High Level Architecture

Pokedex REST API that returns Pokemon information. In Pokedex product scenarios, authentication will be handled centrally. Gateway is a good place to authenticate microservices. To get a response from Pokedex Service, users have to authenticate through an authentication server if token enable is true. After validating credentials, the authentication server will return a token to invoke Pokedex REST API.

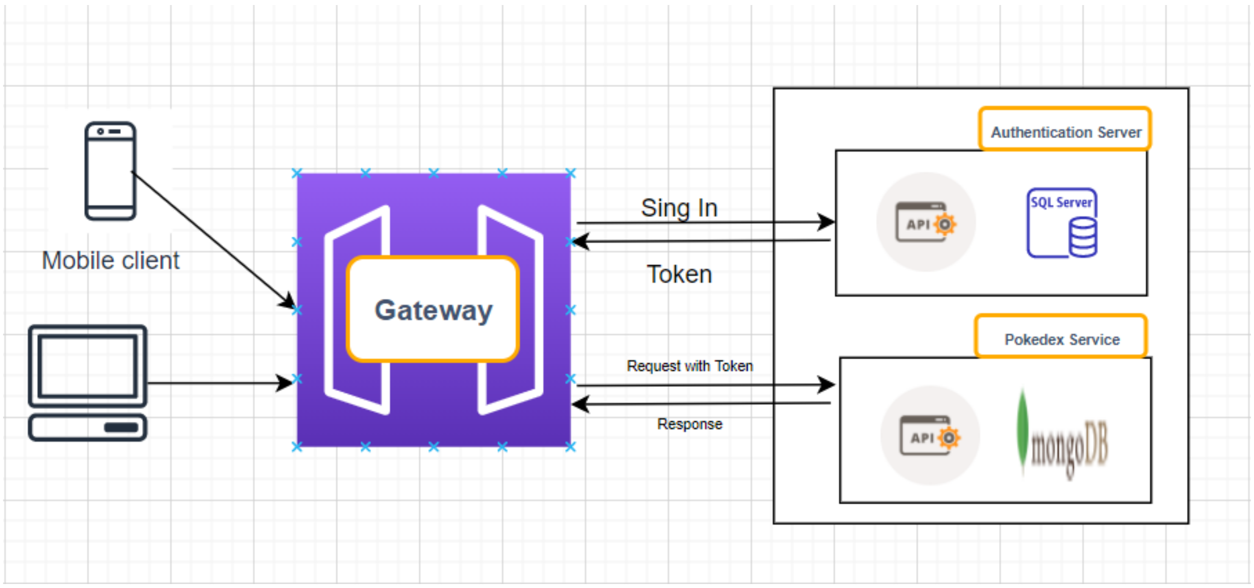
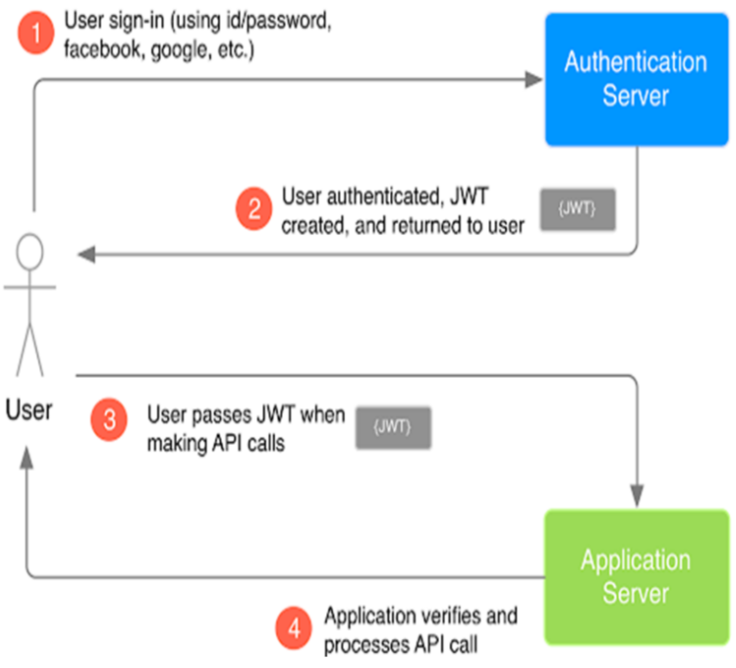


Figure 1: High level architecture of Pokedex

Low Level Design

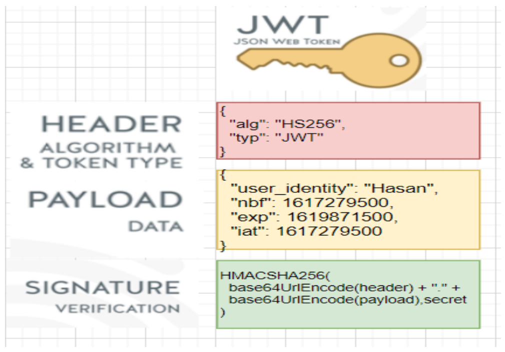
Authentication_{Optional}


1 User sign-in (using id/password, facebook, google, etc.)
Authentication Server
2 User authenticated, JWT created, and returned to user
{JWT}
User
3 User passes JWT when making API calls
{JWT}
Application Server
4 Application verifies and processes API call

Assuming the user sends an authentication request (login request) to the authentication server with credentials (username and password).The Authentication Server takes this information and queries it from the database, combines the user name with its secret key and generates the token.When the user authenticates, the Authentication server returns **token** and the server writes this information to the related user's Token column in the database.The user passes **JWT token** when making API calls into header part of the request and the Pokedex server checks each time whether the token is valid or not.

Pokedex service verifies and processes API calls.

JWT Structure



```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "user_identity": "Hasan",
  "nbf": 1617279500,
  "exp": 1619871500,
  "iat": 1617279500
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),secret
)
```

The token produced with JWT consists of 3 main parts encoded with Base64. These are **Header** (Header), **Payload** (Data), **Signature** parts.

Let's take a closer look at these parts. If we pay attention to the token example on the side, there are 3 fields separated by dots in the token form.

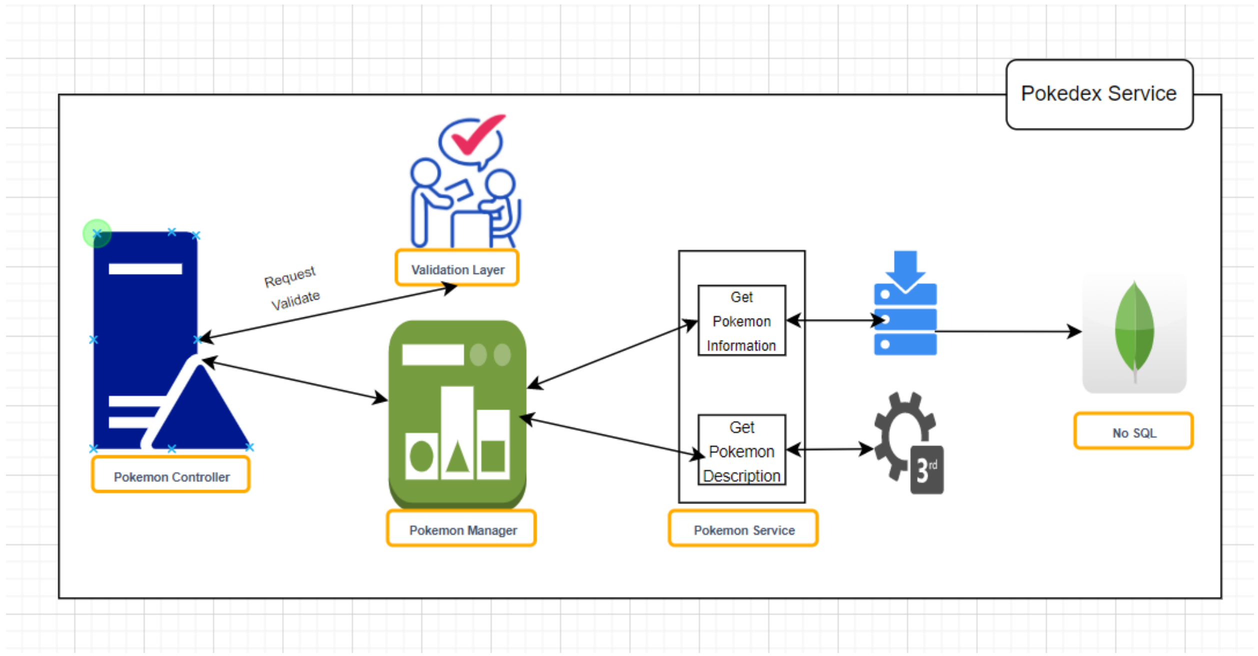
Pokedex Service

Pokedex REST API that returns Pokemon information. For example, Given a Pokemon name, returns standard Pokemon description and additional information and Given a Pokemon name, returns translated Pokemon description and other basic information using the following rules -

1. If the Pokemon's habitat is a cave or it's a legendary Pokemon then will be applied to Yoda translation.
2. For all other Pokemon will be applied to the Shakespeare translation.
3. If you can't translate the Pokemon's description then it will be used as the standard description.

Low level architecture

Pokedex data will be saved in no sql server (MongoDb). For fun translation, It will use external interface like Yoda/Shakespeare Translation.



Database Design

Pokedex database requirements follows :

1. Billions of records need to be saved.
2. Each object will save as small as possible data.
3. There is no need for a relationship between records.
4. The system has a high read rate.

Database Schema

Currently Pokedex has implemented one main table to get pokemon information without any relationship to its entity by MongoDB. It is also possible to design this entity into two tables. One will be Pokemon and another it’s habitat. Pokemon table will contain habitat primary id. If we implement two tables with its relationship then better to implement SQL Server.

Pokemon	
Name	***string***
Description	***string***
Habitat	***string***
IsLegendary	***boolean***

Logging and error handling

The Pokedex system is properly logged scenario basis and has handled all possible exceptions. Detailed error handling code and message will be added to the API specification error code section.