CS4471b/CS9549b – Software Design and Architecture

Project: The PurplePages - Team 1

By: Chengyuan Zhang
Gary Chiu
Matthew Dudycz
Vatsal Shah
Wang Lyu

Progress Report 2

Table of Contents

Service Offered

ID	Title	Description
ID: S1	Ski Resort	What: Find many ski resorts easily from a web portal. Why: Easy to find many ski resorts. Save time and more details will be added over time.
ID: S2	Restaurants	What: Find many restaurants easily from a web portal. Why: Easy to find many restaurants in the US. Save time and more details will be added over time.
ID: S3	Museums	What: Find many museums easily from a web portal. Why: Easy to find many museums in the US. Save time and more details will be added over time.
ID: S4	Software Companies	What: Find all fornute companies easily from a web portal. Why: Easy to find all fortune companies. Save time and more details will be added over time.
ID: S5	GUI	What: Provides a graphical interface to users. Why: Allows for all users to understand and utilize the full value of the system.
ID: S6	Registry and Discovery	What: Identifies services that are active and should be available to users. Why: To facilitate the microservice structure, it is necessary to display only the services that are active, to allow others to be modified.
ID: S7	Authentication service	What: Validate user credentials and provide access to the system. Why: Allows the restriction and coordination of users for the system.

ID: S8	Gateway	What: Facilitates the connection between individual services and the UI. Why: Allows for manipulation of services to facilitate the microservice structure.

Service Requirements

Service	Requirement
S1: Ski Resort	FR1 Lookup resorts that are located in a specified location, e.g. Continent/Country/Country+Province
	FR2 Find resorts whose ticket prices fall within a specified range, e.g price<200
	FR3 Search resort(s) that matches the specified name or the first N type-in characters
	FR4 Search resorts by a specified slope rating.
	QR1 Service can be quickly changed and removed
	QR2 More data can be added in database + enhancement can be done for advance search feature
	QR3 Availability: the service should be built to handle(mask) certain exceptions like invalid query.
	QR4 Modifiability: all components should have low-coupling and high-cohesion relationship.
	QR5 The search function should be responsive to any query within few seconds.
S2: Restaurants	FR1 Look up restaurants that are located in a specified location, e.g. State
	FR2 Search restaurants by a specified cuisine category
	FR3 Search restaurants by a specified name
	QR1 Service can be quickly changed and removed
	QR2 More data can be added in database + enhancement can be done for advance search feature
	QR3 Availability: the service should be built to handle(mask) certain exceptions like invalid query.
	QR4 Modifiability: all components should have low-coupling and high-cohesion relationship.
	QR5 The search function should be responsive to any query within few seconds.
S3: Museums	FR1 Look up museums that are located in a specified location, e.g. State
	FR2 Search museums by a specified name
	QR1 Service can be quickly changed and removed
	QR2 More data can be added in database + enhancement can be done for advance search feature
	QR3 Availability: the service should be built to handle(mask) certain exceptions like invalid query.
	QR4 Modifiability: all components should have low-coupling and high-cohesion relationship.
	QR5 The search function should be responsive to any query within few seconds.

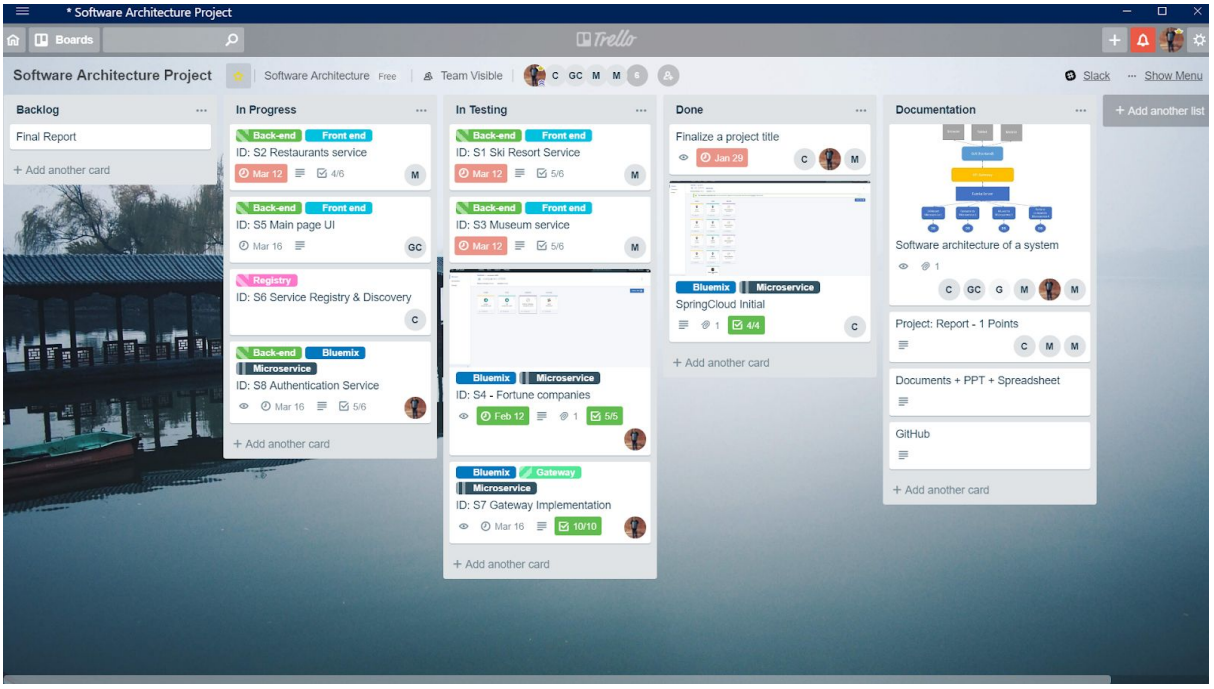
S4: Software companies	FR1 Look up companies that are located in a specified location, e.g. State
	FR2 Search companies by a specified name
	FR3 Search companies by a website
	FR4 Search companies by an industry
	QR1 Service can be quickly changed and removed
	QR2 More data can be added in database + enhancement can be done for advance search feature
	QR3 Availability: the service should be built to handle(mask) certain exceptions like invalid query.
	QR4 Modifiability: all components should have low-coupling and high-cohesion relationship
	QR5 The search function should be responsive to any query within few seconds.
S5: GUI	FR1 Displays available services to users
	FR2 Takes user input for a respective service
	FR3 Returns appropriate data for a query
	QR1 Availability: the service should be built to handle(mask) certain exceptions like invalid query.
	QR2 Modifiability: all components should have low-coupling and high-cohesion relationship
	QR3 The search function should be responsive to any query within few seconds.
	QR4 Usability: Presents services and responses to users in a consumable format
S6: Registry and Discovery	FR1 Identifies active services
	FR2 Communicates to other components which services are active
S7: Authentication service	FR1 Accept user input of credentials
	FR2 Validate credentials against system
	FR3 Provide users with appropriate level of access
	FR4 Creates a user account for the specified credentials
	QR1 Modifiability: all components should have low-coupling and high-cohesion relationship
	QR2 Security: Credentials cannot be accessed by unintended users
	QR3 Security: Correctly restricts access
S8: Gateway	FR1 Facilitates communication between the UI and the respective services
	QR1 Modifiability: all components should have low-coupling and high-cohesion relationship

Project Plan

Trello

V2 - 20190312

A Final project report is in backlog process. In progress indicates the working process for each service. Other microservices like Gateway, authentication and other services are deployed on IBM bluemix. Documentation block represents report, and other credentials.



Spreadsheet

Service ID	Service Name	Task	Status 1	Status 2	Agent
S1	Ski Resort	Find dataset	Done	-	Wayne
		Data upload to database	In Progress	Done	
		Create DAO classes	Not started	Done	
		Create SVC class(es)	Not started	Done	
		Create Controller class		**Done**	

		Perform Testing		**In progress**	
S2	Restaurants	Find dataset	Done	-	Matthew
		Data upload to database	In progress	Done	
		Create DAO classes	Not started	Done	
		Create SVC class(es)	Not started	Done	
		Create Controller class		**In progress**	
		Perform Testing		**Not Started**	
S3	Museums	Find dataset	Done	-	
		Data upload to database	In progress	Done	
		Create DAO classes	Not started	Done	
		Create SVC class(es)	Not started	Done	
		Create Controller class		**Done**	
		Perform Testing		**In progress**	
S4	Software companies	Find dataset	Done	-	Vatsal
		Create node.js project	Done	-	
		Connect database and link with Bluemix	In progress	Done	
		Find deatils from database and return a response to it	Done	-	
		Create bluemix node js server	Done	-	
		Add mongodb (mLab) connection	**-**	**Done**	
		Add api endpoint for customise search functionality	**-**	**In Progress**	
		Deployment on Bluemix server and create endpoint of a service	**-**	**Done**	
S5	GUI (Graphical	Initialize react app	Done		Gary
		Connect react app with Bluemix	In		

	User Interface)		progress		
		Implement dashboard with search functionality for services	In progress		
		Implement results screen for each service			
		Create registration and login page	**-**	**In Progress**	
		Implement results screen	**-**	**In Progress**	
S6	Registry and Discovery	Initialize Eureka Server	Done	-	Chengyuan
		Initialize Eureka Clients	In progress	**Done**	
		Enable Spring cloud Gateway	Done	-	
		Enable RESTful API UI (Swagger)	In progress	**Done**	
		Enable Feign	Not Started	**Done**	
S7	**Authentication Service**	**Create table for a user**	**-**	**Done**	Vatsal
		Connection with database	**-**	**Done**	
		Verify a user with POST event of login API	**-**	**Done**	
		Verify a user when a new request arrive on a server	**-**	**Done**	
		Create a token using JWT and store	**-**	**Done**	
		Deployment of a service on IBM	**-**	**Done**	
		Verify and return response to gateway	**-**	**Done**	
		In Testing & Deployment	**-**	**In Progress**	
S8	**Gateway Service**	**Create gateway (nodejs service) on IBM bluemix**	**-**	**Done**	Vatsal
		Add endpoint of all microservices to access it	**-**	**Done**	
		Create an endpoint for a client to access list and search	**-**	**Done**	
		Create a endpoint to access live services from a registry	**-**	**Done**	
		Authentication process of a user	**-**	**Done**	

Summary of Documentation Beyond Views

Document Control Information

Issuing Organization: Team 01 - CS9549
Version Number: 1.0
Date of Issuing: 13/03/2019
Status: Under development
Change History:

- 13/03/2019: Created

To submit a change request, please email the request to mdudycz@uwo.ca.

Section 1: Documentation Roadmap

Scope and summary

This document is intended to provide insight into the architectural design and reasoning of the PurplePages project. It outlines the details regarding the structure and pattern implementation of each microservice, along with the delivery method, and the intended user interactions.

Within this document are three view templates, corresponding to the Module view, Component-and-Connector view, and Allocation view. There are also two quality views and an outline of the service-oriented architecture pattern.

Organization

Section 2: This section outlines the documentation standards employed throughout this document.

Section 3: This section outlines the system's intended functions and interactions with users. It also contains an outline of the service-oriented architecture view.

Section 4: This section provides an understanding of the various interactions between the views outlined in this document.

Section 5: This section elaborates on the design decisions of the development team, outlined within the document.

Section 6: This section offers reference material to aid in understanding the document.

View Overview

The module view demonstrates all the layers involve in a microservice system. It also includes the interrelationship between each layer. It serves as a guideline for the organization of modules and their responsibilities.

The component-and-connector view is an overall view of a system with the component connection. It's a relationship between all components and with each other. It consists of client, gateway and microservices.

The allocation view has utilized the deployment view of a system.It gives a brief view of how to deploy three parts of the whole project on bluemix and how to deploy them with specific parameters. All the microservices and database are included to make a clear identification on the whole project.

The first quality view is 10_Quality View: Ski Resort Sequence Diagram. It depicts a sequence diagram for the submission and reception of a request to the Ski Resort Service. It is modelled as a RESTful API to facilitate the client-server structure. The elements featured are the user and the respective components that decode the request to access information from the database. The diagram is modelled over instances and time, reflecting the interactions between elements as the sequence progresses.

The second quality view is 11_Quality View: Login Use Case Diagram. It depicts a use case of the login process. The diagram features the specific use cases, and their interrelations in the process to validate a user. It also features actors, in this specific case, the user attempting to log in.

Documentation Relevance

View	Stakeholders and Concerns	Example
Module View	● Users ● Acquirer ● Developers ● Maintainers	● A user wishes to retrieve information about a resort ● An acquirer wishes to integrate their own modules in data retrieval ● A developer wants to integrate a new service using the existing module structure ● A maintainer wishes to know which modules will be affected by a change
Component-and-Connector view	● Users ● Acquirer	● A user wishes to understand the dataflow within the system

	● Developers ● Maintainers	● An acquirer wishes re-direct the dataflow to their own applications ● A developer wishes to integrate new validation information into the dataflow ● A maintainer wishes to identify the source of a resulting error
Allocation View	● Users ● Acquirer ● Developers ● Maintainers	● A user wishes to know which environment the front ends can be executed in ● An acquirer wishes to know if services can be transferred to other environments ● A developer wishes to add a new service to an existing environment ● A maintainer must migrate a service to an new environment
Quality View: Ski Resort Sequence Diagram	● Users ● Acquirer ● Developers ● Maintainers	● A user wishes to retrieve information about a resort ● An acquirer wishes to communicate resort detail to their clients ● A developer wishes to add additional search functions ● A maintainer wishes to update the format in which resort information is sent
Quality View: Login Use Case Diagram	● Users ● Acquirer ● Developers ● Maintainers	● A user wishes to be authenticated to access services ● An acquirer wishes to integrate their users into the system ● A developer wishes to add additional information to the login process ● A maintainer wishes to manually check if a user should be validated

Section 2: How a View is Documented

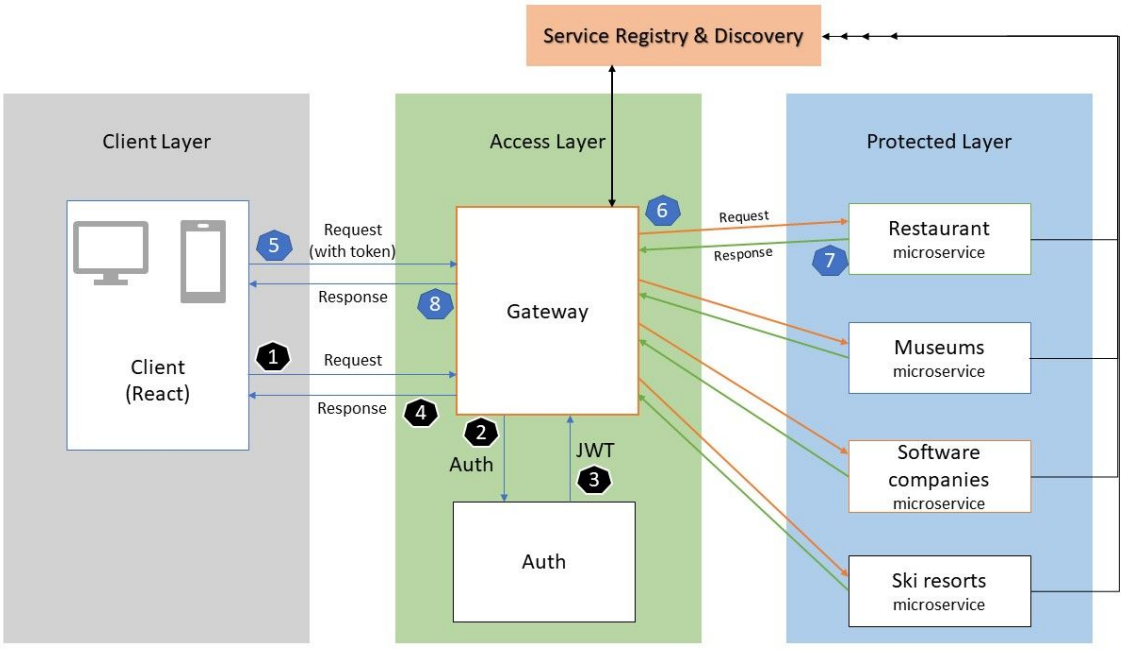
The views contained in this document match that of the textbook “Software Architecture in Practice: Third Edition”, reflecting the documentation philosophy of ISO/IEC/IEEE standard 42010:2011.

Section 3: System Overview

The primary function of this system is to provide directory relevant microservices, imitating that of a directory book such as the Yellow Pages. These services are accessible to multiple independent users seeking information from a web portal. The microservice structure allows for on-the-fly editing, adding, and removal of services by administrators. As the system stands, it offers directory information on fortune 500 companies, ski resorts, museums, and restaurants. Presently, the system relies on information sourced from pre-existing databases, which is hosted and managed through the IBM Cloud.

SOA View:

Software-oriented architecture presents in layers. An end-user can access the client layer. An end-user can access a website from their browser or phone device. Client services send a valid token to a Gateway, and gateway verifies that token. A Gateway sends a request to an Auth service and middleware verifies authentication. If a request is authenticated only then can a user access the protected layer. The protected layer consists of all microservices offered by the system. Each microservice is connected with a database. Service Registry and discovery identifies live/dead microservices.



Section 4: Mapping Between Views

View1	View2	Correlation
Module View	C&C View	C&C View shows the architectural details of the system, while the Module Views unveils the the implementation level details. That is to

		say, the Module view can be used to flesh out the details of each component in the C&C View. In turn, C&C View connects each module with others to shed lights on how each module can fit in the bigger picture which is the overall system.
Module View	Quality View	Quality View talks about the actual implementation methods, strategies and procedures of Module View.
C&C View	Allocation View	Allocation View supports the C&C View when stakeholders planning to deploy the system on the hardware level. At the same time, C&C View requires Allocation View to meet exact conditions in order to run the system smoothly and user-friendly.

Section 5: Rationale

The primary focus of this system is to implement an architectural structure to facilitate the use of multiple microservices. As such, the views reflect a client-server structure, where the client can request information from the server, ranging from which services are available, to the information each service provides.

The core microservices reflecting the functionality of a YellowPages directory all utilize a RESTful implementation to retrieve information from independent databases. This design was specifically chosen to allow individual services to go down, for editing or deletion, without impacting the end user.

All microservices are tested from a postman environment. Each microservice returns a common response with code and message. A 200 code is for success, 500 for a server-side error, 401 for a client error. Moreover, the message contains success, error message, or JSON response of the microservices.

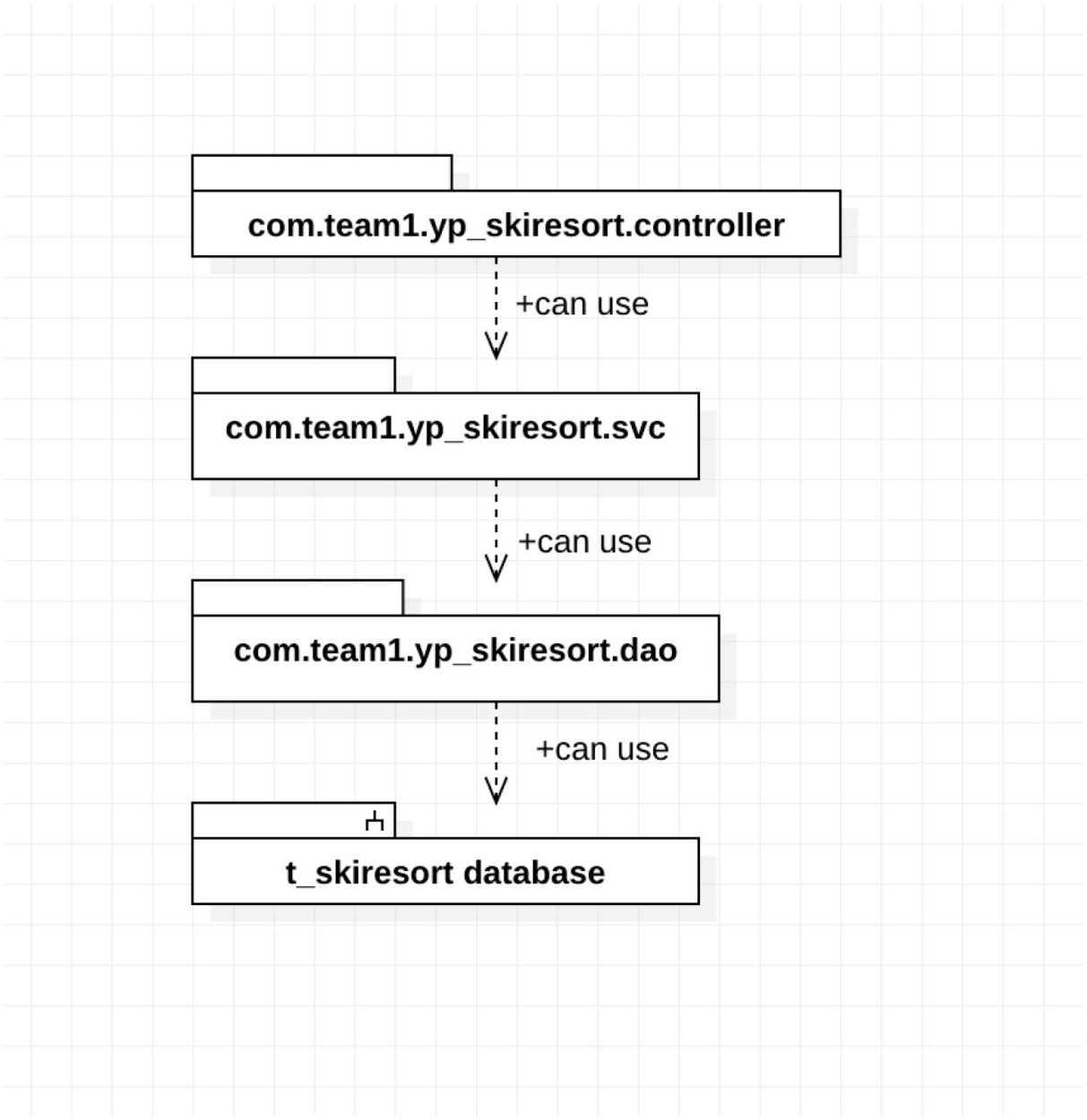
Section 6: Directory -- index, glossary, acronym list

Please see the document-wide directory.

Module View

Layered View of Ski Resort

Section 1: The Primary Presentation



Section 2: The Element Catalog

2.1 com.team1.yp_skiresort.controller

The controller class inside this folder, namely ‘SkiResortController’ in this web service, is built as the HTTP request processor. By annotated with @RestController, this class is equipped with the ability to process the incoming request and generate an HTTP response. In Spring framework, all requests that originated from UI are sent to built-in DispatcherServlet, which is the hub for all HTTP request, and DispatcherServlet would dispatch the request to the controller that is registered for processing a request that sent to a specified endpoint.

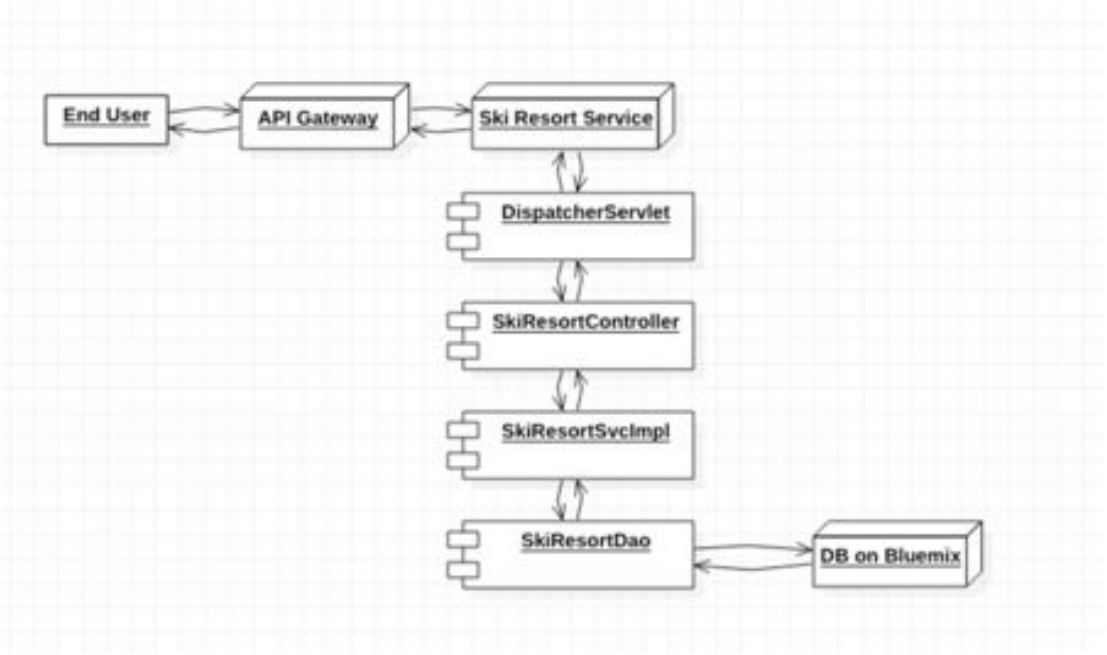
2.2 com.team1.yp_skiresort.svc

The ‘SkiResortSvc’ interface, which locates in the fold, list the functions that the Ski Resort service is bound to offer. The ‘SkiResortSvcImpl’ is the implementation of the previous interface. These functions invoke corresponding functions in the ‘SkiResortDao’ interface, retrieving data from DB.

2.3 com.team1.yp_skiresort.dao

The ‘SkiResortDao’ interface is implemented as a data-accessing-object or ‘Dao’ in short. Functions defined within this interface interact with the database to retrieve data. In the SkiResort service system, the interaction with DB is implemented with Mybatis persistence framework. It maps functions in this interface to the SQL included in the ‘mapper.xml’ file, which significantly simplified the coding compared to traditional JDBC

Section 3: Context Diagram



Section 4: Variability Guide

There are no variation points for this diagram.

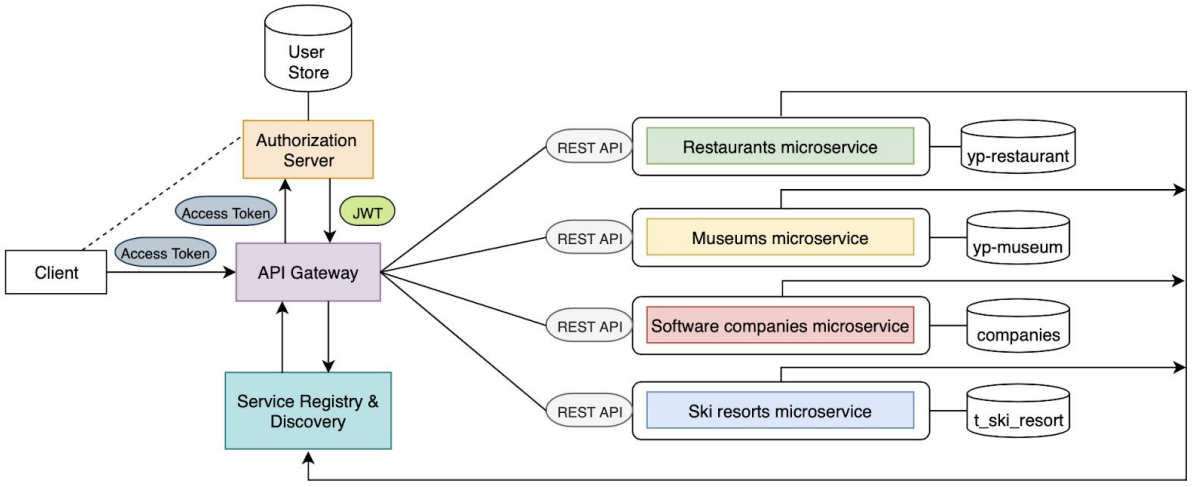
Section 5: Rationale

This diagram demonstrates the layered architecture of the web service. The design is derived from both SOA (service-oriented architecture) and MVC pattern. Specifically, class ‘SkiResortController’ and ‘SkiResortSvcImpl’ are both components locate in the controller layer, while the interface ‘SkiResortDao’ sit one layer below in the module layer. The whole system would be benefited by the layered design by better modifiability and scalability. In addition, the DI feature that comes with the Spring framework can lead to the desired low-coupling design, which would further improve the modifiability and scalability.

Component-and-Connector View

Service-Oriented Architecture (SOA) View

Section 1: The Primary Presentation



Section 2: The Element Catalog

2.1 Client layer

A client layer is accessed from mobile or browser. An only authenticated user can access micro-service. The client sends a token in a header form to a Gateway. It can be accessed from tablet, phone or browser. Client layer handles a return response from the access layer for authenticated user or not-authenticated users.

2.2 Access Layer

The access layer is a combination of Gateway and authentication service. Gateway handles all request from a client and communicates with authentication service. If Gateway get a request without token it will return not-authenticated user response. When a client sends a request with a token, Gateway verifies a request with authentication service and forward to protected layer to access each microservice. Protected layer returns microservice response.

2.3 Protected layer

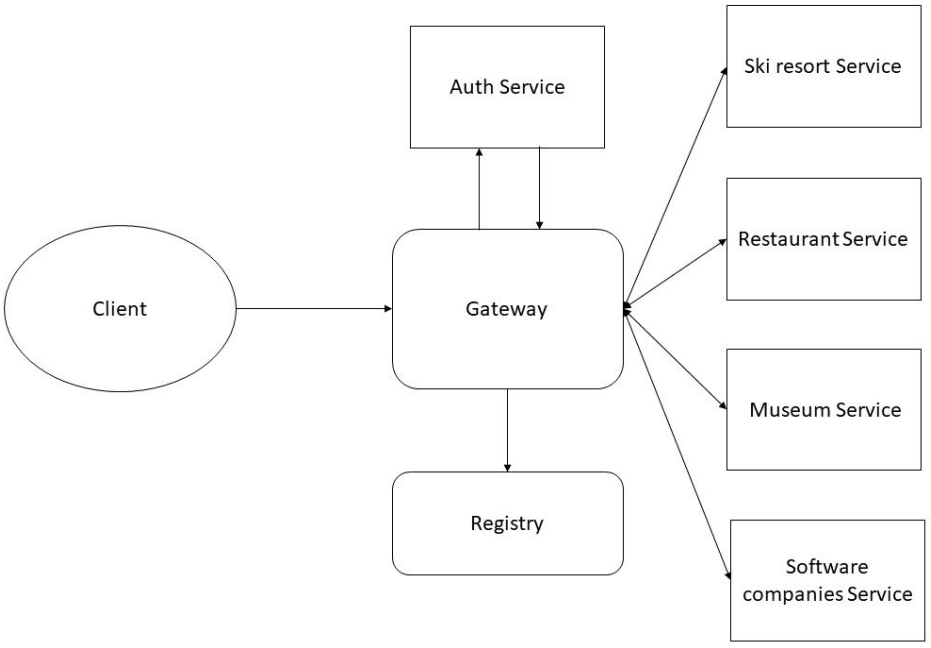
The elements within the protected layer can be accessed only via the API gateway. The requests that originated from the UI will first send to Access layer where the API gateway

locates, then these requests would be routed by the API gateway to the corresponding services based on the entries in the registry. After processing, the responses from microservices in the protected layer would be transmitted back to the API gateway. In this sense, the protected layer is hidden behind the access layer. If any elements need offline for many reasons, the function of that element would be automatically decommissioned in the registry and subsequently API gateway. So without restarting the whole system, we can add/remove elements in the protected layer. This is the flexibility that is provided by the design.

2.4 Service registry & discovery

Service registry and discovery come from a eureka organization and are implemented by Java Spring. Discovery is responsible to receive heartbeats from registries in order to maintain a list of alive microservices. At the same time, in terms of making sure synchronization, the registry will check the status of microservices positively. Finally, API Gateway can access to all microservices registered in the discovery server and return retrieved data to the frontend.

Section 3: Context Diagram



Section 4: Variability Guide

If gateway configurations changes, client connection will break-down. All the services end-points are defined in a Gateway. So, if any microservice API-endpoints are updated it is necessary to update in Gateway configuration file. Moreover, if database connection as login

credentials, URL or database name updates, it should be updated in each microservice. All microservice has a logging system to check logs of each service.

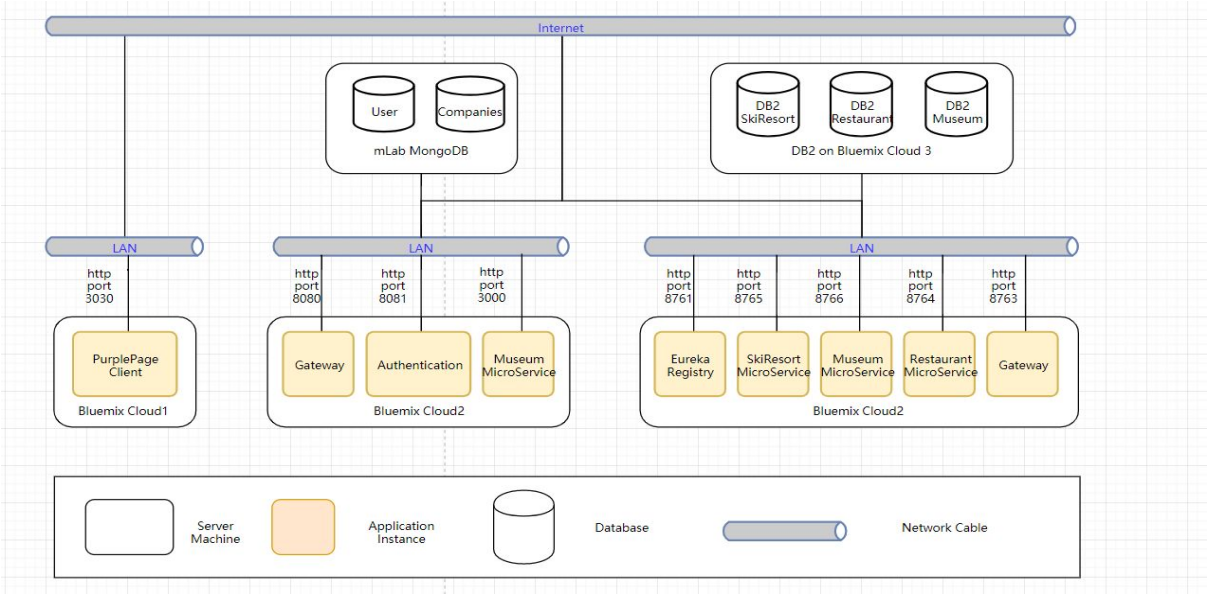
Section 5: Rationale

This identifies components of a system. We used various framework and language as Java Spring, Node.js and React. Gateway is implemented using node.js, Client is in React.js and other microservices are in Java Spring. Moreover, microservice, in terms of a kind of SOA architecture, need a registry to manage all the microservices. Eureka has set the connections between the server and clients by allowing the client to send a heartbeat to the server and requiring the server to check the status of clients lifecycle in an exact time interval. All components are deployed on Bluemix and able to accessed by users anytime and any platform with the responsive layout of a website.

Allocation View

Deployment view of a system

Section 1: The Primary Presentation



Section 2: The Element Catalog

2.1 Elements and their properties

There are three could instance deployed on bluemix in our project. PurplePage Client is the frontend of our project, it is responsible to interact with users and deployed on Bluemix Cloud1.

2.2 Relations and their properties

Both Bluemix Cloud2 and Cloud3 are in charge of some microservices. For the Cloud2, there are Gateway, Authentication and Company Microservice which are implemented by NodeJS. They all connect to the mLab MongoDB that has two tables, user table and companies table. Authentication service shares the same database with login and signup functions. For this part, they have already deployed on bluemix but we design it properly so that it can be deployed in any system, environment, cloud platform.
The last part, including Microservice registry, request gateway, three microservices are all deployed on Bluemix Cloud3. All the Eureka clients consisting of three microservices and gateway will send the heartbeat to notify the action of registering and in Eureka server will check whether the service is still alive periodically.

Section 3: Context Diagram

Totally the same as Section 1.

Section 4: Variability Guide

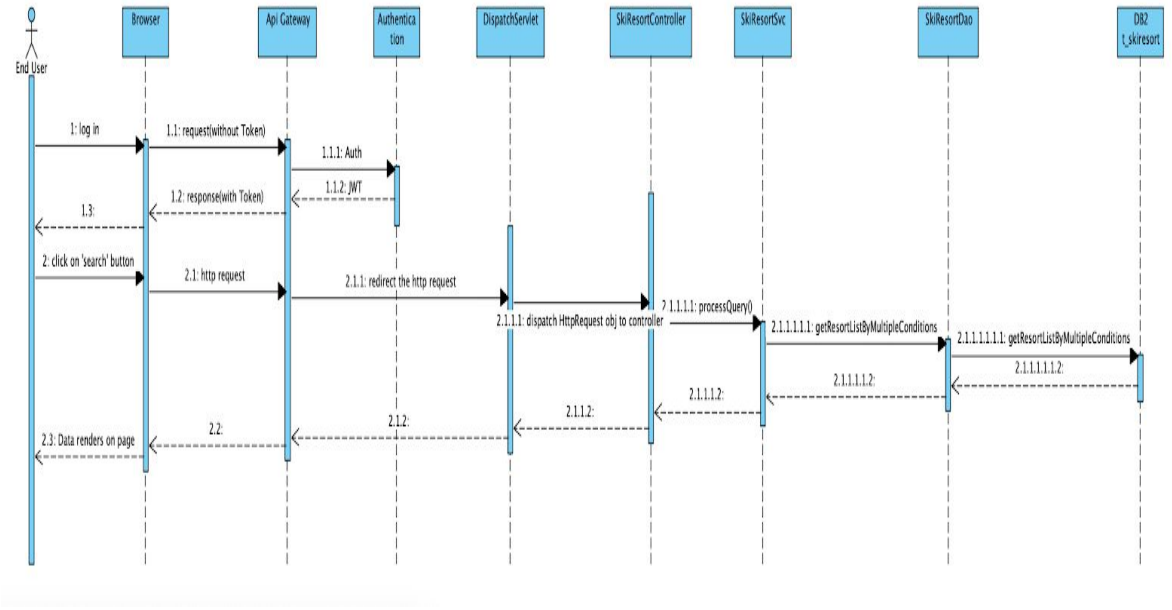
In the Registry Center, we should config the Eureka server host and port number. Furthermore, we need to set some properties to manage the usability of both server and clients. Besides, the source code needs to config the database connection in the main *application.yml.* Therefore, the DAO layer can return data from a database located in different servers or hosts.
As mentioned before, we decide to deploy the applications on bluemix, so that we need to utilize pipeline in bluemix to integrate the compile and deploy procedures. Some parameters like service bound name, runtime memory and service start timing should be set properly according to the account's authority.

Section 5: Rationale

The project the frontend part to interact with users and the NodeJS is an effective language to implement this functionality. Moreover, in order to imitate the actual development environment and requirements, we decide to utilize two server language to implement all the logic layers. Therefore, we try to use bluemix to implement two different kinds of microservice to integrate the project. Moreover, microservice, in terms of a kind of SOA architecture, need a registry to manage all the microservices. Eureka has set the connections between the server and clients by allowing the client to send a heartbeat to the server and requiring the server to check the status of clients lifecycle in an exact time interval. All three parts are deployed on Bluemix and are able to be distributed with domains and accessed by users anytime and any platforms.

Quality View: Ski Resort Sequence Diagram

Section 1: The Primary Presentation



Section 2: The Element Catalog

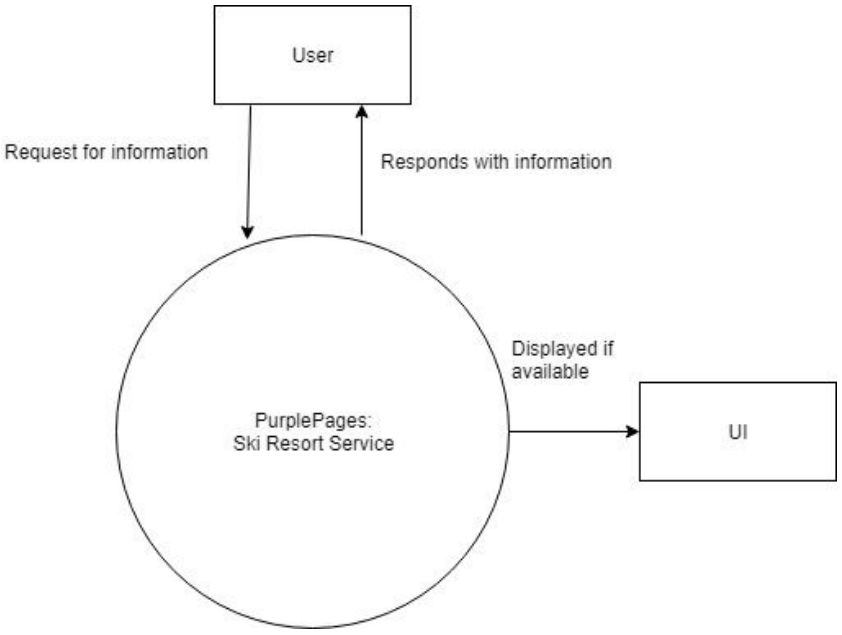
2.1 Elements and their properties

The end user represents the individual initiating the sequence to achieve the end result.
The browser is the web portal through which the end user access the application and sends/receives information.
The API Gateway serves to manage requests from the user and their related responses. It provides separation to facilitate the microservice structure.
The authentication element authenticates the user as approved to use the system in the event a request is made without authorization.
The DispatchServlet manages the requests and their associated responses between the gateway and specific service controllers.
The SkiResortController then processes the query transmitted through the URL so that it can be associated with an instance of the SkiResortSvc.
The SkiResortSvc then sends the appropriate method to the SkiResortDao class, which ultimately retrieves data from the DB2 t_skiresort.

2.2 Relations and their properties

The relations within the sequence diagram fall under the categories of time or instance. Vertically, the diagram depicts the progression of execution to achieve the goal of the sequence. Horizontally, it depicts the flow of information between the instances.

Section 3: Context Diagram



Section 4: Variability Guide

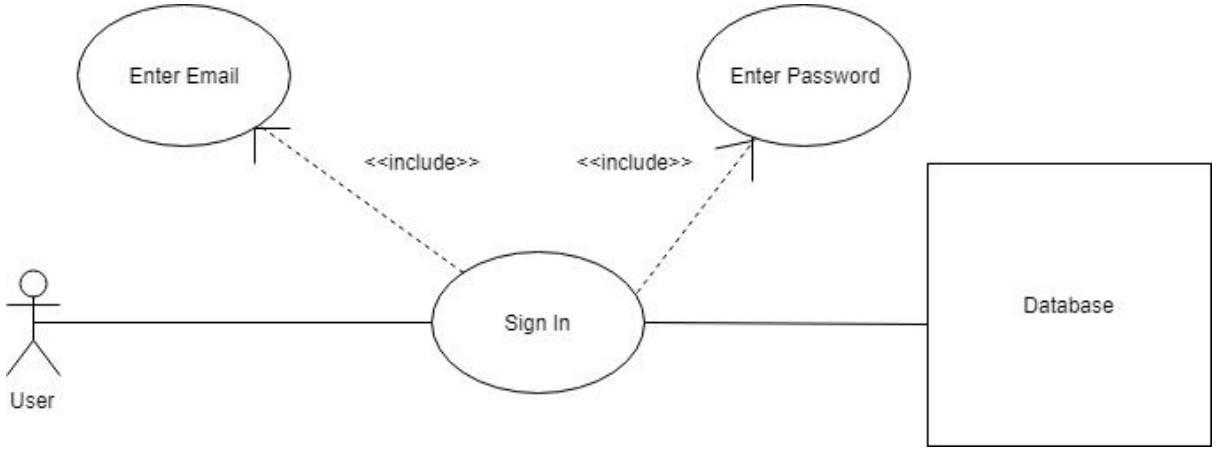
There are no variation points for this diagram.

Section 5: Rationale

This diagram represents the use of an API to retrieve data from a database. This sequence uses a RESTful API design as it is essential to have a separate client and server system to allow for microservices to be added, edited, and removed on the fly.

Quality View: Login Use Case Diagram

Section 1: The Primary Presentation



Section 2: The Element Catalog

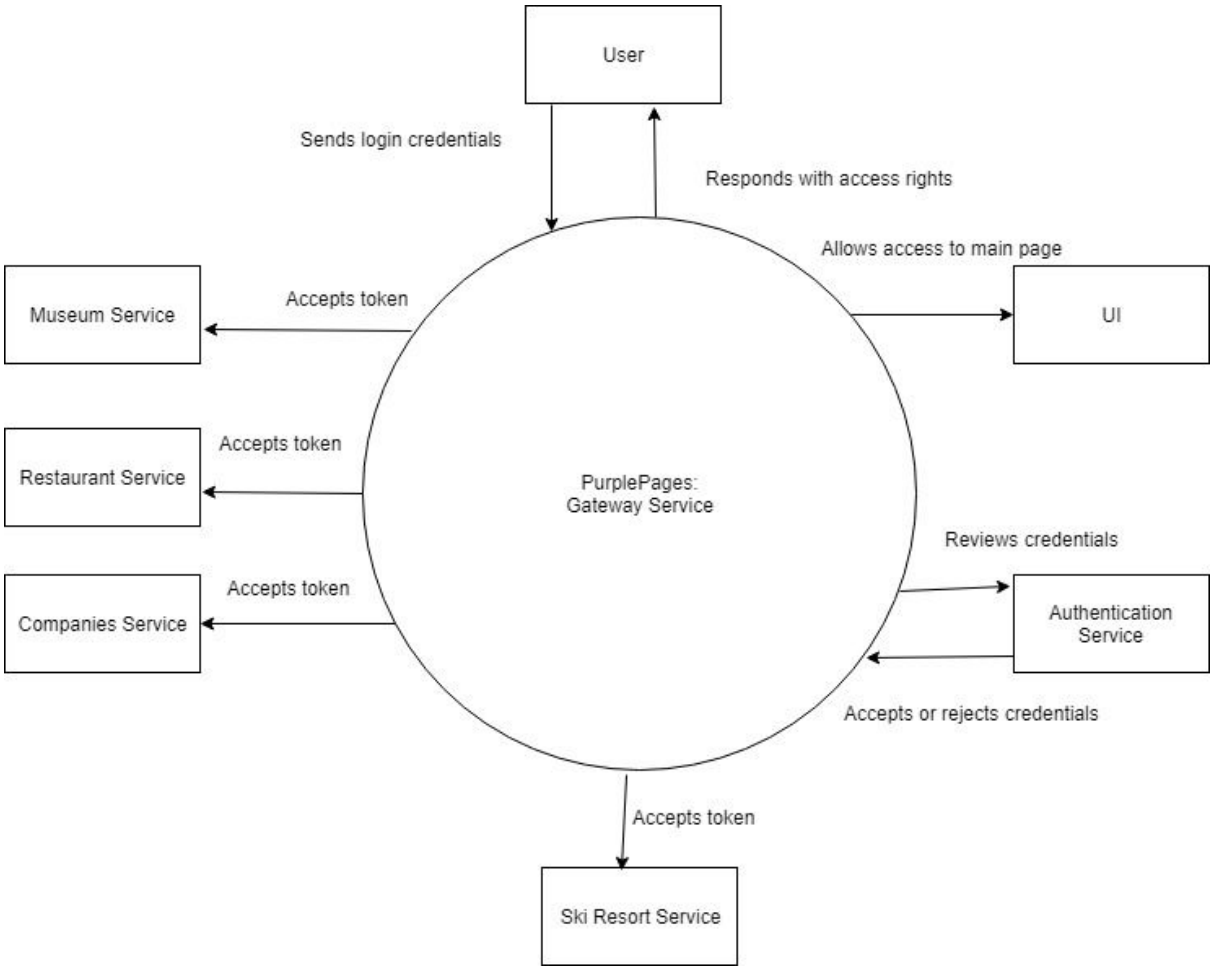
2.1 Elements and their properties

The stick-figure element represents an actor, from which the use case is initiated. Each oval represents a use case, which is a specific situation for which the product can be used. Lastly, there is the database element, which participates in the action by providing data back.

2.2 Relations and their properties

Solid lines represent associations, by which the elements are joined. The segmented lines represent the inclusion of separate use cases, within others.

Section 3: Context Diagram



Section 4: Variability Guide

There are no variation points for this diagram.

Section 5: Rationale

This diagram demonstrates a standard login procedure. Two forms of data are required to validate a user, in this case, an email to identify the user and a password to confirm the identity. This information is then validated server side to prevent unapproved access.

Directory -- Index, Glossary, Acronym List

Glossary and Acronym List

- **API**: Application Programming Interface, a set of functions allowing for the access of features of a system
-
- **C&C View:** Component and connector view
- **DAO**: Data Access Object, an object that provides an abstract interface to some type of database, providing specific data operations without exposing the details of the database
- **DB**: Database
- **DB2**: Relational Database Management System from IBM
- **HTTP**: protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands
- **JDBC:** Java Database Connectivity is an API for the programming language Java, which defines how a client may access a database.
- **JWT:** JSON Web Tokens, used for securing our application through JSON-based tokens
- **Middleware:** A bridge between gateway and authentication service
- **MVC**: Model-View-Controller, design pattern for the separation of data and views
- **REST:** Representational State Transfer, a software architectural style that defines a set of constraints to be used for creating Web services
- **SOA:** Service-Oriented Architecture, involves the deployment of services, which are units of logic that run in a network
- **UI:** User Interface, the medium which the user interacts with the application
- **URL:** Uniform Resource Locator, colloquially termed a web address, is a reference to a web resource