2.2 Key Architectural Decisions 6

2.3 Data Design 6

2.4 Other Architectural Decisions 6

2.5 Architectural Issues 7

3. Quality Attributes 8

3.1 Performance 8

3.2 Availability 8

3.3 Security 8

3.4 Maintainability 8

4. DevOps and Deveopment Lifecycle 8

4.1 Source Control Strategy 8

4.2 Continuous Integration 8

4.3 Continuous Delivery 8

# Introduction

## Background

Fruits Vending Pte Ltd is a premium smart vending machines manufacturer and operator in Singapore. They specialize in providing freshly squeezed orange juice vending services to the customers with consistent, excellent quality at competitive prices.

Currently the company unable to recognize their customers even though thousands of cups of orange juice sold daily through vending machine. They also unable to get the feedback from customers who bought the juice directly from vending machine.

## Business Needs

With this platform, the company will be able to recognize their customers and improve the quality of their business based the customer data/feedback that they will get from this platform.

In addition to that, cloning/replicating the platform to other companies should be effortless so we can setup new business with other companies easily.

## Stakeholders

Fruits Vending Pte Ltd Business Team

Fruits Vending Pte Ltd Development Team

## Project Scope

Briefly describe the scope of the solution. State what is included in the scope and what is out of scope. If relevant, define the phases of the solution and what is delivered in each phase. Provide references to the documented scope. Also clearly define the scope for the designs and implementations if you do not implement everything that you have designed

The solution should include below 5 main functionalities.

1. Account management

Including Register, Update, Withdraw and Login with account. On top of that , 2FA should also be implemented for security.

In the scope: Register (without 2FA), Login

Out of scope: Update, Withdraw

1. E-wallet management

Including top up, deduction , query.

All of the functionalities listed are in the scope.

1. QR & Product management

Including interface with customer company for QR generation , share QR by generating a new QR with specified amount and QR balance management.

All of the functionalities listed are in the scope.

1. Authentication and Authorization

Including OAuth2 password and client credential grant type.

Password grant type is used in communication with services in current project.

Client credential grant type is used by customer companies for generating jwt token to communicate with QR balance deduction API. New companies should also be able to register client credential clientid .

In the scope: Password grant type and client credential grant type JWT token generation

Out of the scope: Company clientid registration.

1. Company registration

Company can register from UI and manage the products that will be sold in the platform.

The functionality is not in the scope.

### Functionality in scope

Briefly list down the functionality and use cases that you include as part of the project

**Use case 1: Purchase QR Code**

**Main flow**

1. User login and open E-wallet UI
2. Select product , input amount of QR
3. Go to QR code slot UI to buy QR code with E-wallet balance
4. QR code generated with the specified balance , E-wallet balance deducted

**Alternative flow (E-wallet balance insufficient)**

1. User login and open E-wallet UI
2. Select product , input amount of QR
3. Buy QR code with E-wallet balance.
4. E-wallet balance is insufficient, re-direct to E-wallet top up page
5. Make payment to top up E-wallet
6. Go to QR code slot UI to buy QR code with E-wallet balance
7. QR code generated with the specified balance , E-wallet balance deducted

**Use case 2:Share QR Code**

**Main flow**

1. Login from UI
2. Query QR code that owned by current user
3. Select QR code that will be shared, enter share amount
4. New QR Code generated with the specified amount, Existing QR code balance deducted.

**Alternative flow(QR code balance insufficient)**

1. Login from UI
2. Query QR code that owned by current user
3. Select QR code that will be shared, enter share amount
4. QR code balance insufficient, transaction failed.

### Functionality out of scope

Briefly list down the functionality and use cases that is out of scope of the project

**Use case 1: Company register from website**

1. Company register from UI
2. Company input contact information
3. Company input QR code generation endpoint , input parameter
4. Company register clientid & client secret from platform auth server
5. Company upload product image and input product properties.
6. System auto connect to new company and display new products for sale.
7. Company use clientid and client secret to connect the platform for QR code balance deduction.

**User case 2: User forget password**

1. User click forget password
2. User input account name , system send email if the account exists.
3. User open link in email and input new password and confirm password
4. User is able to login with new password.

# Solution Overview

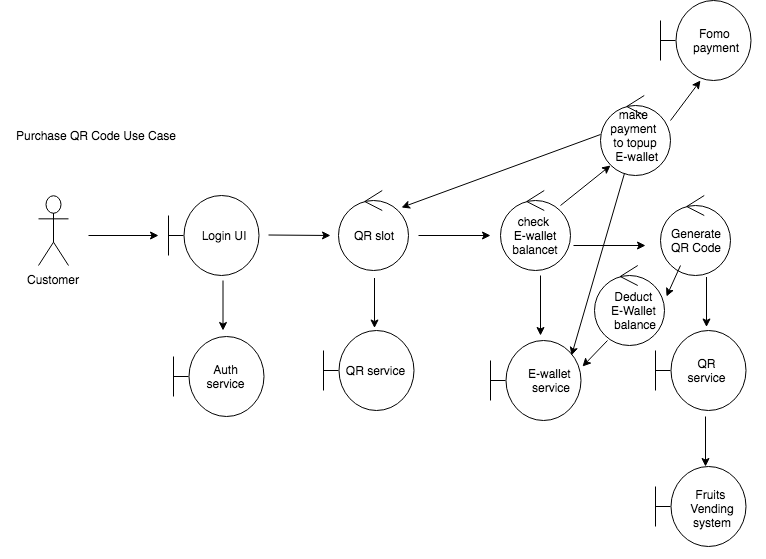
## Logical View

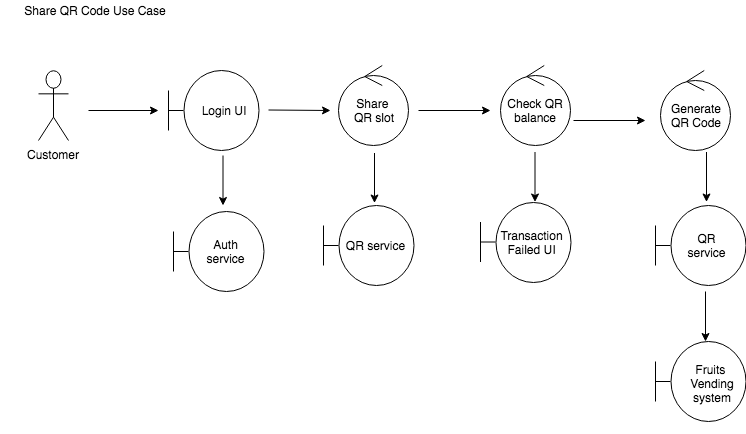
Describe the architecturally significant parts (or elements) of the solution and the design model showing its decomposition into subsystems using logical detail deployment diagram and other diagrams as necessary.

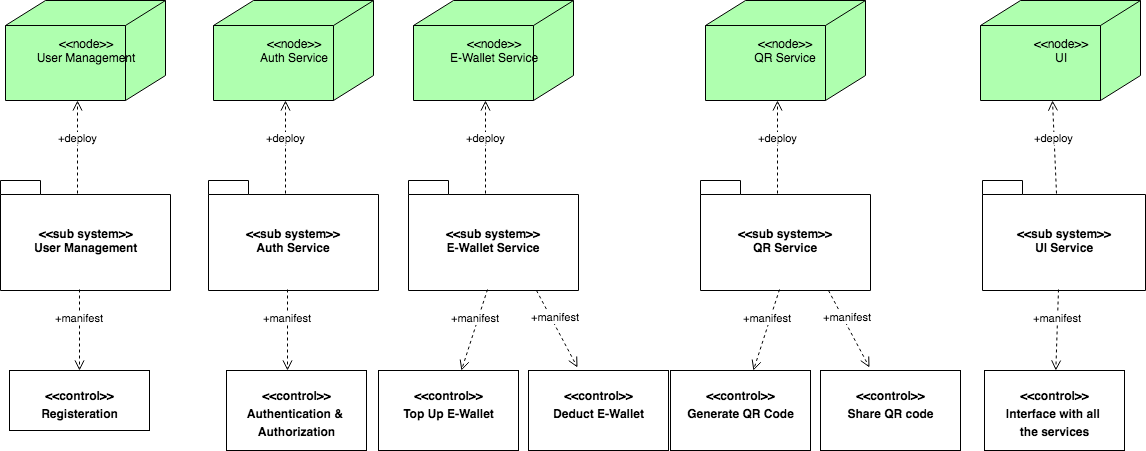
The individual elements of the solution must be clearly defined and assigned to the respective providers. An element (or subset of multiple elements) may be allocated to a Vendor.

A solution element is any part of the architecture of the overall solution, e.g., a COTS product, a custom-built software module, a data repository, a network device, etc. Each element can utilise the services provided by other elements and provide services of its own.

The logical view describes how elements participate in the solution. It includes static and dynamic relationships and interactions between elements. The documentation typically includes a number of diagrams expressing the different kinds of relationships — for example, dependency relationships, usage relationships, interaction relationships, etc







### Key Architectural Decisions

Provide a summary of the significant decisions made in arriving at the architecture of the solution that are related and/or reflected in the logical view. Documented decisions should focus on trade-off choices and rationale.

The key architectural decisions taken are as follows:

|  |  |
| --- | --- |
| Identifier | Description |
| AD-01 | ***Architectural Decision:*** *Develop an Auth service*  ***Alternatives:*** *Merge Auth service with user management service*  ***Justification:****1. Given Auth service will also be used by companies that use the platform with client credential grant type, the auth rule will become more complicated in the future. By decoupling Auth service from user management service, any new rules introduced to the auth service won’t impact user management.*  *2.On top of current business scope, we can also easily provide SSO for all the companies that use this platform by implementing “Authorization Code” grant type in current auth service.* |
| AD-02 | ***Architectural Decision:*** *Develop E-Wallet service*  ***Alternatives:*** *Merge E-Wallet service with QR service*  ***Justification:*** *Given the plan of expanding business to multiple companies, the E-Wallet can be used to purchase products from all the companies in the platform. Splitting E-Wallet increases the flexibility of adding more products in the future.* |

### Subsystems

Describe the architecturally significant parts (or elements) of the solution and the design model showing its decomposition into subsystems.

E-Wallet

QR

### Platform Design

Specifically describe how the platform is designed and interact with the system that leverage the platform (the applications)

1. Company register from UI
2. Company input contact information
3. Company input QR code generation endpoint , input parameter
4. Company register clientid & client secret from platform auth server
5. Company upload product image and input product properties.
6. System auto connect to new company and display new products for sale.
7. Company use clientid and client secret to connect the platform for QR code balance deduction.
8. System generates monthly sales report for each company and pay the bills offline.

### Tiers and Layers

Describe the layers and their responsibility in the system, e.g., Presentation, Business Logic, Data and Integration Tiers. Rename the section is you organized the solutions using different method (not tiered or layered approach)

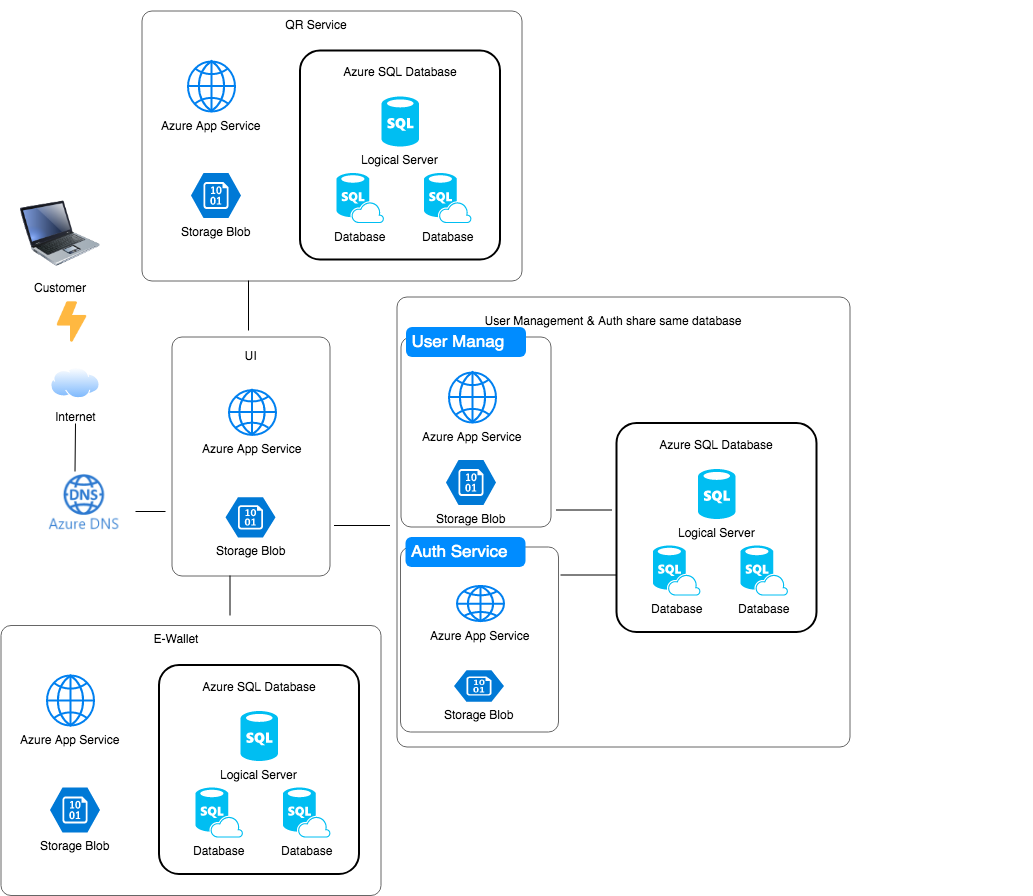
Presentation: UI Service

Business Logic: Integration Tiers are in respective REST API.

Data: SQL database

## Physical View

Describe the architecturally significant components (or elements) of the solution by describing the physical design which may include the network, technologies, nodes and machines where the components are deployed using physical detail deployment and other diagram as necessary. This is where you can also include and describe the cloud services that you use if any.



### Technology and Services

Describe the architecturally significant technology and cloud services that is used in the solution and describe the trade-offs and rationale.

**PaasS (Azure App Service & Redhat Openshift):**

**Platform as a service (PaaS):**

Platform as a Service (PaaS) is a category of cloud computing services that provides a [platform](https://en.wikipedia.org/wiki/Computing_platform) allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

**Trade-offs:**

1)With a PaaS, data security needs to be considered and scrutinised, as information is stored off-site.

2)A less than ideal PaaS provider could leave your company feeling frustrated.

**Azure App Service** is considered an excellent Platform as a Service (PaaS), offering an application platform for developers to build Web, mobile and API applications.

The platform handles **scaling** when your application needs additional compute resources and, at the same time, it handles the **load balance** traffic across multiple instances of your application.

**OpenShift** Online is Red Hat's public cloud application development and hosting platform that **automates the provisioning, management, and scaling of applications** so that you can focus on writing the code for your business, startup, or next big idea.

**.Net Core**

NET Core is an open-source, general-purpose development platform maintained by Microsoft and the .NET community on [GitHub](https://github.com/dotnet/core). It's cross-platform (supporting Windows, macOS, and Linux) and can be used to build device, cloud, and IoT applications.

**Trade-offs:**

**Azure SQL Database**

Azure SQL Database is a general-purpose relational database-as-a-service (DBaaS) based on the latest stable version of Microsoft SQL Server Database Engine. SQL Database is a high-performance, reliable, and secure cloud database that you can use to build data-driven applications and websites in the programming language of your choice, without needing to manage infrastructure.

**Trade-offs: Less secure**

In SQL Azure Database, if you have the credentials you can basically stole all the information from any device by default. When you need to connect with other tools, it is secure and it requires certificates or other secure tools, but the Web Portal can be accessed anytime and anywhere.

# IdentityServer4

IdentityServer4 is an OpenID Connect and OAuth 2.0 framework for ASP.NET Core.

OAuth 2.0 is the industry-standard protocol for authorization.

OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.

**Trade-offs:**

### Key Architectural Decisions

Provide a summary of the significant decisions made in arriving at the architecture of the solution that are related and/or reflected in the physical view. Documented decisions should focus on trade-off choices and rationale. You should continue the numbering from the previous architectural decisions table

The key architectural decisions taken are as follows:

|  |  |
| --- | --- |
| Identifier | Description |
| AD-03 | ***Architectural Decision:*** *Use PaaS*  ***Alternatives:*** *Use IaaS*  ***Justification:***  *1) PaaS can improve the speed of developing an app.*  *2)PaaS offers auto scaling and load balancing*  *3)* PaaS offerings may also include facilities for application design, application development, testing and deployment. |
| AD-04 | ***Architectural Decision:*** *Use .NET Core as development platform*  ***Alternatives:*** *SpringBoot based on Java*  ***Justification:***  1)The majority of developers in the team are familiar with C#  2)The sponsor required using c#. |
| AD-05 | ***Architectural Decision:*** *Use Azure SQL Server*  ***Alternatives:*** *SQL Server installed in VM*  ***Justification:***  1) No worry about the Administration of the Database  2) We do not have many resources or time to administer a SQL Server Database  3) Low budget |
| AD-06 | *Architectural Decision: Use IdentityServer4 as Auth service framework* ***Alternatives:*** *Simply generate JWT token*  ***Justification:***  *1)Given .Net Core was selected as the development platform,IdentityServer4 runs on top of .Net Core.*  *2)Given there are multiple grant types in OAuth2, “Password” and “Client Credentials” grant type meets our current requirement.*  *We can easily utilize this auth service to provide Single Sign On by adding “Authorization Code” grant type support.* |

## Data Design

State how data is managed in the solution, e.g., in a relational database. Describe any special considerations for transaction management, concurrency, etc. Data persistence — indicate which parts of the data model must persist beyond a transaction or session and the period of persistence. Provide a logical data model for the persistent data storage used in the solution.

Describe the information architecture of the solutions describing the type of data inventory and classification that you have and how the data will be stored, secured and handled. E.g.:

<Data Group>

Data format: <explain the different type of data that belong to this group and how the data is stored as record or flat file or other form of data, in what format(JSON, XML, etc)

Proposed storage technology: <explain the technology used to store this data e.g. Oracle RDBMS, MongoDB document database, etc with the rationale>

Security requirements and controls: <explain in high level the security requirement for this data and what are the security controls that are applicable for this data>

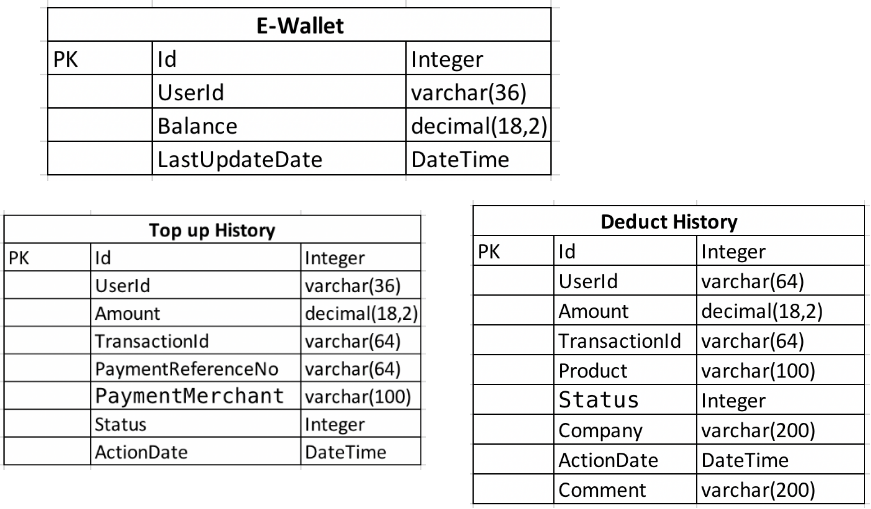
Trade-off consideration: <document the trade-off consideration that has been considered in making this decision.

<E-Wallet>

1. The data is stored in relational database (MSSQL Server).
2. The add/update of Top up History and Deduct History must be in same transaction as E-Wallet table.

One E-Wallet has many Top up History and many Deduct History.

1. Refer to below diagram of table design.
2. Security control: Only the app service in current user account is allowed to access the database.
3. Logging is saved in file which can be saved in Aaure Storage Blob.



## Other Architectural Decisions

Provide a summary of the other significant decisions made in arriving at the architecture of the solution that has not been mentioned. If you explain the rationale of the decisions in other section (such as the quality attribute sections), you can make a reference to those sections.

The key architectural decisions taken are as follows:

|  |  |
| --- | --- |
| Identifier | Description |
| AD-03 | *…* |
| AD-04 | *…* |

## Architectural Issues

(Optional but it’s important to acknowledge any known limitation of your solution before any stakeholder point them out to you) Include any outstanding issues from a solution architecture and high-level design perspective which need to be resolved. For resolutions that extend beyond existing project plans, there should be a roadmap with a timeline for closing the issue.

The key outstanding architectural issues in the solution are as follows:

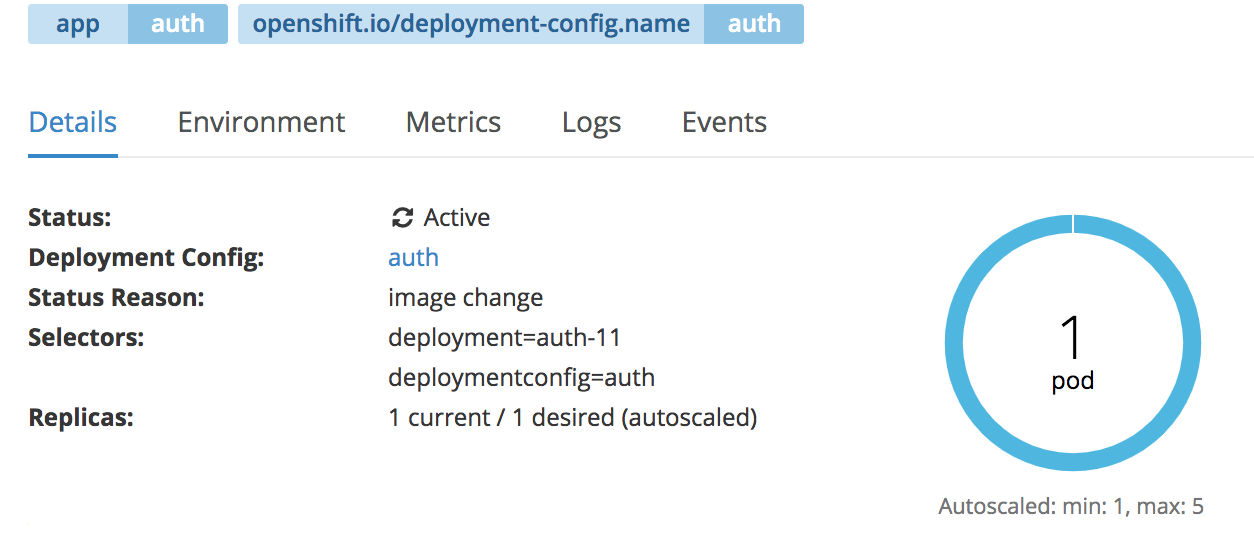
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Identifier | Issue and impact | Description | Resolution | Owner | Status |
| AISS–01 | Auth service not able to deploy to Azure App Service | Auth service throws anonymous logon error when deploying to Azure App Service | Deploy Auth Service to Openshift online | Zou Xuan | Closed |
| AISS–02 | … | … | … | … | … |

# Quality Attributes

## Performance

The purpose of this section is to describe the solution handles the performance requirement of the system. You can describe the use of caching, load balancing, any optimization that you plan/have done in the design of the system

1. With the use of PaaS, we have added auto scale out for Auth service, when there is intensive traffic coming in, openshift will auto scale to max 5 instances (based on below setting).



2. We plan to use same strategy for other services.

## Availability

The purpose of this section is to describe how the solution handles high availability requirement.

Deploy the application in cloud automatically inherits the feature of high availability.

## Security

The purpose of this section is to describe the security that will be incorporated into the solution which includes any security services: IAM, encryption, hashing, IDS, IPS, etc.

1. Password hashing.
2. Secure each service with JWT token.
3. Set firewall in each database to allow specific IP address or App service to access.
4. Configure Certificate and connection string in environment variable instead of committing to code repository.

## Maintainability

The purpose of this section is to describe the solution handles the maintainability requirement. How the platform and new application can be enhanced and maintained in the future. You can refer to the platform section if necessary.

Changeability:

1. Use ORM (Entity Framework Core) to connect to database which makes database replaceable.
2. Separation of concerns : Split the application into 5 micro services.
3. REST : Calling service with REST protocol which makes the development programing language decoupled from the application.
4. Using flexible JSON data format between services invocation.

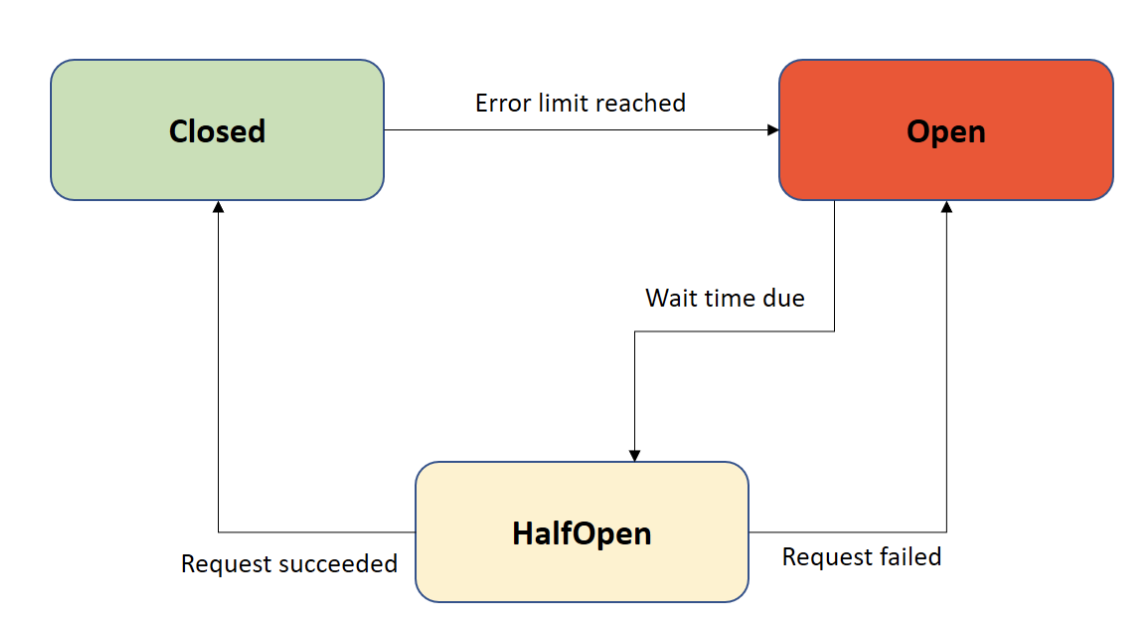
Analysability:

1. Writing application log to file

2. Can enable Azure diagnose functionality to diagnose issue.

Stability: Plan to implement circuit breaker pattern.

Circuit Breaker: The circuit breaker is a proxy that controls flow to an endpoint. If the endpoint fails or is too slow (based on your configuration), the proxy will open the circuit to the container. In that case, traffic is routed to other containers because of load balancing. The circuit remains open for a preconfigured sleep window (let’s say two minutes) after which the circuit is considered “half-open”. The next request attempted will determine if the circuit moves to “closed” (where everything is working again), or it it reverts to “open” and the sleep window starts again. Here’s a simple State Transition Diagram for the circuit breaker:



Testability:

1. Use Repository pattern and Unit of Work pattern together to increase maintainability, flexibility and testability

1. Automated testing: Each service should have Unit Test (Only one service implemented UT for now), and we plan to have automated Integration Test.
2. CI/CD: The services deployed in Azure App Services are configured with Azure DevOps.

# DevOps and Deveopment Lifecycle

## Source Control Strategy

Describe the strategy that you use to manage your source code and other artefacts in your configuration management. Include your project structures, repository strategy, branching strategy, authentication of develoeprs, etc.

1. Each micro service has one repository.
2. Use git as version control system.

## Continuous Integration

Describe the pipeline on how you perform continuous integration in your project including the trigger for each integration jobs, the various jobs, tests, etc.

Azure DevOps to 1. Compile on each code push

## Continuous Delivery

Describe the pipeline on how you automate deployment to different environment including how many environments are setup, who is responsible for approving promotion between environments and how deployment get verified.

Azure DevOps1. After Compile and unit test passed

# <other things to be highlighted>