Web Service Registry

A Major Qualifying Project Report

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted on
Monday, January 12, 2015

Project Sponsor: **JPMorgan Chase & Company**

Submitted to:

On-Site Liaisons: **Martin Brownlie,**
Michael Wright,

Project Advisors: **Prof. Jon Abraham**, Department of Mathematical Sciences, WPI
Prof. Arthur Gerstenfeld, Department of Management, WPI
Prof. Micha Hofri, Department of Computer Science, WPI
Prof. Xinmin Huang, Department of Electrical and Computer Engineering, WPI
Prof. Kevin Sweeney, Foisie School of Business, WPI

Submitted by:

John Bosworth, Computer Science

Ibraeim Bukhamsin, Industrial Engineering

Heather Jones, Actuarial Mathematics

Arin Kulvanit, Electrical and Computer Engineering

Jordan Wetzel, Computer Science

Abstract

To assist JPMorgan employees in searching for a company created service application, our team created an internal service registry from scratch to organize existing services in one central location. Our team developed a user-friendly front-end web-based application that displays current services as well as allows employees to add new services. We also focused on designing a back-end for interfacing with our newly designed database that stores all information about current services.

Acknowledgement

We would like to thank Martin Brownlie and Michael Wright at JPMorgan Glasgow for their advice and support throughout the completion of the project. We would also like to thank Professors Jon Abraham, Arthur Gerstenfeld, Micha Hofri, Xinming Huang and Kevin Sweeney for comments and suggestions offered during each week's updates while we developed and completed this project. We are thankful to all members of the Core Component Architecture (CCA) team under the Assets Management group who helped us in the process of developing the Web Service Registry. Finally, we would like to thank JPMorgan for the resources they have provided us to make this project a success.

Exec Summary

JPMorgan's Asset Management line of business offers a variety of industry leading financial services and manages roughly $1.7 trillion USD. To efficiently run their business, Asset Management uses Service Oriented Architecture (SOA); this means it depends on roughly two hundred internal applications (services) that support many day-to-day business operations. Asset Management has multiple development teams that create new services based on business needs.

Currently, there is no organization of these internally created services. The only way for an employee to know about a particular service is for the service to have been created specifically for them or to hear about it from a colleague. When a business unit desires a new service, there is no way for them to check if a similar application already exists. Within Asset Management many business functions overlap and there is potential for services to be used across multiple business units.

This project aimed to assist JPMorgan in organizing and cataloging their existing services as well as creating a platform to add information about new services as they are developed. The goal of the project was to build a front-end for our service registry that displays current services as well as a back-end to store all information about services and their development teams. Our service registry was designed to help organize the SOA and encourage more services to be developed.

To develop our service registry we focused on three main sections. First, we built a front-end user interface using AngularJS, which is the standard web development language defined by JPMorgan. Second, we designed a server in Java to effectively communicate between the front-end and the database. Third, we constructed a database using SQL and Oracle to store all user entered data about services and development teams. These three levels were integrated to form a fundamental service registry.

The registry allows users to search through existing services for a particular business need. The service registry interface was designed to be simple and intuitive to use. It allows an employee to create and manage a service by first adding their team to the registry and then attaching a service to their team. Connecting a service to its individual development team was important in order for a service user to be able to obtain it from the team after finding it on the service registry. In addition, if there are any issues with the service then a service user will know which team to contact. Services can have multiple versions that can be cataloged on the registry as they are developed. One service can also depend on another service; this relationship can be entered on the registry and viewed by all users. Services and teams can be interactively managed through adding, editing, and deleting.

The service registry we created will greatly impact the Asset Management division of JPMorgan. Our registry will be moved to the production phase of development one month after we depart. The production phase will deploy the website and allow all employees to have access and the ability to start adding their services. Our service registry assists employees by enabling them to find a desired service from one central location so that they do not have to ask multiple teams if it has been developed. The service registry facilitates a more efficient application development process because the services are easy to find and can be more widely used. The service registry will assist employees in finding new services that can be applied to their business unit and overall improve the workflow by spreading SOA.

Authorship

The writers are listed to the right of the corresponding sections of this report. We edited each section of the paper together.

Cover Page	Arin Kulvanit
Exec Summary	Heather Jones
Abstract	Heather Jones
Acknowledgements	Arin Kulvanit
1.0 Intro	Heather Jones
2.1 JPMorgan Chase & Co	Heather Jones
2.2 JPMorgan and Services	Jordan Wetzel
2.3 Service Registry	Arin Kulvanit
2.4 Technologies Review	John Bosworth
3.0 Methodology Intro	Heather Jones
3.1 User Focus	Arin Kulvanit
3.2 Features	Ibraeim Bukhamsin
3.3 Coding practices	Heather Jones
3.4 User Interface	John Bosworth
3.5 Java API	Jordan Wetzel
3.6 Database Design and Implementation	Ibraeim Bukhamsin
3.7 Administrative Reports	Heather Jones
3.8 Summary	Arin Kulvanit
4.0 Results Introduction	Heather Jones
4.1 User Interface	John Bosworth
4.2 Java Server	Jordan Wetzel
4.3 Database	Ibraeim Bukhamsin
5.0 Recommendations Introduction	Heather Jones
5.1 Administrative Reports	Heather Jones
5.2 Registering an External Service	Ibraeim Bukhamsin
5.3 Notifications Tool	Heather Jones
5.4 Education and Expansion	Ibraeim Bukhamsin
6.0 Conclusion	Heather Jones
Appendix A	Heather Jones
Appendix B	Arin Kulvanit
Appendix C-H	All

Table of Contents

Table of Figures

1.0 Introduction

JPMorgan is a leading financial services firm and banking institution in the United States and internationally. JPMorgan develops many internal services to help them easily query large databases for only specific information and to support business operations, for example retrieving necessary country codes and current exchange rates from a database of worldwide information. Services provide an added layer of security for the database because individuals do not access the database directly and thus cannot unintentionally change the database or see information they do not have access to. The Asset Management division within JPMorgan has many development teams that have created services for different uses.

The Private Bank sector within Asset Management was looking for a way to catalog all of their available internal services. Employees had no means to find a service or even know of its existence unless they were part of the team that asked for its development. JPMorgan saw the opportunity to raise awareness about available services in order to improve reusability and prevent similar services from being created by multiple teams. A third-party vendor was found that offered a service registry system to catalog the Private Bank's existing services; however, this system lacked features JPMorgan was looking for and it was extremely costly. The goal of our project was to develop a brand new internal online service registry to improve the employee awareness of available services, centralize information about them, and increase access to them.

We developed a user-friendly web-based application that displays current services as well as allows employees to add a new service they have created. We designed a back-end for interfacing with our newly designed database that stores all information about current services. We built the service registry to have a web-based user interface that communicates to a server that connects to a database as seen in figure 1. This three-part system allows the application to take in information about a service on the front end and send it to the database to be stored. Also, this system allows the front end to effectively call the backend through the server and retrieve any needed information to display.

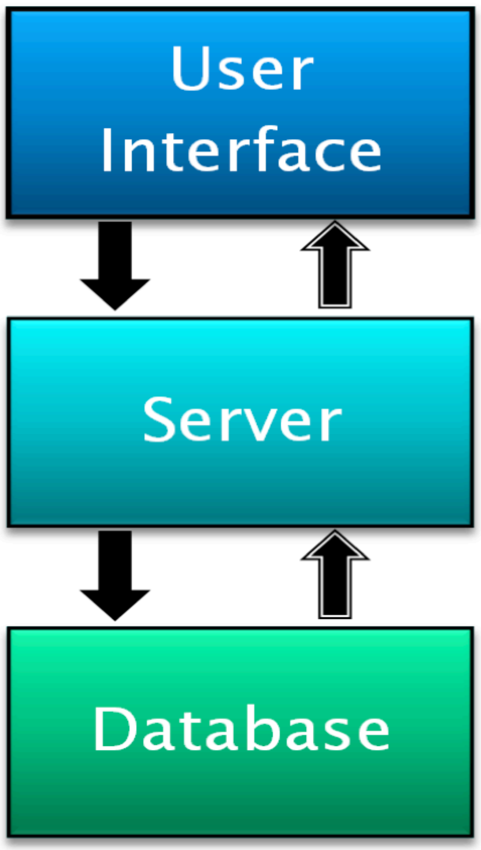


Figure 1: Service registry design

The registry allows users to search through existing services for a particular business need. The service registry interface was designed to be simple and intuitive to use. It allows an employee to create and manage a service by first adding their team to the registry and then attaching a service to their team. Connecting a service to its individual development team was important in order for a service user to be able to obtain it from the team after finding it on the service registry. In addition, if there are any issues with the service then a service user will know which team to contact. Services can have multiple versions that can be cataloged on the registry as they are developed. One service can also depend on another service; this relationship can be entered on the registry and viewed by all users. Services and teams can be interactively managed through adding, editing, and deleting.

The service registry we created will greatly impact the Asset Management division of JPMorgan. Our registry will be moved to the production phase of development one month after we depart. The production phase will deploy the website and allow all employees to have access and the ability to start adding their services. Our service registry assists employees by enabling them to find a desired service from one central location so that they do not have to ask multiple teams if it has been developed. The service registry facilitates a more efficient application development process because the services are easy to find and can be more widely used.

2.0 Background

2.1 JPMorgan Chase & Co.

JPMorgan Chase & Co. is a leading financial services firm and one of the largest banking institutions in the United States of America and worldwide. JPMorgan is included in the Dow Jones Industrial Average and holds $2.4 trillion in assets along with $211.2 billion in stockholders' equity. Fortune named JPMorgan their number one "Most Admired Company" in the world among Megabanks in 2012, 2013, and 2014. JPMorgan could not succeed as well as it does today without its solid history of financial strength and leadership (JPMorgan Chase & Co., 2013).

JPMorgan is built on over two hundred years of strong banking tradition with roots tracing back to the Manhattan Company chartered in 1799 in the U.S. JPMorgan has endured and shown leadership during times of economic growth and financial uncertainty. During the nineteenth century, JPMorgan developed relationship banking, where customer loyalty is the key to success. JP Morgan Chase & Co. is the result of many strategic mergers of financial companies allowing it to survive in times of instability and grow to be the international company we know today (Company History, 2014).

JPMorgan is a current leader in investment banking, financial services for consumers and small businesses, commercial banking, asset management, and private equity. Internationally, JPMorgan has grown with roots in Europe tracing back to 1838 and in Asia since 1872. JPMorgan is an industry leader in financial services as well as cutting edge technology; their European Technology Center in Glasgow, Scotland recently celebrated its 15 year anniversary. JPMorgan's rich history has provided the firm with a strong foundation to lead the financial services and banking industry today and maintain a strong brand name (JPMorgan Chase & Co., 2013).

2.2 JPMorgan and Services

Services are created primarily for the purpose of obtaining information from a remote storage location such as a database. Some databases are accessible directly from a client via the use of SQL statements or other methods; however, services provide a layer of abstraction and protection between the client and storage location. The client can make a call to a particular service, such as an ISBN lookup for a library, and that service will take care of the retrieval of the specified information from the storage location and return the result back to the client. This added level of abstraction is useful for several reasons, mainly security and ease of use. On the security side, services prevent the client from making direct interactions with the database by using predetermined calls set by the creators of the service. Services also prevent users from accessing or modifying unauthorized information by limiting the calls to ones that return a specific grouping of data. In addition, services allow for much more user-friendly interactions with a database than if a client had to provide the calls necessary to obtain the information directly.

In JPMorgan, services are primarily used to query large data towers which house internal information such as investment strategies for various products, business areas involved in these strategies, company reference data, and currency codes. This data is useful to different lines of business within the company, specifically internal application designers who need to include this information in their own products. For example, developers at JPMorgan could query a service to obtain data that would then be displayed in a drop-down menu for the user select.

2.3 Service Registry

Service registries are developed as a way to catalogue existing services to allow users to locate these services; without the service registry, finding services is not always easy. As part of

JPMorgan Assets Management, the Core Component Architecture (CCA) team seeks to build their own service registry to organize their internal services. A service registry creates a central location for which services can be accessed and relocates all useful information into a single application (Treiber, 2007). If one is looking for a type of service, one can use a service registry as a search engine by entering keywords. Each entry can contain connectivity information; in this case a service will be connected to its development team (Bou-Ghannam, 2007). At first JPMorgan wanted to buy a service registry from IBM; however, JPMorgan decided not to due to high expenses and lack of flexibility regarding modifications. JPMorgan decided to create its own web service registry that would improve delivery time, increase productivity, and allow for future expansion.

The creation of an in house service registry began with the proof of concept developed by former summer interns at JPMorgan. The proof of concept lacked useful desired features such as team view, team members, and service version. The interns' design failed to incorporate desired technologies to develop the service registry, thus requiring an entirely new registry to be created. A service registry is an important part of improving JPMorgan's service-oriented architecture.

2.4 Technologies Review

Numerous technologies were used throughout the development and completion of the project. A brief literature review of each tool and language is provided below.

2.4.1 Languages

Java

Java is an Object-Oriented programming language designed around the ideal "write once, run anywhere." In this project, Java is used as the server language connecting the front-end and the database. Java's Object-Oriented ideals allow for access to the database with less risk of exposing details of the database to malicious users. Java's versatility and compression of complicated tasks, such as garbage collecting, has made it one of the most popular languages used in software development today.

JavaScript

JavaScript is a prototype-based scripting programming language with support for Object-Oriented, imperative, and functional programming styles. JavaScript's primary use is for dynamic interactions with websites. With the popularization of JavaScript, websites have become faster and more interactive because they don't have to load new pages in order to display minor changes. JavaScript can make requests to servers and dynamically display the server response. JavaScript, beyond name, has no relation to Java, following very different paradigms, syntaxes, and semantics.

SQL

Structured Query Language (SQL) is a special-purpose programming language designed for maintaining data held in database systems. SQL handles all interactions between the server and the database including but not limited to: a search query around specific criteria; inserting new information into a database; or deleting unwanted information from a database. There are many variations of SQL, such as MySQL and T-SQL, but they all follow similar syntaxes.

HTML

Hypertext Markup Language (HTML) is the standard markup language for the creation of webpages, but is not considered a programming language. HTML arranges webpages using nested tags that hold many attributes. A web browser is able to parse the HTML tags and display the results according to a semi-standardized list of rules. Every browser treats HTML slightly differently especially with the onset of HTML5. The lack of standardization between web browsers can cause display issues and make a fully functional webpage in any web browser difficult to create.

CSS

Cascading Style Sheets (CSS) is a style sheet language used for the stylization of HTML pages, but is also not considered a programming language. CSS is a list of instructions for browsers to follow when displaying an HTML page and provides the majority of the look and formatting of a web page. CSS can be used to help mitigate any cross-browser display issues from pure HTML5. However, there is also a lack of standardization between web browsers on the reading of CSS, forcing developers to often create different CSS files for different web-browsers.

2.4.2 Tools

Eclipse

The Eclipse Integrated Development Environment (IDE) acts as an editor, debugger, compiler, and interpreter for computer programmers, best known for working with the Java language which comes pre-installed with the Eclipse software development kit (SDK). Eclipse is a free and open source program used to develop applications including web servers. Eclipse allows programmers to work in a single environment with all the tools they need; any tools not pre-installed can be downloaded as plug-ins. Eclipse is also able to integrate itself with other tools, such as the Tomcat web server, and the Maven build tool which includes the Junit testing and code coverage suite (Are you ready for Java, 2014).

Maven

Maven is a software project management and build automation tool for Java projects maintained by the Apache Software Foundation. Maven tracks both how the software is built and any dependencies that the software may require. The automation allows for consistency across computers and simplifies the process by collecting dependent libraries and building the project. Maven also automates testing, reporting, and documentation across the project, allowing for developers to easily run all unit tests in a project and identify errors (Maven, 2014).

Sublime Text

Sublime Text is a text and source code editor, and syntax highlighter for many languages. Sublime Text is used in this project mainly for the editing of HTML, CSS, and Javascript files. Though possessing no ability to build and execute code, or report errors, Sublime Text is useful for increasing code readability by consistently highlighting alike syntaxes and maintaining proper whitespace (Sublime Text, 2014).

Squirrel SQL

Squirrel SQL is a graphical Java program that allows the user to view tables of the database and browse the data of these tables. Squirrel SQL allows the user to run SQL commands containing the structure of the database and view a graphical representation of all tables, relationships, and keys. In addition, Squirrel SQL supports visual database testing by allowing the user to observe the impact SQL statements have on tables (Squirrel, 2014).

DBMSs

A Database Management System (DBMS) is a computer program that enables the user to store, modify, and extract information from the database. A DBMS is where the final design of the database is implemented to allow interactions with the server. An example is Hyper SQL (HSQLDB), which is a small size database that runs in-memory to allow fast data transactions by the user. Another example is Oracle database, which is reliable for a large amount of data in a multiuser environment.

PowerDesigner

PowerDesigner is a collaborative enterprise-modeling tool for windows with a plugin for Eclipse. PowerDesigner allows users to design database architecture in an easy, visual manner. Once the data structure is created visually, it can be exported into the SQL database language and the design imported into the project database. When a developer makes changes to the database in PowerDesigner, instead of attempting to rebuild the entire database, PowerDesigner will create SQL update code for the database to prevent loss of data already stored (Powerdesigner, 2014).

2.4.3 Language Libraries

JUnit

JUnit is a unit-testing framework for the Java language. Unit testing is a software testing method, characterized as being a collection of tests of the smallest testable parts of an application while still testing expected functionality (JUnit, 2014).

AngularJS

AngularJS, short for Angular Javascript and often referred to as Angular, is an open-source web application framework created by Google. Angular is designed to help overcome many of the challenges in developing a dynamic web page. Angular uses the Model-View-Controller architecture to organize its code by first reading an HTML page with embedded Angular tags and attributes, and then interpreting the directives and binding JavaScript variables to HTML. As the JavaScript variables change, the display of the HTML changes as well (AngularJS, 2014).

Karma

Karma is an open-source unit testing JavaScript library, which extends AngularJS and allows a user to easily test the functionality of their JavaScript code. The unit tests aim to be easy to read and give useful responses in the event of a failed test. Karma allows a web developer to develop in a test-driven manner, a previously near-impossible task.

Bootstrap

Bootstrap is an open-source toolkit intended to help simplify and streamline the process of creating a website's layout and style. The toolkit is comprised of HTML and CSS templates for components such as forms, buttons, and columns. Many sites developed since Bootstrap's release in 2010 utilize the pre-created components, most notably being the social media site Twitter (Bootstrap, 2014).

Protractor

Protractor is a tool used for AngularJS end to end testing, which tests the interactions of different components of a system and ensures that they work together properly. The tests are written in Javascript and used by Protractor to simulate user actions in a web browser and determines if this created any errors in the application.

JSON

JavaScript Object Notation (JSON) is a text format that is used for transmitting data objects between a server and a website. JSON is human readable and smaller in size when comparing to corresponding text formats, such as XML. Also, files that have the extension of .JSON are noticeably faster when parsing to natural language or computer language (JSON, 2014).

MyBatis

MyBatis is a Java framework that links Java objects with stored SQL statements. MyBatis eliminates the manual writing of all the Java Database Connections (JDBC) that handles the setting of parameters. MyBatis utilizes simple XML or Annotations for configuration and map primitives.

Spring Framework

The Spring Framework is an open source application framework for Java. Spring's main features can be called by any Java application. This framework has extensions for building web applications based on the Java EE platform. The web extension uses a model-view-controller framework to provide code annotations for development and customization of web applications and RESTful web services.

Jersey

Jersey is an open source RESTful web services framework. This production quality framework supports JAX-RS APIs. Jersey provides its own API that has additional features and utilities to further simplify RESTful service and client development (Jersey, 2014).

Moneta

Moneta is an internal JPMorgan collection of libraries that utilizes a number of industry leading libraries combined with additional code to provide many additional features. Moneta assists with authentication and validation in Java. Moneta also provides flexible logging across deployment environments along with integration throughout to assist in data handling.

3.0 Methodology

This project resulted in the delivery of an internal service registry to assist JPMorgan employees in the search for a company created service application. The Assets Management division at JPMorgan originally looked into purchasing a service registry from an outside vendor but that proved to be too costly and did not provide all the functions they wanted. Figure 2 shows the service registry's progression from an idea to reality. The idea of creating a service registry was then passed to the Core Components and Architecture (CCA) team to develop. The CCA team's interns developed a proof of concept that was used to prove feasibility of building the registry. Next our project focused on building the entire service registry as highlighted in figure 2. Our service registry was moved to the production phase after we departed and it is scheduled to launch in January, 2015.

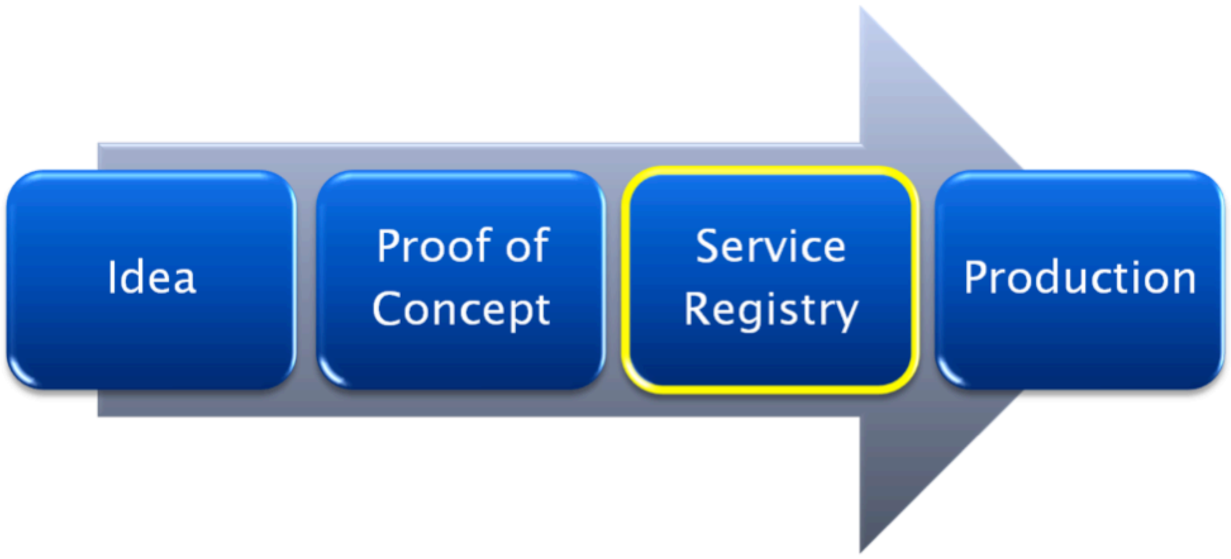


Figure 2: Service registry development

The service registry we created took into account the needs of all possible users to develop its functionality. In six weeks, our team developed a user-friendly front-end web-based application that displayed current services and their development teams as well as allowing employees to upload a new service they have created. We designed a back-end for interfacing with our database that stored all information about current services and teams. The service registry gathers all company created services in one central location to make it easy for users to find the service they are looking for.

3.1 User Focus

This project based its objective on the satisfaction of user stories. User stories are statements used in everyday or business language that illustrates what users need to do as part of their job. For example, if a service owner wants to build a team for services management then its user story would be, “As a service owner, I would like to create my team to allow me to manage my services”. Our sponsor provided us with a base set of user stories to build our project from. These user stories focused on the needs of the expected users of our service registry. The more the project satisfied the end user, the more the service registry was going to be easily recognized. The end-user would prefer an interface that is easy to learn and use.

3.2 Features

Features of the service registry were obtained from user stories. We simplified these features in order to understand all the steps required for implementation. Access to features of the service registry depends on the user type, which is distinguished by an authentication process.

3.2.1 Main Features

The main features that JPMorgan desires in a service registry are: [1] searching and browsing services; [2] creating a new team; [3] editing or deleting an existing team; [4] adding a new service; [5]

editing an existing service and adding a new service version; [6] assigning a dependency for a service version; [7] deleting a service; and [8] deleting a version.

Searching and browsing services are basic features of the service registry. All users can search for a service and read its description without any authentication process.

A team is defined as a group of employees, comprised of two or more owners and any number of members, who are providing a service. A team can upload one or more services to the service registry allowing it to be searchable by other teams. Each team only has control over the services it has created, which prohibits modifications of other teams' services. The process of adding a team to the service registry is illustrated in figure 3.

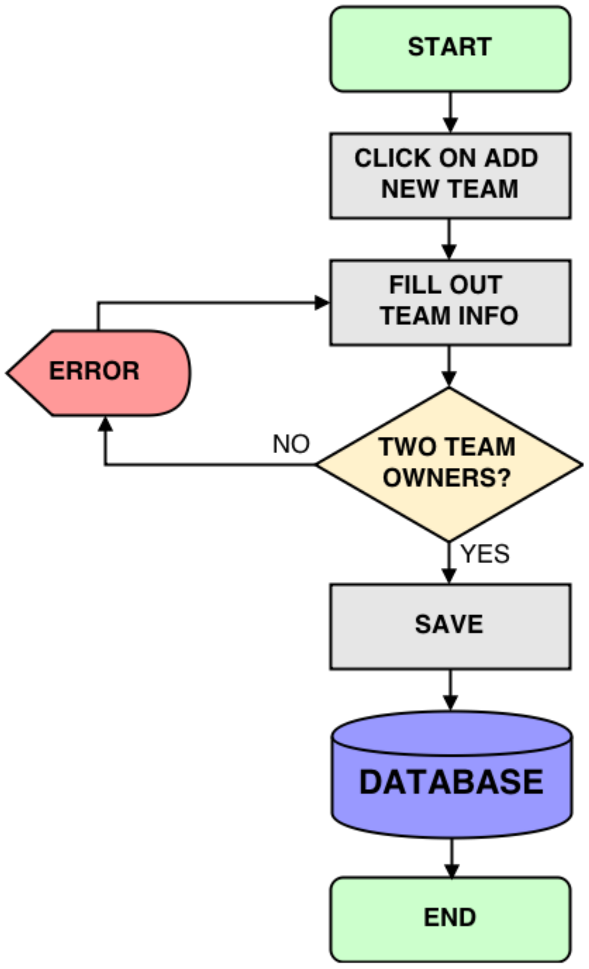


Figure 3: Add team flowchart

Editing an existing team is a feature that is restricted to the team owner only. This is the only condition for editing a team. On the other hand, deleting a team is a more complex process that is illustrated by figure 4.

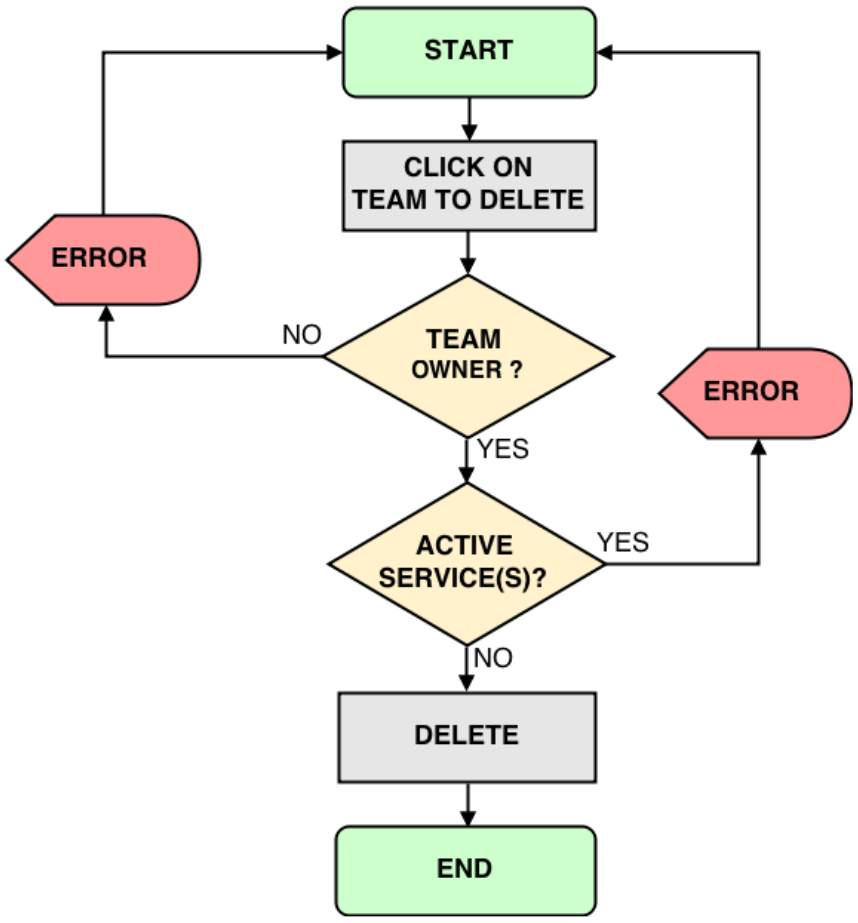


Figure 4: Delete a team flowchart

Adding a new service to the service registry is an essential feature and is shown by figure 5.

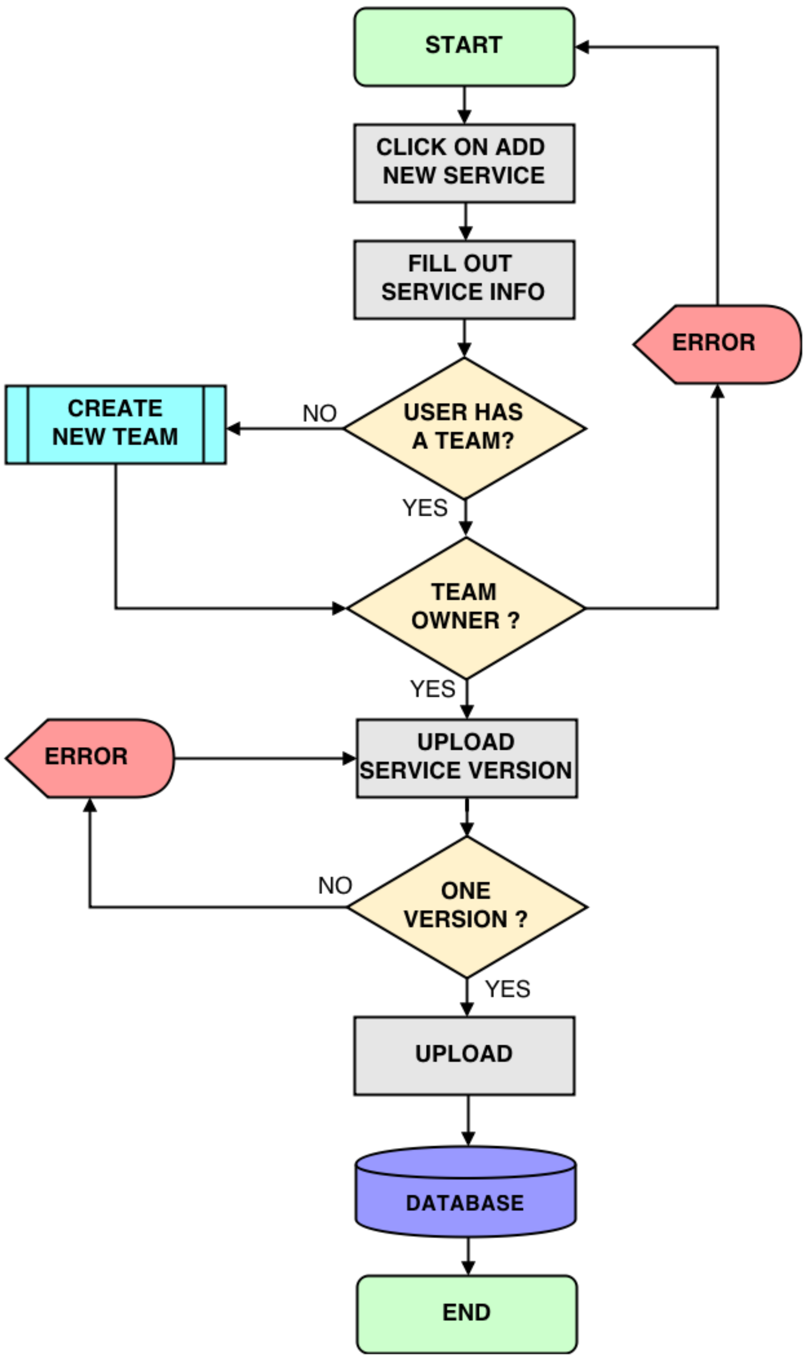


Figure 5: Add a service to the registry flowchart

Editing an existing service and adding new service versions are restricted features to the team owner; however, a service version might depend on the existence of other versions. Dependent versions might belong to different services or to the same service. The dependent can be viewed as a child that is linked to a parent version, which is called the dependency. The process of linking two service versions is called assigning dependencies and it is exclusive to the team owner. This process allows subscribers of a service to receive notifications about updates such as deletion, changed status, and availability of new versions.

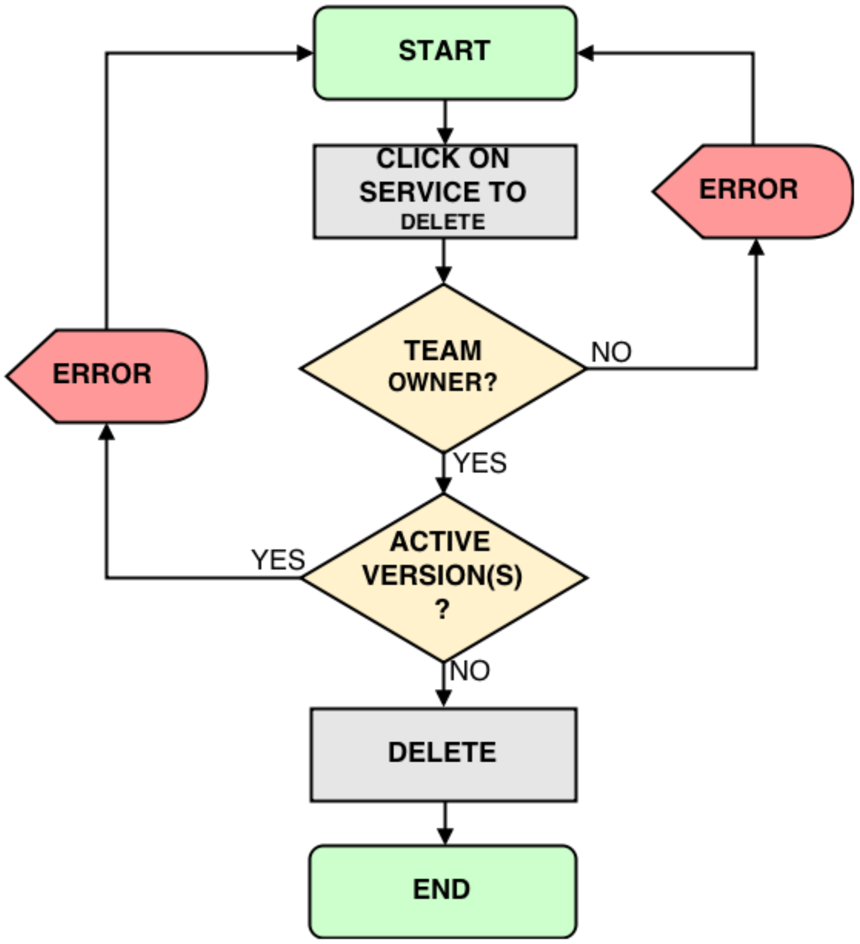


Figure 6: Delete an existing service flowchart

As seen in figure 6, an existing service can be deleted if and only if all of its versions have been deleted as well. Understanding the status of a service version is important to the process of deleting a version. As displayed in figure 7, the status of a version is written in the description so all users are informed about its current condition. Once all dependencies of a service are removed, the version can then be deleted, as shown in figure 8.

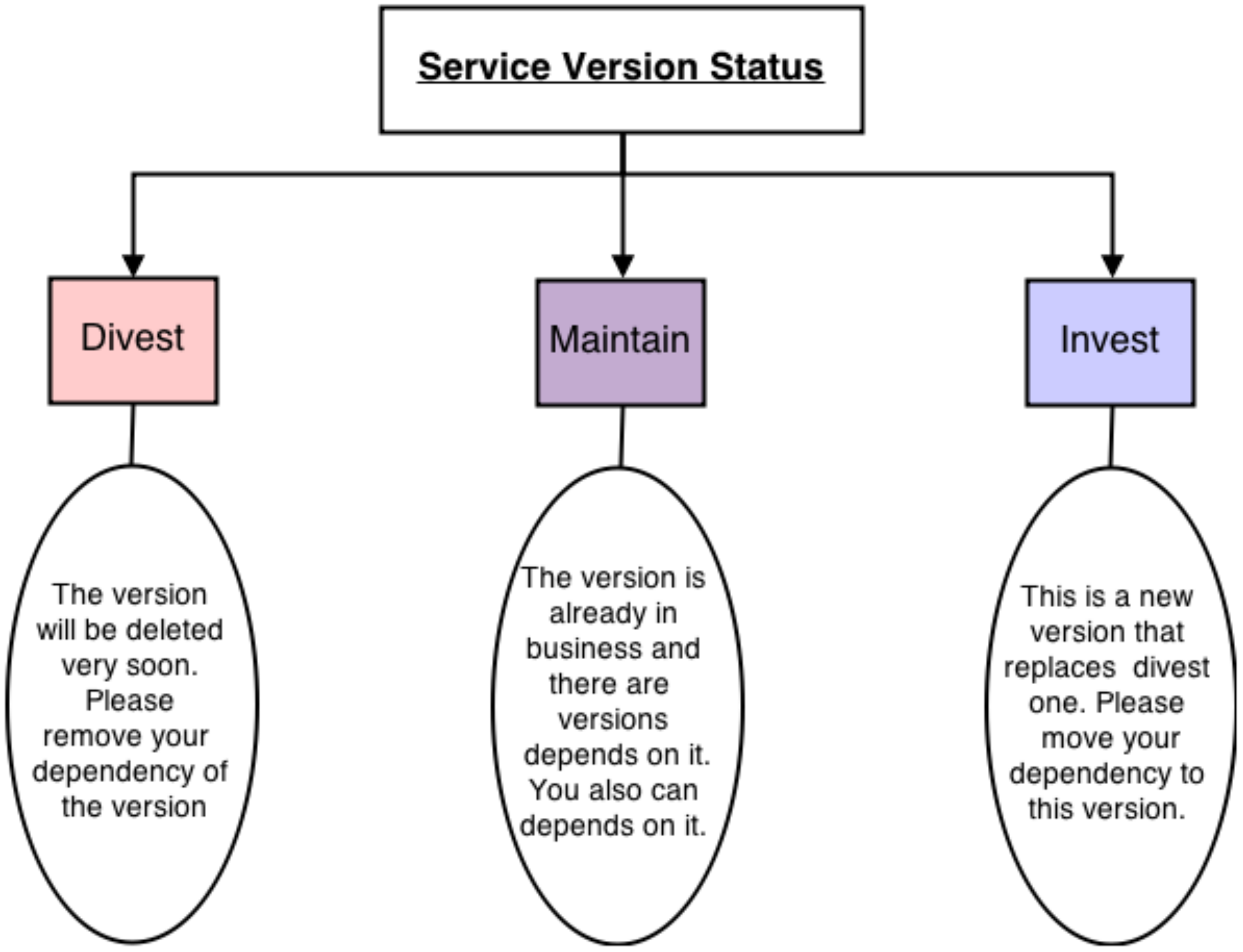

Service Version Status
Divest
Maintain
Invest
The version will be deleted very soon. Please remove your dependency of the version
The version is already in business and there are versions depends on it. You also can depends on it.
This is a new version that replaces divest one. Please move your dependency to this version.

Figure 7: Service version statuses

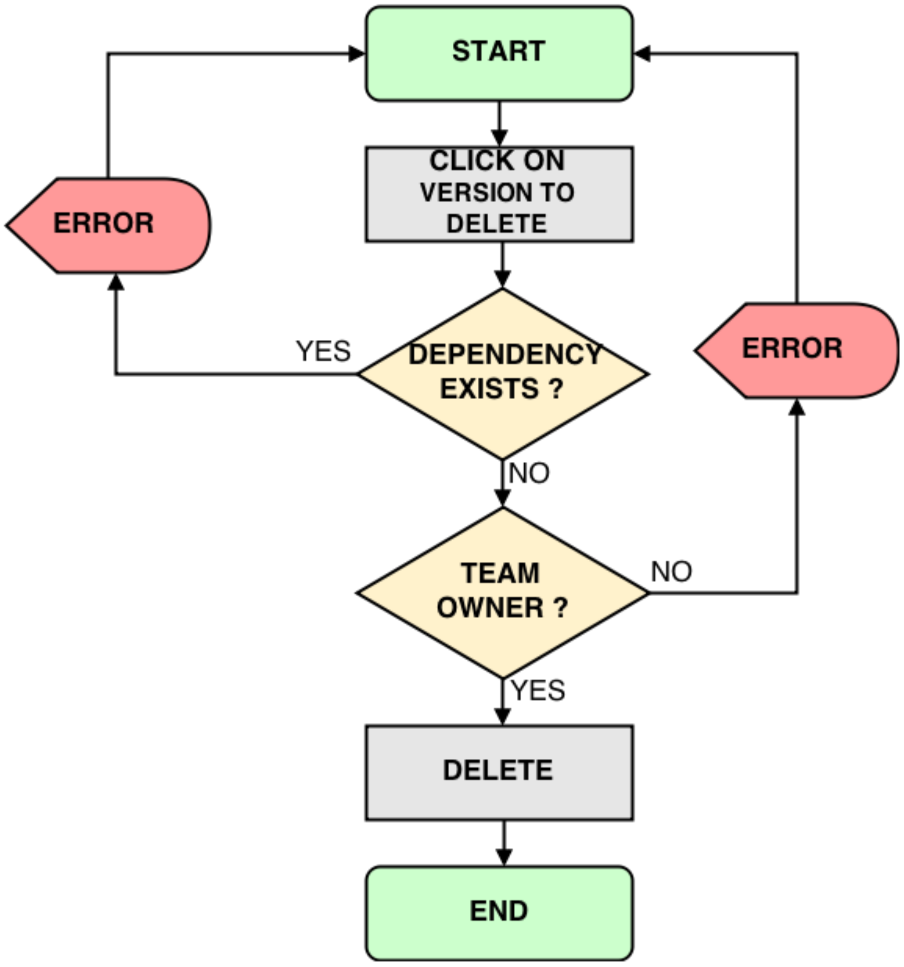


Figure 8: Delete an existing version flowchart

3.2.2 Features by user type

Features of the service registry are distinctive based on the user type. There are four user types: [1] business analyst; [2] team owner; [3] team member; and [4] administrator. Users will have to be authenticated in order to access specific views of the website containing the designated features of their user type. Each user type and their associated features are displayed in figure 9.

A business analyst is any JPMorgan employee who is not authenticated as an administrator, team member, or team owner. In addition, a business analyst can be any user who does not log in to pass the authentication process. This type of user is limited to searching and browsing services.

A team owner is a leader of the team who has access to extra features such as creating and deleting services; whereas, a team member has more limited access to the team's features.

Finally, an administrator is the person who is responsible for controlling, monitoring, and maintaining the service registry. Administrators have access to all features due to their critical role.

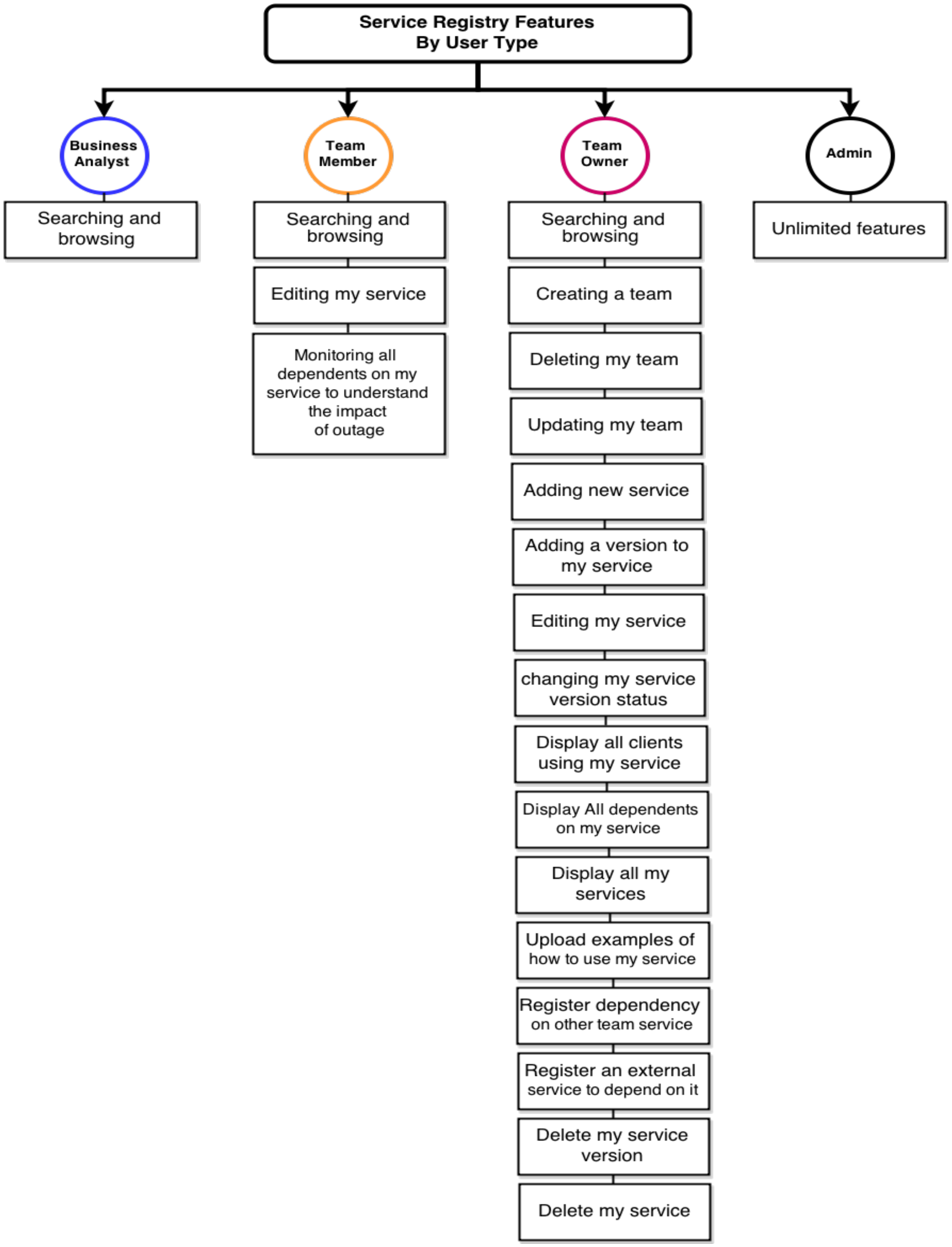

Service Registry Features By User Type
Business Analyst
Team Member
Team Owner
Admin
Searching and browsing
Searching and browsing
Editing my service
Monitoring all dependents on my service to understand the impact of outage
Searching and browsing
Creating a team
Deleting my team
Updating my team
Adding new service
Adding a version to my service
Editing my service
changing my service version status
Display all clients using my service
Display All dependents on my service
Display all my services
Upload examples of how to use my service
Register dependency on other team service
Register an external service to depend on it
Delete my service version
Delete my service
Unlimited features

Figure 9: Features by user type

3.3 Coding practices

3.3.1 Test Driven Development

For our project we followed the Test Driven Development (TDD) style of programming. TDD allowed our group to efficiently write code by providing constant feedback on our code's functionality. TDD is the process of writing simple failing tests before writing the functional code to complete these tests. The next step is to write just enough functional code to pass the new tests. Once the tests pass the next step with TDD is to restart the cycle and write another simple failing test. Figure 10 shows the TDD process from start to finish. Development cycles until all functionality requirements are completed.

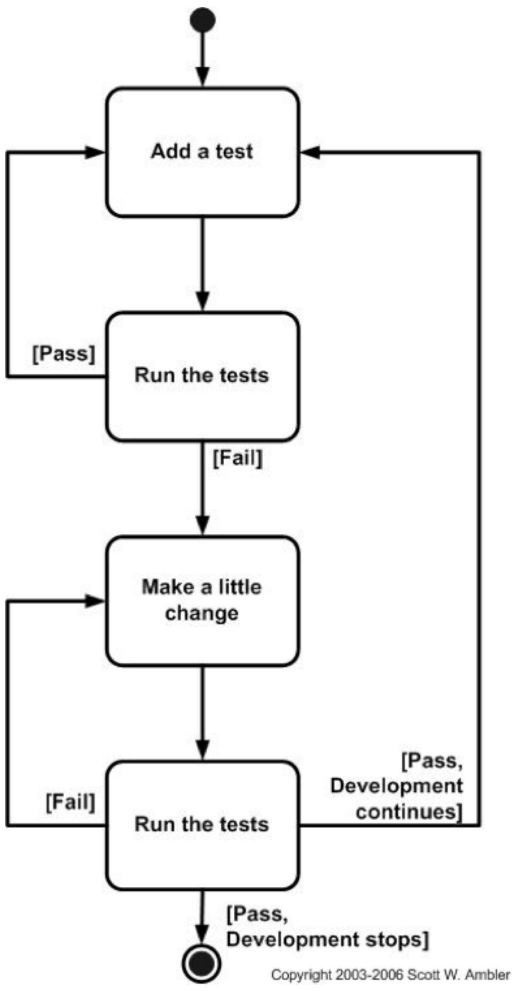

Add a test
[Pass]
Run the tests
[Fail]
Make a little change
[Fail]
Run the tests
[Pass, Development continues]
[Pass, Development stops]
Copyright 2003-2006 Scott W. Ambler

Figure 10: Test driven development

The main goal of TDD is to require a programmer to write code using small steps and prevent excess code. Traditional programming is done by writing code first, then writing tests that in theory should pass. When using traditional methods, debugging a failed test can be very time consuming due to difficulty finding the problematic code. TDD allows the code design to adapt to the programmers changing understanding of the problem more easily than traditional development. Test Driven Development allowed our group to stay focused on small tasks at a time and progress in an iterative manner. We used TDD over the course of our entire project as long as the situation allowed.

3.3.2 Pair Programming

Our group used pair programming to write our code, which involved two of our group members sharing one computer and writing code together. The programmer at the keyboard is called the “driver” and the other individual is the “navigator”. Pair programming allows both individuals to share knowledge, exchange ideas, and help each other stay focused on the current task. Pair programming greatly improved our design process and code quality by forcing discussion, and allowing questions to be raised, on the best practices to complete a task. Two sets of eyes were always looking at the code, which reduced the number of errors and improved our programming efficiency.

3.3.3 Scrum Planning

Our team used the Scrum framework to plan out and execute our project. The Scrum framework outlines the process to successfully complete a complex problem as shown in figure 11.

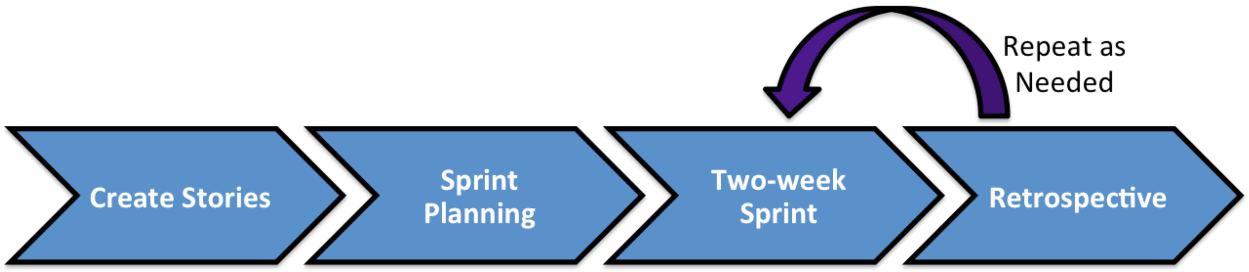


Figure 11: Scrum framework

The first step is to write up all user stories necessary to complete the project. Our sponsor created user stories to fulfill the ultimate design and functionality for the service registry based on user requirements. The Scrum framework groups the stories into sections called sprints which, for our project, lasted two weeks each. We assessed the stories by difficulty and assigned relative values to each story. In Scrum, each sprint should ideally have stories with comparative levels of difficulty. Also, the stories within each sprint are organized by dependencies on other stories to provide a clear path for completion. During each sprint there are daily meetings called stand-ups where each member of the team says what they accomplished yesterday and what they plan to do today; as well as, any current problems they are having in accomplishing their story. Figure 12 displays the layout of the scrum board and demonstrates the movement of tasks as they are completed. The daily scrum meeting allowed our team to discuss the status of the stories.

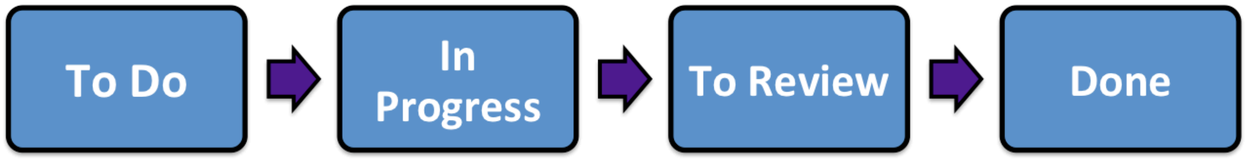


Figure 12: Scrum board layout

Before a story can be moved into the done category on the board it must meet the requirements that our team agreed upon. Figure 13 illustrates the steps that must be completed in order to meet our team's definition of done. During our sprint-planning meeting, we also discussed the steps in figure 13 and decided that integration tests are optional to meet the definition of done because they are not always feasible for a particular story.

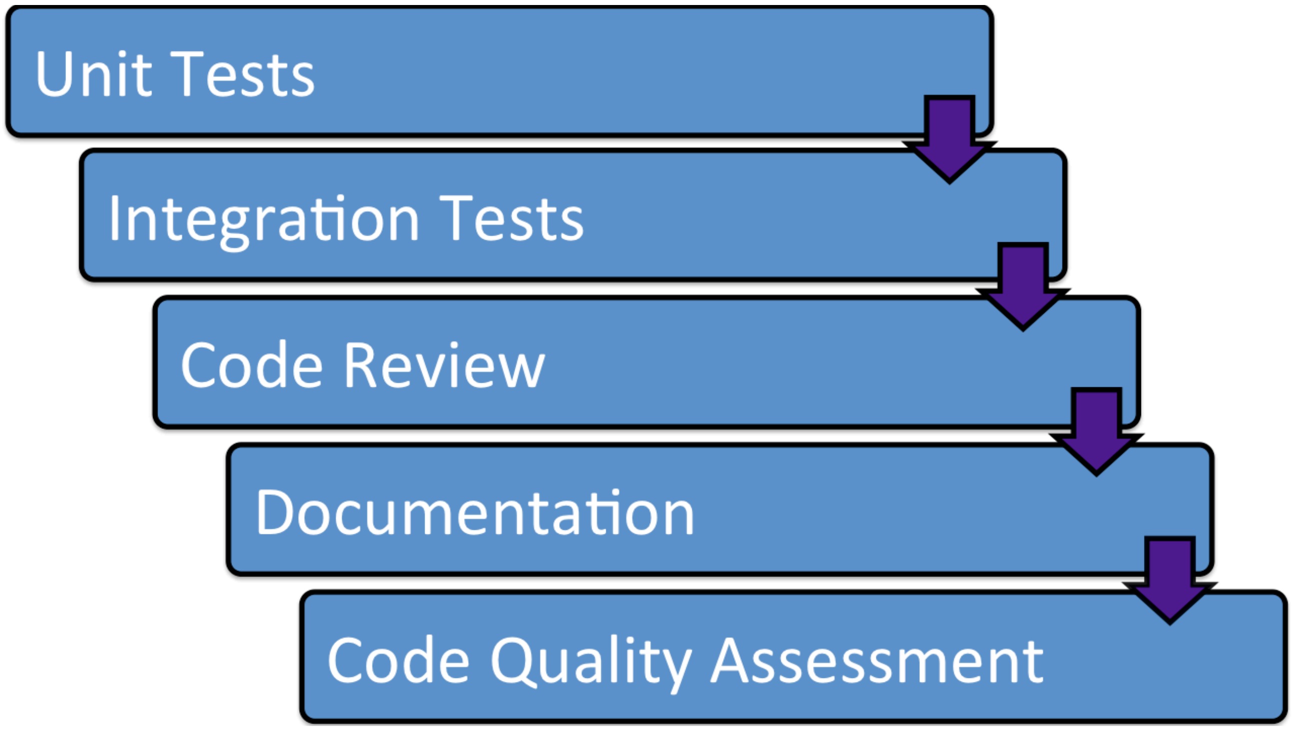


Figure 13: Definition of done

At the end of each two-week sprint our goal was to have a deliverable that can be presented in its entirety with all the stories completed. We aimed to present a demonstration (demo) to our sponsor at the end of each two-week sprint. Also, at the end of a sprint our team has a retrospective that reviews how the sprint went. For this discussion each team member wrote down actions they wanted the group to stop doing, do less of, keep doing, do more of, and start doing. Then we would discuss each category (Stop, Less, Keep, More, and Start) as a group and set goals about how we wanted to improve the next sprint.

Our project involved using the Scrum framework to create an internal service registry for JPMorgan employees to use when looking for a company created service application. Our team was responsible for building a web-based application that will display current services; as well as, allow employees to upload a new service they have created. Also, we focused on designing a back-end for interfacing with our newly designed database that stores all the information about current services.

We broke our project up into three sprints that were each two weeks long. For the first sprint, we focused on setting up the database schema in Powerdesigner and creating the framework for database interactions using Java. For the second sprint, we worked on setting up the user interface and connecting it with the back-end. For the third sprint, we concentrated on adding more functions to our service registry. The Scrum framework kept our team focused and gave us a clear plan of the order to complete tasks in.

3.4 User Interface

In order for our project to achieve its overarching goal of easily informing users of available services and allowing access to them, we provided an intuitive, web-based Graphical User Interface (GUI) to interact with our service registry. The challenge with designing a GUI is accounting for the time required to program it. If a GUI is being designed for a web-based business, then a functional and visually appealing GUI makes every detail of an artist's design required; however, our project is not being designed for a web-based business. Our project is focused more on being functional than being displayed in a fanciful manner. In fact, unlike most web development projects, our project lacks a designer entirely. Instead, we were tasked with creating a bare bone user interface that properly displayed functionality in a user-friendly manner that would not take much time to develop.

To achieve a functional GUI we used the built in constructs of Bootstrap, the HTML toolkit discussed in section 2.4.3. Following Bootstrap's default design patterns: curved edges; similar color schemes; and the layout of navigation bar, title area, main content area, and footer from top to bottom respectively, we were able to create a functional and user friendly website in minimal time.

However, a sleek and user friendly GUI is only half of the front-end. The other half deals with the interactions of the user and the web page. To deal with the user interactions we incorporated Bootstrap into an AngularJS framework. Angular follows the Model-View-Controller Computer Science paradigm as seen in figure 14; of which, Bootstrap handled the view, what is seen by the user. Angular and Javascript handled the model, where information is stored, and the controller, where information is manipulated and the view updated.

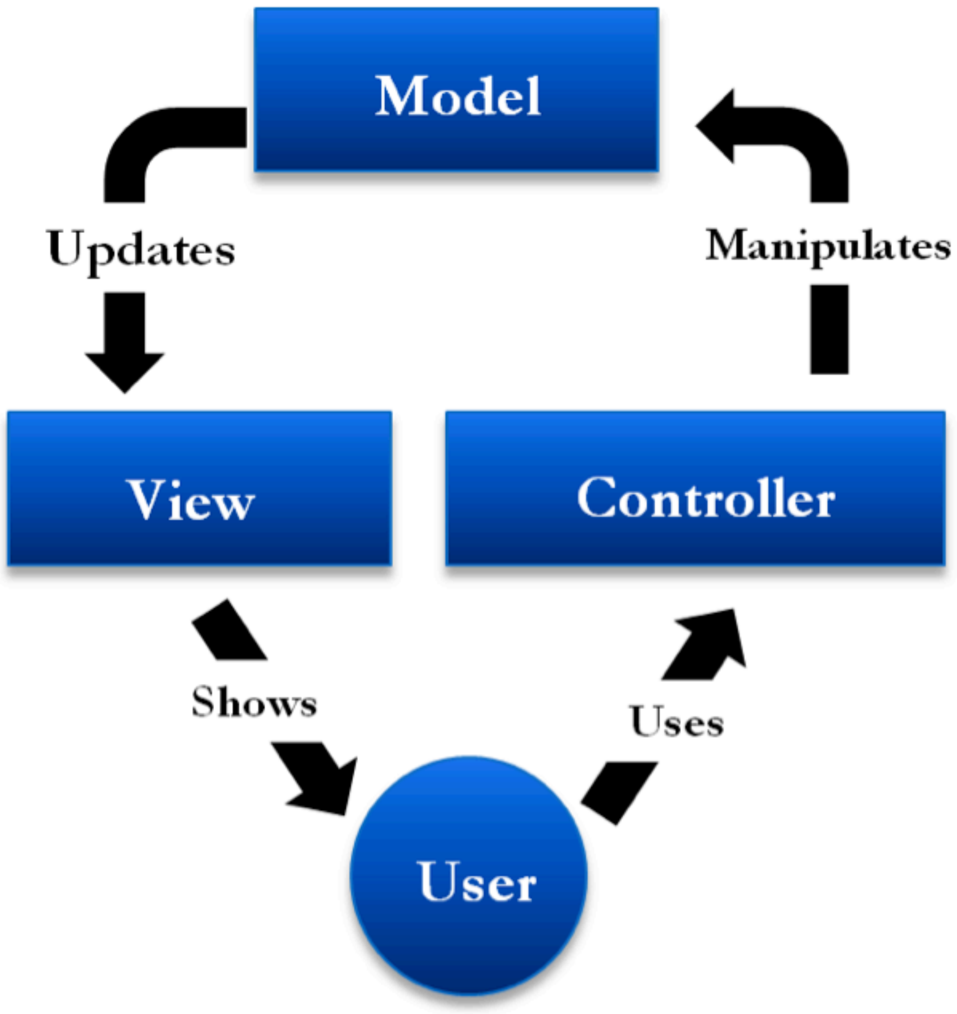


Figure 14: Model-View-Controller

Approaching this project, we spent much of the focus of the frontend working in Angular, an area we were the least familiar with. A large amount of time was devoted to learning Angular as fast as possible. As the front end was programmed, it quickly became clear that there was much we did not know about Angular, and we were forced to spend more time learning than originally anticipated. We chose to use Angular for a large portion of our project, despite knowing little of its workings, because it was an indisputable requirement of the project upon arrival. Our sponsor required the use of Angular because it is a newer technology that is quickly gaining popularity among web-developers.

The integration of Bootstrap and Angular gave our project a modular design, allowing common code to be abstracted and used in many locations. Abstraction is important for code understandability, consistency, and future expansions.

The last aspect of our front-end design was unit and end to end testing, which is not part of the GUI. Testing was handled entirely by Angular plugins and used to ensure that changes to code keep desired functionality. If a test fails, then the code is no longer acting as expected and needs to be fixed. Testing is a vital part of the user experience, though the user will never actually see it reflected in the GUI.

3.5 Java API

To create an effective method of communication between the user interface and the database system, we designed a RESTful application programming interface (API) which links the two elements. REST (representational state transfer) is an architectural style used to develop software and web applications. For web applications such as our service registry, the design involves a base URL with links to various extensions that call standard HTTP methods such as GET, POST, PUSH, and DELETE as shown in figure 15. These calls send a JSON (JavaScript Object Notation) string with the request type and any relevant data to a RESTful API that processes the request and tells the database how to act. Once the database returns the query results, it is sent back through the API to the front-end of the web application.

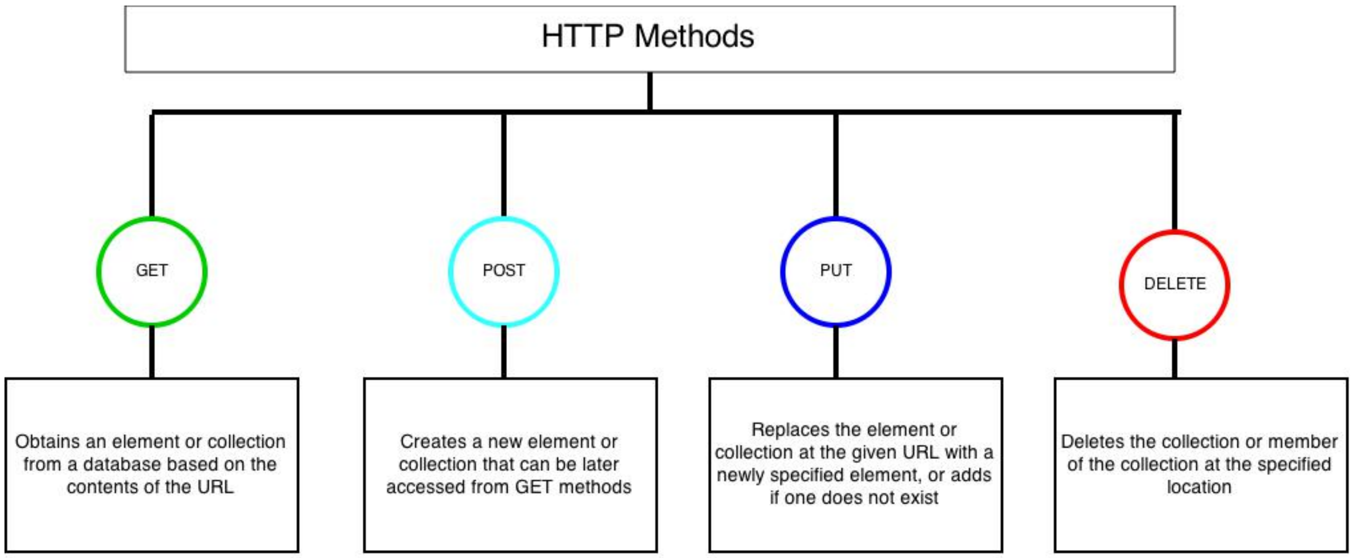

HTTP Methods
GET
POST
PUT
DELETE
Obtains an element or collection from a database based on the contents of the URL
Creates a new element or collection that can be later accessed from GET methods
Replaces the element or collection at the given URL with a newly specified element, or adds if one does not exist
Deletes the collection or member of the collection at the specified location

Figure 15: HTTP methods

We decided to use the Java programming language because of its object-oriented capabilities. Java allowed us to capture data from the user interface or the database in an easily understandable format, by creating instances of individual objects with specific properties. In addition, Java objects can be quickly translated into formats that are readable by both the database and front-end with the use of Spring Dependency Injection. Spring gives the capability to annotate Java code with tags that, when compiled, tell the program where and how to send its information.

Another important reason we chose Java was the testing capabilities that the language provides. With the JUnit testing suite, instances of objects such as services and teams can be created each time a new test is run. Scenarios can then be built up which test how the code handles successful outcomes, validation, and error checking.

3.6 Database Design and Implementation

3.6.1 Requirements analysis

Requirements analysis specifies what the database is required to do in order to be considered complete based on JPMorgan's needs. The core goal of our requirements analysis was to fully understand all needs by using business language instead of technical language. Our requirements analysis included two main activities, collecting and recording requirements. Requirements analysis was applied several times during the process of our database design in order to clarify unclear language, and remove redundant or unimportant details (Buxton, 2009).

The collecting phase included both data and functional requirements. The data requirements specified all user groups and application areas as column names in the database, called attributes; as well as, operations that were applied to the data; such as, adding, deleting, and updating. Data requirements are important because they are used as the basis of the database. On the other hand, functional requirements specified all the tools and software that were used in the process of designing and implementing the database. Our group analyzed the collected requirements by holding several meetings to discuss and focus our thoughts.

Recording requirements is the process of documenting requirements in various forms. We recorded our collected requirements by creating user stories in easily understandable business terms.

3.6.2 Logical data model

The logical data model (LDM) describes the structure of the collected requirements. The LDM includes criteria such as attributes' details, key statuses, and relationship types (Database Design Essentials, 2011). Before the implementation of the physical database, our group designed an LDM on paper. We began our design of the logical data model by separating the collected data into child and parent tables. A child table includes the data that is dependent on the data of a parent table. Once all tables were constructed, primary keys were specified for all tables.

The primary key is a unique attribute that identifies each record in a table (Buxton, 2009). We used three methods of specifying tables' primary keys in the design of our database. First, we looked for a unique attribute within each table to assign this attribute as the primary key. Second, when a table did not include any unique attributes, then we used the server to automatically generate a unique sequence to be used as the primary key. When this was the case the generated primary key was added as an additional attribute to the table. Third, if a table had two or more attributes that uniquely identified the table's records, a compound of them was used as the primary key.

A foreign key is an attribute in one table that references the primary key of another table. The main goal of assigning a foreign key is to be used as a retrieval key for the data of the parent table. The process of assigning a foreign key creates a relationship between a child and parent table.

A one-to-many relationship means that a single record of a table is connected to many records of a second table through a foreign key. In other words, the foreign key recorded in the second table can be used to reference a record in the first table. Our group designed the database to have one-to-many relationships between all tables because it was the simplest and most useful relationship (Moss, 2014).

3.6.3 Specifying DBMSs

A Database Management System (DBMS) is computer software used to organize, store, and retrieve data of the database. A DBMS is where the final design of the database is implemented to allow interactions with the server such as creation, querying of data, and maintenance of the database. Our group decided to use two types of DBMSs, Hyper SQL for the development phase and Oracle DB for the production phase.

We used the Hyper SQL database (HSQLDB) for the development phase. HSQLDB is a small database engine with in-memory and desk-based tables that supports embedded and server modes. Since HSQLDB is loaded into memory with a fresh set of tables every time it is started, it is more easily implemented into the development cycle than a standard database such as Oracle. Also, HSQLDB allows multithreading, which is the ability to manage its use by more than one user at the same time (HSQLDB, 2014). Due to these reasons, we used HSQLDB as our DBMS in our development phase.

We chose the Oracle database for the production phase because of its well-known reputation among enterprises. The Oracle database can be relied on to manage a large amount of data in a multiuser environment, so that many users can concurrently access stored data. The Oracle database offers high performance and a high level of standards regarding data security (Oracle, 2014).

3.6.4 Physical data model

The physical data model (PDM) defines how the data will be stored in a pre-specified DBMS (Connolly, 2014). We used the PowerDesigner software to design the PDM, which translated the LDM into a language that could be understood by the DBMS. The physical data model specifies data types, field constraints, and alternate keys. Data types are a way for the database to understand the form of data entered by users, such as integers, characters, and boolean values. Setting field constraints is the process of specifying a set of certain entries to a data value. For example, our PDM utilized these constraints to restrict entries of a service's version status to invest, divest, or maintain. An alternate key is similar to a primary key in that it must be unique for each entry in a table, and is used as a secondary method for preventing the duplication of data.

After adding all configuration details to the database design, PowerDesigner allowed for the generation of multiple SQL files for the specified type of DBMS. Two SQL files were generated for our DBMSs, one for the HSQLDB and the other for the Oracle Database. The SQL file for HSQLDB was then inspected using a software tool called Squirrel SQL, a graphical Java program that allowed viewing and browsing of database tables. The inspection was essential to guarantee valid connectivity between the DBMS and the server; as well as, remove SQL statement errors before proceeding to the real connection process. After the inspection, HSQLDB was connected to the server in order to start the development phase. The database was updated several times during the project, and the process of updating the database design is illustrated in figure 16.

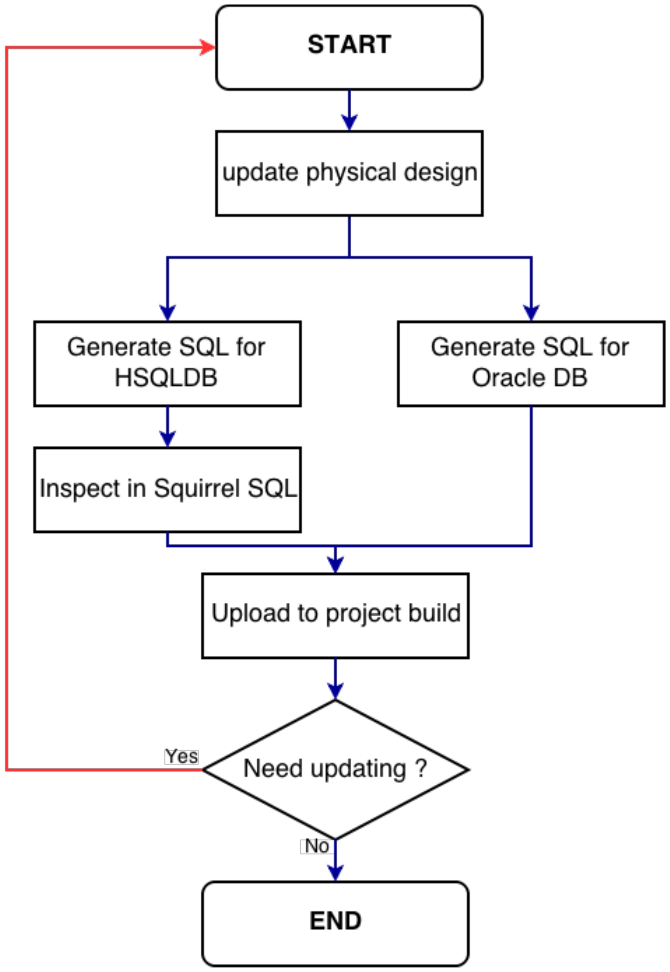


Figure 16: Database update process

3.7 Admin Reports

One focus of this project was to create functions on our service registry that can generate administrative reports and spreadsheets about the uploaded services and teams. These reports will be useful to track available services and teams that are actively creating services. One example is a report that shows service versions with unachievable Service Level Agreement (SLA) time due to their dependencies. This report would allow a risk operative to see the team that created the service in order to contact them and have them updated the service to reflect the proper SLA. Another example is the generation of an administrative report that can be used to warn individuals and teams that depend on a particular service if that service happens to temporarily stop working. The warning will help to keep the proper workflow and prevent any unexpected business losses. Reports, such as those highlighted in the previous examples, will help administrators, risk operatives, and project managers to monitor internal JPMorgan services and keep all information up to date. Administrative reports will be useful to many individuals at JPMorgan and help to improve the use of services.

3.8 Summary

Within the timeframe of six weeks, split into three sprints, we created the service registry which satisfied many user stories by incorporating required features. The features included the customization of services, service versions, and teams; as well as, assigning dependencies for services. All users: business analysts; teams; team members; and administrators, should be able to make adjustment for their specific needs. To accommodate the specifications of our sponsor, we first had to plan out how to approach the development process.

We accomplished the tasks in each sprint by using TDD, pair programming, and scrum planning concepts. TDD allowed us to autonomously test the code and thus save time debugging. Additionally, pair programming improved the design quality, satisfaction, and learning process.

Scrum planning introduced effective iterations and leveraged team efforts for successful sprints. Furthermore, a retrospective at the end of the sprint allowed for group discussions, improved awareness of task challenges, and provided recommendations for the next sprint.

The user interface allows the users to interact with the service registry in a simple manner. Our project focused more on functionality than design; hence, we relied on Bootstrap's default templates to promptly create a standard website. Our team took the time to learn Angular because it is a new web-development tool, providing invaluable experience and assistance for designing a web application. Before front-end design could be considered done, unit testing, provided by Angular plugins, was implemented to ensure desired functionality and make sure that the code was working properly. Whenever the user interface wants to communicate with the database, our RESTful API will answer the request. This API was implemented with Java due to the language's object oriented capabilities. An SQL query would then perform all the database related tasks.

Requirements analysis, the collection and recording of requirements, helped us to understand our sponsor's needs, remove unimportant details, and distinguish data and operations during the database design. A logical data model (LDM) is an abstract design that describes the structure of the collected requirements. We used primary keys and foreign keys to connect child and parent tables in our database. We specified two types of DBMS, HSQLDB for the development phase, and the Oracle database for production phase. The physical data model (PDM) focused on translating the LDM into a language that can be understood by DBMSs. After we created the PDM, PowerDesigner generated the SQL for our DBMSs.

Administrative reports would keep the data monitored and help the data be kept up to date. Administrators, risk operative, and project managers would be able to regularly understand the usage of services and make any necessary changes.

4.0 Results

The result of our project was a basic service registry designed for the internal use of JPMorgan employees. Our service registry was developed in three tiers, as seen in figure 17: a front-end, which acts as the view for the user; a server, which handles information from the frontend and database; and a database, which stores information. The user interface is the front-end of the application, built in AngularJS, that creates the website of the service registry. The server was built using Java to effectively pass information from the user interface to the database. The database stores all information added to the service registry about existing services and teams. All three parts of our design constantly interact with one another to create our functional service registry.

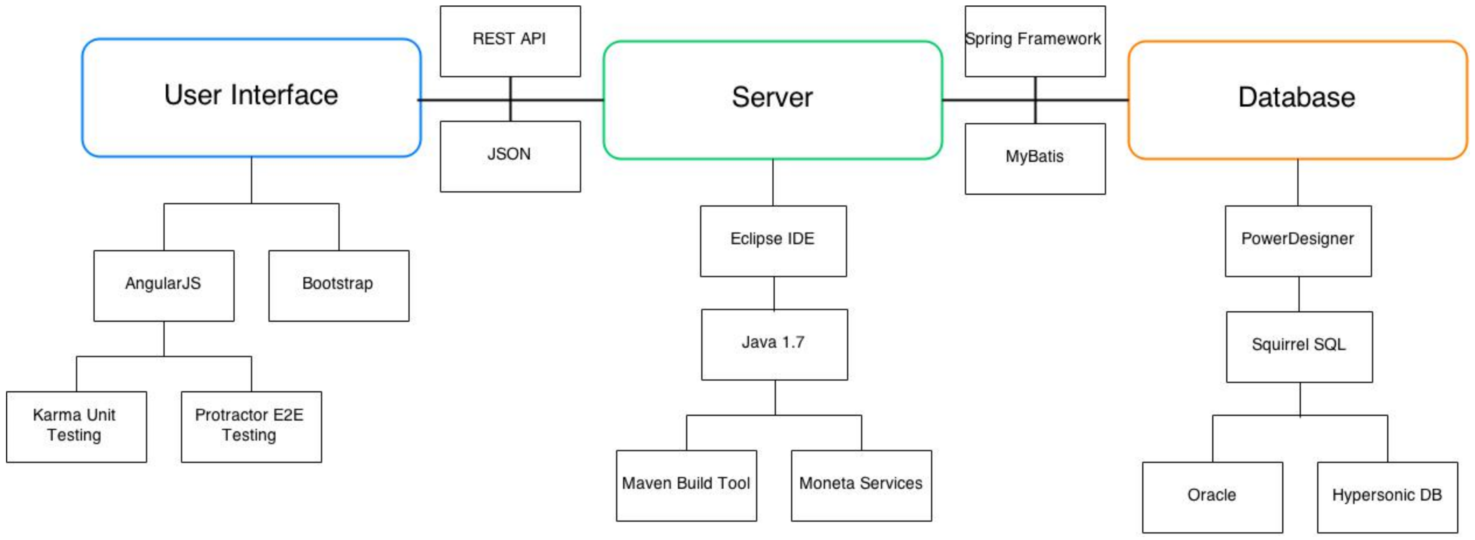


Figure 17: Service registry technologies

Approaching this project, our team was aware that its completion was unlikely. Our sponsor estimated that the project would take a minimum of ten weeks to complete and we were only able to work on it for six. In the six weeks we had, we completed 21 stories, mostly completed 5 stories, and did not begin 10 stories. The 5 mostly completed stories were left unfinished because they had pieces that depended on the completion of at least one story we did not start. The 21 completed stories were often composed of many smaller tasks that needed to be finished in order to complete the story. Supporting our completed and mostly completed stories were 97 JUnit tests providing 90% code coverage of coverable Java code, 86 Karma unit tests, and 12 Protractor end-to-end tests.

4.1 User Interface

Our user interface, from a computer science perspective, is separated into three reasonably distinct sections: the model, the view, and the controller. The view is handled by Bootstrap, HTML, and CSS, with elements of Angular that interact with both the model and the controller. The model is handled by Angular, particularly by Angular defined services which are created into objects for information storage. The controller is similarly handled by Angular, particularly by Angular defined controller classes, which handled all interactions with the view. Following the Model-View-Controller paradigm allowed for common code to get reused and for quick expansion of functionality when needed.

From a user perspective, our front end is separated into several functional views that each contains multiple layers based on user type. The functional views are focused on the different aspects of the service registry, such as services or teams; whereas, the user layers alter the display of the view based on the privileges of the current user. Ideally, users would only see functionality that is accessible by their privilege level; however, currently all user types can see the links to all functional areas of the web page. Due to time constraints, the ability to hide higher privileged areas was considered non-vital, and replaced with the less time consuming method of redirecting underprivileged users to an access-denied page.

4.1.1 Services

Viewing, browsing, and searching for services is allowed for all user types: business analyst, team member, team owner, or administrator. The bottom section of the front page of the website, as seen in Appendix C, displays a list of current services, with pagination and the option of increasing the length of the displayed list. Above the list of services is a search bar, for easily browsing the contents of the list. Typing into the search bar will filter the services by all fields, including those that are not shown.

When users find the service they are looking for, they can select the service name and will be directed to a page containing more information on that service, as seen in Appendix D. On the service view page, users can view more detailed information of a service including version information. The service version information is displayed on the bottom half of the page in Appendix D and allows users to see information on each version including dependencies and dependents of the version with links to all dependencies and dependent services. The dependency and dependent views can be seen in Appendix D. Due to time constraints, the view shown in Appendix D is lacking several fields of information, such as links to documentation, as the incorporation of uploading documentation was not implemented by the end of our time with the project.

The privilege of adding a service is given to team owners and administrators. In order to add a service, the user must select a valid SEAL ID. SEAL is a catalog of applications used by JPMorgan, and a valid SEAL ID must be entered when creating any service on the registry. The SEAL selection screen has a visible dropdown menu as seen in Appendix G. The next page of the add a service wizard, as shown in Appendix G, allows the user to enter information about the service, such as the service name and the team associated with it. Ideally, only team owners would be able to create services for teams they are a part of; however, the prerequisite of user validation was completed too late in the project, and the limiting of which teams can be selected was put on the backlog for after our departure. The front end checks for the correct format of the data and, if any errors exist, will not allow the user to proceed until the problem is corrected. Once valid service information has been entered the user will be able to progress to the version information page as seen in Appendix D. Ideally, the version information page would have fields for uploading documentation, but as stated earlier, the functionality of uploading documentation could not be fully implemented due to time limits. Once all information has been entered, the user is sent to a confirmation screen, as seen in Appendix G, where the user can submit the service to the database and be brought to the service view page of the newly created service.

Users can add or delete service versions along with dependencies on those versions; however, editing a service is currently largely unimplemented. The view of a service page with privileges to edit can be seen in Appendix H. Adding a service version is not fully implemented, due to time constraints, and lacks of version validation. Ideally, a service can only have two different major versions, attempting to add a third major version would result in the deletion of the oldest major version, but this functionality was put on the backlog for after we leave because the prerequisites required to implement it were not completed until our final sprint. Adding a dependency is almost fully implemented, but lacks small validation preventing dependencies to multiple versions of the same service. Finishing the validation of dependencies should not be a difficult task, but it was put on the backlog due to being considered less vital for completion before we left. The deleting of both service versions and dependencies are fully implemented.

4.1.2 Teams

The view teams page, as shown in Appendix F, is accessible by any team member, team owner, or administrator, each with slightly different functionalities. Team members and team owners can view any team they are assigned to via the view teams page. Team members may not edit team information; whereas, team owners can edit only teams they are owners of, and administrators can edit any team, as shown in Appendix H. Currently, teams can be searched for by team name, but the table cannot be order alphabetically by team name. Due to time constraints, our sponsor considered the functionality of ordering the team view table by team name as too time-consuming and insignificant.

The add a team page, as shown in Appendix E, allows any user to create a team. A team must have a valid and unique name as well as at least two team owners. The add a team page has a button to allow users to add as many members to their team as is required. The add a team page is fully functional and has visual error checking on all fields.

4.2 Java Server

To begin the design of our API, we utilized the Moneta Services template created by the CCA team at JPMorgan. As mentioned in the Technologies Review, Moneta contains libraries and tools to implement a RESTful service in Java. JPMorgan also provided an example of a Moneta service on which we based our design. The example emulated a service that dealt with the passing of information between a database and a client related to libraries, books, and authors. Creating a template about a subject that was simple to understand allowed our team to quickly apply the information to a new service based around our needs. The documentation that the CCA team provided on the Moneta website, included a walkthrough on how to create a service and add resources, also proved to be very useful.

In our Java code, we created multiple layers of abstraction to clearly separate the flow of data from the user interface to the database system and vice versa. The main purpose for this abstraction was to allow the addition of components within the application in the case of future modifications by JPMorgan. The abstraction also separates error checking into levels that prevent propagation of data in the event of a problem. In addition, the separate layers make the code much more readable in the event that JPMorgan employees outside of our team need to make changes to the service registry.

As shown by figure 18, data from the client is originally received at the resource level. This level decides where to pass the data based on the type of HTTP message the client is sending. The structure of the resource level was developed by using Jersey to annotate our code with tags that, when compiled, tell Java which resource method to use. The data is then passed to a service level that serves only as a buffer between the above and below levels. In the event that changes need to be made to either, the opposite level will require minimal modifications to continue functioning properly. From there, the data continues to the database access object (DAO) which contains various calls to the database in order to retrieve, post, or modify data and also provides error checking that can be passed up to the user interface in the case of invalid requests or input. Finally, the data arrives at the database mapper, which uses MyBatis to transform the request into an SQL query that performs the correct action at the database level. We chose MyBatis because it integrates well with the Spring framework used to connect many other aspects of our server.

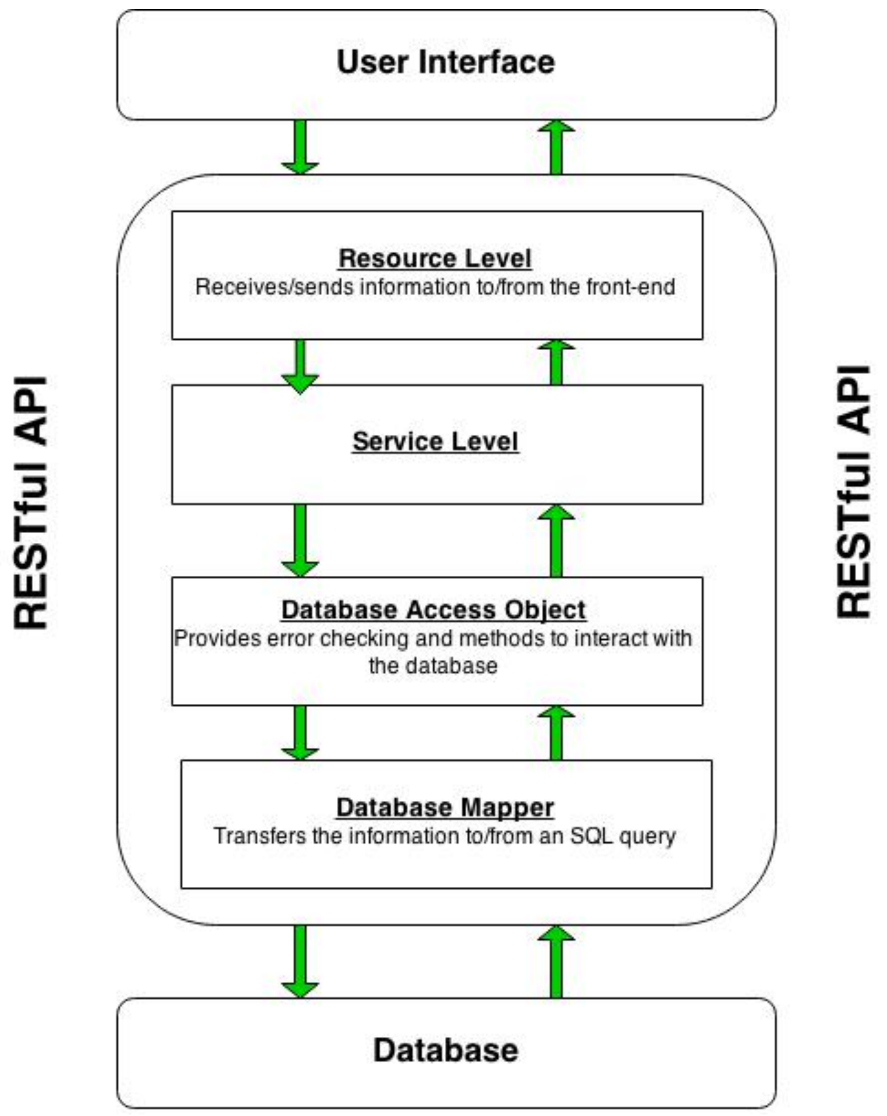

User Interface
RESTful API
RESTful API
Resource Level
Receives/sends information to/from the front-end
Service Level
Database Access Object
Provides error checking and methods to interact with the database
Database Mapper
Transfers the information to/from an SQL query
Database

Figure 18: RESTful API levels

4.3 Database

Database design included seven tables, which are services, service versions, lead design authority (LDA), dependencies, teams, members, and administrators. The table of administrators was not connected to any other table because there was no need to interact with its data. The table of administrators served as a source to authenticate administrators when they access the website. All other tables were connected by one-to-many relationships as shown in figure 19.

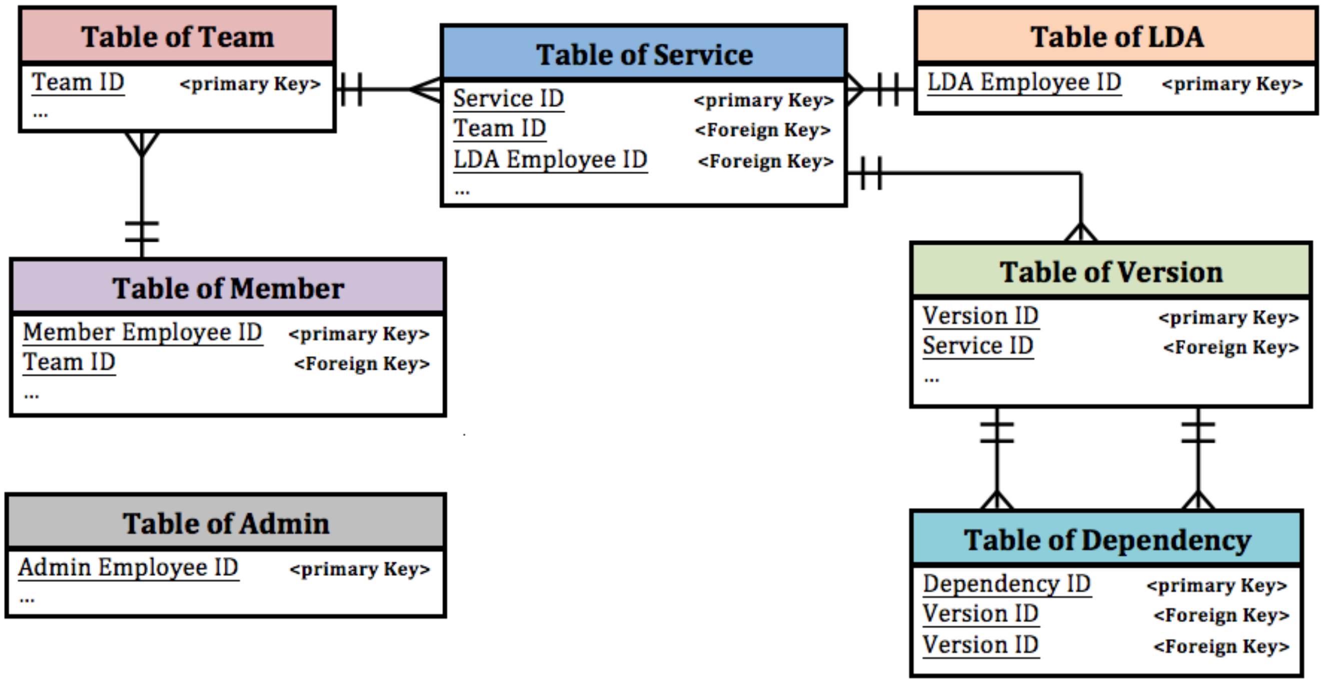


Figure 19: Database schema

After constructing the tables, we assigned a primary key for each table using the three methods stated in section 3.6.2. First, an automatic generated primary key was assigned to tables that do not include any unique attribute such as employee ID. Second, we used employee ID as a primary key in tables where it exists as an attribute because it is unique for each record. Third, a combined primary key was used in the member table, which includes both employee ID and Team ID. The combined primary key was necessary in order to not duplicate an entry of a member on the same team. After that, foreign keys were assigned by placing primary keys in other tables for the purpose of retrieving records. An alternate key was assigned to each table to prevent duplicate entries to the said tables. Alternate keys were usually constructed of more than one attribute since one attribute might not guarantee the preventing of duplicated entries.

5.0 Recommendations

While this project resulted in a basic service registry for JPMorgan, several recommendations can be made: adding the capability to pull administrative reports; to send notifications to teams; registering external services; educating employees about the registry and expanding the use of the registry beyond Asset Management.

5.1 Administrative Reports

We strongly recommend that the CCA team at JPMorgan continue to improve the service registry by adding a feature to create administrative reports. Originally we planned on implementing this feature while working at JPMorgan, but due to time constraints it was out of the scope for our project. Based on the needs of administrators, risk operatives, and project managers we see high value in creating reports to help them view the current services and all information about them. One example is to create a report that shows dependencies on a particular service. In the event of a new service version being created the administrator can contact the appropriate team leaders to alert them of the change. As services continue to be developed at JPMorgan the service registry allows employees to find what they are looking for and administrative reports will help keep everything monitored and up to date. Without these reports it would be very time consuming for administrators to go through each service or team to find what they are looking for; however, an automatically generated report can display desired information in a simple format such as a spreadsheet. Based on user needs we highly recommend adding a feature to the service registry to create diverse reports.

5.2 Registering External Services

In addition to registering internal services, we highly recommend adding the functionality of registering external services from third-party vendors. Adding this functionality would limit the amount of duplicate external services because they are grouped in a central location. When a team depends on an external service, the team would be able to register their usage of the service. If other teams need to assign their dependencies of the same external service, the service will be discoverable through the registry. Registering external services will increase the use of the registry since teams can keep track of all types of services whether internal or external.

5.3 Notifications Tool

We recommend adding the functionality to send email notifications from the service registry to reach out to users and inform them of any important changes with services they use. For example, when a service gets marked as divest the registry would send out notifications to all services that are dependent on it. This notification would let the dependents know that they should update to a newer service version because the one they currently depend on is outdated. These notifications will allow the registry to stay updated and give users warning before service versions are deleted.

5.4 Education and Expansion

One reason that JPMorgan's reputation is increasing from year to year is the amount of courses provided to employees. Example of these courses are understanding fraud transactions, understanding corruption in stock market, learning a new programming tool or coding style, and learning a new technology used in the firm. Courses have mutual benefits for both the firm and the employees; therefore, we strongly recommend providing a short online course to educate employees about the service registry. The course will educate employees of the service registry's existence as well as explain the advantages and outcomes of using it.

6.0 Conclusion

The internal service registry created a platform to organize existing services, search through existing services, and record newly developed services and development teams. The service registry allows JPMorgan to track all current versions of services and their availability. When adding a new service to the registry all information provided by the user is validated for correctness, and for certain fields the registry calls other databases to validate the information.

Before the existence of the service registry, a JPMorgan employee would have no way to find new services except for asking colleagues. Services created for a specific team were only used within that team when they could potentially be expanded for use throughout the company. The service registry we created will hold information about all the services developed for Asset Management in one central location. The service registry will allow employees to easily find new services that can be applied to a business process they use. The service registry promotes the use of existing services and encourages employees to further develop a service oriented architecture.

Throughout the project our team built a service registry from the ground up which will be moved to production after we depart. Our team combined our knowledge from each of our different areas of study to collective build the service registry. Approaching the project from a computer science, actuarial mathematics, industrial engineering, and electrical and computer engineering prospective allowed our group to develop and discuss the best ideas for how the service registry would operate. Once the service registry is deployed in the production phase, it will be open for all employees to use and add services to. We built a basic model for this service registry, but there are still some additions to be done in the future. The service registry can be improved with additions such as adding the capability to upload documentation on the service, to create administrative reports, and to send notifications to a team when necessary. Finally, the service registry is only for employees in the Asset Management department of JPMorgan and we suggest expanding the service registry company wide.

References

AngularJS. (n.d.). Retrieved December 20, 2014, from https://angularjs.org/

Are you ready for Java™ 8? (2014). Retrieved December 20, 2014, from https://eclipse.org/

Bootstrap. (n.d.). Retrieved December 20, 2014, from http://getbootstrap.com/

Bou-Ghannam, A. A., Creamer, T. E., Moore, V. S., Narayan, V., & International Business Machines Corporation. (2007).*Telecommunication service registry*

Buxton, S. (2009). *Database design: Know it all.* Amsterdam: Morgan Kaufmann /Elsevier. Retrieved December 19, 2014.

Company History. (2014). Retrieved from https://www.jpmorgan.com/pages/company-history

Connolly, T., & Begg, C. (2005). *Database systems: A practical approach to design, implementation, and management* (4th ed.). Harlow, Essex, England: Addison-Wesley. Retrieved December 19, 2014.

Database Design Essentials. (2011, April 1).*SQL Server Magazine.* Retrieved December 19, 2014

HSQLDB. (n.d.). Retrieved December 19, 2014, from http://hsqldb.org

Jersey. (n.d.). Retrieved December 20, 2014, from https://jersey.java.net/

JPMorgan Chase & Co. (2014). 2013 *complete annual report of JPMorgan Chase & Co.* Retrieved from http://investor.shareholder.com/JPMorganChase/annual.cfm

JSON. (n.d.). Retrieved December 20, 2014, from http://www.json.org/

JUnit - About. (n.d.). Retrieved December 20, 2014, from http://junit.org/

Maven. (n.d.). Retrieved December 20, 2014, from http://maven.apache.org/

MOSS, K. (n.d.). The Entity-Relationship model. *WEB Site Simulations4Teaching.co.uk,* 6-6. Retrieved December 19, 2014, from http://ieeexplore.ieee.org/Xplore/home.jsp

Oracle | Hardware and Software, Engineered to Work Together. (n.d.). Retrieved December 19, 2014, from http://www.oracle.com/index.html

PowerDesigner. (n.d.). Retrieved December 20, 2014, from http://www.powerdesigner.de/en/

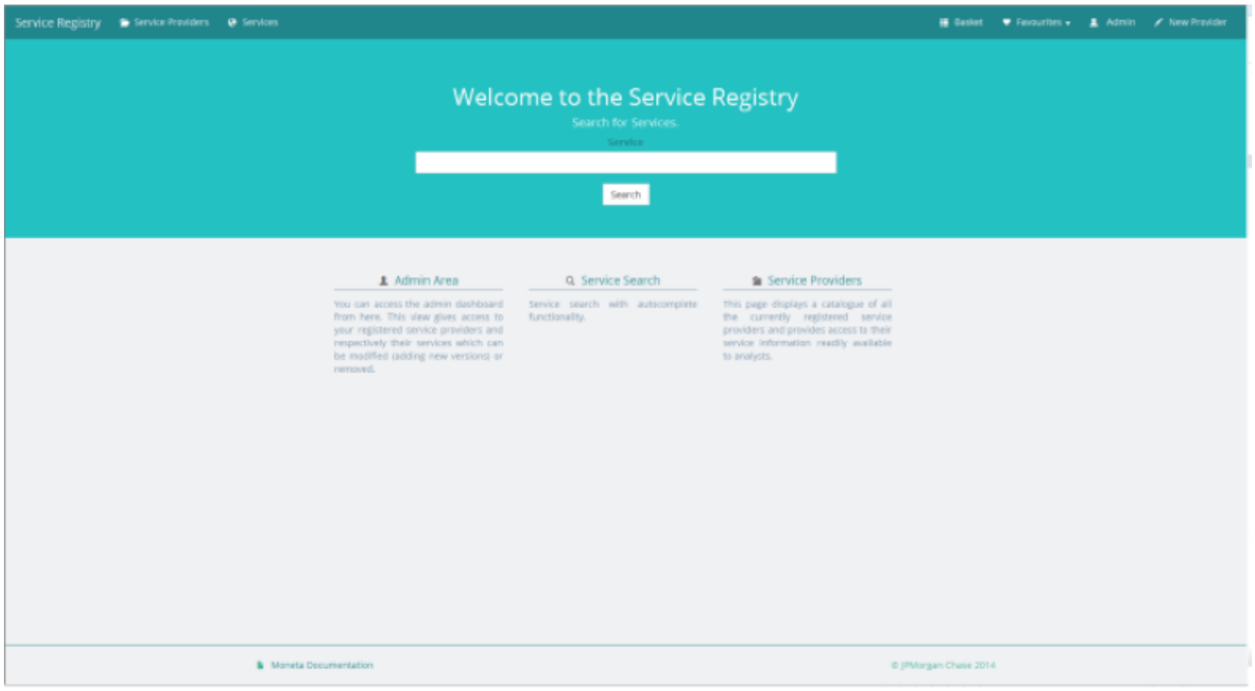
Squirrel. (n.d.). Retrieved December 20, 2014, from http://squirrel-sql.sourceforge.net/

Sublime Text. (n.d.). Retrieved December 20, 2014, from http://www.sublimetext.com/

Treiber, M., Treiber, M., Dustdar, S., & Dustdar, S. (2007). Active web service registries. *IEEE Internet Computing, 11*(5), 66-71. doi:10.1109/MIC.2007.99

Appendix A: Proof of Concept

Homepage:



Admin Area

You can access the admin dashboard from here. This view gives access to your registered service providers and respectively their services which can be modified (adding new versions) or removed.

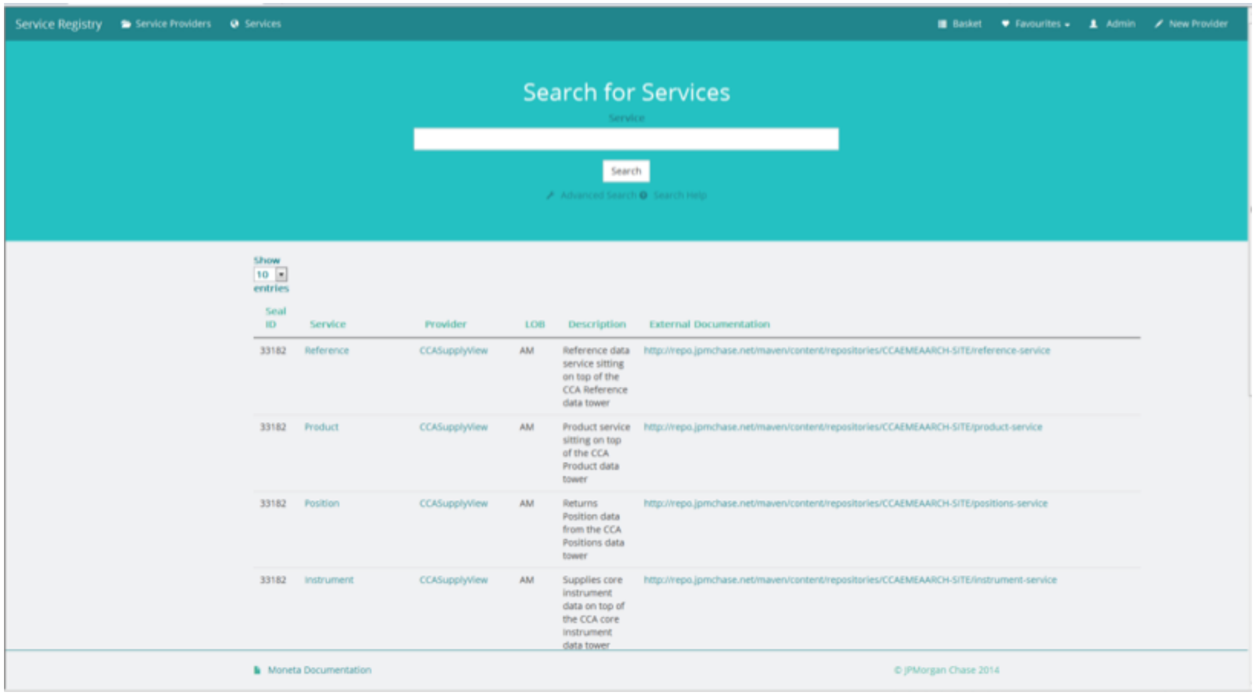
Service Search

Service search with autocomplete functionality.

Service Providers

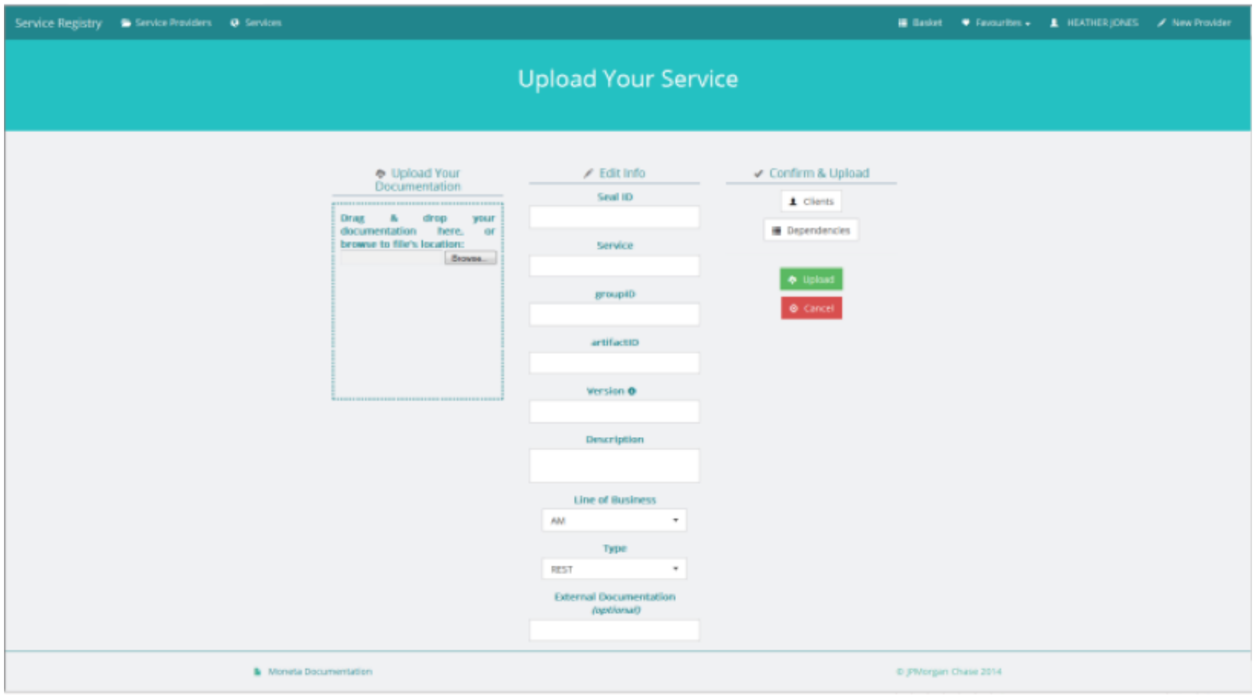
This page displays a catalogue of all the currently registered service providers and provides access to their service information readily available to analysts.

Search page:



Seal ID	Service	Provider	LOB	Description	External Documentation
33182	Reference	CCASupplyView	AM	Reference data service sitting on top of the CCA Reference data tower	http://repo.jpmchase.net/maven/content/repositories/CCAEMEAARCH-SITE/reference-service
33182	Product	CCASupplyView	AM	Product service sitting on top of the CCA Product data tower	http://repo.jpmchase.net/maven/content/repositories/CCAEMEAARCH-SITE/product-service
33182	Position	CCASupplyView	AM	Returns Position data from the CCA Positions data tower	http://repo.jpmchase.net/maven/content/repositories/CCAEMEAARCH-SITE/positions-service
33182	Instrument	CCASupplyView	AM	Supplies core instrument data on top of the CCA core instrument data tower	http://repo.jpmchase.net/maven/content/repositories/CCAEMEAARCH-SITE/instrument-service

Upload service page:



Drag & drop your documentation here, or browse to file's location:

Line of Business: AM

Type: REST

Moneta Documentation

© JPMorgan Chase 2014

Appendix B: Sprint Stories

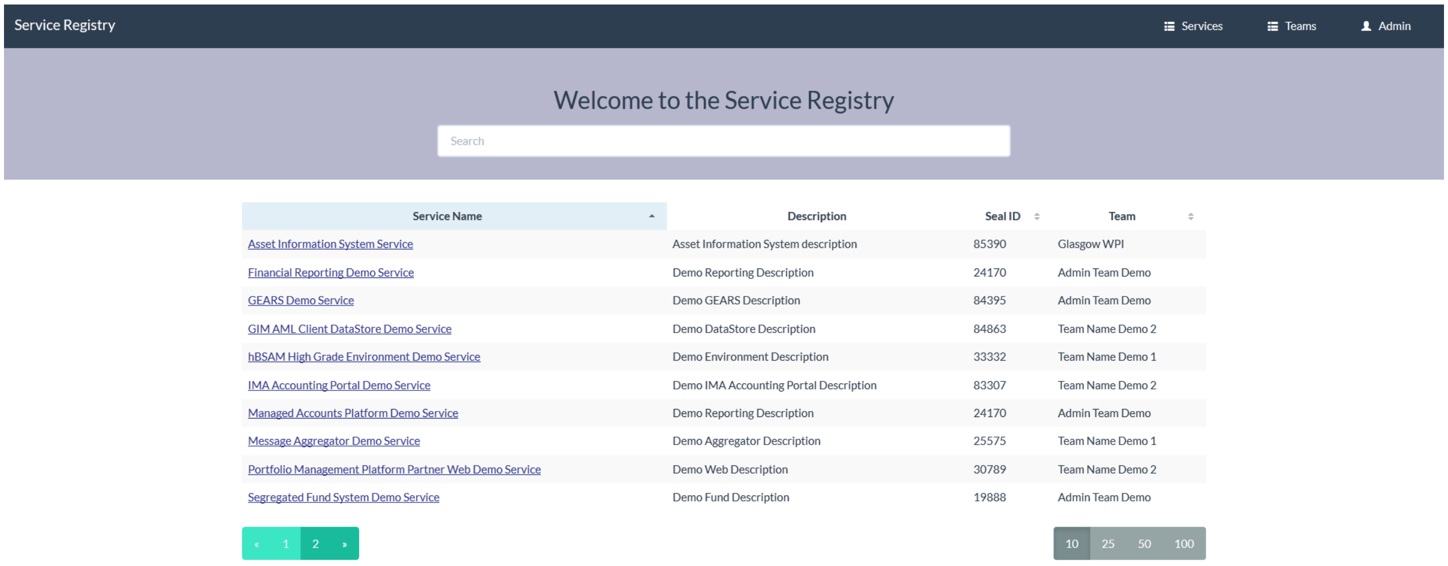
Role	Service Owner	Business Analyst	Project Leader	Others
Sprint 1	Create a team to allow service management	Browse all available services	Add a brand new service	Assign a team to a service
	Delete a team I own		Add a new version of my service	I would like to add the Swagger documentation related to my service version.
	Delete unused or old service versions			
	Register Dependencies my services may have on other services			
	See which teams have registered a usage of my services.			

Role	Service Owner	Business Analyst	Others
Sprint 2	Promote my service version from Test to Production	Find services by words in either their name or their description (Filtering)	I would like to upload the WSDL and schema information associated with my service version.
	Delete a service that has dependencies		As an Operate member, I want to understand the dependents of a service so I understand the impact in the event of an outage.
	Display all clients that are dependent on my service		As part of the service team, I would like to edit my service.
	Mark a service as divest so that I can start to migrate all clients to a new version		As a client of a service, I would like to register my service usage to allow me to be informed when services are being changed or retired.
	Edit my team		
	See all services I own		
	Upload examples on how to use my service		

Role	Service Owner	Risk Operative	Project Leader	Others
Sprint 3	Want swagger documentation built into his development cycle via maven.	Understand which services have unachievable SLAs so that this can be remediated	Register external services or vendor service, so that I can subsequently register my dependencies on the service.	As a Project Management Operator user, I would like to run various reports.
				Create and Maintain Administrative Team

Appendix C: Service Registry Homepage

Homepage:



Service Registry — Services — Teams — Admin

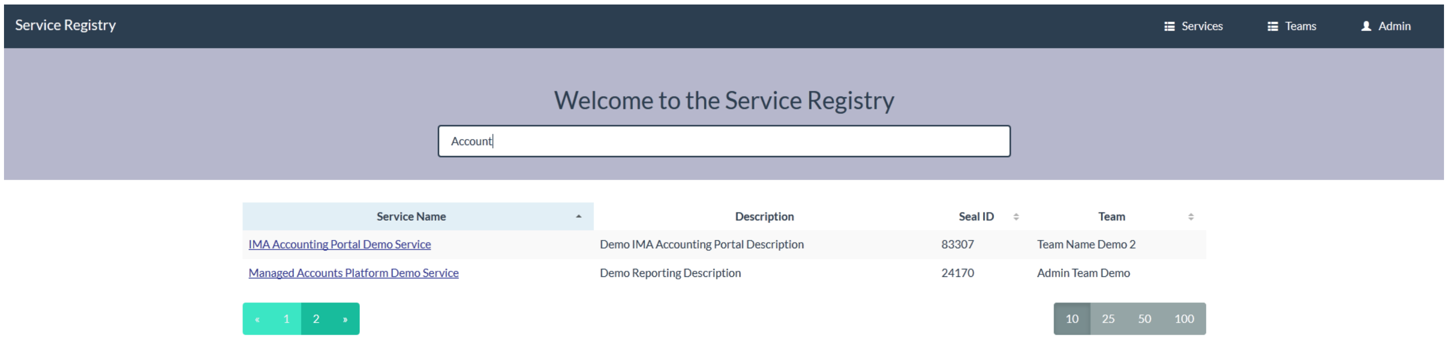
Welcome to the Service Registry

Search

Service Name	Description	Seal ID	Team
Asset Information System Service	Asset Information System description	85390	Glasgow WPI
Financial Reporting Demo Service	Demo Reporting Description	24170	Admin Team Demo
GEARS Demo Service	Demo GEARS Description	84395	Admin Team Demo
GIM AML Client DataStore Demo Service	Demo DataStore Description	84863	Team Name Demo 2
hBSAM High Grade Environment Demo Service	Demo Environment Description	33332	Team Name Demo 1
IMA Accounting Portal Demo Service	Demo IMA Accounting Portal Description	83307	Team Name Demo 2
Managed Accounts Platform Demo Service	Demo Reporting Description	24170	Admin Team Demo
Message Aggregator Demo Service	Demo Aggregator Description	25575	Team Name Demo 1
Portfolio Management Platform Partner Web Demo Service	Demo Web Description	30789	Team Name Demo 2
Segregated Fund System Demo Service	Demo Fund Description	19888	Admin Team Demo

« 1 2 » 10 25 50 100

Homepage Search by Name:



Service Registry — Services — Teams — Admin

Welcome to the Service Registry

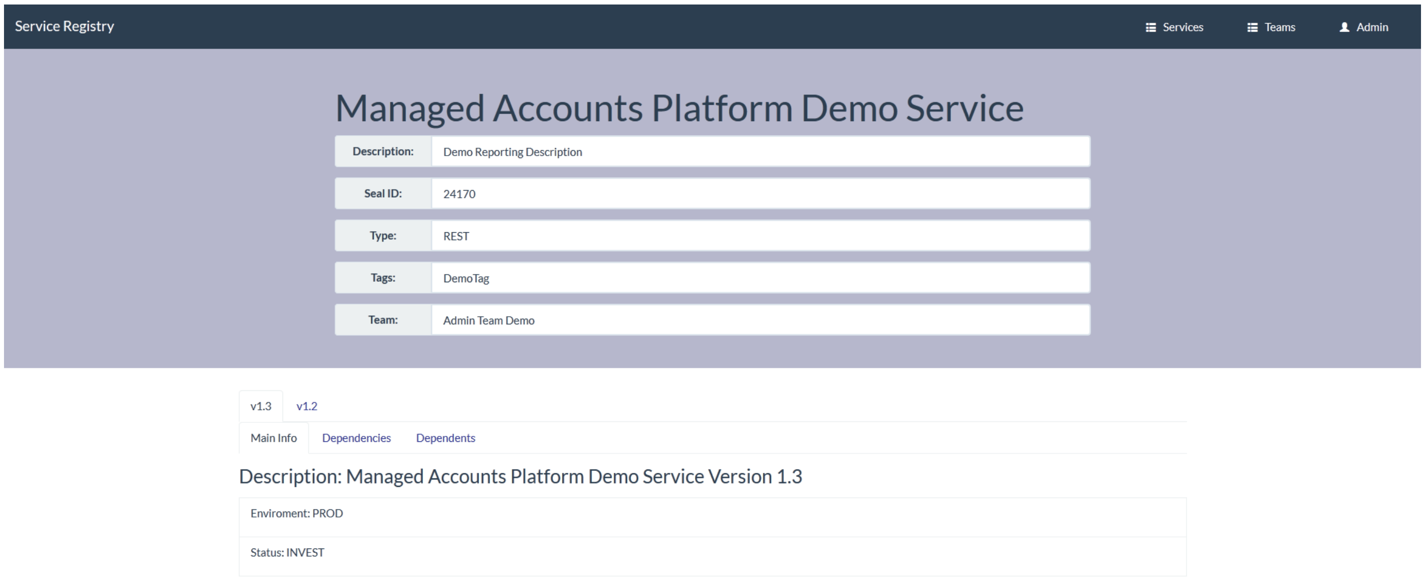
Account

Service Name	Description	Seal ID	Team
IMA Accounting Portal Demo Service	Demo IMA Accounting Portal Description	83307	Team Name Demo 2
Managed Accounts Platform Demo Service	Demo Reporting Description	24170	Admin Team Demo

« 1 2 » 10 25 50 100

Appendix D: Service View

Service View Main:



Managed Accounts Platform Demo Service

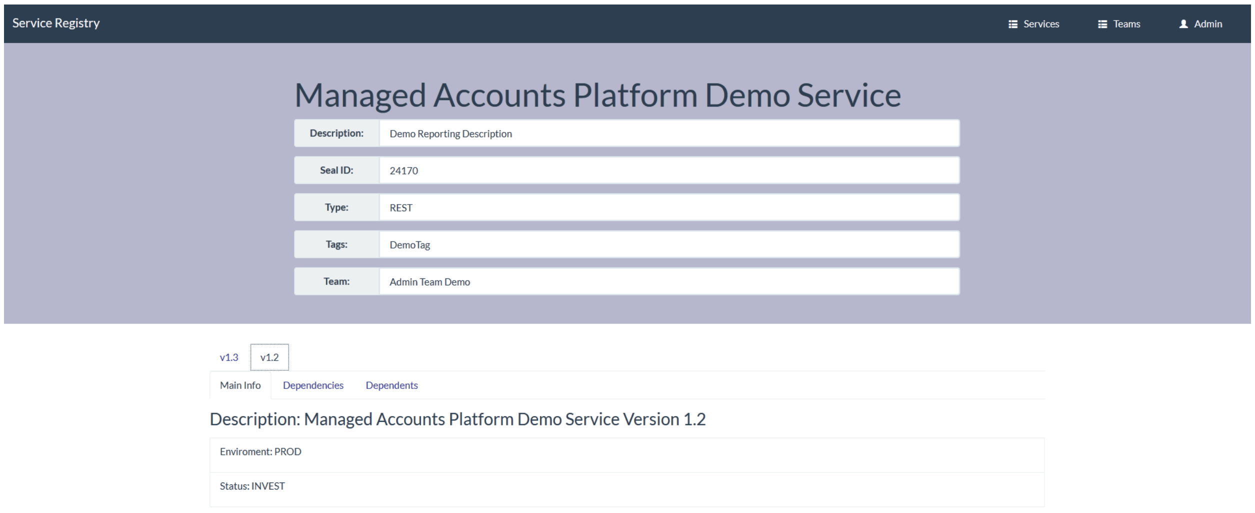
Description:	Demo Reporting Description
Seal ID:	24170
Type:	REST
Tags:	DemoTag
Team:	Admin Team Demo

Description: Managed Accounts Platform Demo Service Version 1.3

Enviroment: PROD

Status: INVEST

Service View Second Version:



Managed Accounts Platform Demo Service

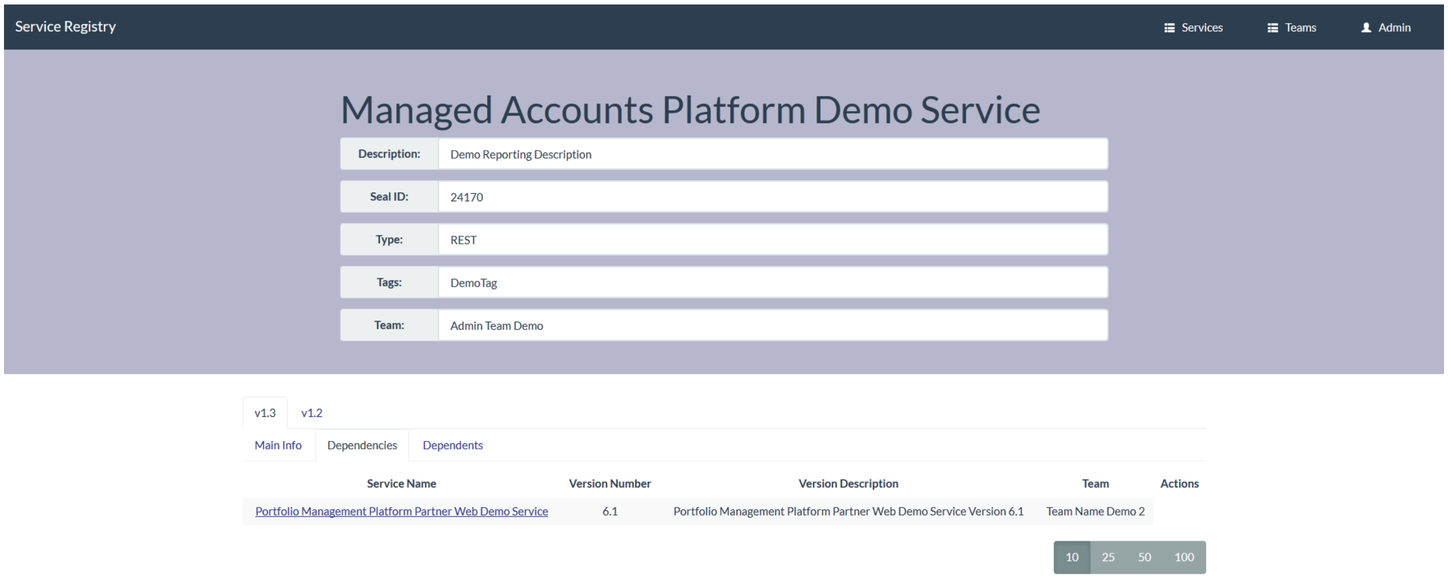
Description:	Demo Reporting Description
Seal ID:	24170
Type:	REST
Tags:	DemoTag
Team:	Admin Team Demo

Description: Managed Accounts Platform Demo Service Version 1.2

Enviroment: PROD

Status: INVEST

Service View Dependencies:



Service Registry

Services Teams Admin

Managed Accounts Platform Demo Service

Description:	Demo Reporting Description
Seal ID:	24170
Type:	REST
Tags:	DemoTag
Team:	Admin Team Demo

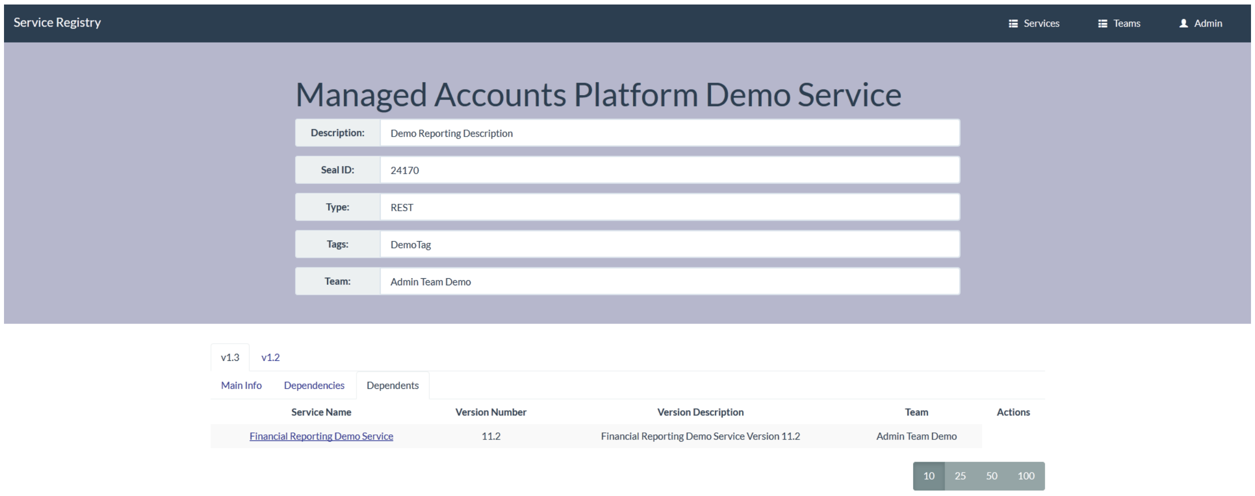
v1.3 v1.2

Main Info Dependencies Dependents

Service Name	Version Number	Version Description	Team	Actions
Portfolio Management Platform Partner Web Demo Service	6.1	Portfolio Management Platform Partner Web Demo Service Version 6.1	Team Name Demo 2	

10 25 50 100

Service View Dependents:



Service Registry

Services Teams Admin

Managed Accounts Platform Demo Service

Description:	Demo Reporting Description
Seal ID:	24170
Type:	REST
Tags:	DemoTag
Team:	Admin Team Demo

v1.3 v1.2

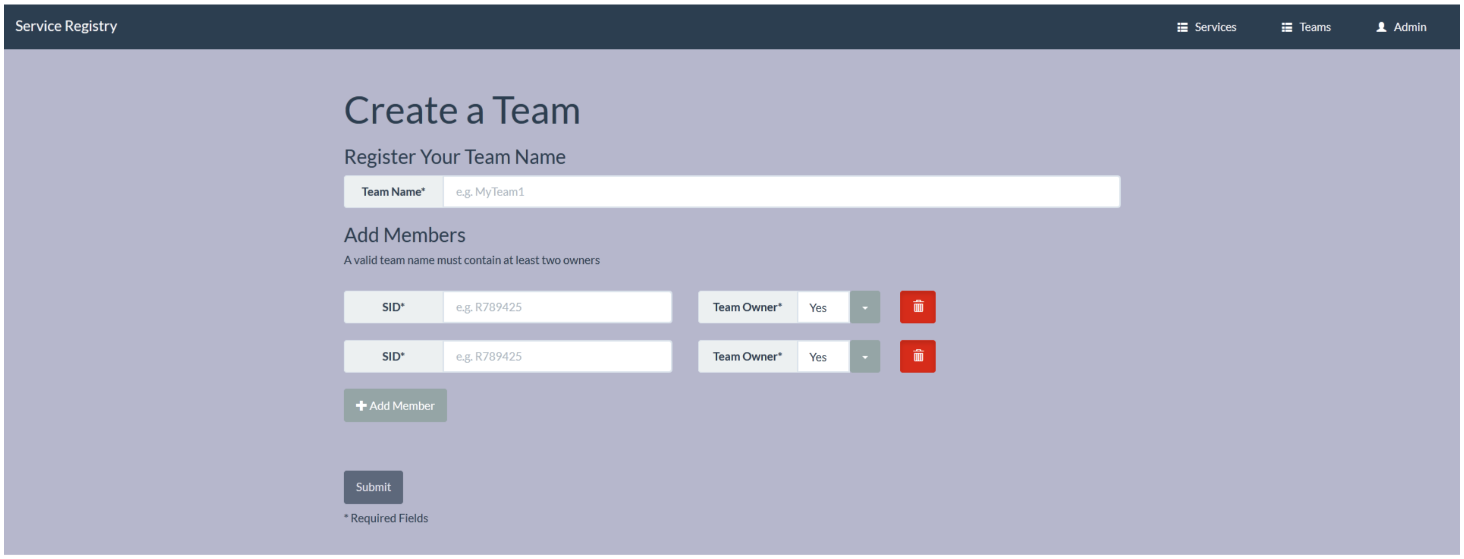
Main Info Dependencies Dependents

Service Name	Version Number	Version Description	Team	Actions
Financial Reporting Demo Service	11.2	Financial Reporting Demo Service Version 11.2	Admin Team Demo	

10 25 50 100

Appendix E: Add Team Wizard

Add Team Main View:



Service Registry

Create a Team

Register Your Team Name

Team Name* e.g. MyTeam1

Add Members

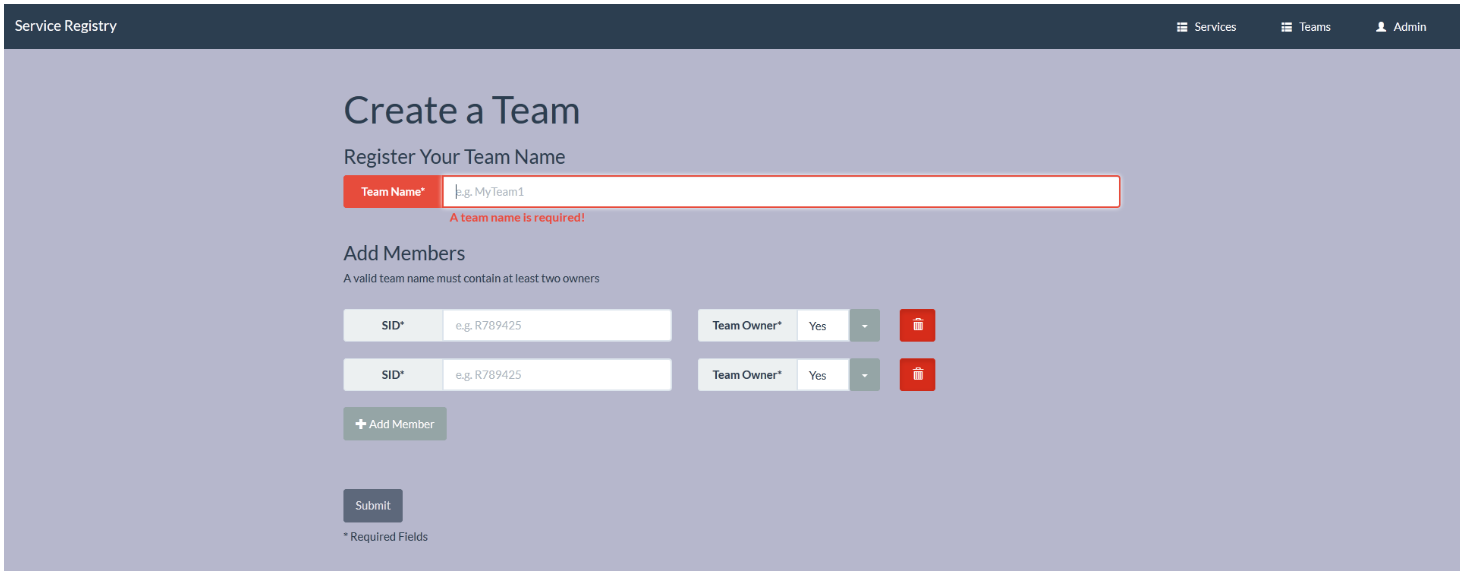
A valid team name must contain at least two owners

SID* e.g. R789425 Team Owner* Yes

SID* e.g. R789425 Team Owner* Yes

* Required Fields

Add Team With Error Checking:



Service Registry

Create a Team

Register Your Team Name

Team Name* e.g. MyTeam1

A team name is required!

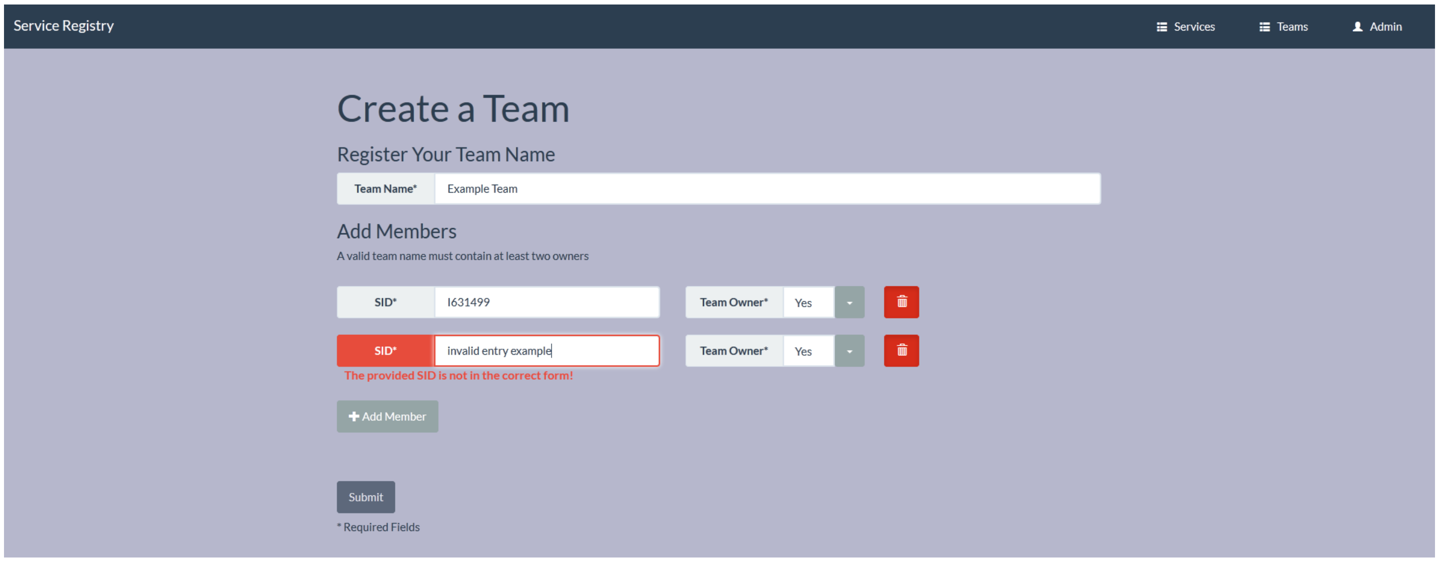
Add Members

A valid team name must contain at least two owners

SID* e.g. R789425 Team Owner* Yes

SID* e.g. R789425 Team Owner* Yes

* Required Fields



Create a Team

Register Your Team Name

Team Name*	Example Team

Add Members

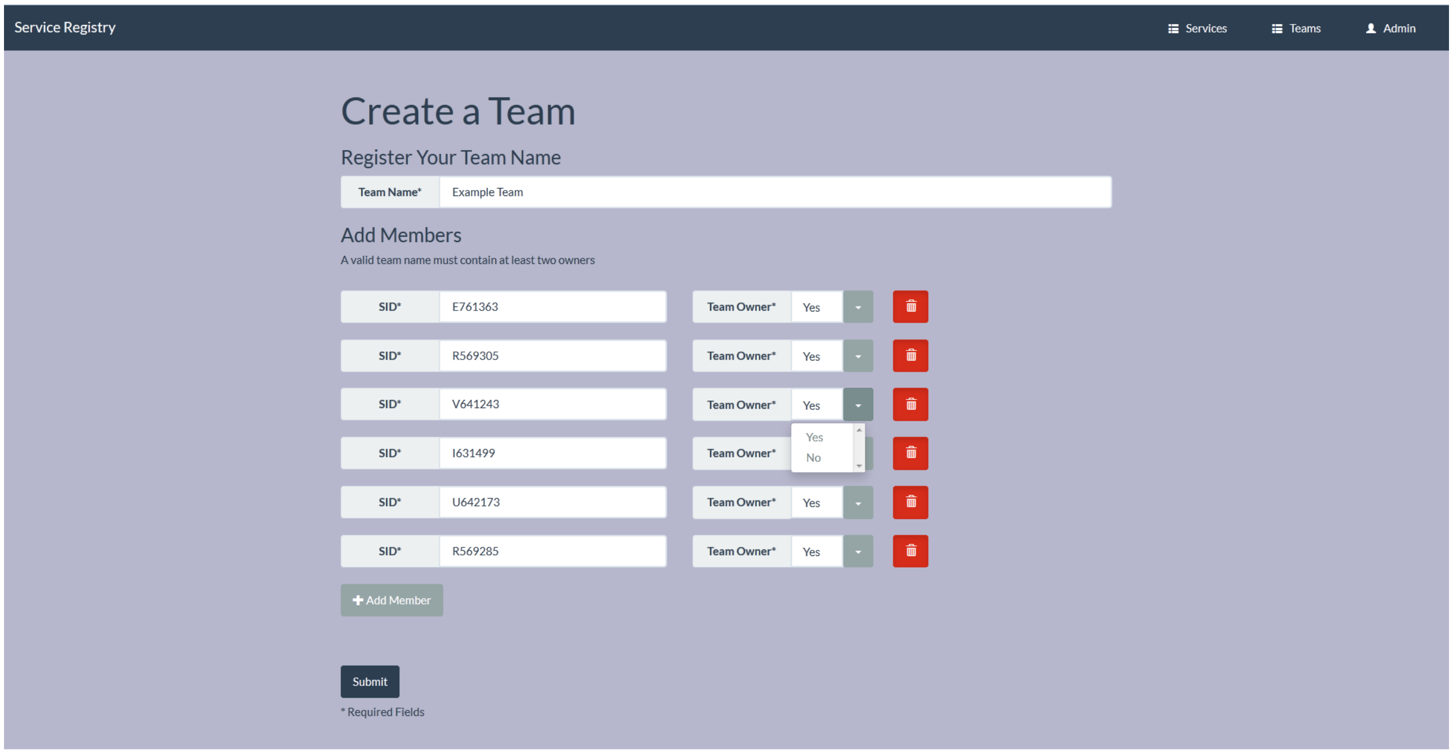
A valid team name must contain at least two owners

SID*	I631499	Team Owner*	Yes
SID*	invalid entry example	Team Owner*	Yes

The provided SID is not in the correct form!

* Required Fields

Add Team Multiple Members:



Create a Team

Register Your Team Name

Team Name*	Example Team

Add Members

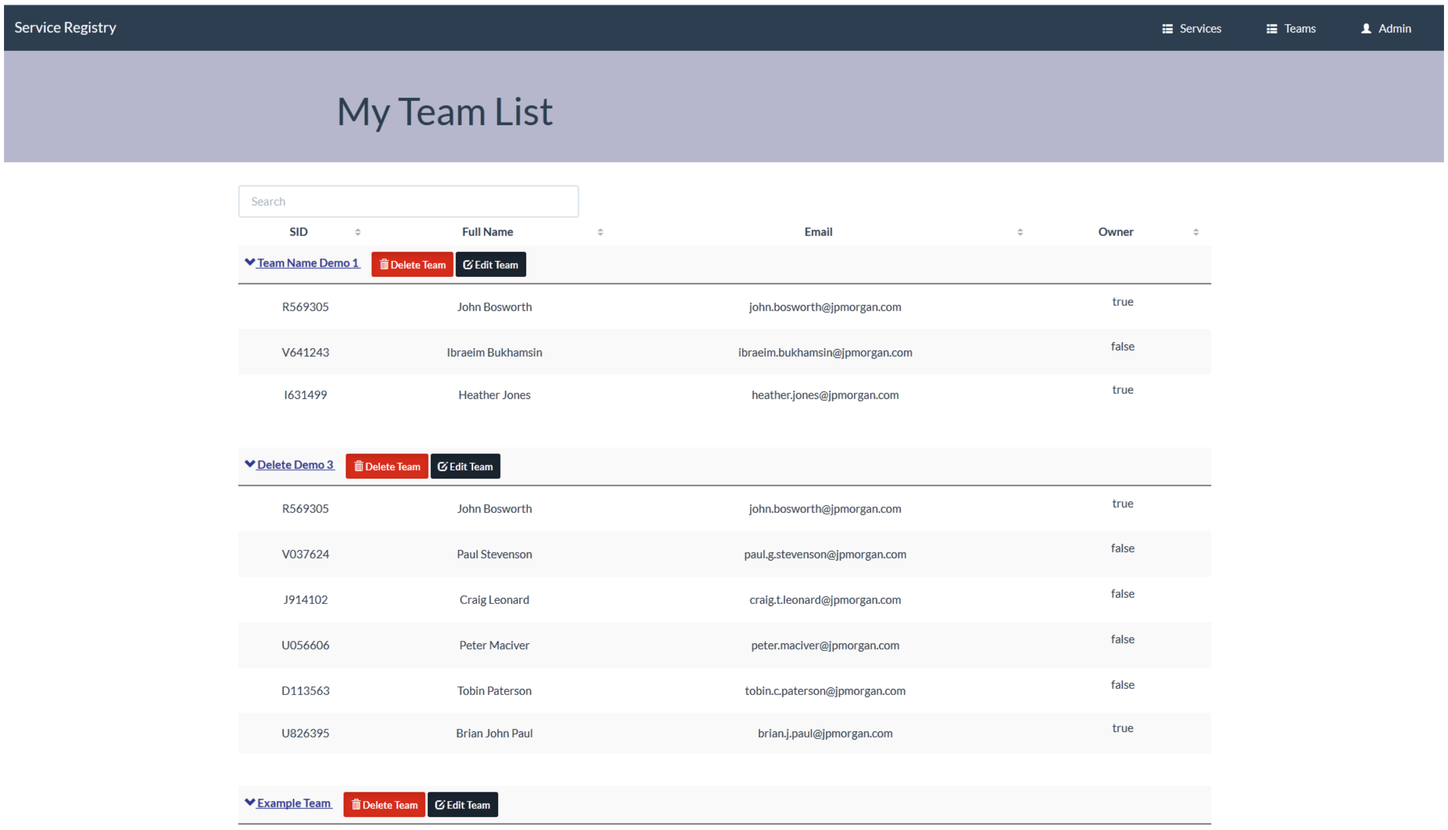
A valid team name must contain at least two owners

SID*	E761363	Team Owner*	Yes
SID*	R569305	Team Owner*	Yes
SID*	V641243	Team Owner*	Yes
SID*	I631499	Team Owner*	
SID*	U642173	Team Owner*	Yes
SID*	R569285	Team Owner*	Yes

* Required Fields

Appendix F: My Team View

My Team Main View:



Service Registry

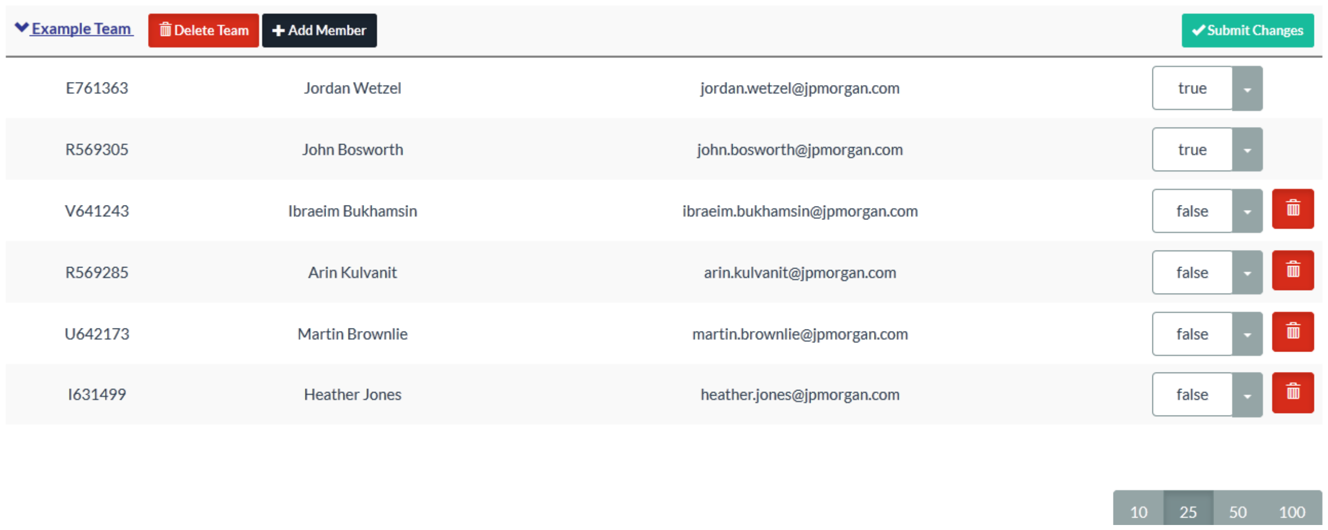
Services Teams Admin

My Team List

Search

SID	Full Name	Email	Owner
Team Name Demo 1 Delete Team Edit Team			
R569305	John Bosworth	john.bosworth@jpmorgan.com	true
V641243	Ibraeim Bukhamsin	ibraeim.bukhamsin@jpmorgan.com	false
I631499	Heather Jones	heather.jones@jpmorgan.com	true
Delete Demo 3 Delete Team Edit Team			
R569305	John Bosworth	john.bosworth@jpmorgan.com	true
V037624	Paul Stevenson	paul.g.stevenson@jpmorgan.com	false
J914102	Craig Leonard	craig.t.leonard@jpmorgan.com	false
U056606	Peter Maciver	peter.maciver@jpmorgan.com	false
D113563	Tobin Paterson	tobin.c.paterson@jpmorgan.com	false
U826395	Brian John Paul	brian.j.paul@jpmorgan.com	true
Example Team Delete Team Edit Team			

My Team Editing:



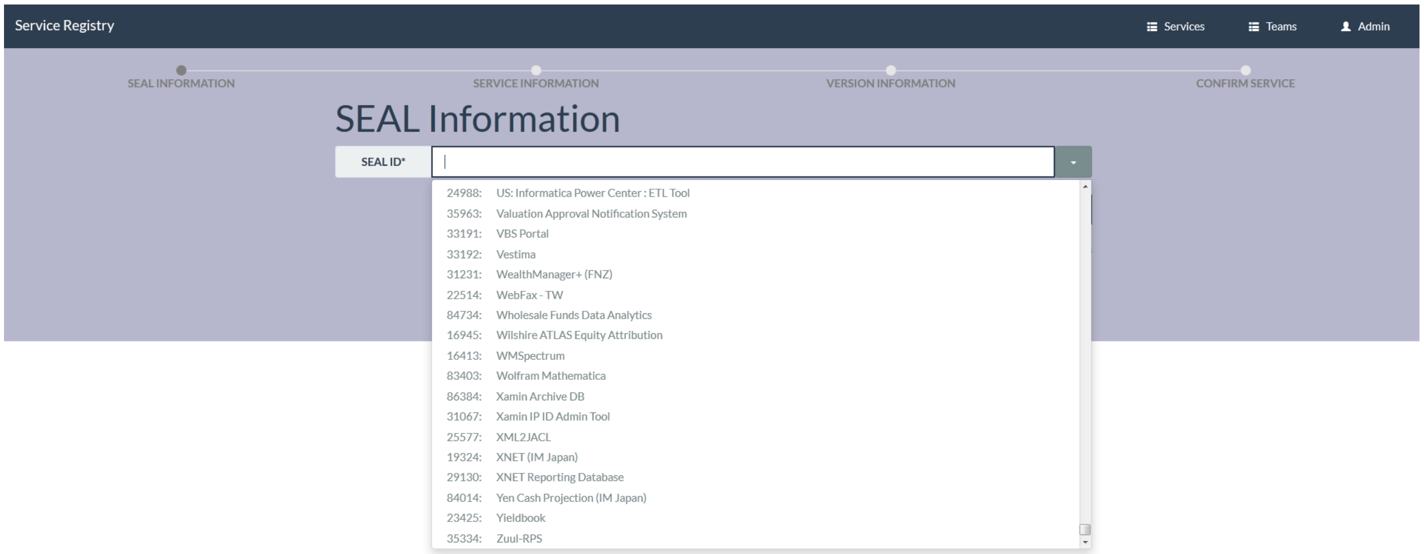
Example Team Delete Team Add Member Submit Changes

E761363	Jordan Wetzel	jordan.wetzel@jpmorgan.com	true
R569305	John Bosworth	john.bosworth@jpmorgan.com	true
V641243	Ibraeim Bukhamsin	ibraeim.bukhamsin@jpmorgan.com	false
R569285	Arin Kulvanit	arin.kulvanit@jpmorgan.com	false
U642173	Martin Brownlie	martin.brownlie@jpmorgan.com	false
I631499	Heather Jones	heather.jones@jpmorgan.com	false

10 25 50 100

Appendix G: Add Service Wizard

Add Service SealID:



Service Registry

Services   Teams   Admin

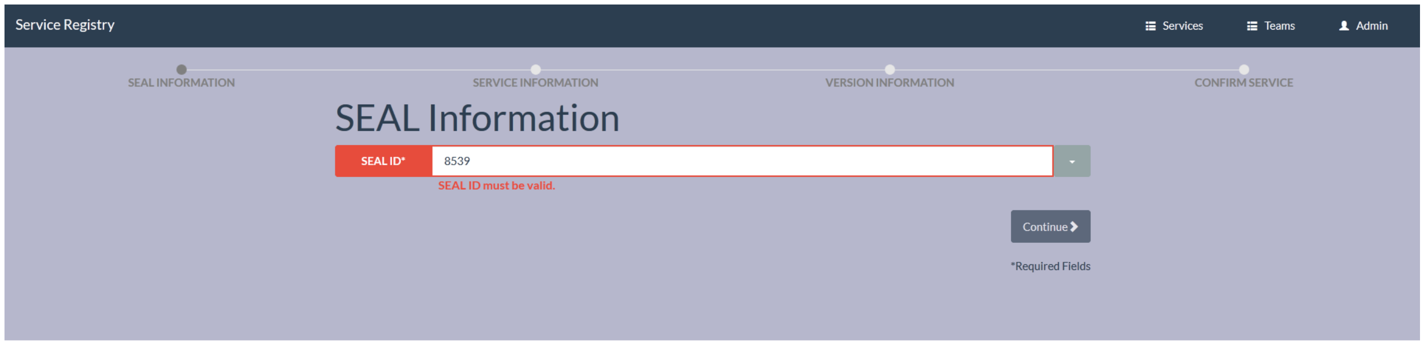
SEAL INFORMATION   SERVICE INFORMATION   VERSION INFORMATION   CONFIRM SERVICE

SEAL Information

SEAL ID*

24988:	US: Informatica Power Center : ETL Tool
35963:	Valuation Approval Notification System
33191:	VBS Portal
33192:	Vestima
31231:	WealthManager+ (FNZ)
22514:	WebFax - TW
84734:	Wholesale Funds Data Analytics
16945:	Wilshire ATLAS Equity Attribution
16413:	WMSpectrum
83403:	Wolfram Mathematica
86384:	Xamin Archive DB
31067:	Xamin IP ID Admin Tool
25577:	XML2JACL
19324:	XNET (IM Japan)
29130:	XNET Reporting Database
84014:	Yen Cash Projection (IM Japan)
23425:	Yieldbook
35334:	Zuul-RPS

Add Service SealID Error Checking:



Service Registry

Services   Teams   Admin

SEAL INFORMATION   SERVICE INFORMATION   VERSION INFORMATION   CONFIRM SERVICE

SEAL Information

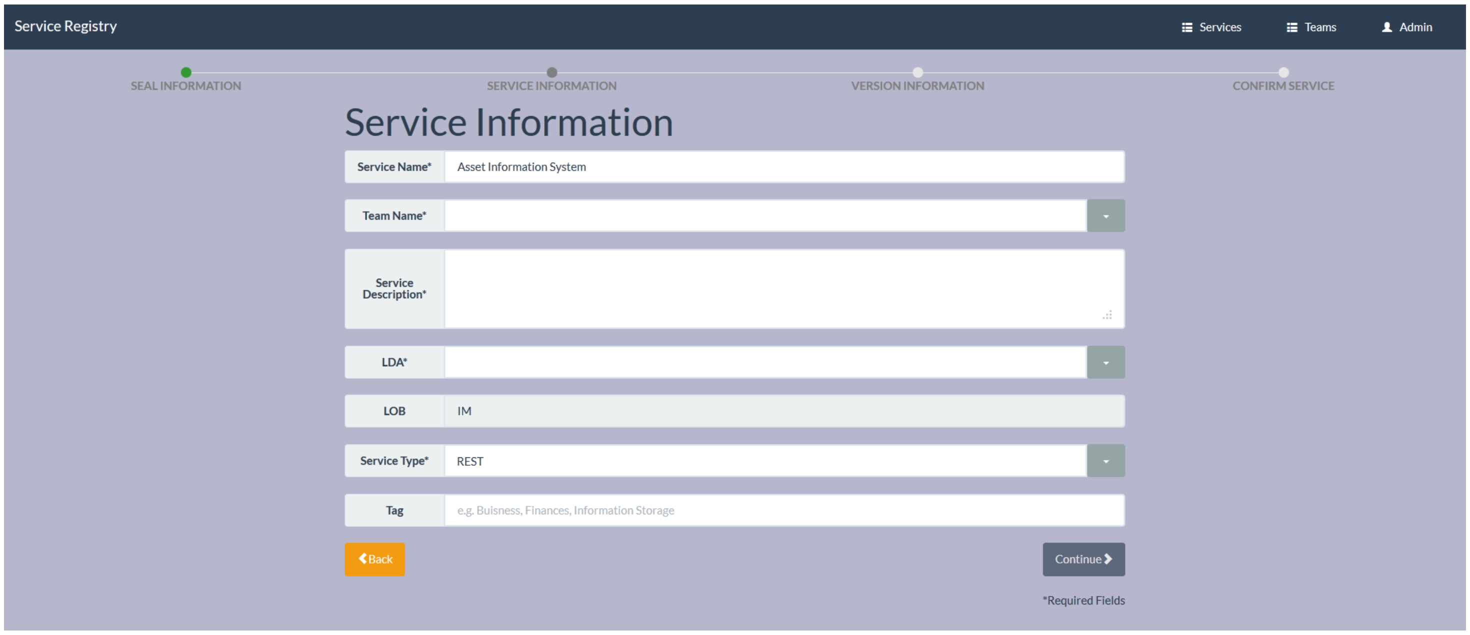
SEAL ID*   8539

SEAL ID must be valid.

Continue

*Required Fields

Add Service General Information:

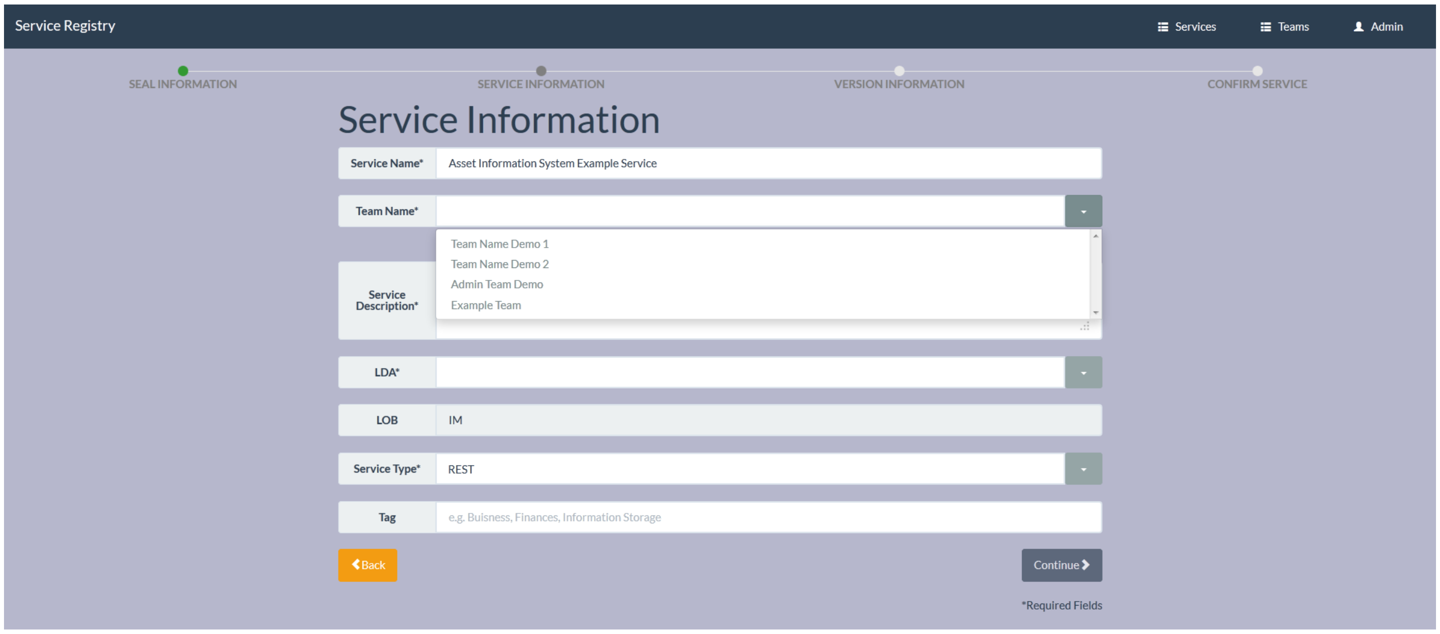


Service Information

Service Name*	Asset Information System
Team Name*	
Service Description*	
LDA*	
LOB	IM
Service Type*	REST
Tag	e.g. Buisness, Finances, Information Storage

*Required Fields

Add Service Drop Down of Available Teams:



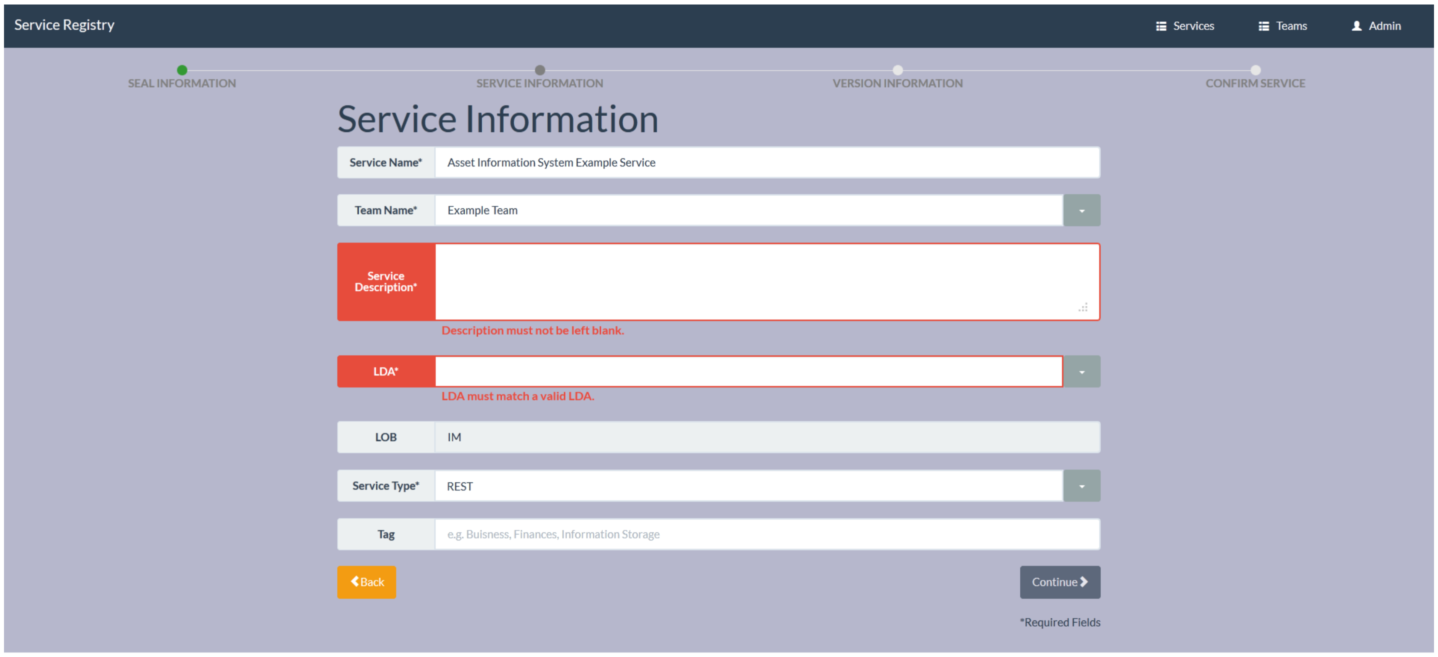
Service Information

Service Name*	Asset Information System Example Service
Team Name*	
Service Description*	
LDA*	
LOB	IM
Service Type*	REST
Tag	e.g. Buisness, Finances, Information Storage

Team Name Demo 1
Team Name Demo 2
Admin Team Demo
Example Team

*Required Fields

Add Service Error Checking:



Service Information

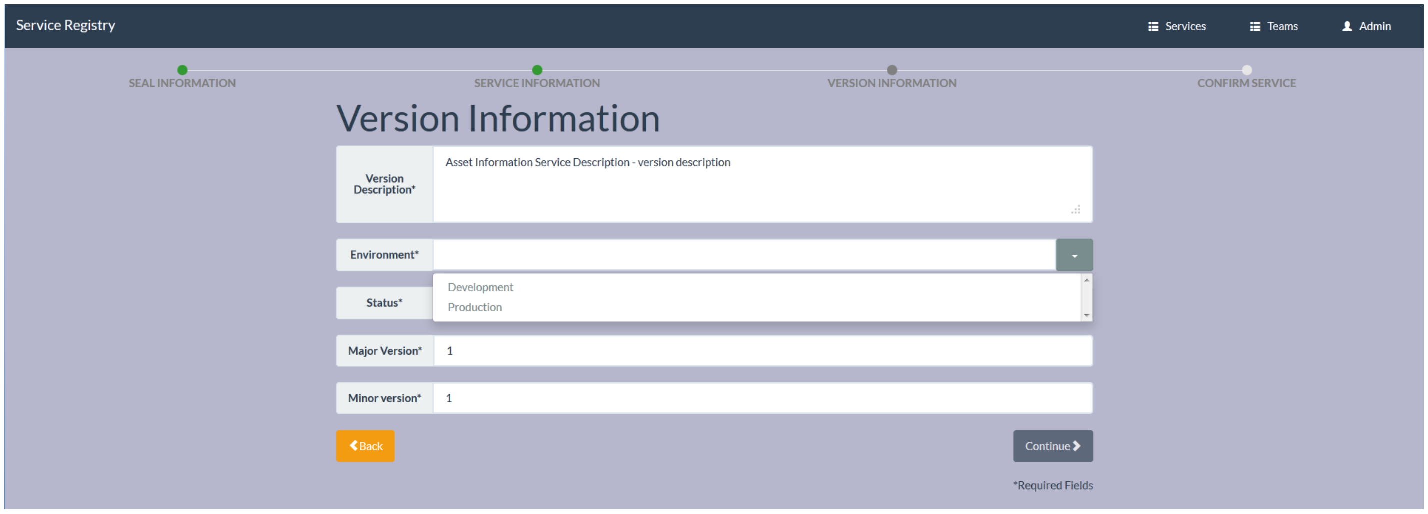
Service Name*	Asset Information System Example Service
Team Name*	Example Team
Service Description*	
LDA*	
LOB	IM
Service Type*	REST
Tag	e.g. Buisness, Finances, Information Storage

Description must not be left blank.

LDA must match a valid LDA.

*Required Fields

Add Service Version:

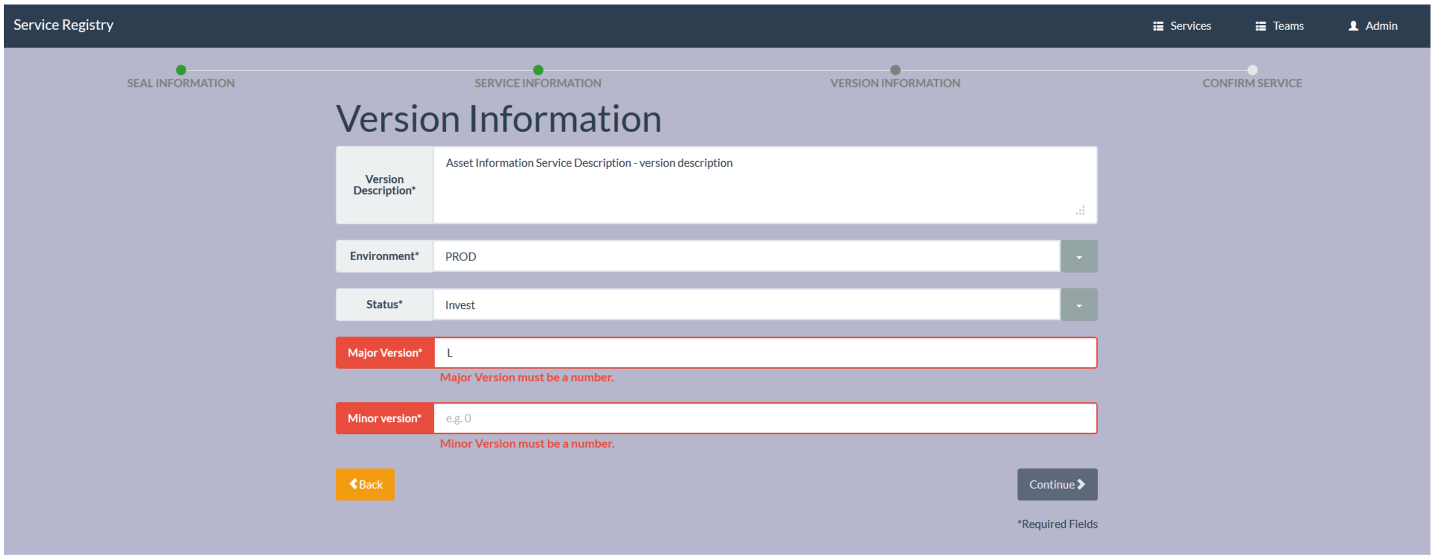


Version Information

Version Description*	Asset Information Service Description - version description
Environment*	
Status*	
Major Version*	1
Minor version*	1

*Required Fields

Add Service Version Error Checking:

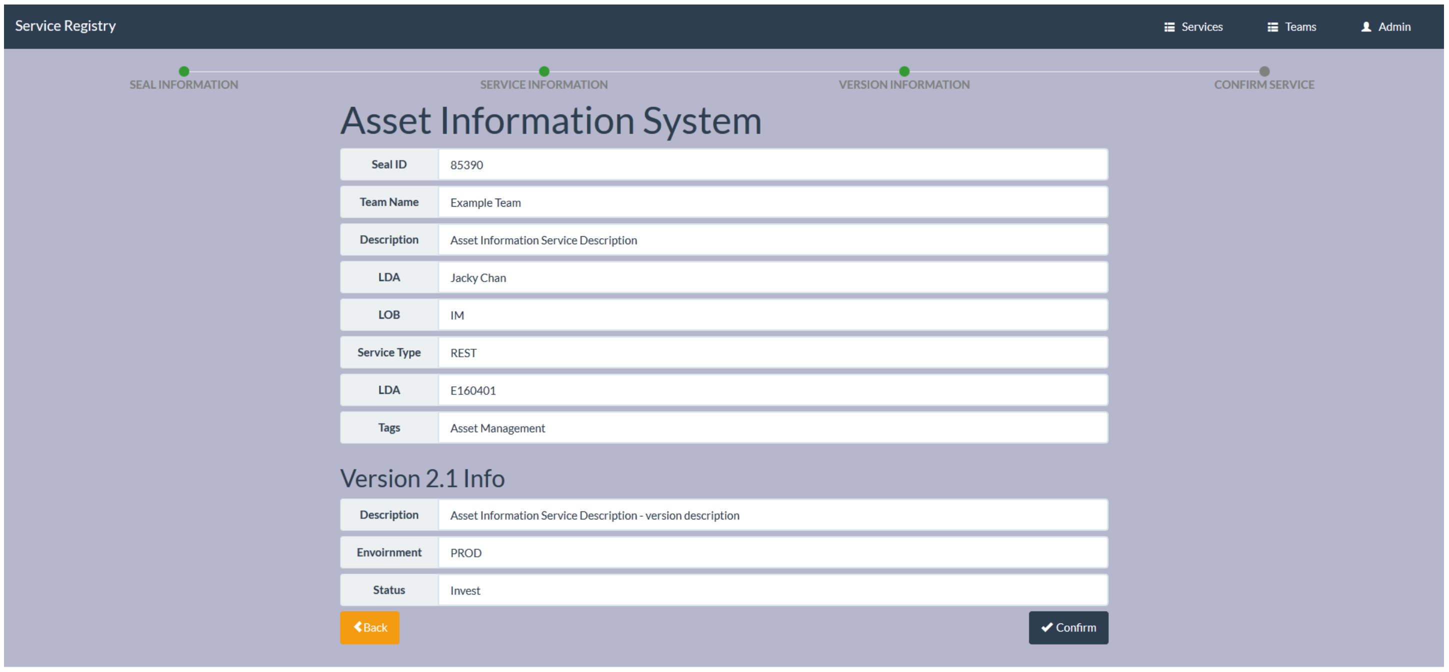


Version Information

Version Description*	Asset Information Service Description - version description
Environment*	PROD
Status*	Invest
Major Version*	L
	Major Version must be a number.
Minor version*	e.g. 0
	Minor Version must be a number.

*Required Fields

Add Service Conformation Page:



Asset Information System

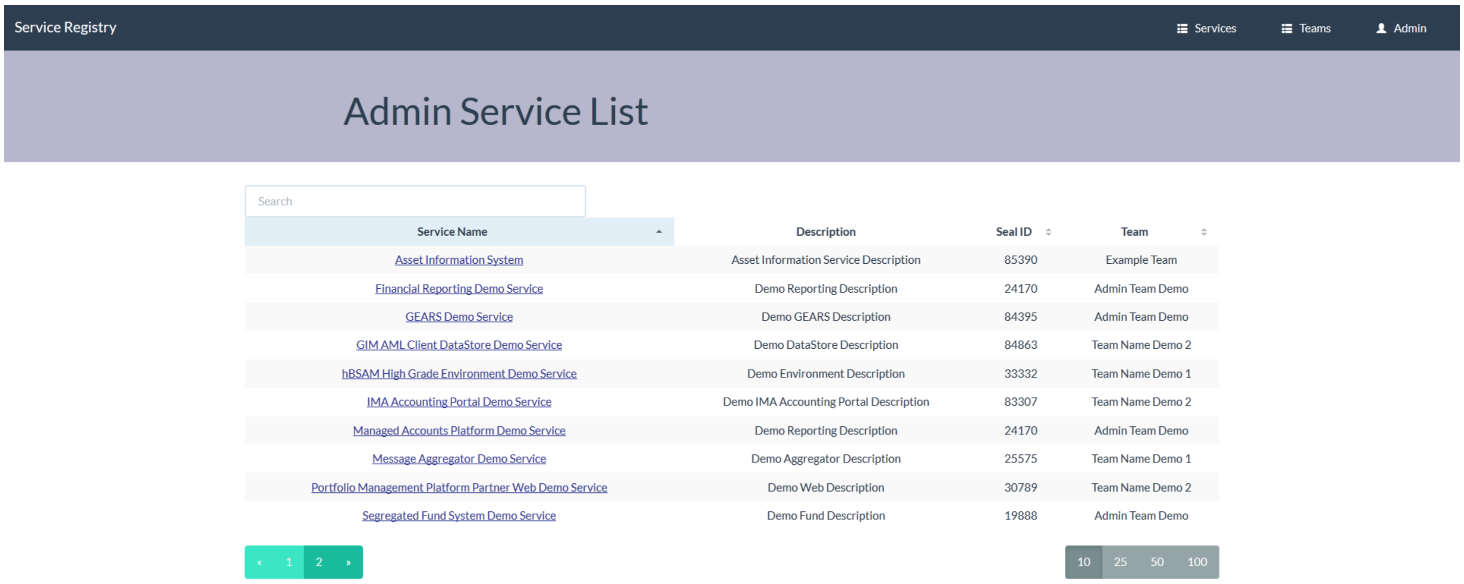
Seal ID	85390
Team Name	Example Team
Description	Asset Information Service Description
LDA	Jacky Chan
LOB	IM
Service Type	REST
LDA	E160401
Tags	Asset Management

Version 2.1 Info

Description	Asset Information Service Description - version description
Envoirnment	PROD
Status	Invest

Appendix H: Administrator View

Admin Services View:

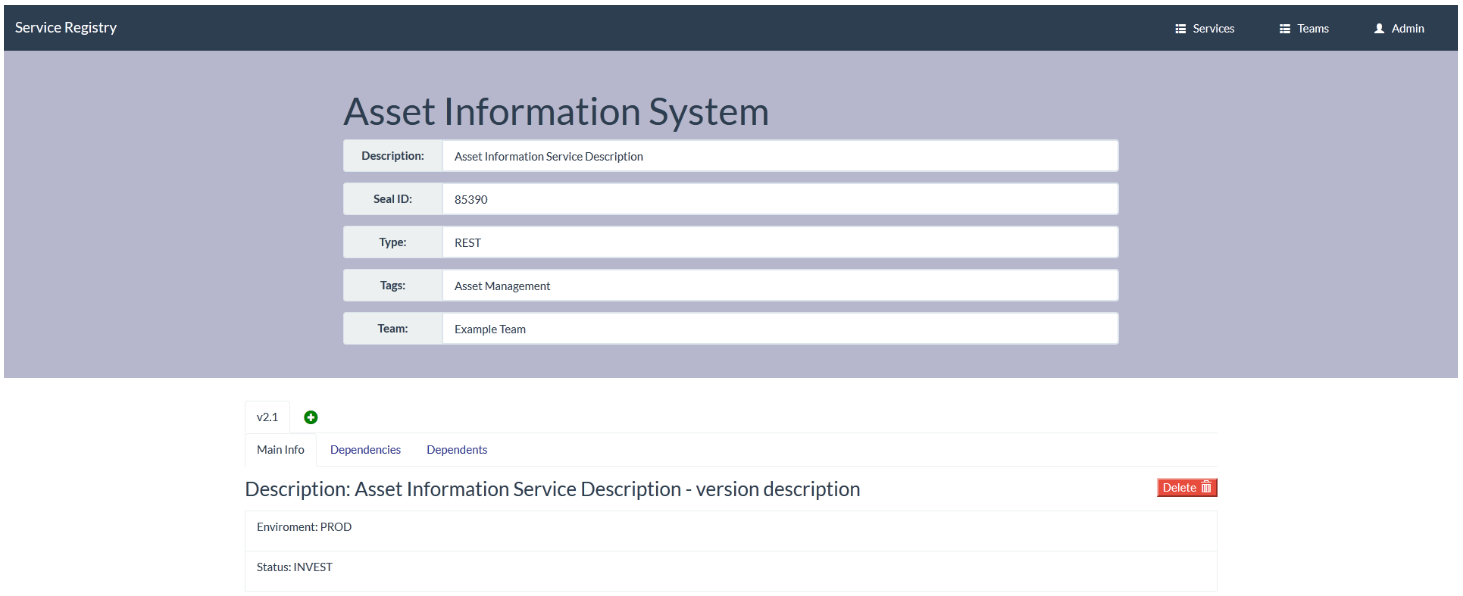


Admin Service List

Search

Service Name	Description	Seal ID	Team
Asset Information System	Asset Information Service Description	85390	Example Team
Financial Reporting Demo Service	Demo Reporting Description	24170	Admin Team Demo
GEARS Demo Service	Demo GEARS Description	84395	Admin Team Demo
GIM AML Client DataStore Demo Service	Demo DataStore Description	84863	Team Name Demo 2
hBSAM High Grade Environment Demo Service	Demo Environment Description	33332	Team Name Demo 1
IMA Accounting Portal Demo Service	Demo IMA Accounting Portal Description	83307	Team Name Demo 2
Managed Accounts Platform Demo Service	Demo Reporting Description	24170	Admin Team Demo
Message Aggregator Demo Service	Demo Aggregator Description	25575	Team Name Demo 1
Portfolio Management Platform Partner Web Demo Service	Demo Web Description	30789	Team Name Demo 2
Segregated Fund System Demo Service	Demo Fund Description	19888	Admin Team Demo

Admin Service View:



Asset Information System

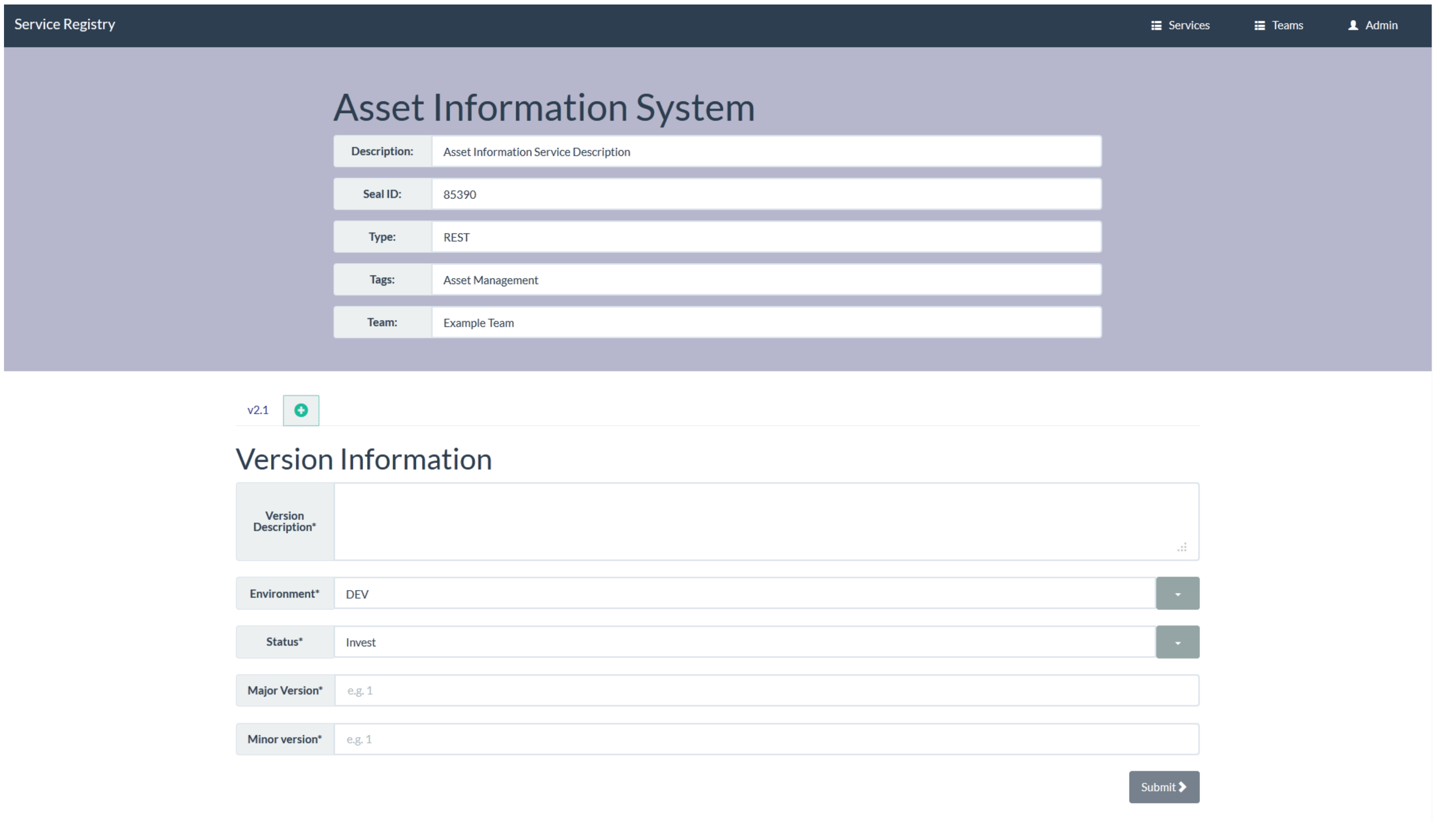
Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

Description: Asset Information Service Description - version description

Enviroment: PROD

Status: INVEST

Admin Add Version:



Asset Information System

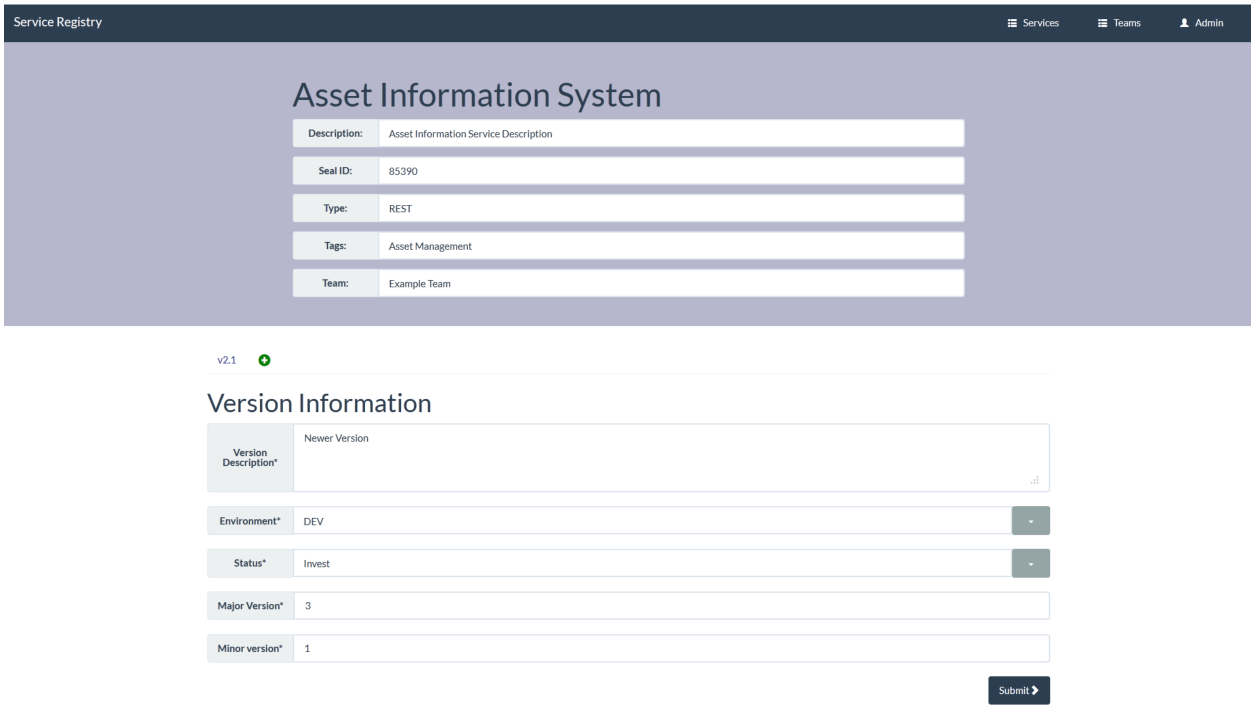
Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

v2.1

Version Information

Version Description*	
Environment*	DEV
Status*	Invest
Major Version*	e.g. 1
Minor version*	e.g. 1

Admin Add Version Information:



Asset Information System

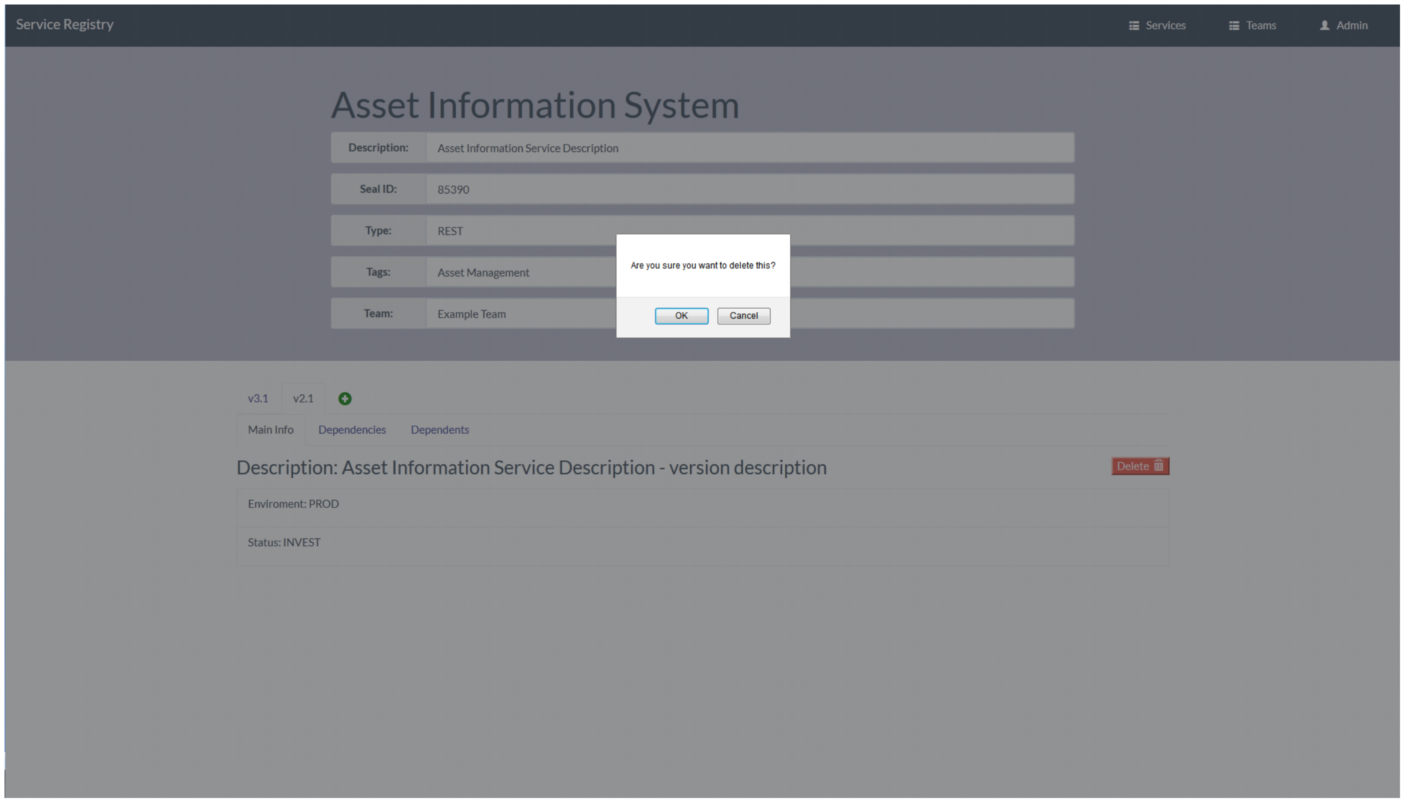
Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

v2.1

Version Information

Version Description*	Newer Version
Environment*	DEV
Status*	Invest
Major Version*	3
Minor version*	1

Admin View Deleting a Version:



Asset Information System

Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

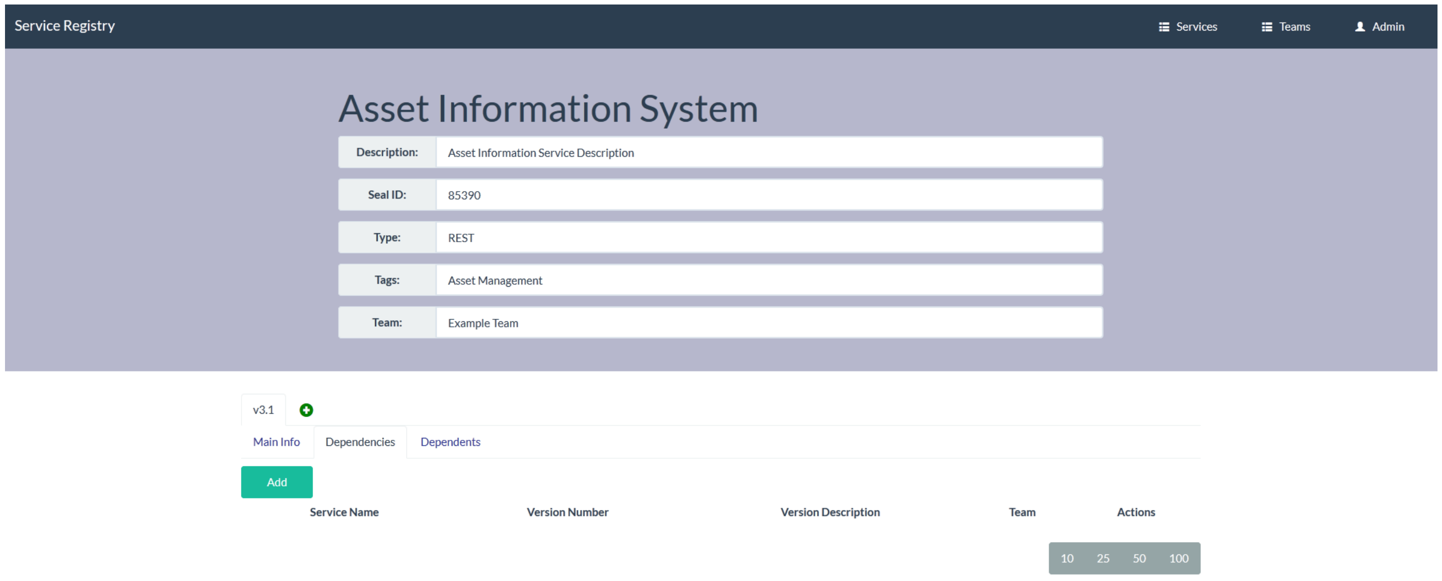
Are you sure you want to delete this?

Description: Asset Information Service Description - version description

Enviroment: PROD

Status: INVEST

Admin View Add Dependency:

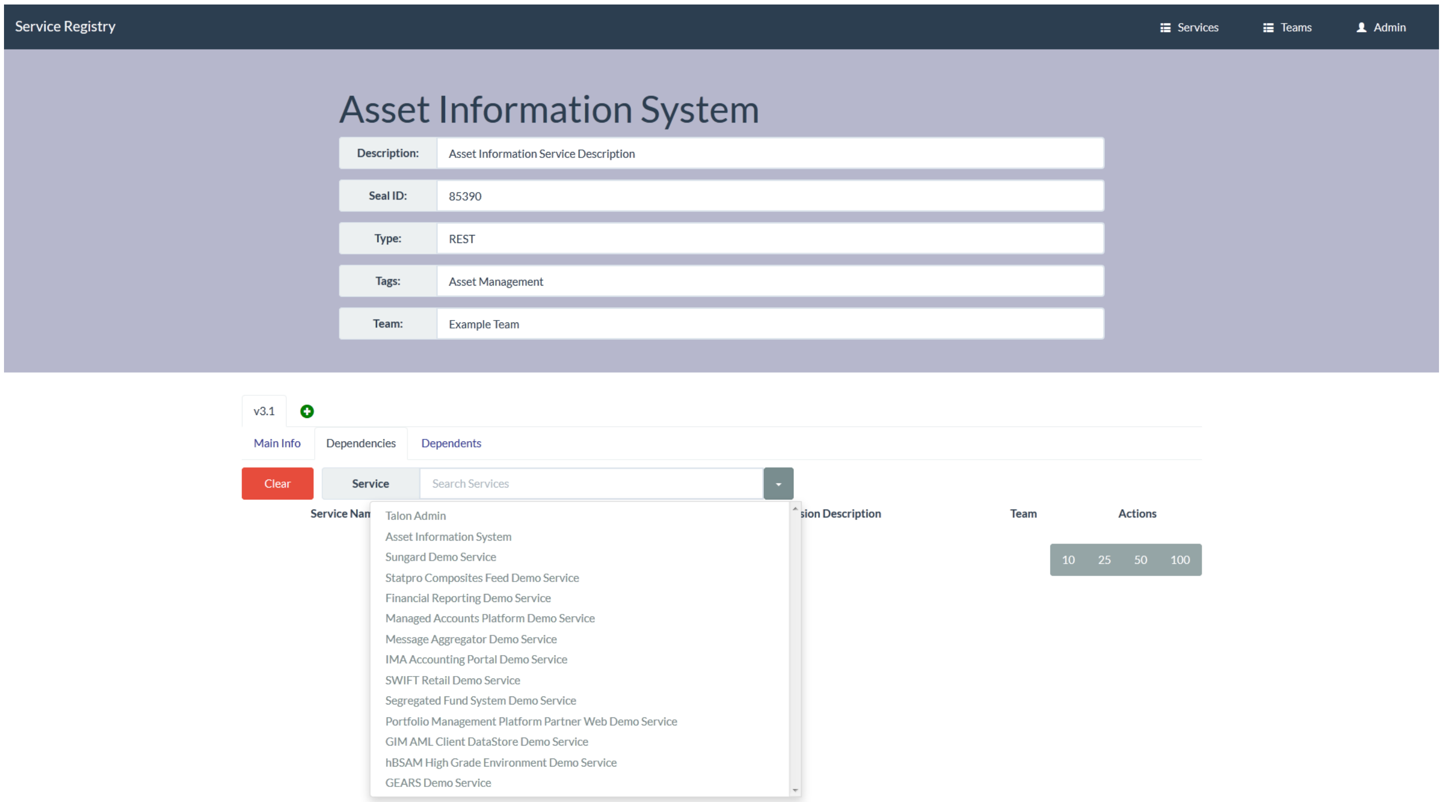


Asset Information System

Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

Service Name	Version Number	Version Description	Team	Actions

Admin View Add Dependency Select Service:

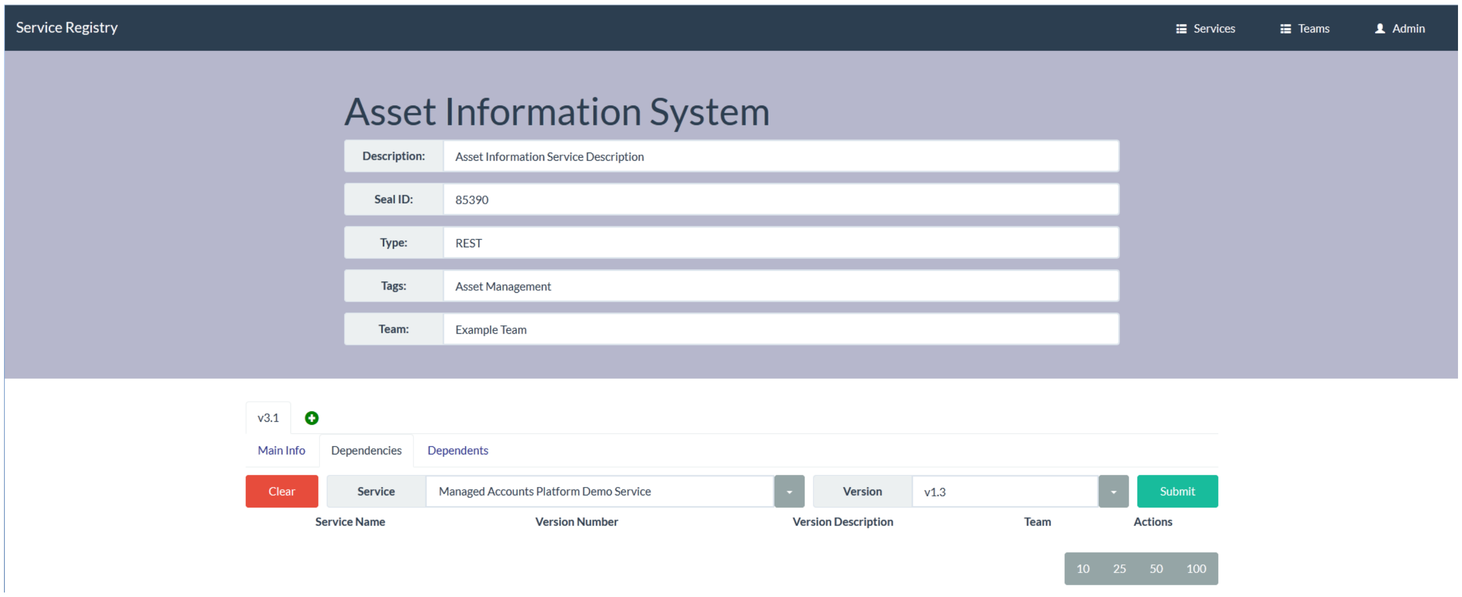


Asset Information System

Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

Talon Admin
Asset Information System
Sungard Demo Service
Statpro Composites Feed Demo Service
Financial Reporting Demo Service
Managed Accounts Platform Demo Service
Message Aggregator Demo Service
IMA Accounting Portal Demo Service
SWIFT Retail Demo Service
Segregated Fund System Demo Service
Portfolio Management Platform Partner Web Demo Service
GIM AML Client DataStore Demo Service
hBSAM High Grade Environment Demo Service
GEARS Demo Service

Admin View Add Dependency Select Version and Submit:



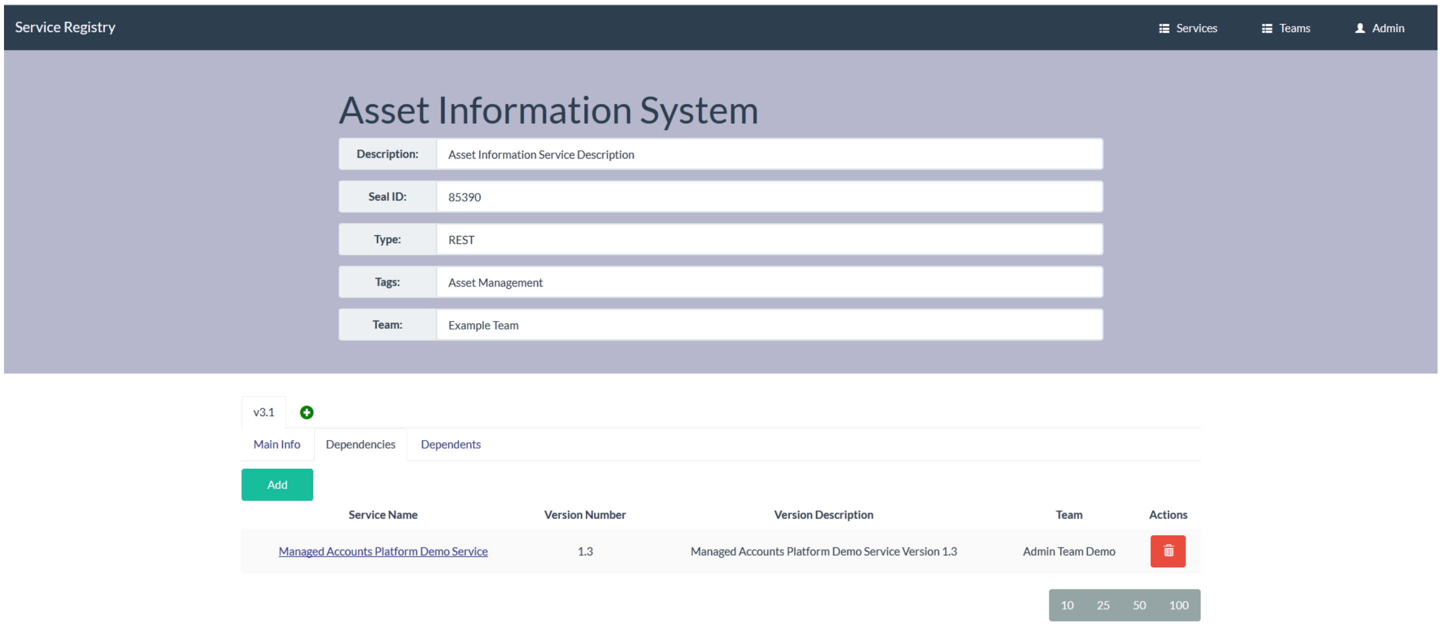
Asset Information System

Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

Service: Managed Accounts Platform Demo Service
Version: v1.3

Service Name	Version Number	Version Description	Team	Actions

Admin View with New Dependency:



Service Registry

Services | Teams | Admin

Asset Information System

Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

v3.1

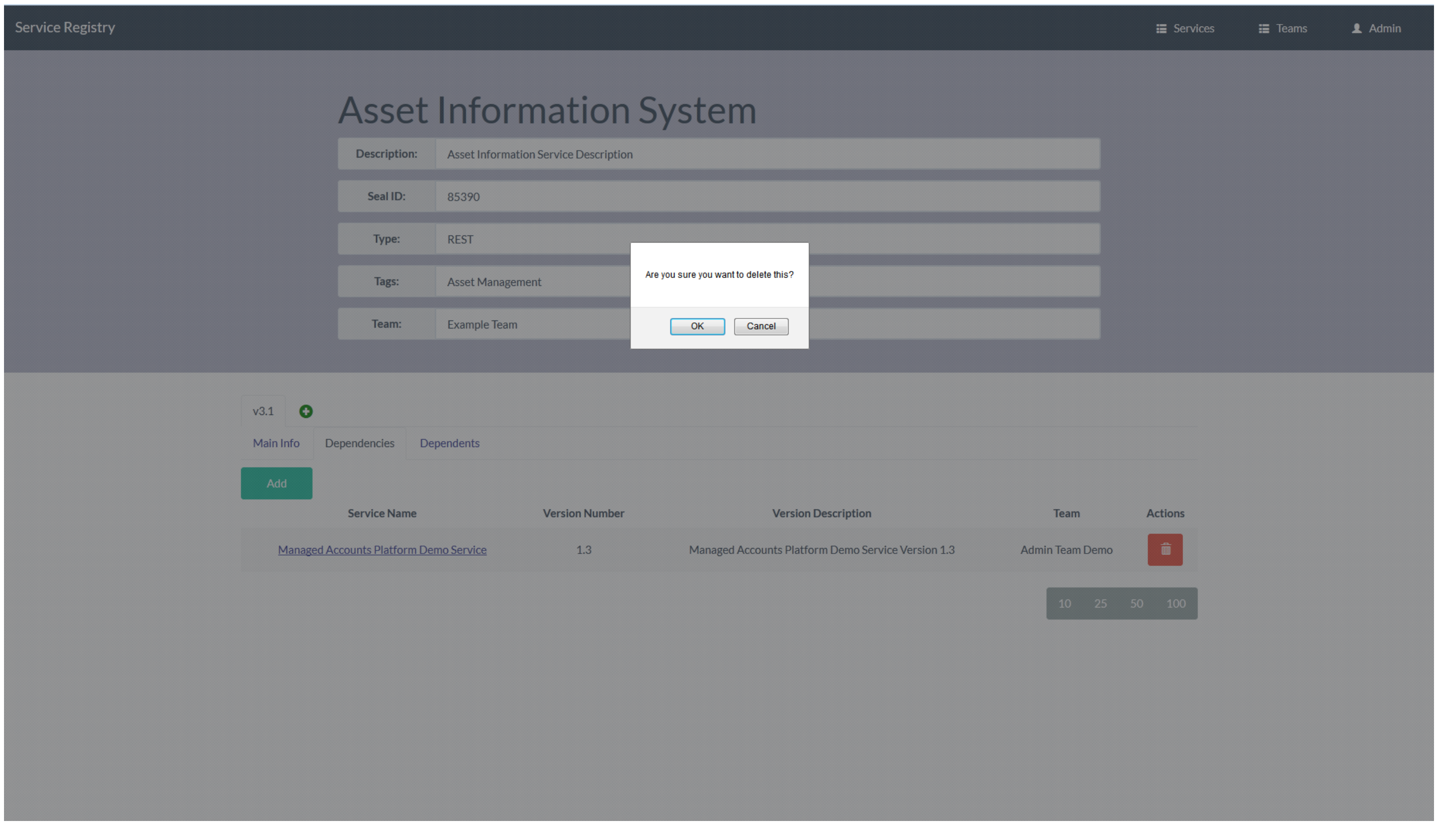
Main Info | Dependencies | Dependents

Add

Service Name	Version Number	Version Description	Team	Actions
Managed Accounts Platform Demo Service	1.3	Managed Accounts Platform Demo Service Version 1.3	Admin Team Demo	

10 25 50 100

Admin View Deleting a dependency:



Service Registry

Services | Teams | Admin

Asset Information System

Description:	Asset Information Service Description
Seal ID:	85390
Type:	REST
Tags:	Asset Management
Team:	Example Team

Are you sure you want to delete this?

OK Cancel

v3.1

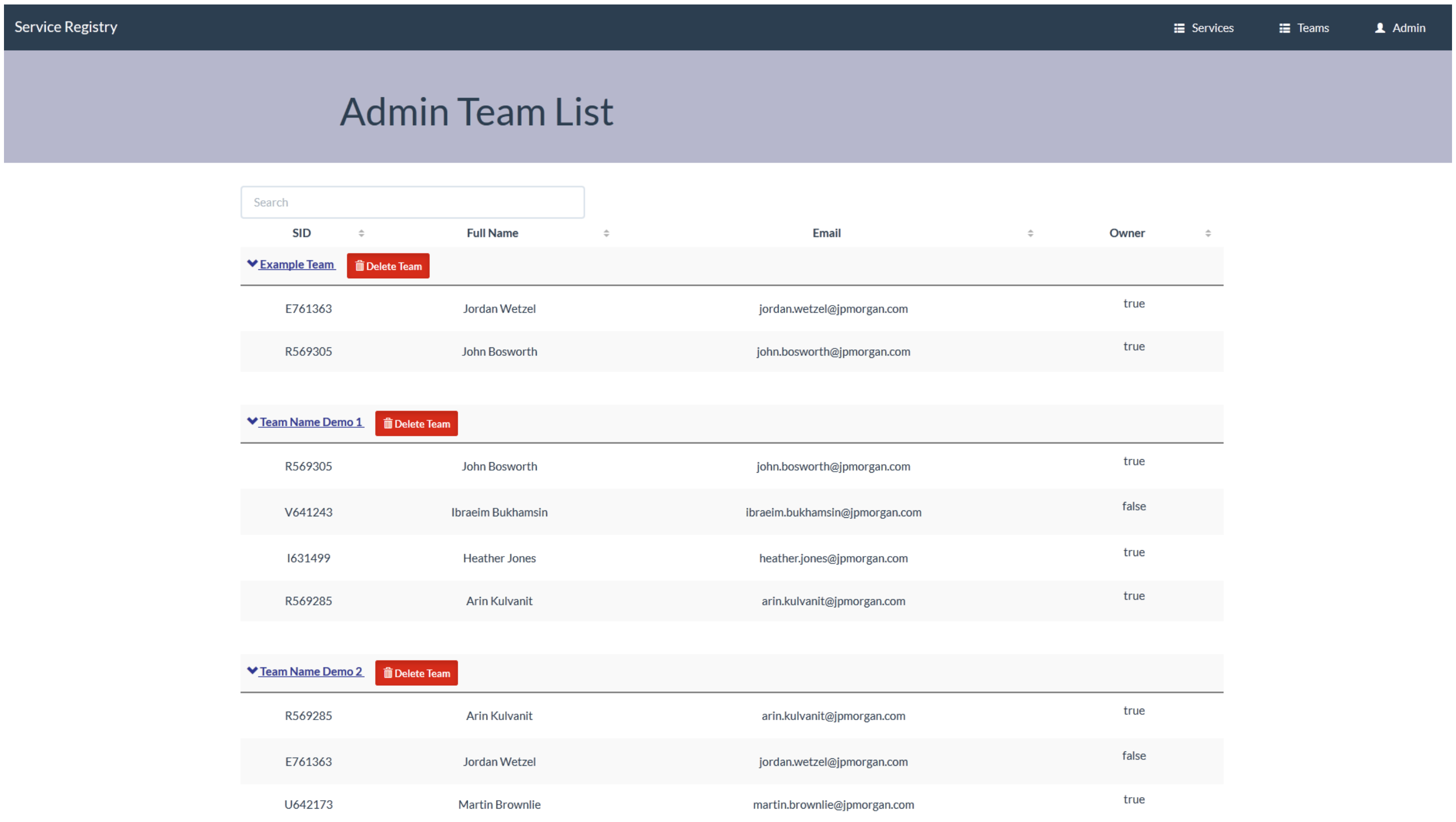
Main Info | Dependencies | Dependents

Add

Service Name	Version Number	Version Description	Team	Actions
Managed Accounts Platform Demo Service	1.3	Managed Accounts Platform Demo Service Version 1.3	Admin Team Demo	

10 25 50 100

Admin Team View:



Service Registry

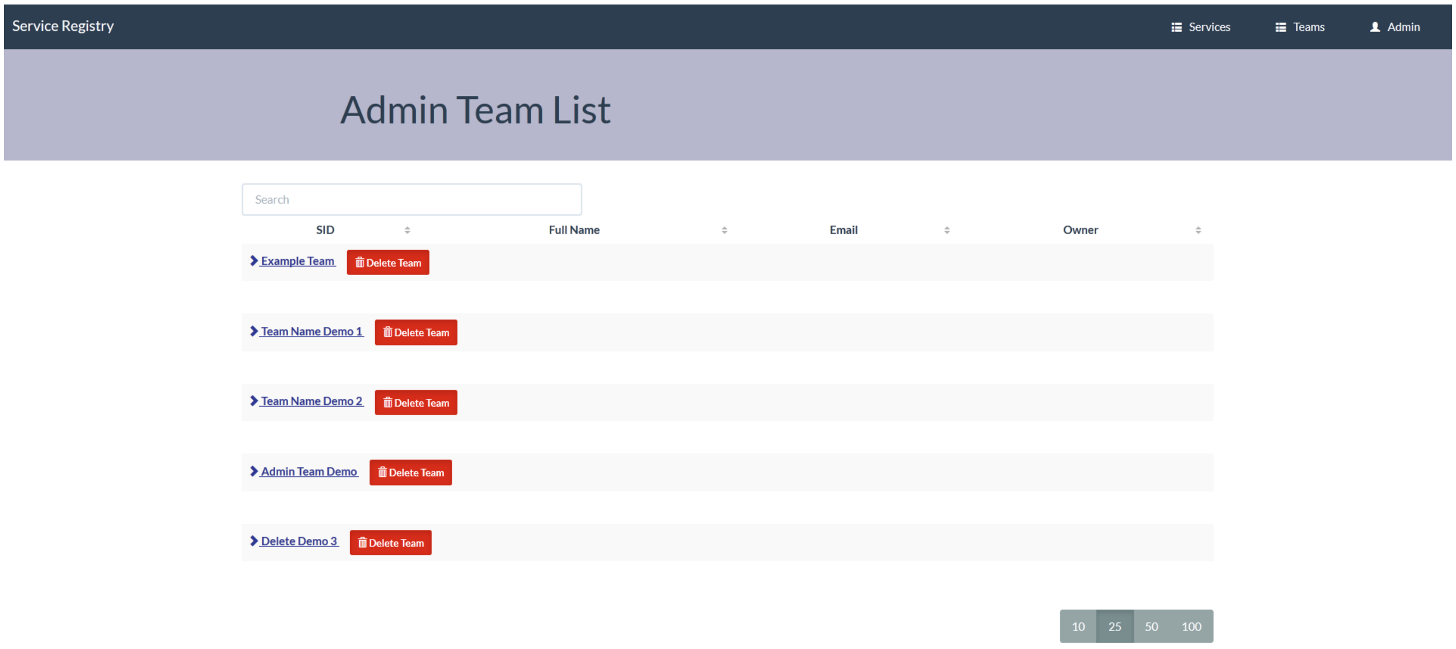
Services | Teams | Admin

Admin Team List

Search

SID	Full Name	Email	Owner
Example Team (Delete Team)			
E761363	Jordan Wetzel	jordan.wetzel@jpmorgan.com	true
R569305	John Bosworth	john.bosworth@jpmorgan.com	true
Team Name Demo 1 (Delete Team)			
R569305	John Bosworth	john.bosworth@jpmorgan.com	true
V641243	Ibraeim Bukhamsin	ibraeim.bukhamsin@jpmorgan.com	false
I631499	Heather Jones	heather.jones@jpmorgan.com	true
R569285	Arin Kulvanit	arin.kulvanit@jpmorgan.com	true
Team Name Demo 2 (Delete Team)			
R569285	Arin Kulvanit	arin.kulvanit@jpmorgan.com	true
E761363	Jordan Wetzel	jordan.wetzel@jpmorgan.com	false
U642173	Martin Brownlie	martin.brownlie@jpmorgan.com	true

Admin Team View Collapsed:



Service Registry

Services | Teams | Admin

Admin Team List

Search

SID	Full Name	Email	Owner
Example Team (Delete Team)			
Team Name Demo 1 (Delete Team)			
Team Name Demo 2 (Delete Team)			
Admin Team Demo (Delete Team)			
Delete Demo 3 (Delete Team)			