Project Report

|  |  |
| --- | --- |
| Project Title | Design and develop a website portal using Liferay |
| Qualification Name (NICF) | Advanced Certificate in Web Development using Platforms |
| Product Name | Triple – A (AAA) Hosting |
| Module Name (NICF) | Development using Platforms |

|  |  |  |  |
| --- | --- | --- | --- |
| Student name | | Assessor name | |
| Wildan Luqmanul Hakim | |  | |
| Date issued | Completion date | | Submitted on |
| 10 November 2022 | 23 November 2022 | | 23 November 2022 |
|  | |  | |
| Project title | Design and develop a website portal using Liferay | | |

|  |
| --- |
| Learner declaration |
| I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.    Student signature: |

Content

1. Project background
2. Project Objectives
3. Project Requirement Specifications
4. Task 1
5. Task 2
6. Task 3
7. Task 4
8. Task 5
9. Project Background

AAA Website Hosting always guarantees 99.9% server uptime quality, 24-hour CS support, and provides the lowest prices in its class. We are also committed to always fulfilling various customer needs, both professional and personal.

AAA Website Hosting mission is to help millions of people take advantage of the potential that exists on the internet to build online success through stable, fast and easy web hosting services. It provides all types of services which include domain name registration, Shared Hosting, Reseller Hosting, Cloud Hosting, VPS Hosting,  
Dedicated Hosting, and Colocation service, etc.

1. Project Objective

This Project is used for Summative Assessment of student in the Module ‘Web  
Development using Platforms’ of the NICF Course “Applied Degree in Software  
Engineering” This Project considers the skills required to Design, Implement, Test & Document a website for a Used Car Sales Portal using Spring Framework, MySQL server and test the system by adopting Risk Based Testing (RBT).

Tools & platform used

1. Microsoft Word
2. Microsoft Power Point
3. Liferay Developer Studio
4. MySQL Workbench
5. Web Browser Google Chrome
6. Diagram.net
7. Project Requirements Specifications

**3.1 Project Scope**

**There are two types of users in this portal. They are**

1. Administrator

2. Site Member (Staff)

**Administrator** **should be able to perform following functions in the portal,**

1. Manage all of the site contents and pages.

2. Update the theme and layout.

3. Manage the customer data portlet and control permission.

4. Manage all user (Site Member and Customer) roles and permission

**Site Member** **should be able to perform following functions in the portal,**

1. Update AAA company site contents.

2. Add and Update the AAA’s customer data.

**3.2 Functional Requirement**

**The AAA website consists of the following Key pages**

1. Home Page

2. Registration Page

3. Login Page

4. Our services Page

a. Domain name service page

b. Shared Hosting service page

c. Reseller Hosting service page

d. Cloud Hosting service page

e. VPS Hosting service page

f. Dedicated Hosting service page

g. Colocation service page

5. Contact us Page

6. About us Page

7. Terms and Conditions Page

1. Task 1

Task Statement:

Create the following items Under “Program Paradigm and Design Pattern” in Project Presentation

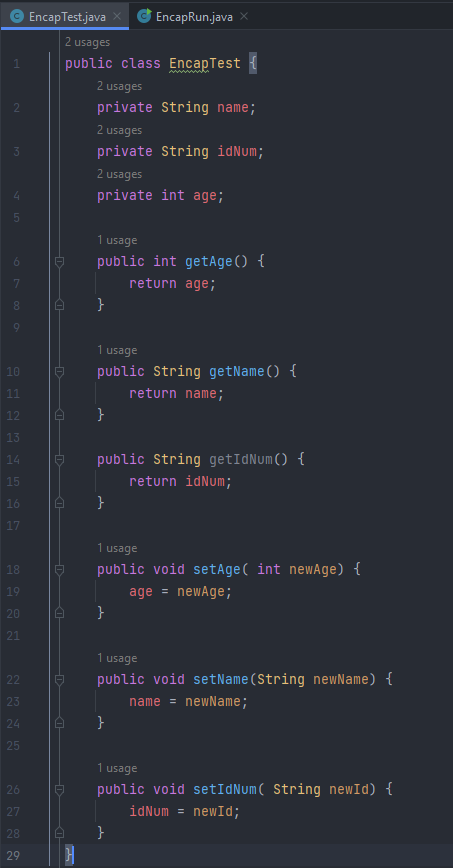
1. Identify and explain the characteristics of the object-orientated paradigm and relationship between the various classes from a given code scenario.
2. Identity the suitable design patterns base on the project scenario.
3. Examine how the object-orientated paradigm and its key principle is identified in each of the design patterns.

Solution:

1. Explain briefly Object-Oriented Programming Paradigm

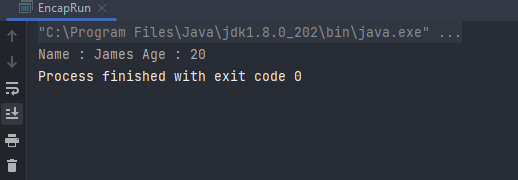
Object-oriented programming (OOP) is a style of programming characterized by the identification of classes of objects closely linked to the methods (functions) with which they are associated. It also includes ideas of inheritance of attributes and methods. It is a technique based on a mathematical discipline, called "abstract data types," for storing data with the procedures needed to process that data. OOP offers the potential to evolve programming to a higher level of abstraction.

1. Examine characteristics of Object-Oriented Programming Paradigm
   * 1. **Encapsulation**  
        Encapsulation is the wrapping of data or hiding personal data from an object so that it cannot be accessed from other objects.

Example:   
EncapTest.java  


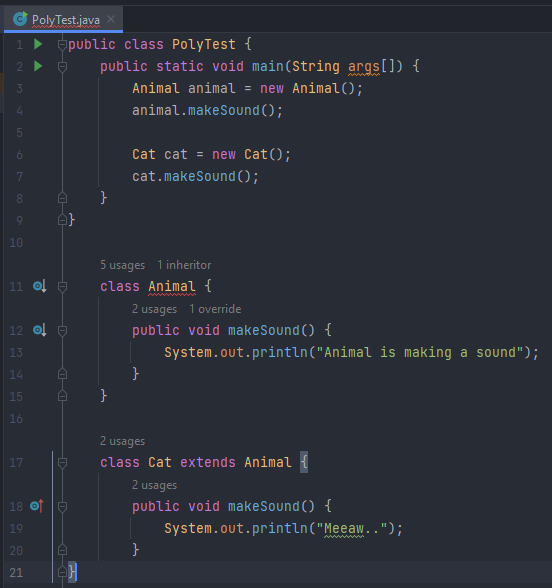
EncapRun.java

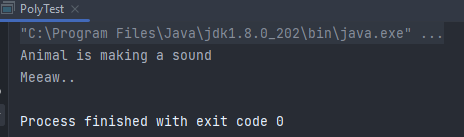


Result :  


* + 1. **Polymorphism**  
       Polymorphism (polymorphism) consists of two syllables, poly means many and morphism comes from the word morph means form. So that polymorphism in OOP programming is a term that explains that a class can have more than one method name that is the same with different contents or method bodies.

Example :  
PolyTest.java

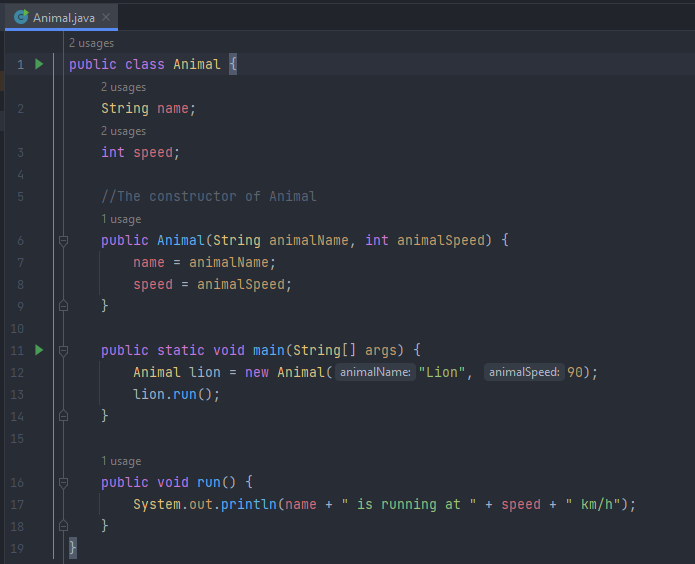


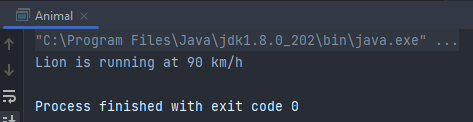
Result:  


* + 1. **Constructor/Destructor**

Constructor is a special method that is used for initialization when creating an object. The constructor is called as soon as the new object is created. The feature of the constructor is that the method name exactly matches the class name.

Example :  
Animal.java

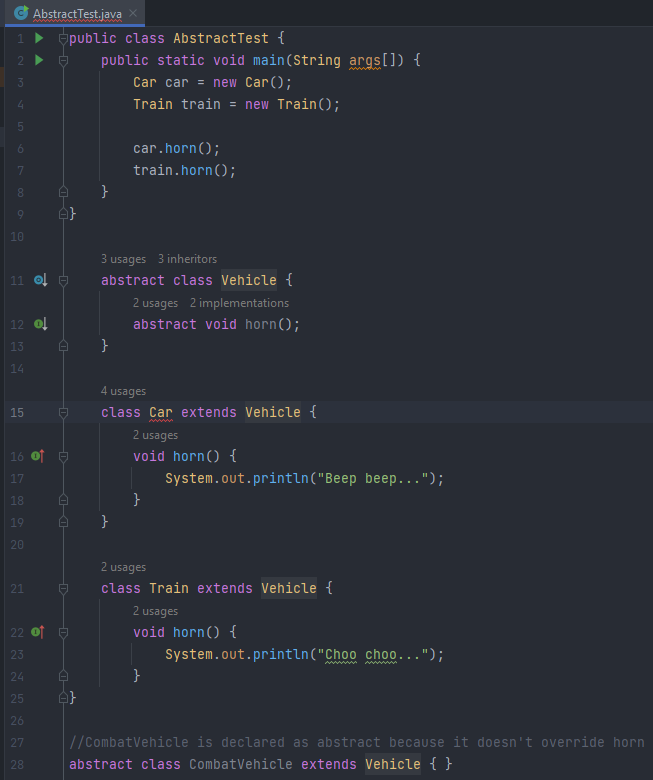


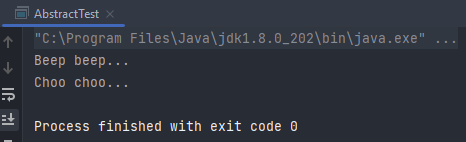
Result :  


* + 1. **Abstract**

Abstract is a declaration without providing details on what is declared. Abstract can be implemented in classes or methods using the abstract keyword. The opposite of abstract is concrete (concrete) where classes and methods are declared along with their details clearly.

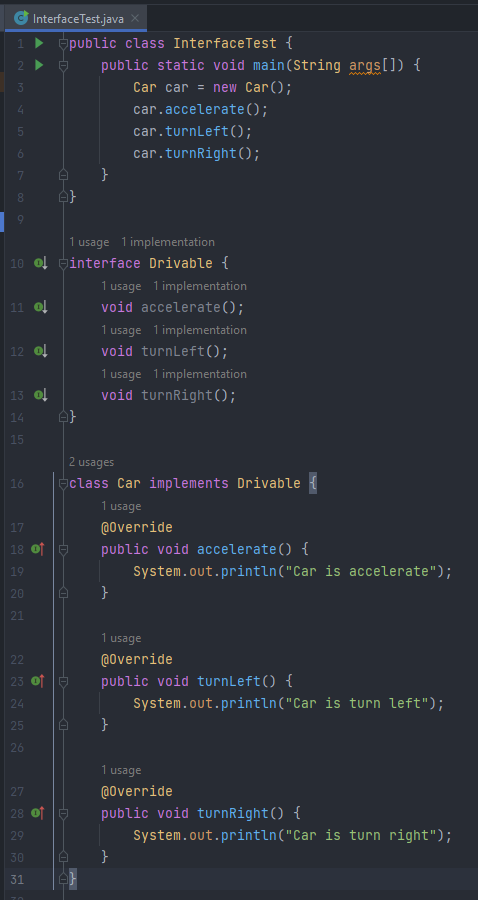
Example :

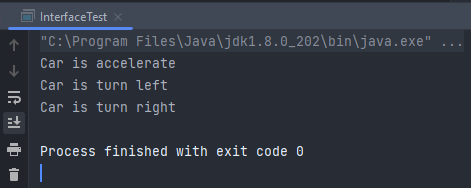
AbstractTest.java  


Result :  


* + 1. **Interface**The interface is a class-like form in which it consists of a collection of empty methods and constants. Interfaces cannot be made objects but can only be implemented. If the class declares an object, then the interface generally expresses capabilities such as Readable, Runnable, Comparable, etc.

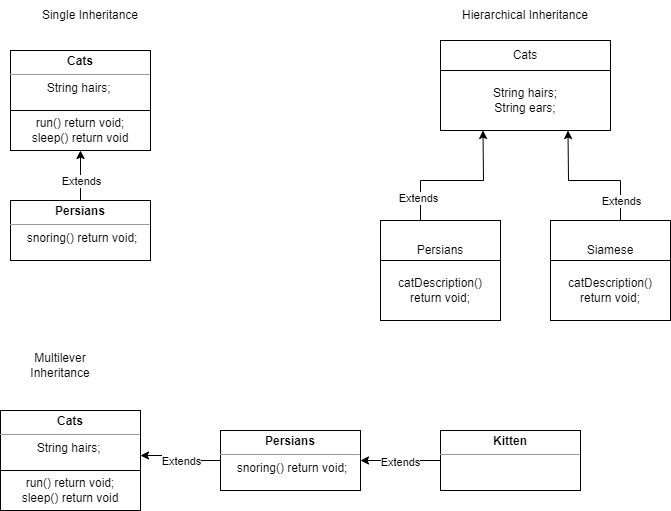
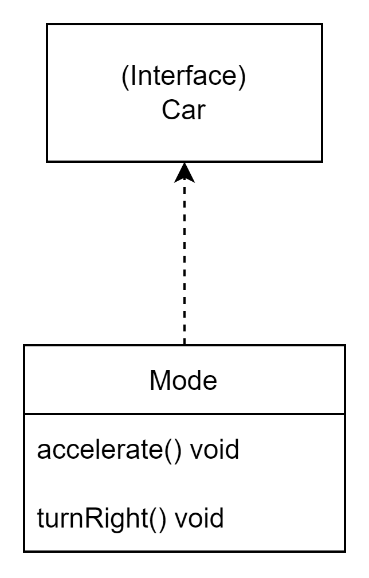
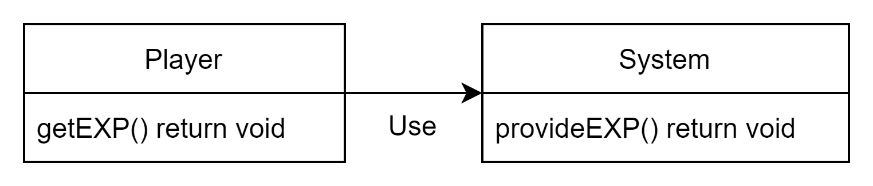
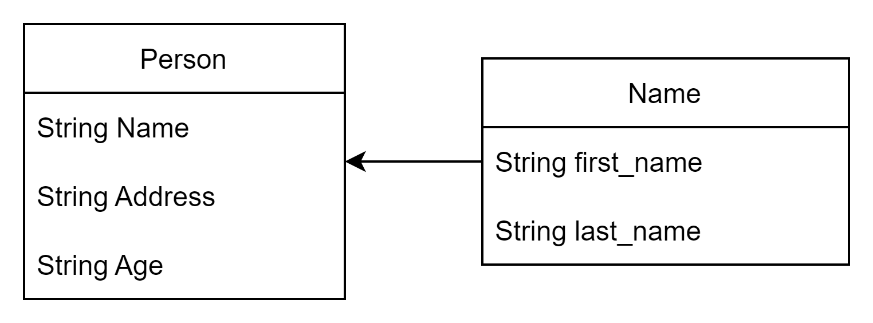
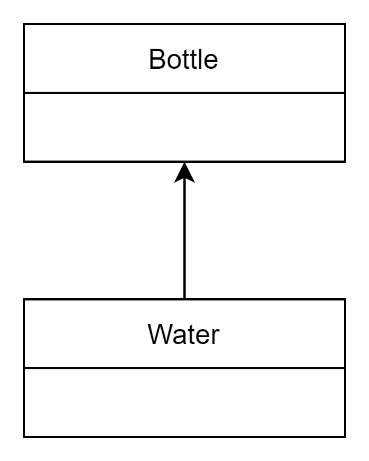
Example :

InterfaceTest.java  


Result:  


* + 1. **What are the differences between abstract classes and interfaces?**

|  |  |  |
| --- | --- | --- |
| NO | Abstract | Interface |
| 1 | Can contain both abstract and non-abstract methods. | It can only contain abstract methods. |
| 2 | The modifiers must be written alone. | There is no need to write public abstract in front of the method name. Because implicitly, modifiers for methods in interfaces are public and abstract. |
| 3 | Can declare constants and instance variables. | Can only declare constant. Implicitly the variables declared in the interface are public, static and final. |
| 4 | Methods can be static. | Methods cannot be static |
| 5 | An abstract class can only extend one other abstract class. | An interface can extend one or more other interfaces. |
| 6 | An abstract class can only extend one abstract class and implement several interfaces. | An interface can only extend other interfaces. And cannot implement other classes or interfaces. |

1. Examine Class Relationships using UML diagrams
   * 1. **Inheritence**
     2. **Realization  
        **
     3. **Depedency  
        **
     4. **Aggregation  
        **
     5. **Composition  
        **
2. Examine and list out class relationships which are used in project scenario.
   * + 1. **Inheritence :**

Administrators, Site Members, and customer classes have

common variables and methods, they all use inheritance from the User, a

new super class with generic variables and methods.

* + - 1. **Association :**

Administrators can only manage customers and site members, and site members can only handle customers, so the relationship between administrators and customers, administrators and site members, and site members and customers is unidirectional.

* + - 1. **Composition :**It is used between the Customer and Service classes because they have a relationship, which means that when the Service class changes something, the Customer class also changes.

1. Identity the suitable design patterns base on the project scenario.

Determine a design pattern from each of the creational, structural and behavioral pattern types. Design pattern is a method created to assist the development team in finding solutions to common problems that arise when software development is in progress.

Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation.

List out creational design pattern

|  |  |
| --- | --- |
| [Singleton](https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples) | This singleton ensures that a class has only one instance and provides a global access point to that instance. There are two forms of this singleton pattern, namely Early Instantiation and Lazy Instantiation. |
| [Factory](https://www.journaldev.com/1392/factory-design-pattern-in-java) | The factory method specifies an interface or abstract class for creating an object, but still allows subclasses to change the type of object being created. It can also be called a Virtual Constructor. |
| [Abstract Factory](https://www.journaldev.com/1418/abstract-factory-design-pattern-in-java) | This pattern can only define an interface or an abstract class to create an object without having to specify its sub-class. |
| [Builder](https://www.journaldev.com/1425/builder-design-pattern-in-java) | Builder is a creational pattern that creates a complex object from a simple object using a step-by-step approach. |
| [Prototype](https://www.journaldev.com/1440/prototype-design-pattern-in-java) | This pattern allows you to copy an object using an existing prototype instance by creating a new object and can be customized according to your needs without relying on the original object.. |

1. Identify suitable design pattern for “AAA Portlet”
   * + - Creational Design Pattern

* Factory Design Pattern
* The Factory Method pattern will be used to seta runtime interface in project to develop an object
* Factory patterns make code more resilient, easier to modify, less dependent.
* For instance, we may simply modify the implementation of a class because the client unaware of it. After all, it just offers code for the interface rather than implementation
* Prototype Design Pattern
* The prototype Is used to cover the complexities of creating new client instances.
* The objective is to clone an already existing object instead of generating a new instance, which might comprise expensive actions
* This technique saves time and funds especially if the creation of object is a major operation

1. What is structural design pattern?

Structural Design Patterns are Design Patterns that facilitate design by identifying simple ways to realize relationships between entities. This pattern category addresses the composition between modules or components that allows one to develop or build a larger system. Structural patterns define how each component or entity should be structured so as to have highly flexible interconnecting modules that can work together within the larger system.

1. Identify suitable behavioral design pattern for “AAA Portlet”

* Template method / pattern

We could create a template for an abstract class with the majority of the functions predefined, and then create another class with a slight modification in accordance with the desired outcome, because we use many classes, most of which share relatively similar code.

* Observer method

Because the portlet contains a one-to-many relationship, for example, one customer may have more than one service, the observer pattern may be useful in keeping track of the state of the object in the portlet.

1. Examine how the object-orientated paradigm and its key principle is identified in each of the design patterns.

The Problem – The OOP's problem-solving methodology is entirely different. Because it focuses parts of the system rather than solving them. In the real world, we can use objects such as cars and cats as examples.

Because object-oriented programming (OOP) is a programming technique or philosophy that divides code into objects and their connections. Design patterns provide tried-and-true methods of creating types or objects to solve a specific problem in software.

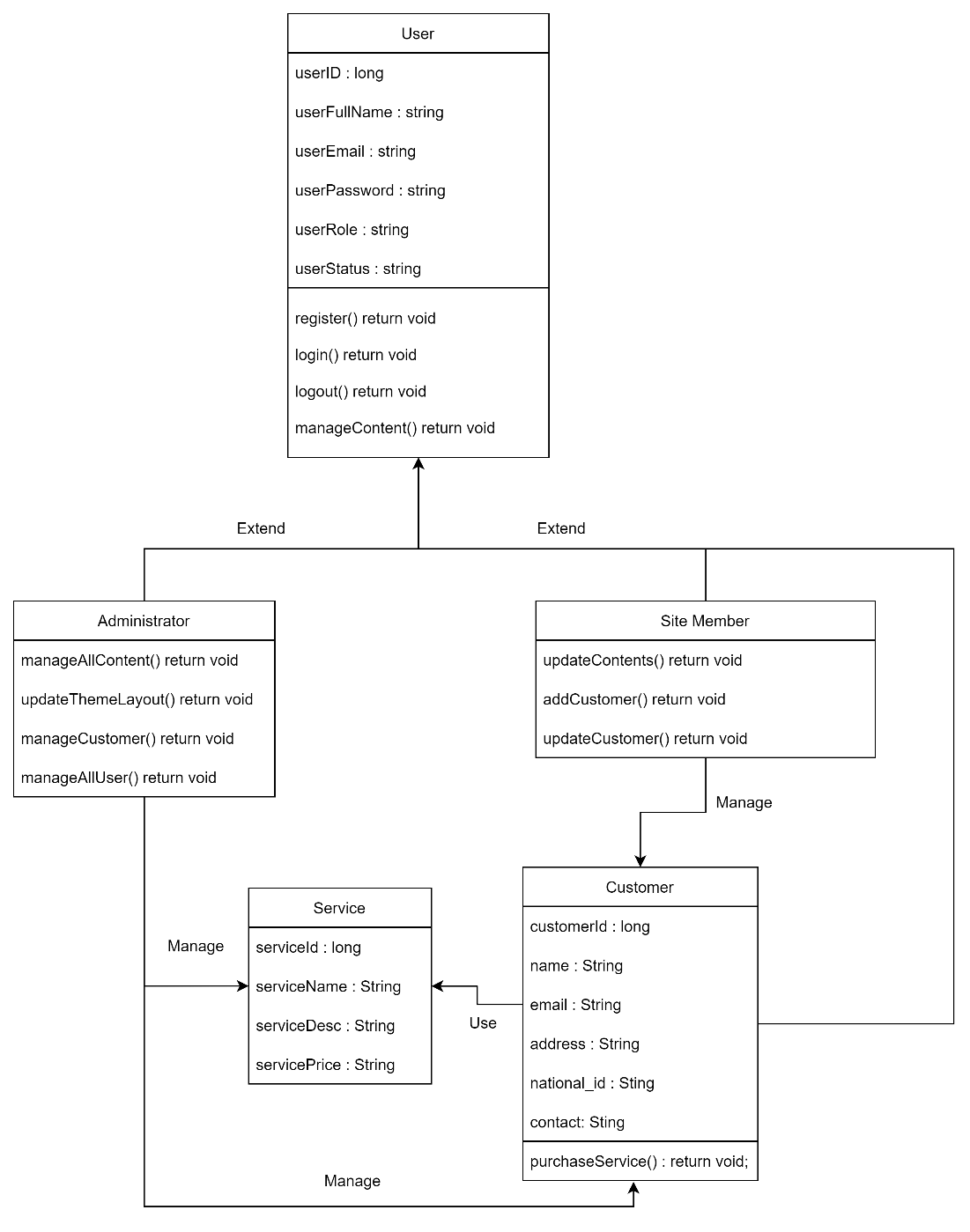
1. Task 2

Task Statement:

Create the following items in “System Design” Section in Project Report

1. Draw class diagrams that represent a simple structure based on the project scenarios using a UML tool.
2. Identify possible situations where design patterns would be beneficial and then develop the UML diagrams reflecting the design patterns.
3. Observe how the class diagrams are reflected from a given scenario by using the UML tool.

Solution:

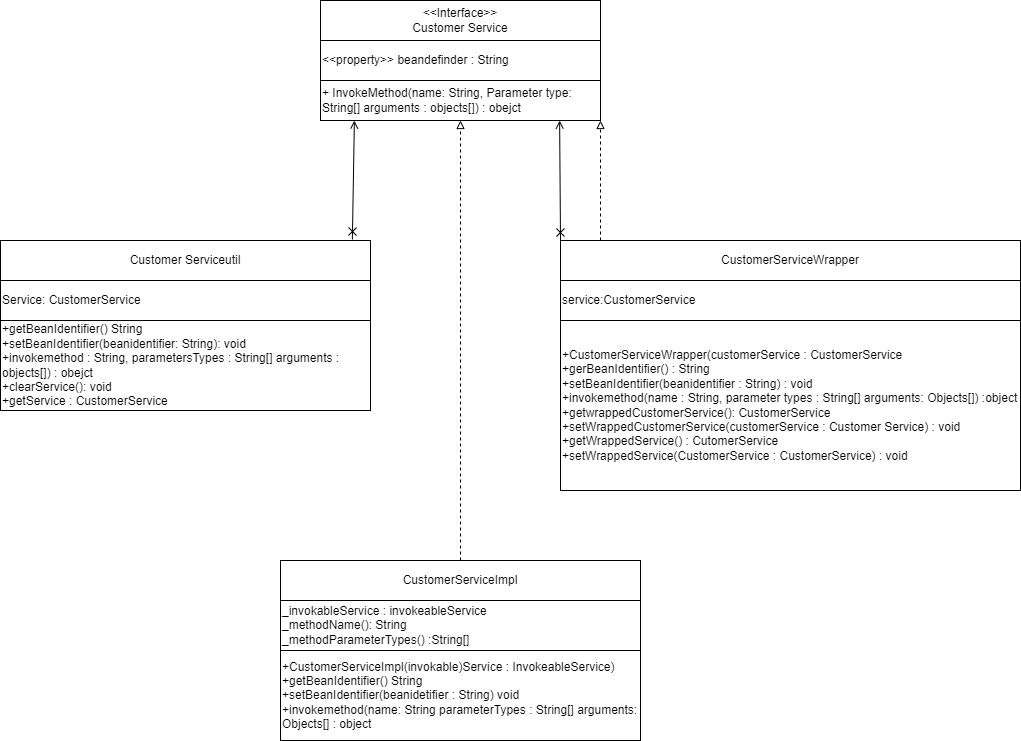
1. Draw class diagrams that represent a simple structure based on the project scenarios using a UML tool.  
   
2. This website has two users: an administrator and a site member (staf)
3. AAACustomer and AAAServices can be managed by the administrator.
4. AAACustomer is under the control of a Site Member.
5. AAA Services and AAA customers have a compositior relationship in the sense that AAA services are the composition of customers. If someone requires services, customers exist or are created.
6. Identify possible situations where design patterns would be beneficial and then develop the UML diagrams reflecting the design patterns.

Benefits of Design Pattern

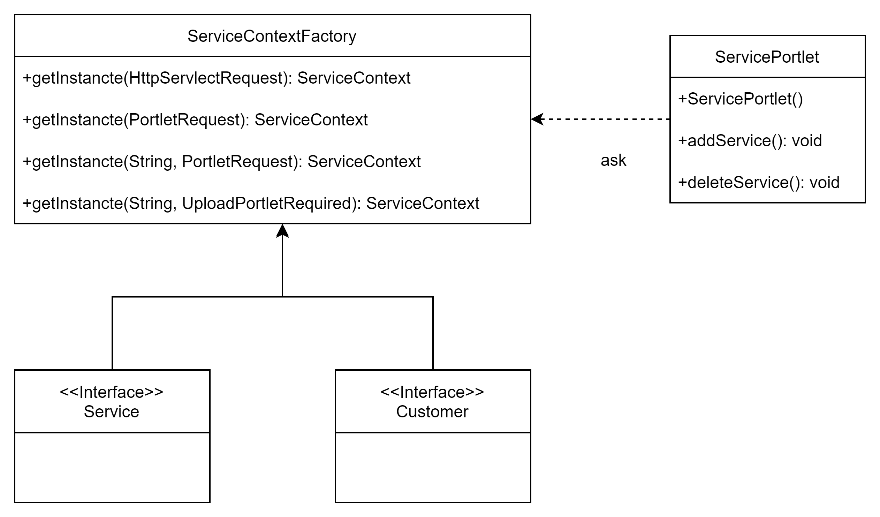
Design Patterns make it easier to modify software and reuse objects. It is easier to reuse and modify loosely connected items. A wide range of tiny specialized objects are used to create design patterns. The system object performing the requested function is assigned functional responsibilities.

* Maintaining binary compatibility with later versions is made easier with his feature.
* Its simple to product and solve future issues.
* Design pattern help designers and developers communicate more clearly and precisely.

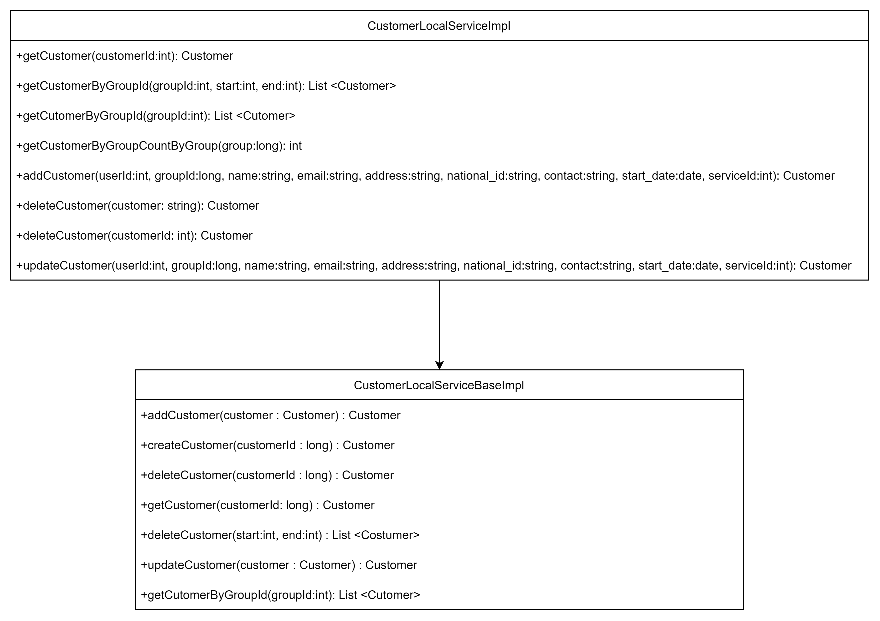
Façade Patterns



Factory Patterns



Template Patterns



1. Observe how the class diagrams are reflected from a given scenario by using the UML tool.

The factory design pattern code is the authoring design pattern in the service portlet. A service portlet's ServiceContext gets a service class from a ServiceContextFactory. Similarly, the factory design pattern is used in the customer portlet. A class diagram includes class names, properties, class links, and methods. As a result, the class diagram displays the actual source code for the application. As the next type of design pattern code for creation, the prototype design pattern code clones an existing object rather than creating a new object.

CustomerModelImpl clones the clone() function of the parent class CustomerModel. CustomerModelImpl clones the clone() function of the parent class CustomerModel. I quickly created a class diagram using UML tools after analyzing the code. As a result of structural design patterns. The observer design pattern is used in source code. For starters, the LocalServiceImpl class extends and overrides the methods of the hyper-abstract CustomerLocalServiceBaseImpl class. As shown in the diagram, the last design pattern I created is the Observer pattern..

1. Task 3

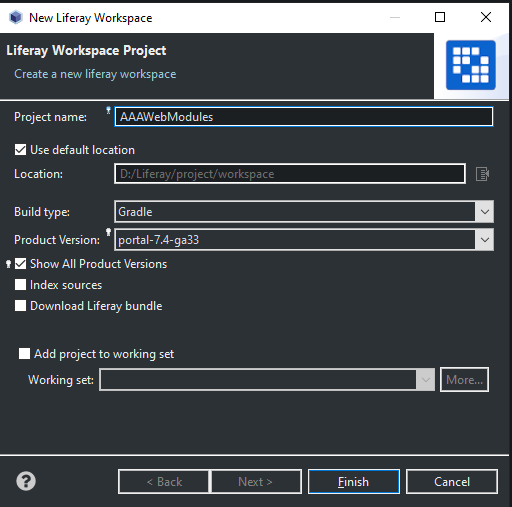
Task Statement:

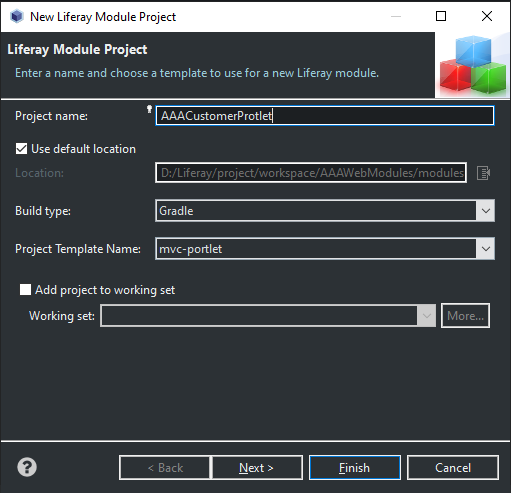
Create the following items under “System Development” in Project Report

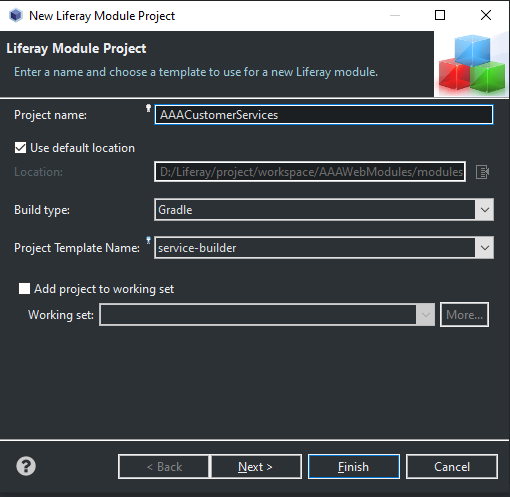
1. Set up a project, build an application based on your derived UML class diagrams and design patterns. Provide screenshots as evidence.
2. Develop an application that implements design patterns and utilizes techniques to produce secure code. Provide the implemented code as evidence.

Solution:

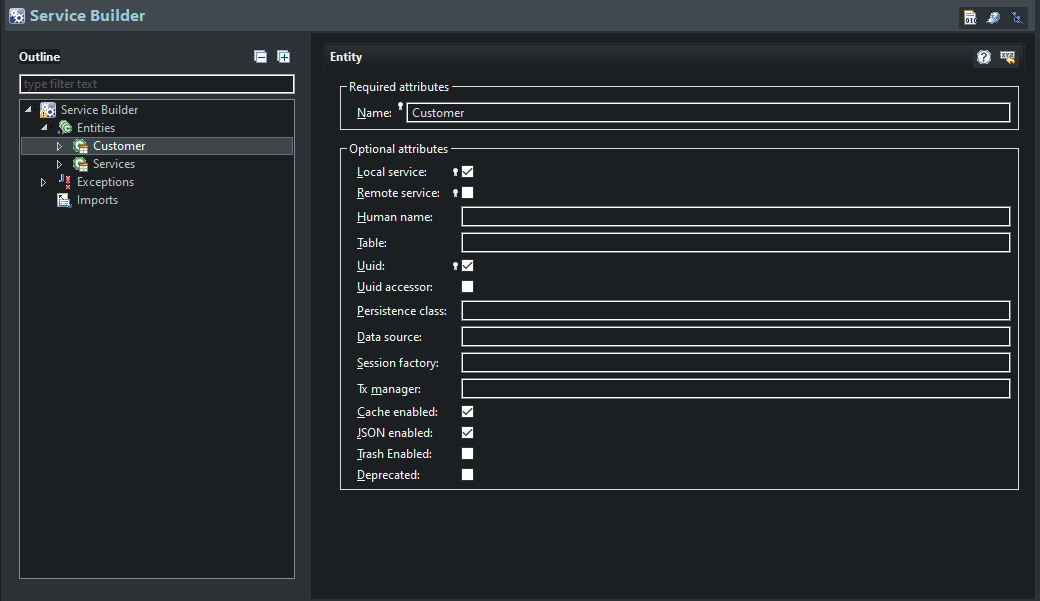
Create MVC Portlet and Service Builder to manage AAA Customer’s data

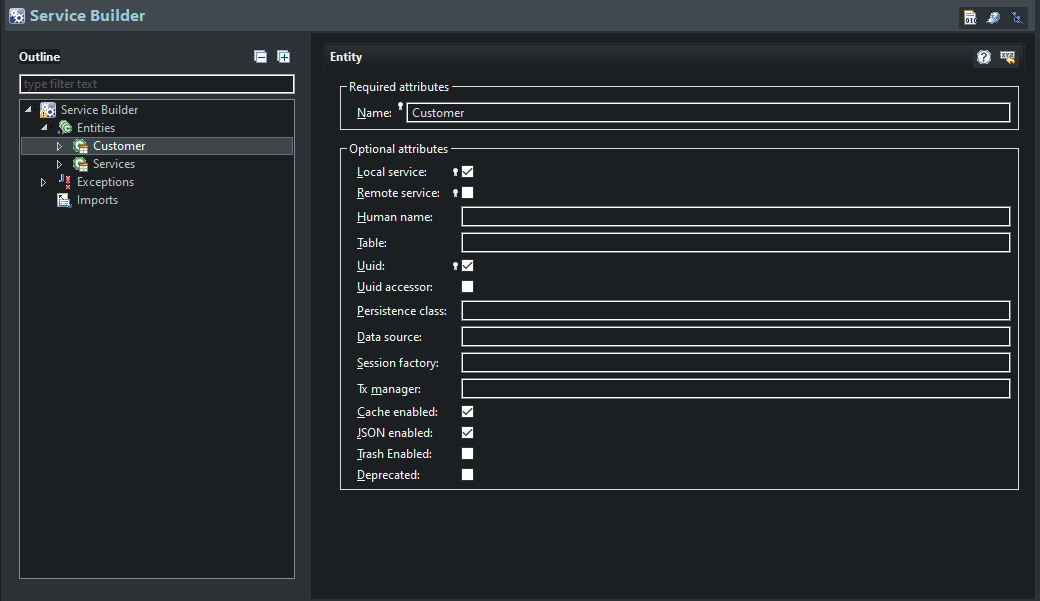




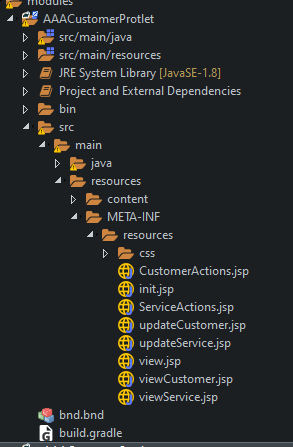


Create required entities and their properties

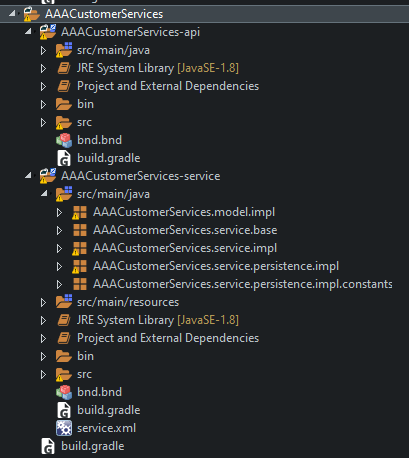


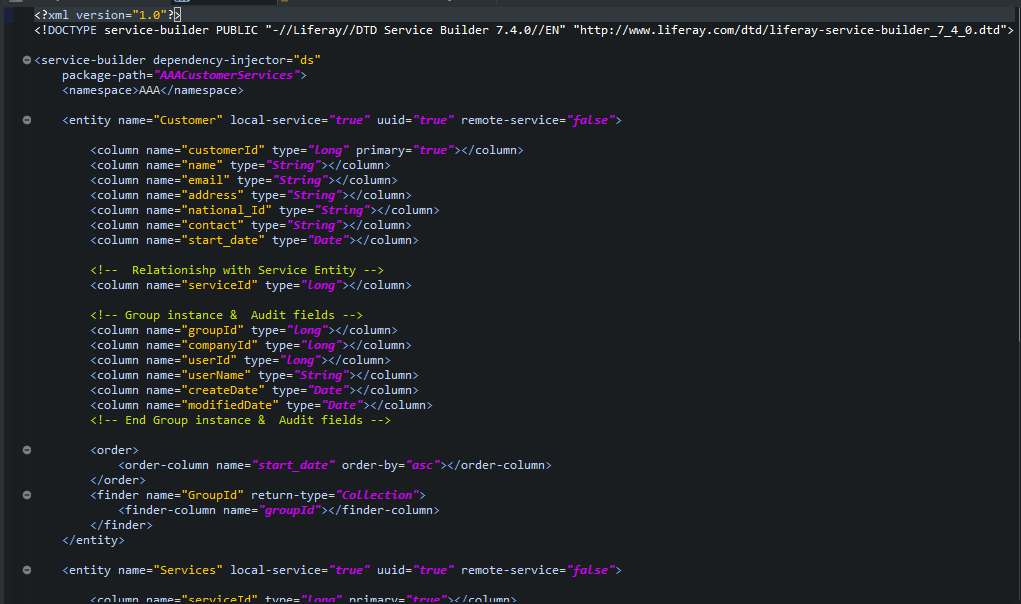


Customer Portlet

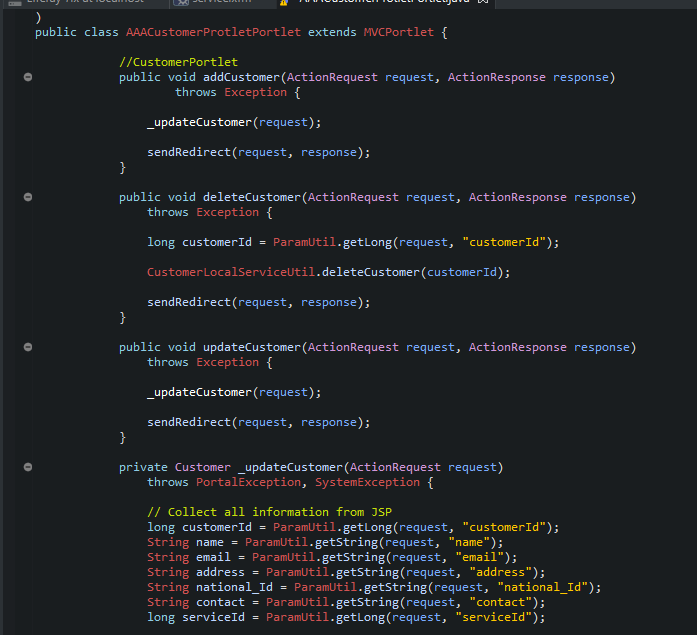


Customer Service Builder

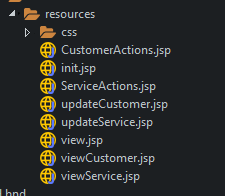


Service.xml  


Portlet Controller



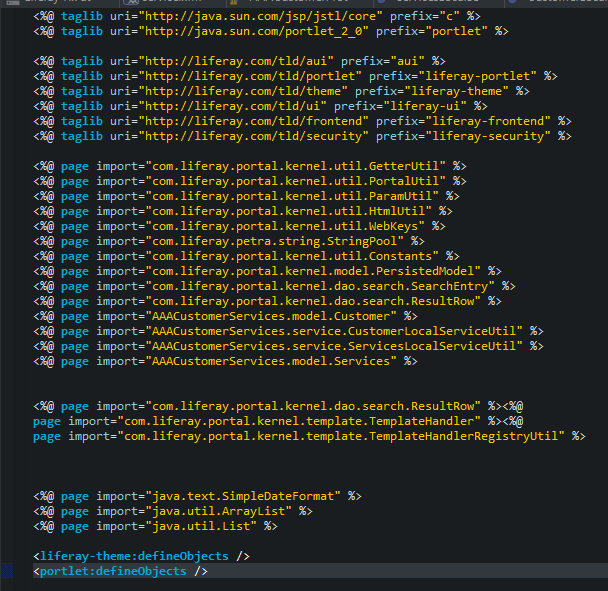
View (Jsp Files)



CustomerActions.jsp



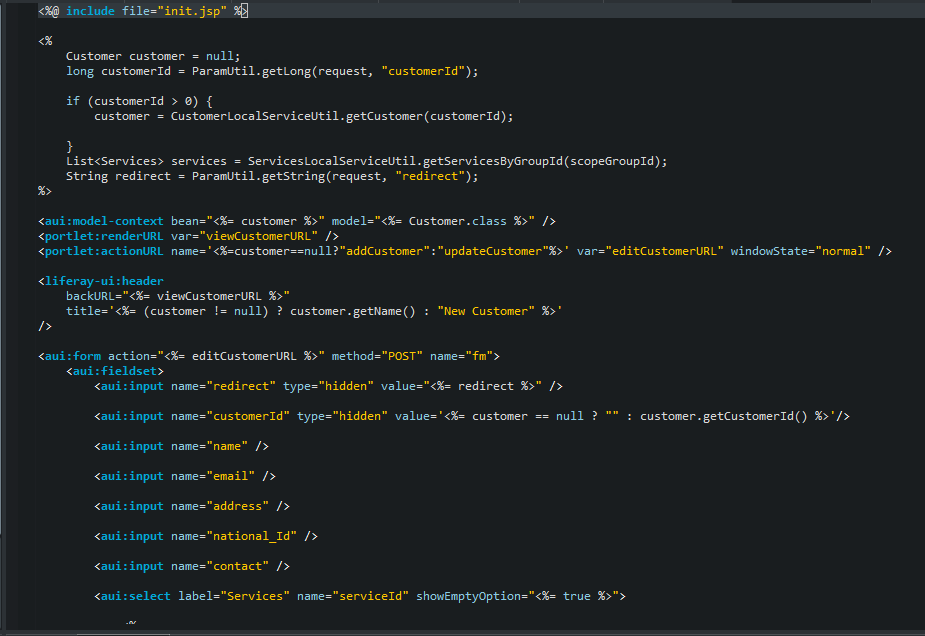
Init.jsp



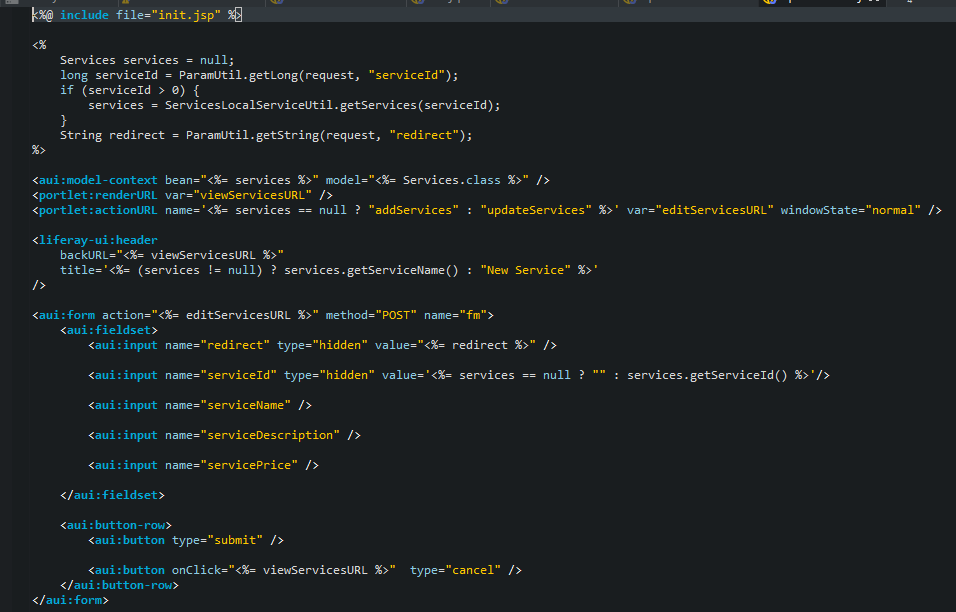
ServiceActions.jsp



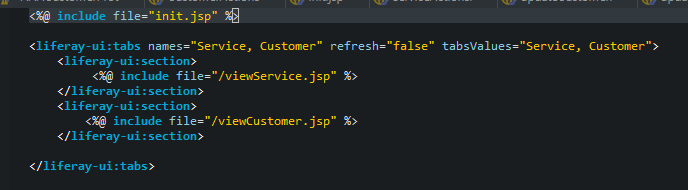
updateCustomer.jsp



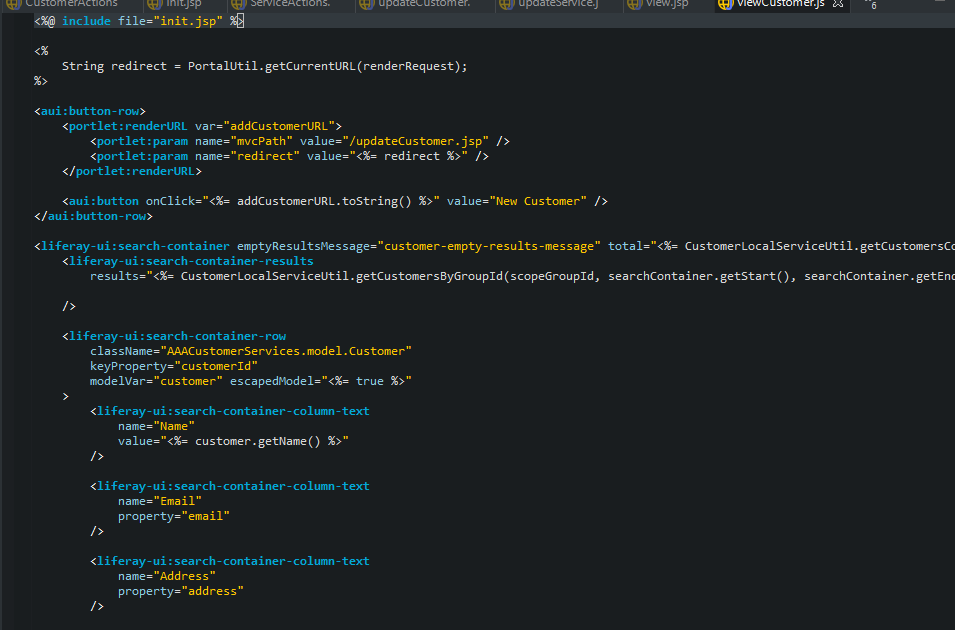
updateService.jsp



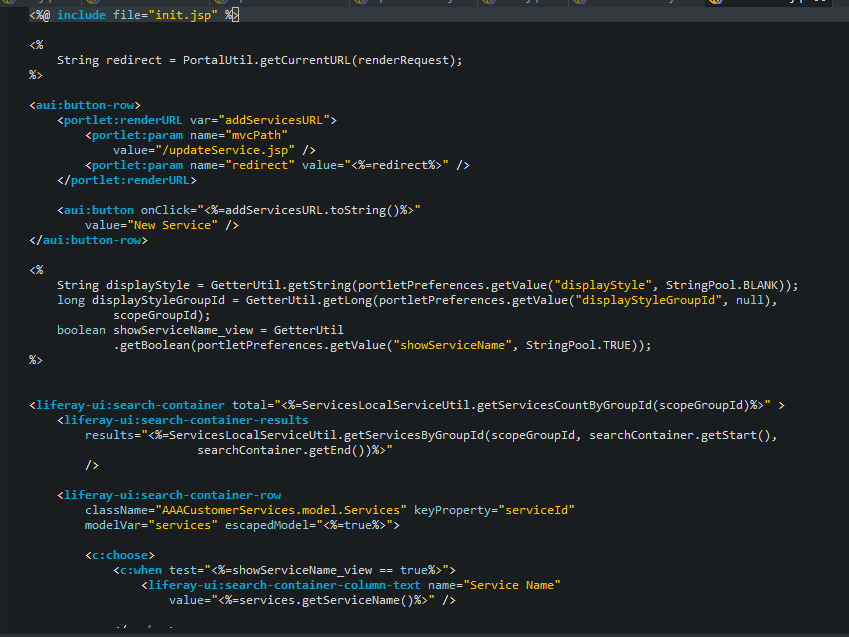
view.jsp



viewCustomer.jsp



viewService.jsp



1. Task 4

Task Statement

Create the following items under “Evaluation of Design Pattern” in Project Report

1. Discuss the use of design patterns for the given purpose and consequences by applying design patterns.
2. Investigate how different design patterns can work within a range of different scenarios.
3. Identify the appropriate design pattern from the investigation.
4. Evaluate and justify the design patterns that you had identify in each of the scenarios.

Solution :

1. Discuss the use of design patterns for the given purpose and consequences by applying design patterns.
2. Discuss usage of design pattern in “AAA Customer Management Portlet”

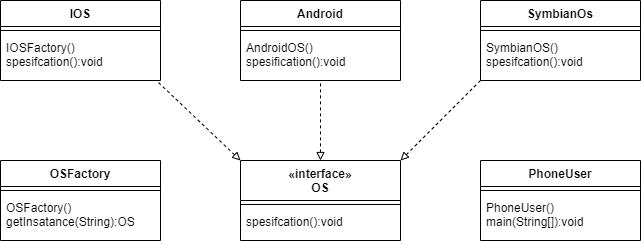
* The use of design patterns in the Triple A Portlet is critical for maintaining neat and clean code.
* Even though the portlet has many layers of services and classes, not every design pattern will be useful when used by the portlet.
* We must use this information to determine the correct and optimal design pattern for the portlet. Because of the multiple layers of java classes, the portlet has already used the Factory pattern. And, because we don't use any filtering functions in the portlet, using the filter pattern is probably not the best pattern to adapt into it.
* Especially given how the portlet implements the method and the fact that there are multiple layers of java classes.

1. The consequences of design patterns which used in the project

* Using the factory pattern in a portlet or any OOP project will result in cleaner and tidier code, but if used excessively will result in a large number of required classes for main class implementation.

1. Investigate how different design patterns can work within a range of different scenarios.

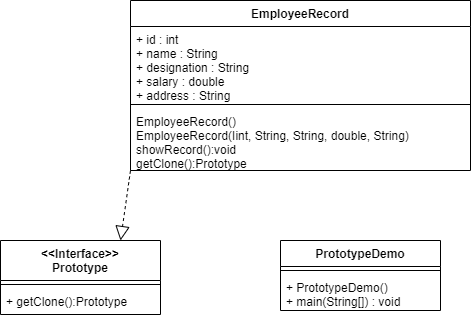
Factory Pattern



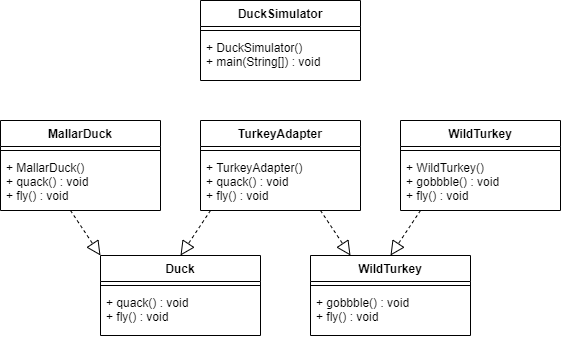
**Singleton**

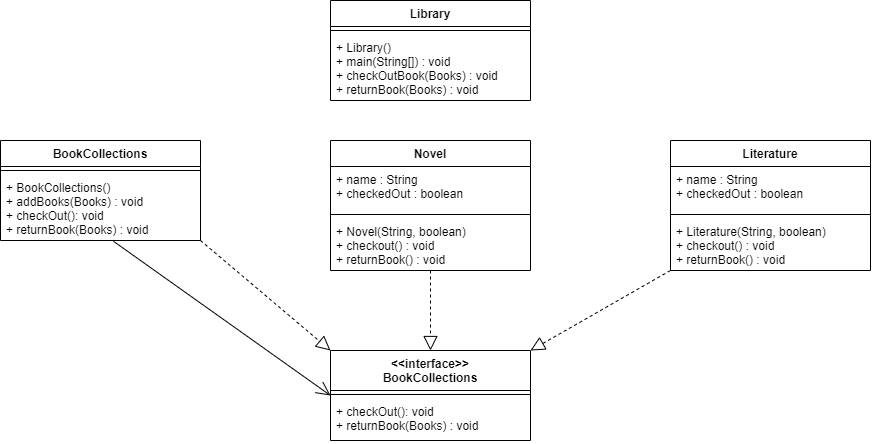
****

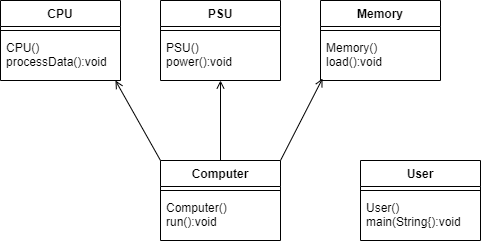
**Prototype**

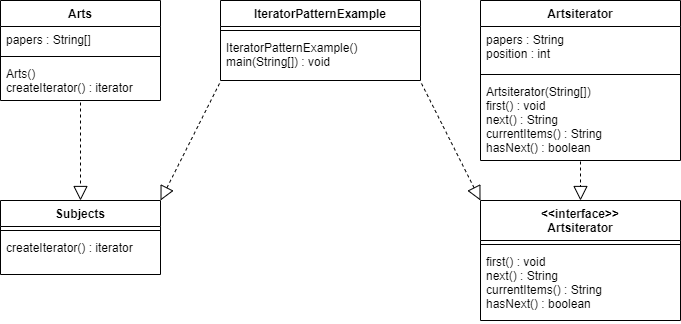


**Adapter Pattern**



**Composite Pattern**

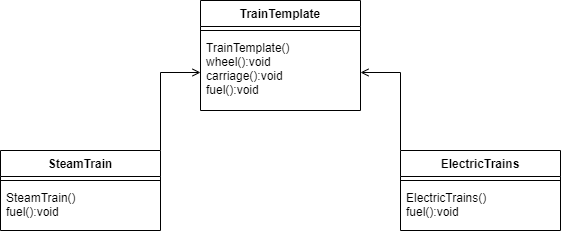
**Façade Pattern**

**Iterator pattern**

**Observer Pattern**



**Template Pattern**



1. Identify the appropriate design pattern from the investigation.

Factory Pattern

In a project, the Factory Method pattern will be used to create an object by defining a runtime interface. Factory patterns make code more resilient, adaptable, and less reliant. For example, because the client is unaware of a class, we could simply change its implementation. After all, it only provides interface code and does not provide implementation..

Prototype

Is used to describe the difficulties in creating new client instances. The goal is to clone an existing object rather than creating a new instance, which may necessitate costly actions, and this technique saves time and money, especially if object creation is a large operation.

Façade Pattern

The facade pattern is a design pattern that streamlines program operation by combining all of its functionalities into a single interface. Because it simplifies the creation of user objects and their subclasses, this pattern would be useful in the AAA Portlet. We could, for example, create a new administrator object and hide its properties and methods behind an interface.

Observer Pattern

Because the portlet contains a one-to-many relationship, such as one customer having multiple services, the observer pattern may be useful in tracking the state of the object in the portlet.

1. Evaluate and justify the design patterns that you had identify in each of the scenarios.

* **Factory methods**, The subclass is in charge of creating class instances in this pattern. Subclasses can make changes to the object being built even if the interface provided when the project is built is a superclass.
* **Prototype**, This pattern allows you to clone existing objects without writing any code. This implies that this operation is not class dependent..
* **Facade pattern** conceals the system's complexity and provides the customer with an interface through which the customer can access the system This design pattern is classified as a structural pattern because it adds an interface to an existing system in order to conceal its complexity..
* **Observer pattern** can have a one-to-many relationship

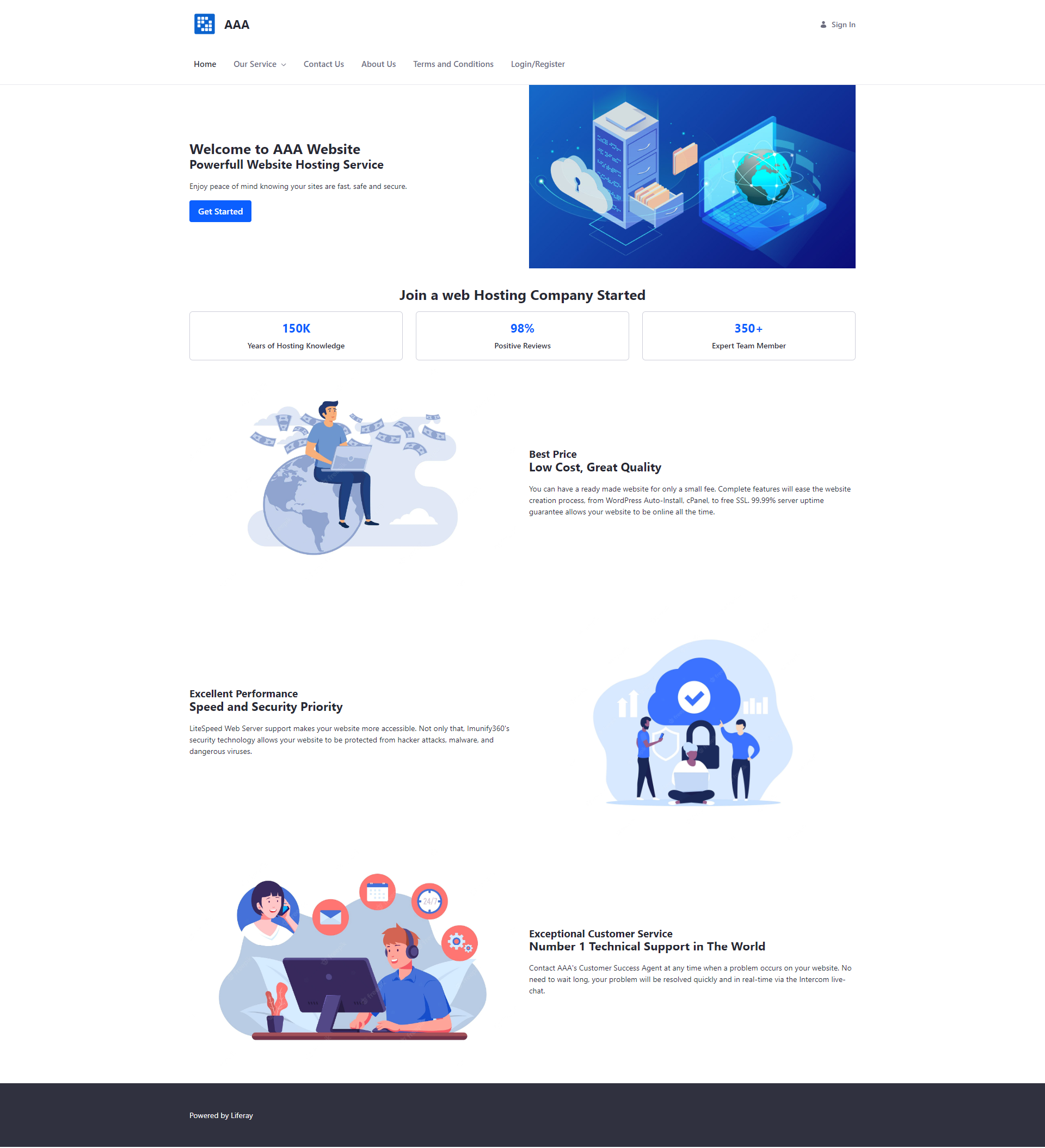
1. Task 5

Task Statement

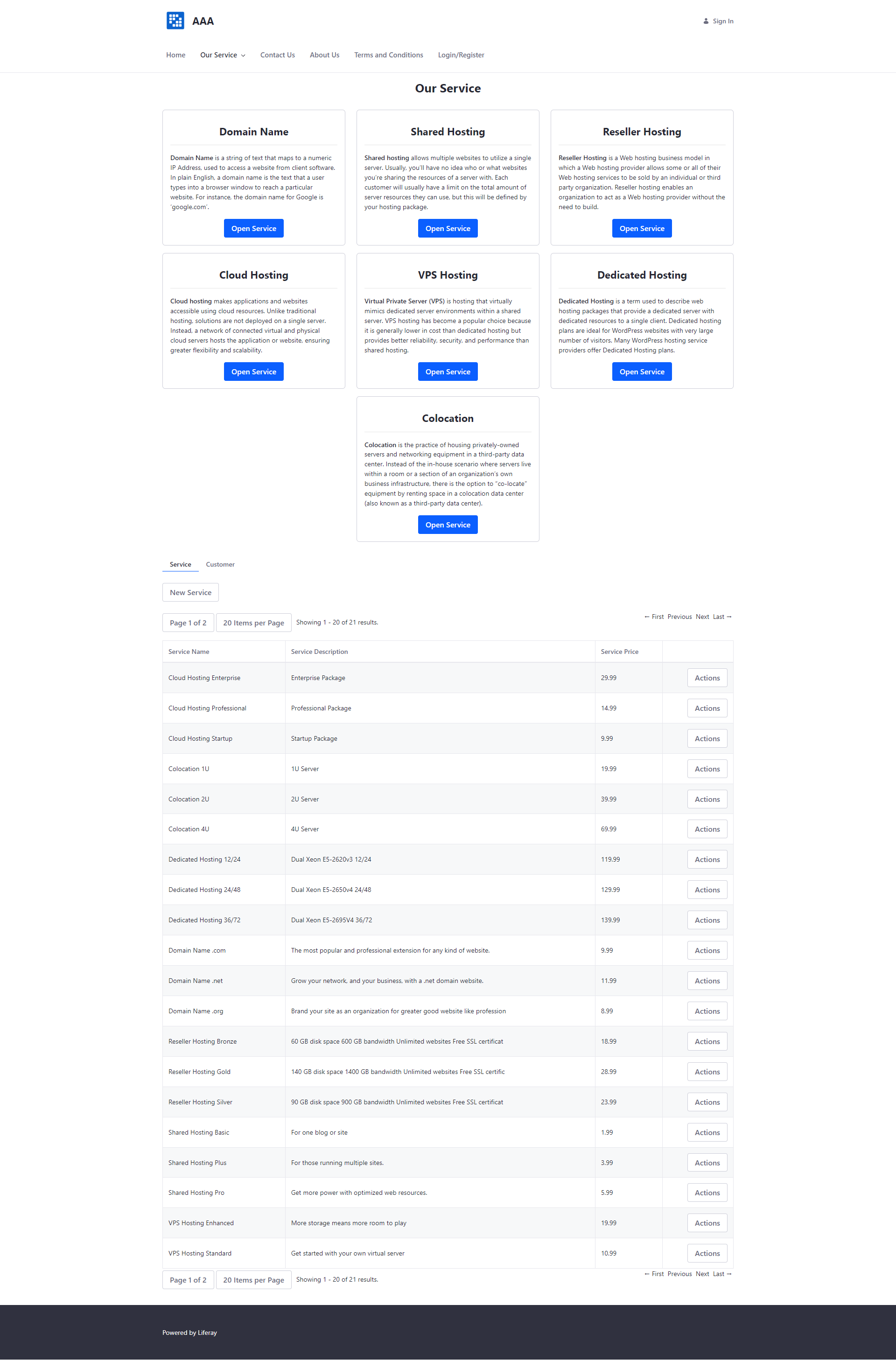
1. Provide screen capture of developed pages and hosted application in Project Presentation

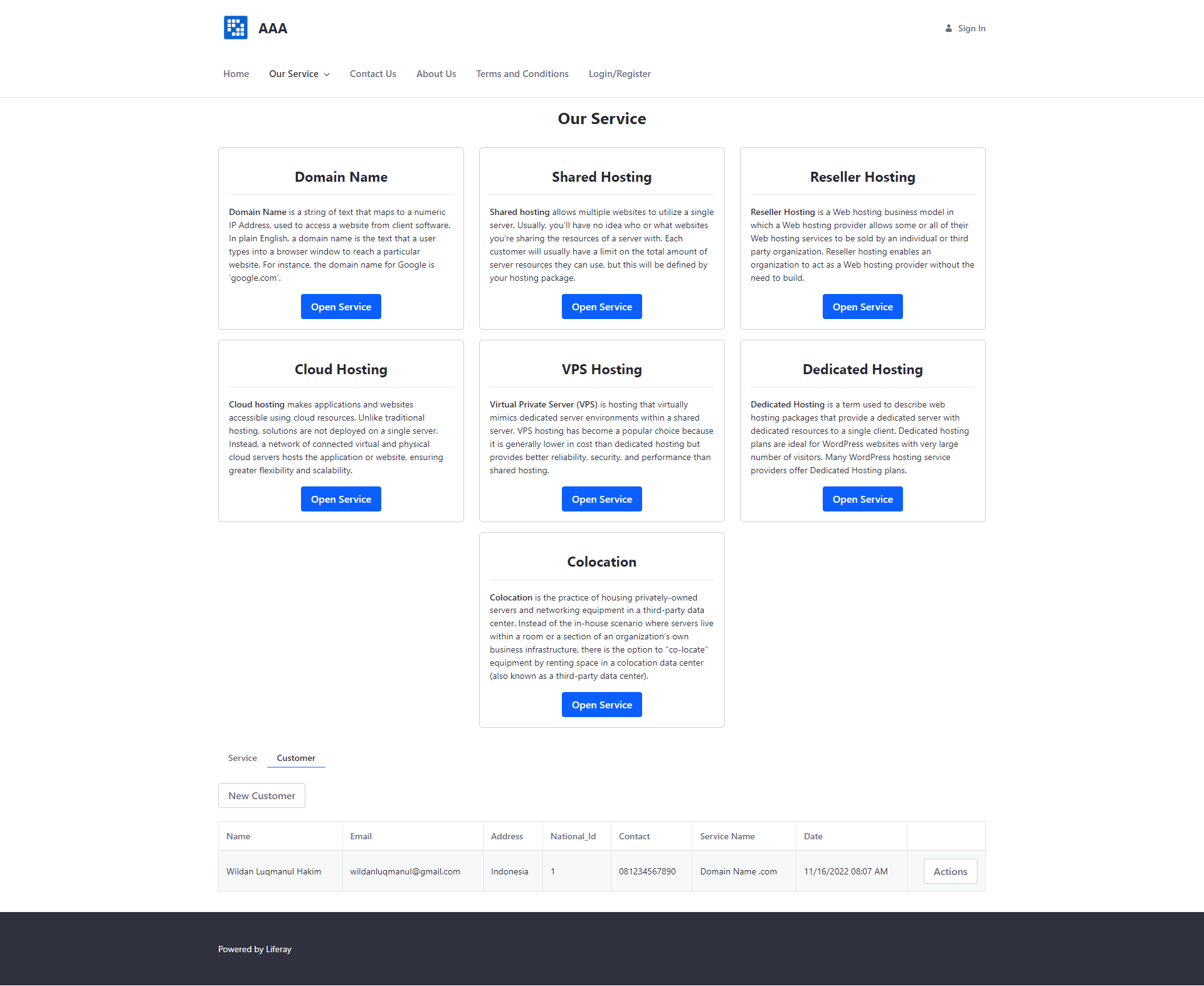
Solution

1. Provide screen capture of developed pages and hosted application in Project Presentation

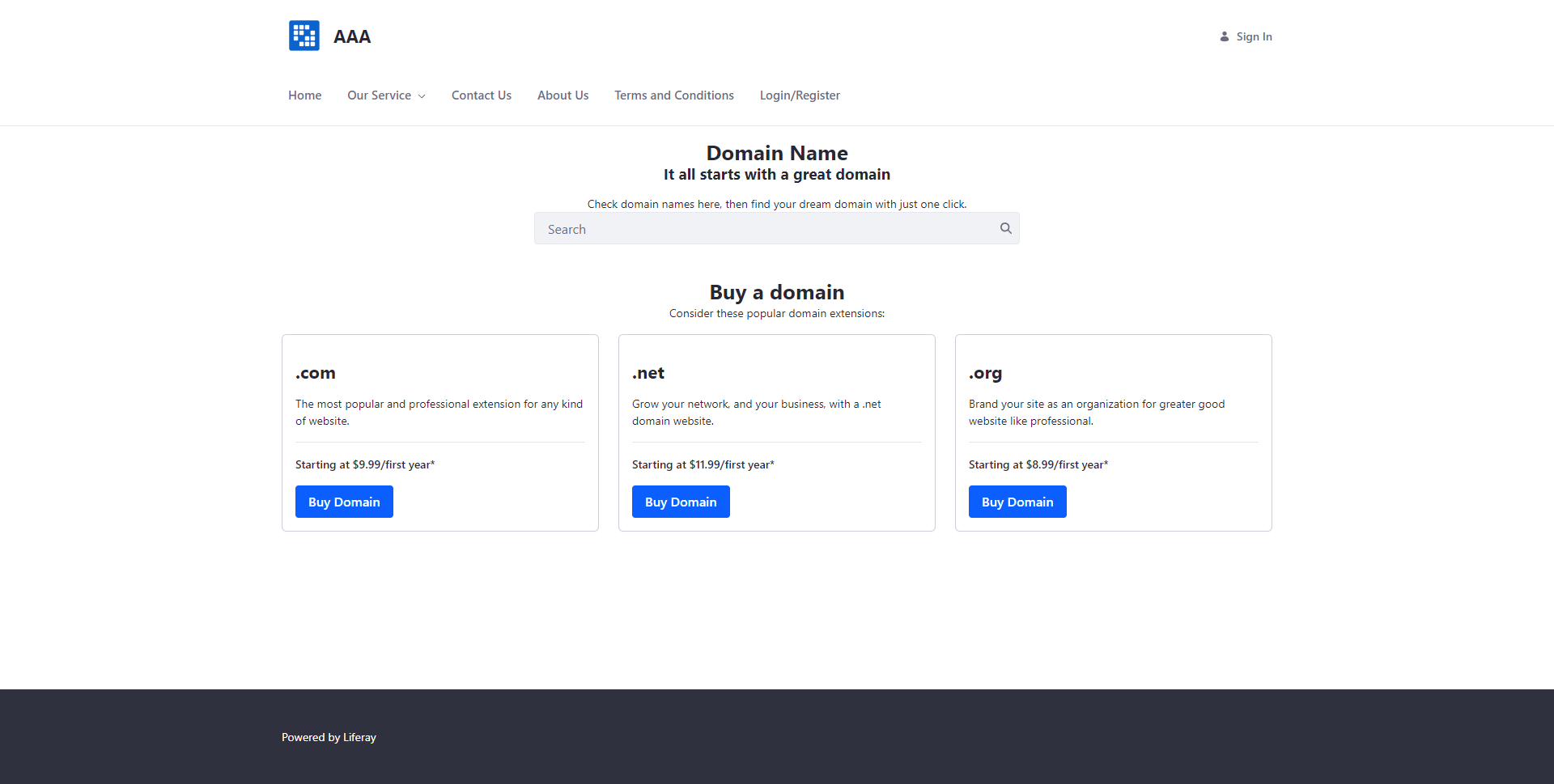
HomePage  


Our Service Page (Including Customer and Service Portlet)

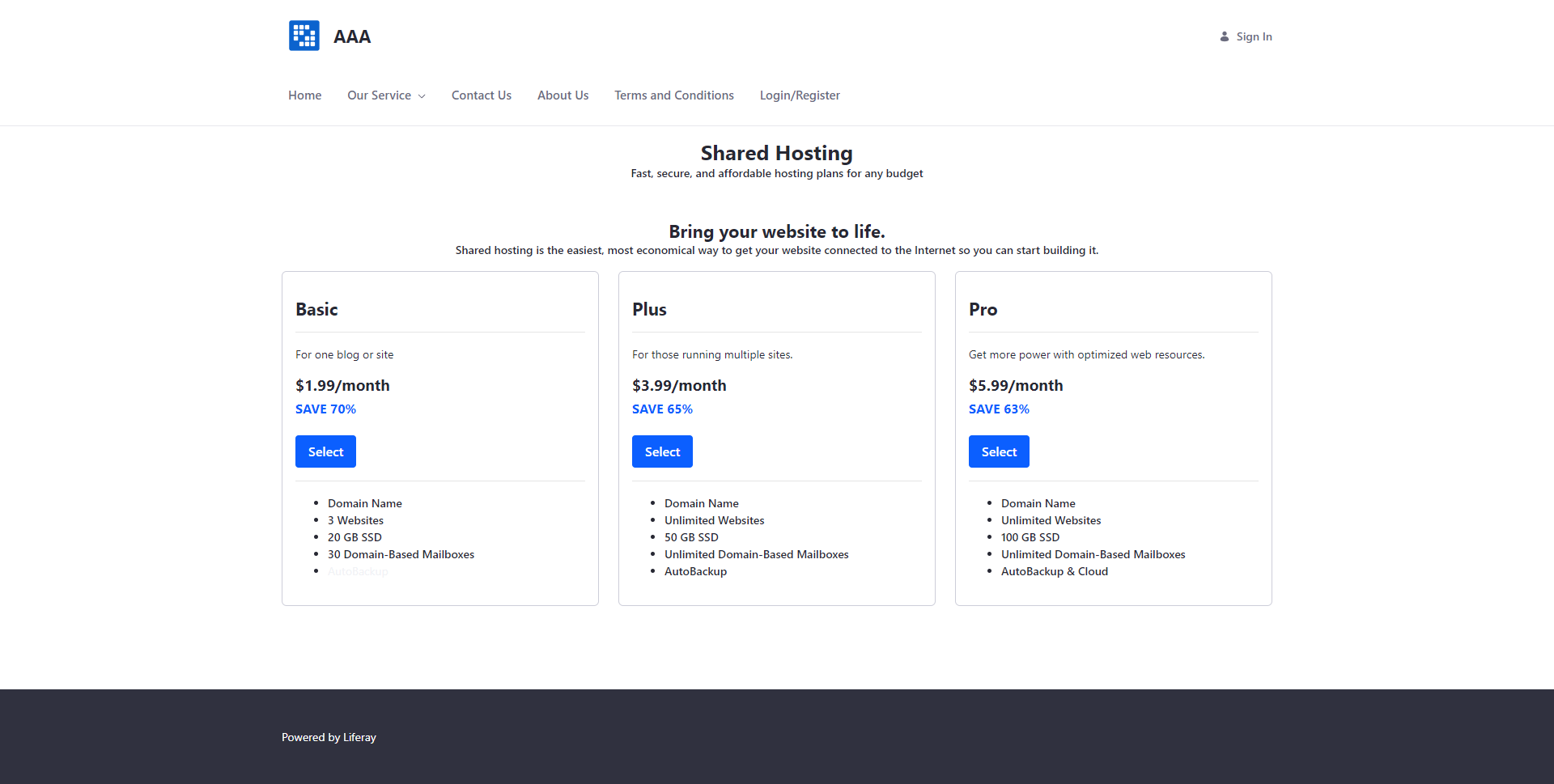
Service  


Customer  


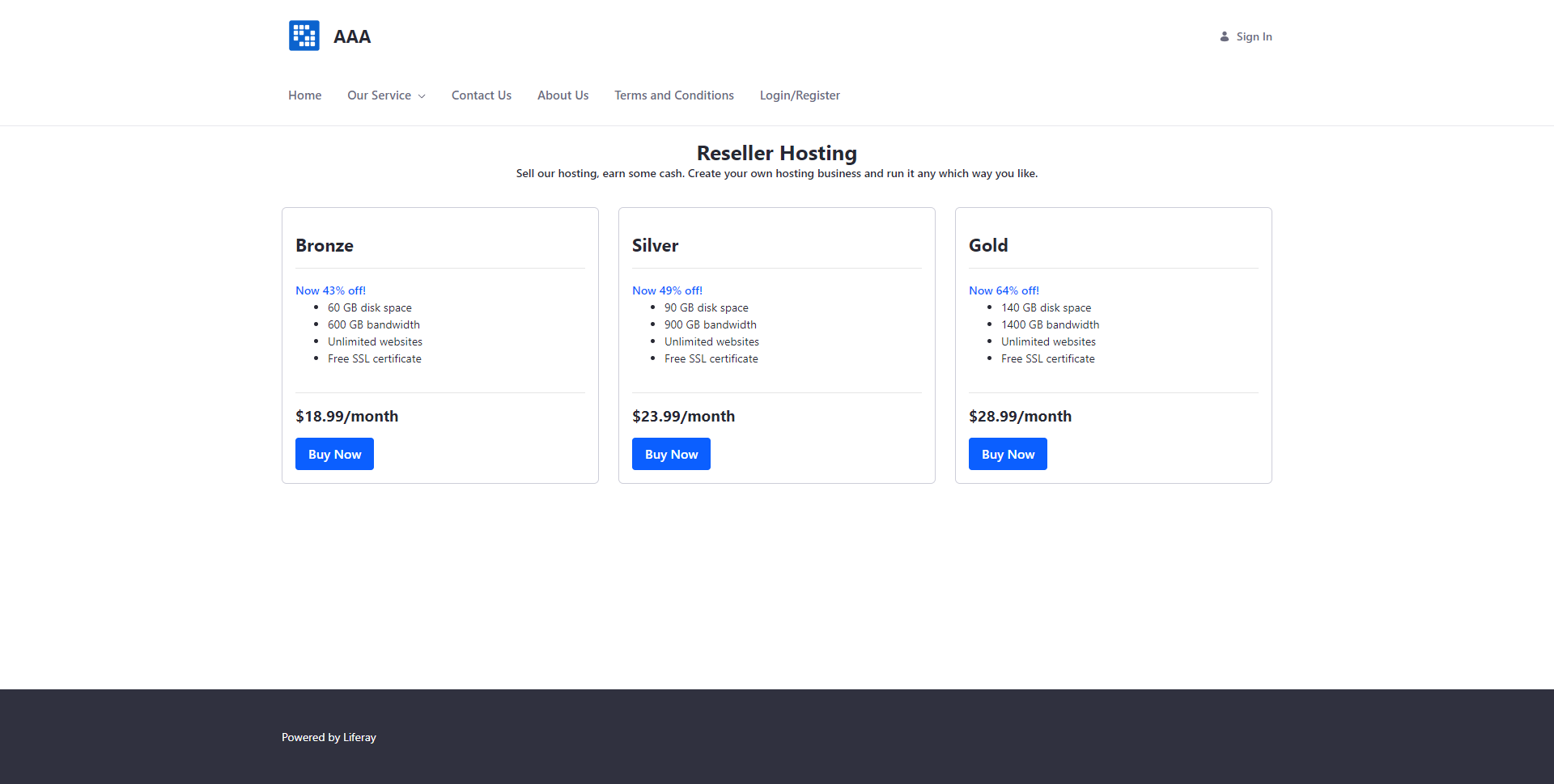
Domain Name Service



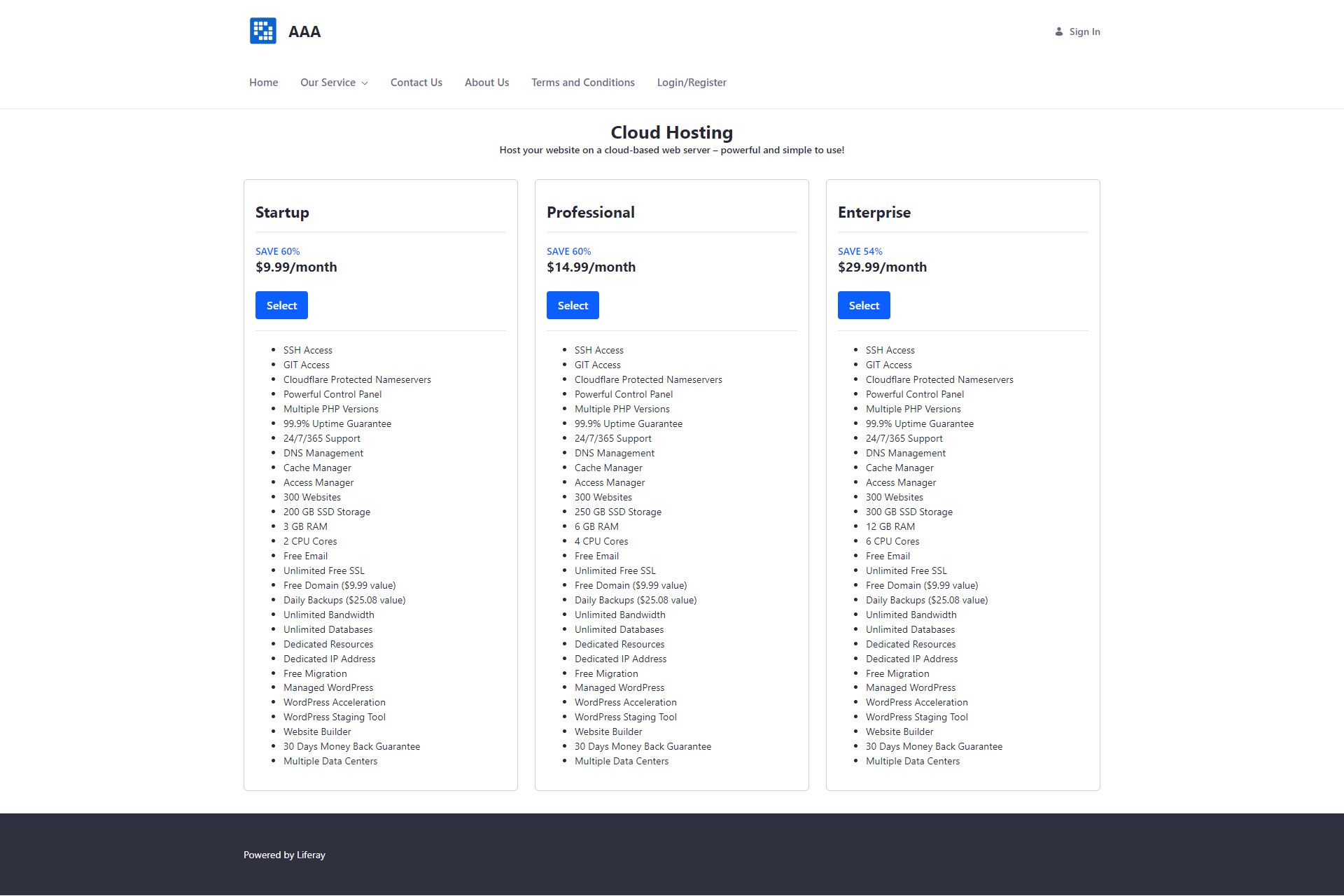
Shared Hosting Service



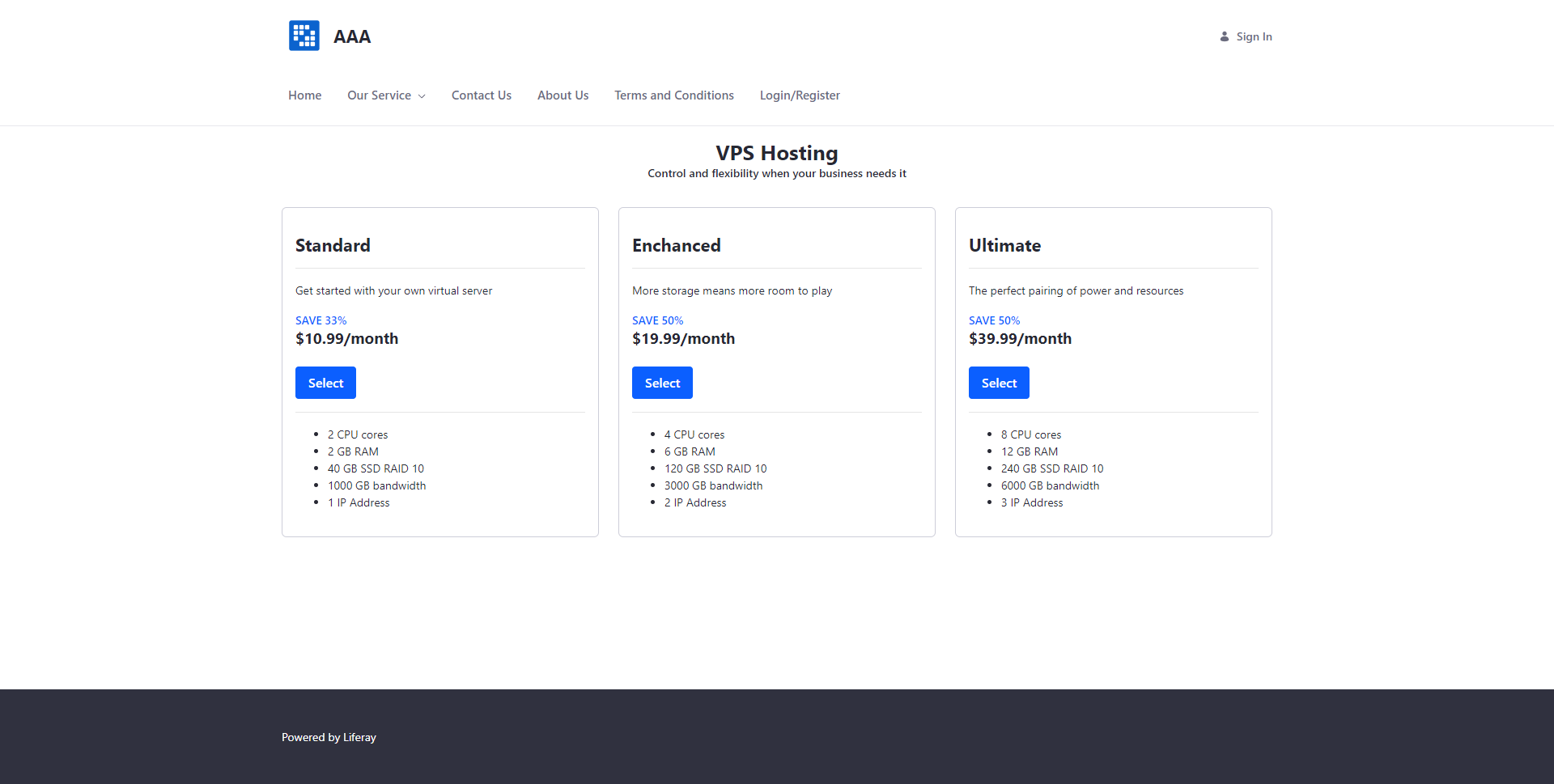
Reseller Hosting Page



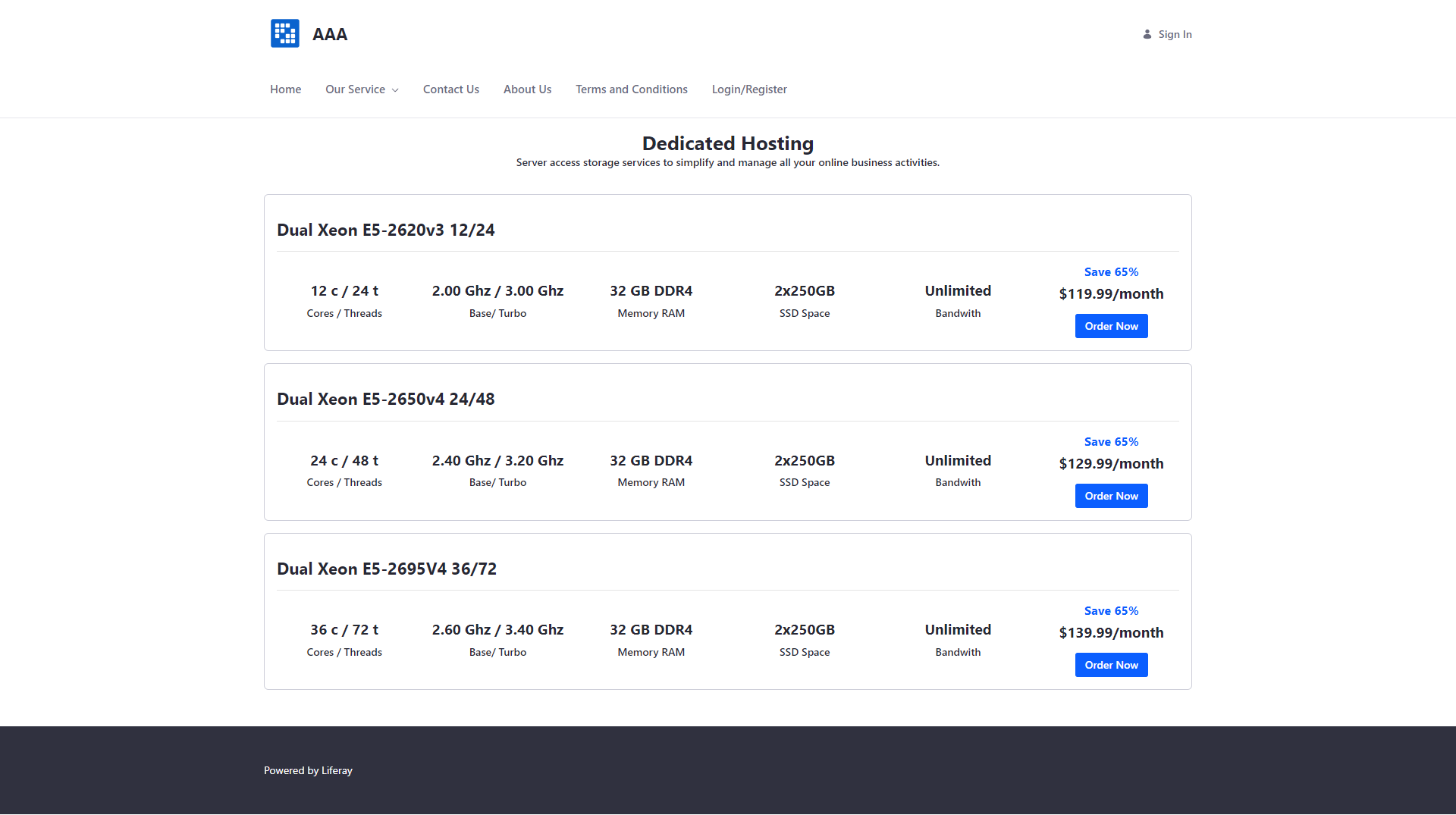
Cloud Hosting Page



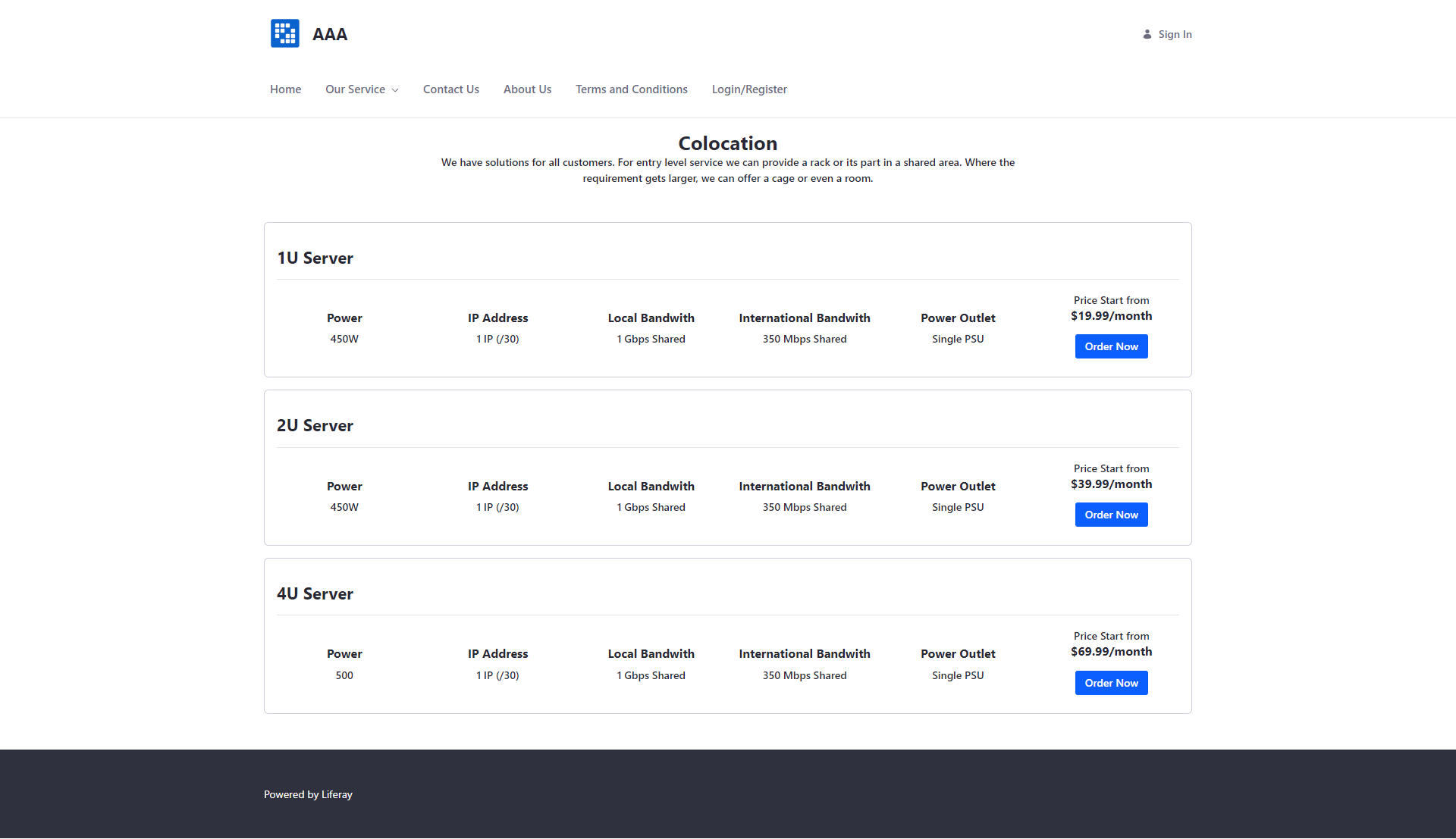
VPS Hosting Page



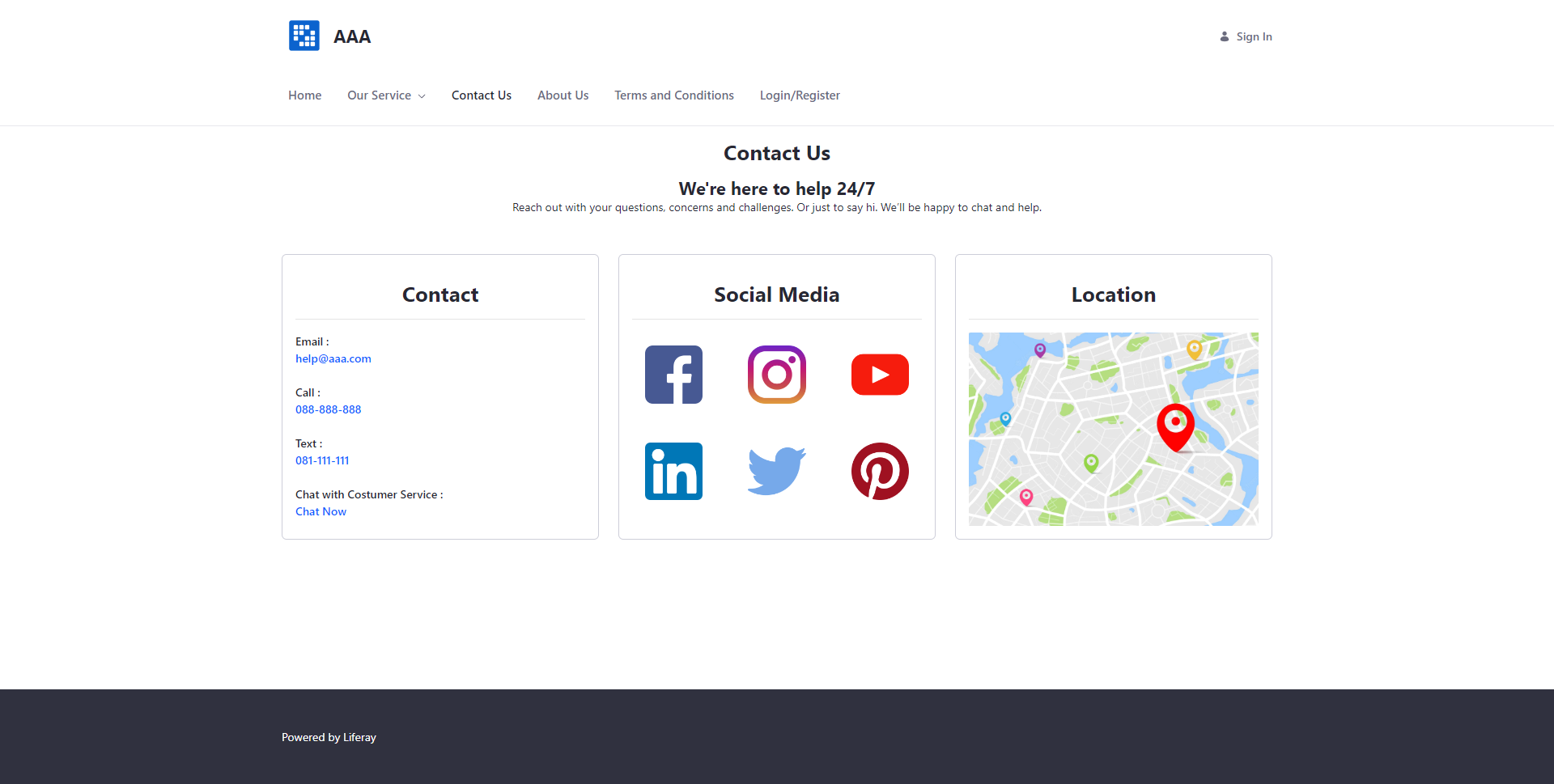
Dedicated Hosting Page



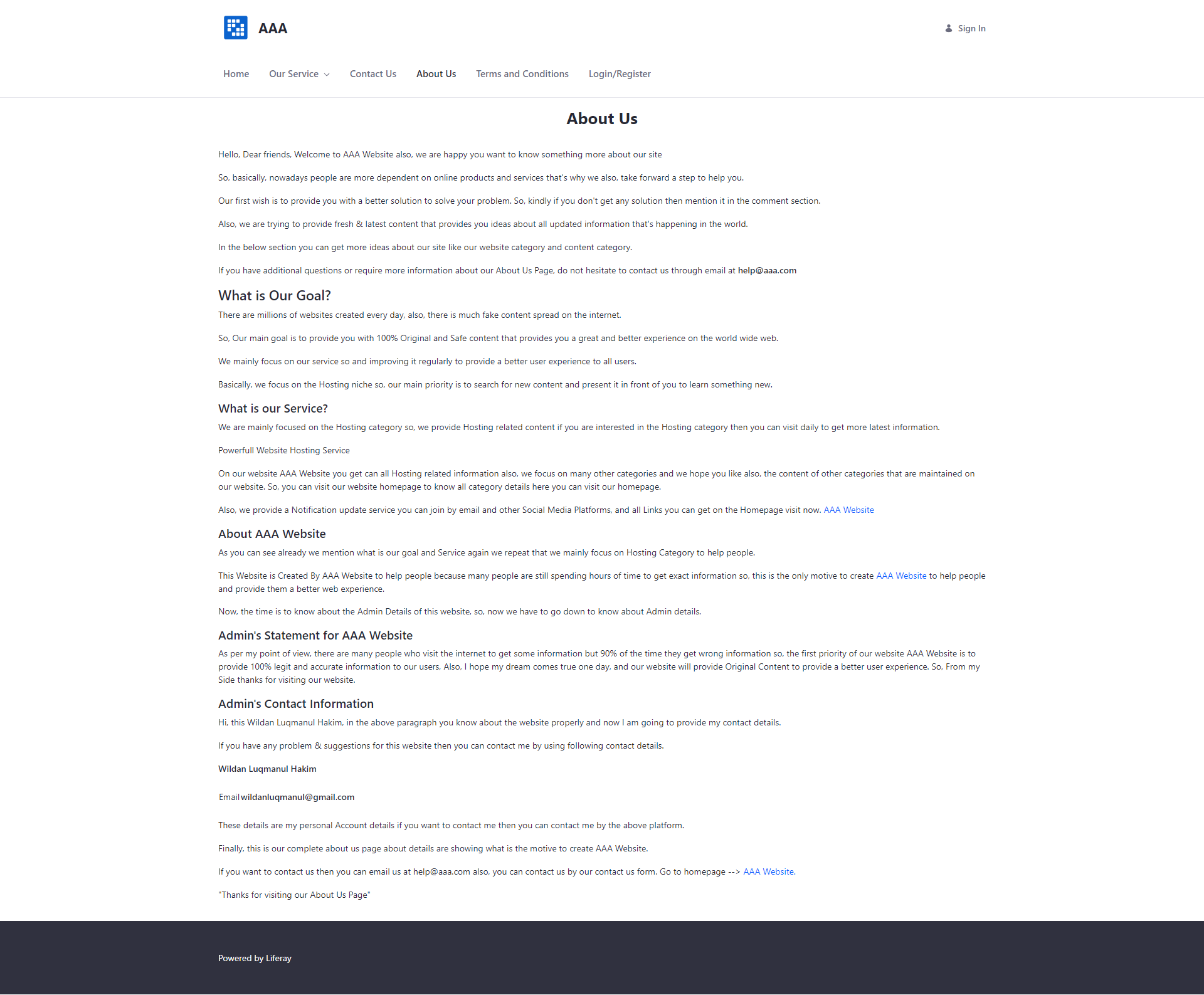
Colocation Service



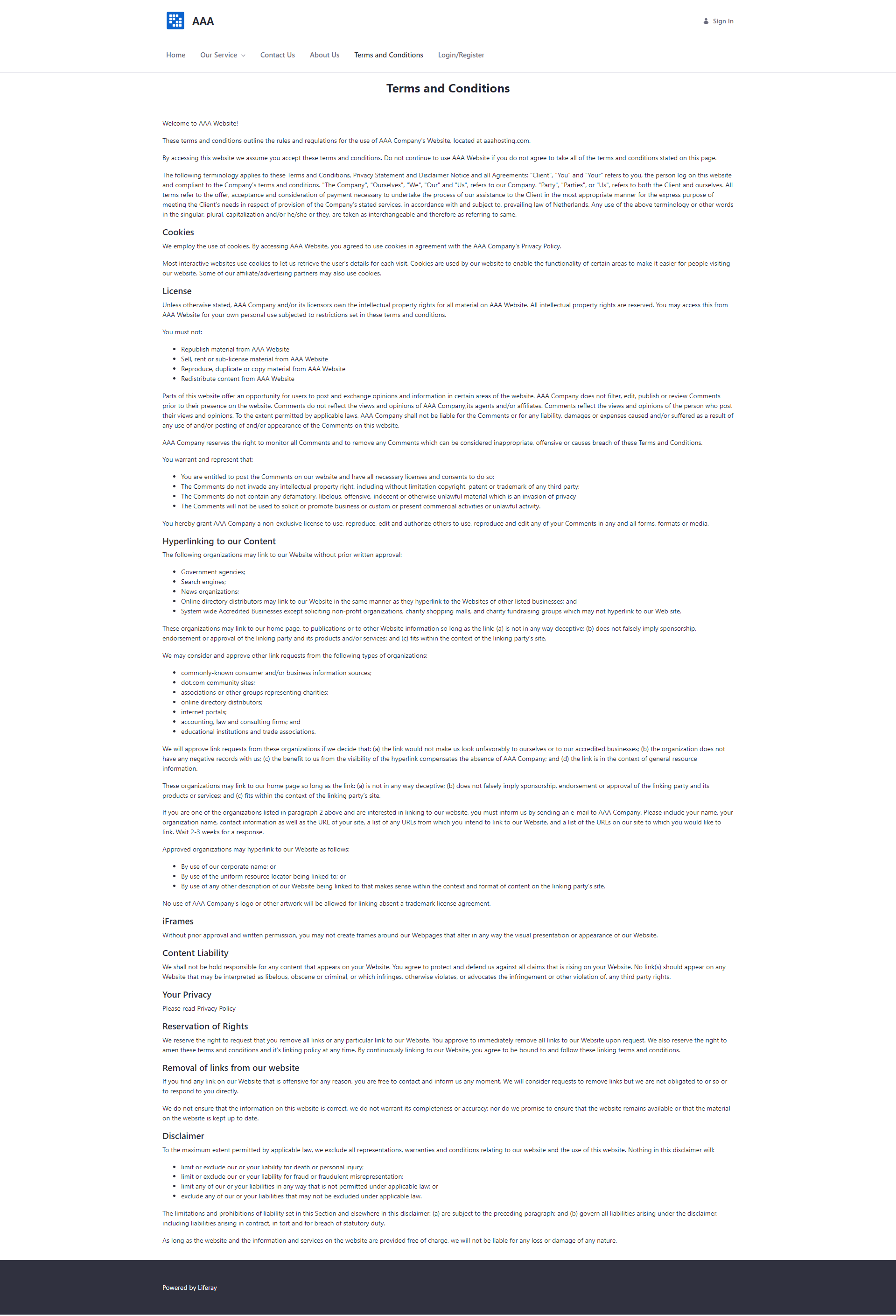
Contact Us Page



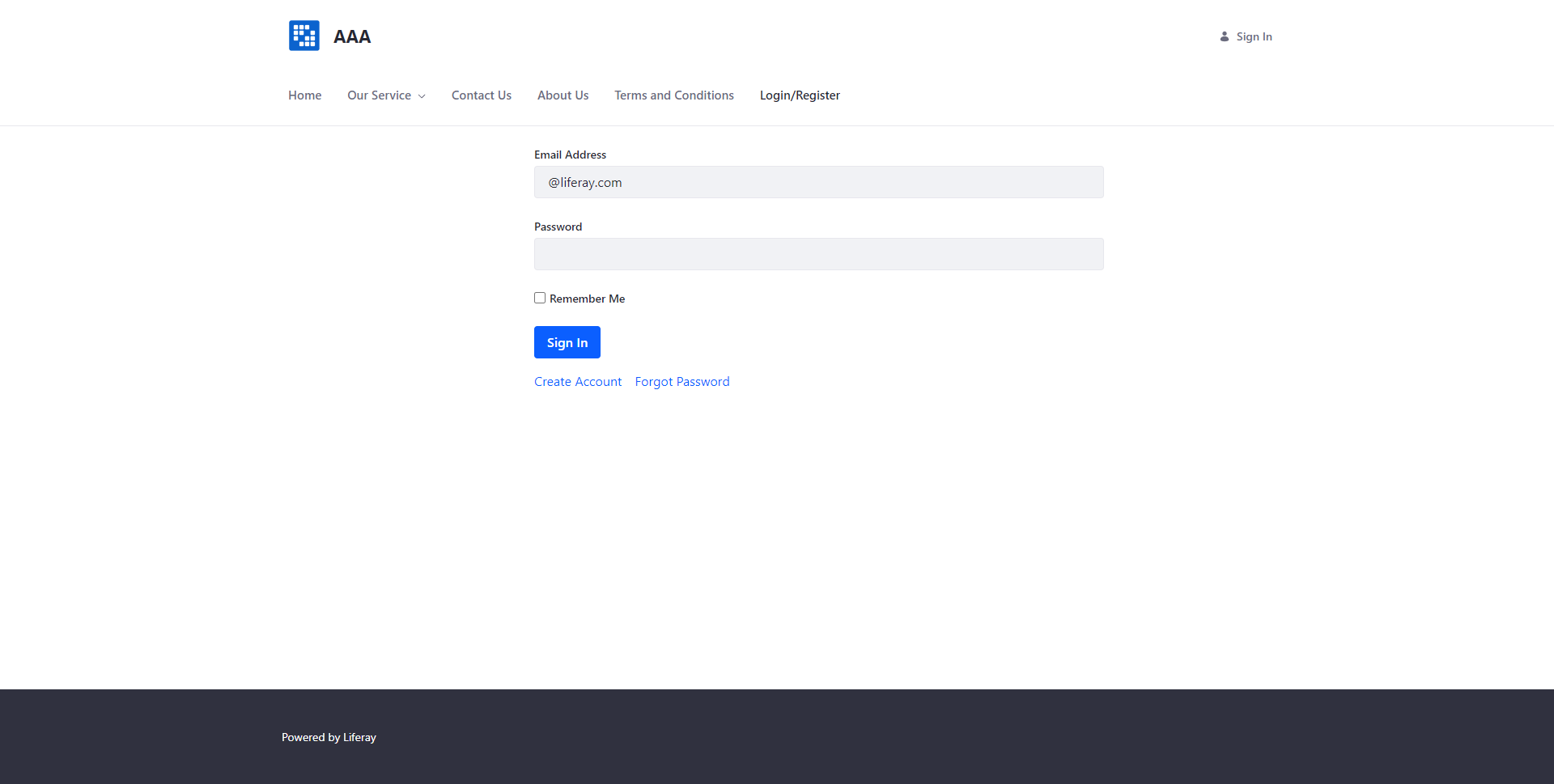
About Us Page



Terms and Condition Page



Login Page



Register Page

