INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT027-3-3-EPDA

ENTERPRISE PROGRAMMING FOR DISTRIBUTED APPLICATION

APU3F2311CS(AI)

Name: LOW SIM CHUAN
TP Number: TP065697

INSTRUCTIONS TO CANDIDATES:

1 **Submit your assignment at the administrative counter.**

2 **Students are advised to underpin their answers with the use of references (cited using the American Psychological Association (APA) Referencing).**

3 **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**

4 **Cases of plagiarism will be penalized.**

5 **The assignment should be bound in an appropriate style (comb bound or stapled).**

6 **Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**

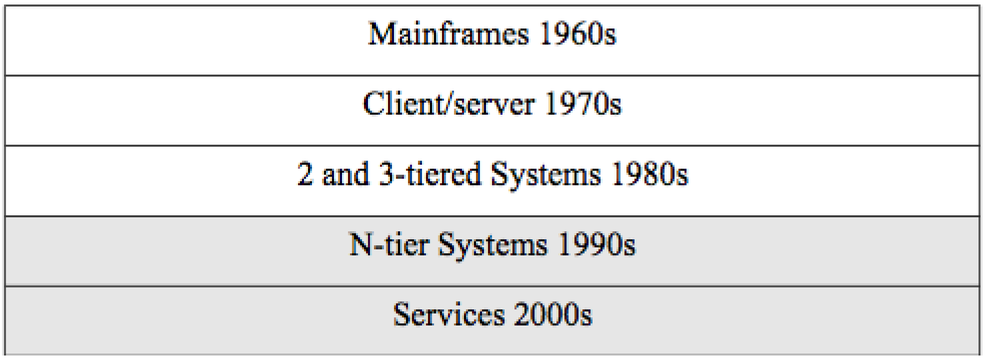
7 **You must obtain 50% overall to pass this module.**

Contents

Part A: Evaluation Report/ Research

(a) Brief history on distributed computing and discussion on architectural evolution of distributed computing.

The history of distributed computing can be summarized by the following timeline:



Mainframes 1960s
Client/server 1970s
2 and 3-tiered Systems 1980s
N-tier Systems 1990s
Services 2000s

Figure 1: Wang, J.W. (n.d.). Timeline for distributed systems. umbc.edu. https://userpages.umbc.edu/~jianwu/is651/651book/is651-strapdown.php?f=is651-Chapter02.md

In the 1960s, computers were mainframe systems, and the connection between mainframe systems were done over dedicated connections (Wright, 2021). *Figure 2* shows the architecture of communication between terminals and the mainframe host through the low-speed access lines, and the mainframe host contains all the applications needed by the users (master, 2019). Such way of connecting computers is fault-prone and highly centralized, that's why U.S. military was thinking about creating a fault-tolerant computer network (Wright, 2021). The researchers began experimenting with idea of sharing resources across multiple computers (Kumar S. , 2023) and the system must be more fault-tolerant, that is operating without needing the dedicated connections (Kanade, 2023). Most of the distributed systems at that time were used for research and development (Landscape, 2023). For example, APRANET was a distributed system invented for academic and research purposes. The modern computer networks are still using plenty of protocols developed for APRANET (Wright, 2021). In 1966, APRANET allowed communications between hosts and passing of messages between hosts within the same network (Wright, 2021).

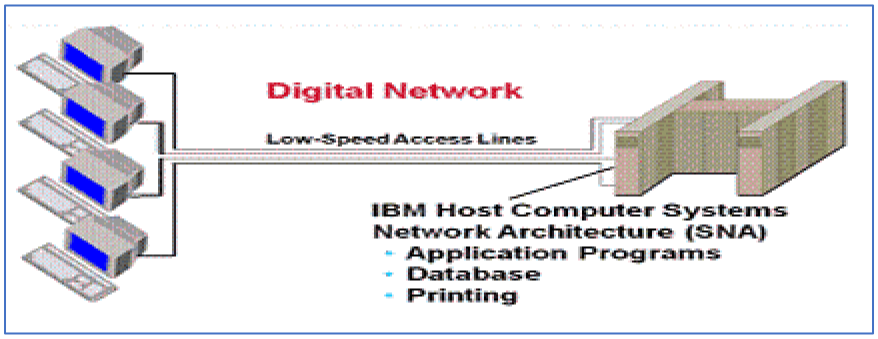


Figure 2: master (2019). Architecture of communication between terminals and the mainframe. E-tutes. https://e-tutes.com/lesson1/1960-s-1970-s-communication/

In the 1970s, minicomputers were smaller computers such as workstations or PCs. The existence of Ethernet made connections between minicomputers feasible, forming a local-area network (LAN). The LAN allows minicomputers to communicate as client and server, a client is a requestor process (not a machine) and the server is a responder process (not a machine or host) where both must have CPUs in a 2-tier architecture (Wang, n.d.). The basic TCP/IP stack was formalized in 1978, allowing the communication between different networks (Wang, n.d.).

In 1980s, the advent of IBM PC allows everyone to possess a computer, and the computers could be networked as a LAN through the operating system NetWare (Wang, n.d.). Mainframe and minicomputers used to connect to APRANET, in 1982s it's LANs connecting to APANET instead using TCP/IP (Wang, n.d.). *Figure 3* shows a 2-tier system, which is about several PCs connecting to the NetWare server through 2 network interface cards, typing commands in any PC can retrieve corresponding programs from the NetWare server and run the programs on the PC. In the late 1980s, the invention of bridges and routers created 3-tier systems, which is about a middleware (application server) between client and server offering services to the distributed systems. One of the 3-tier systems developed during the 1980s until 1990s is the World Wide Web, where its architecture can be represented by *Figure 4* (Silva, 2021).

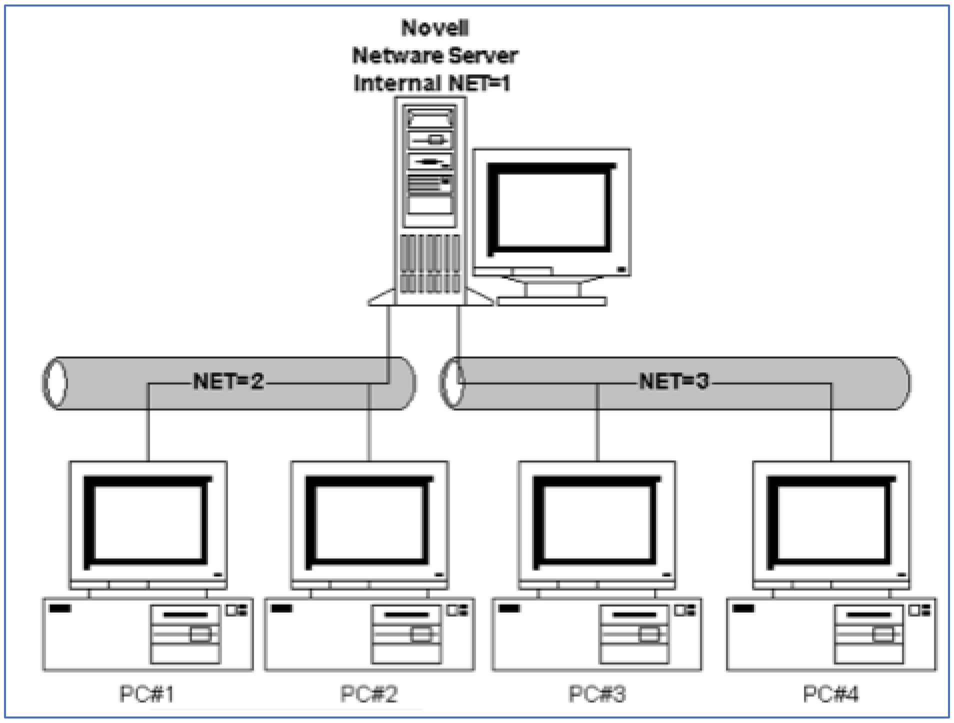


Figure 3: Wang, J.W. (n.d.). A Netware network. umbc.edu. https://userpages.umbc.edu/~jianwu/is651/651book/is651-strapdown.php?f=is651-Chapter02.md

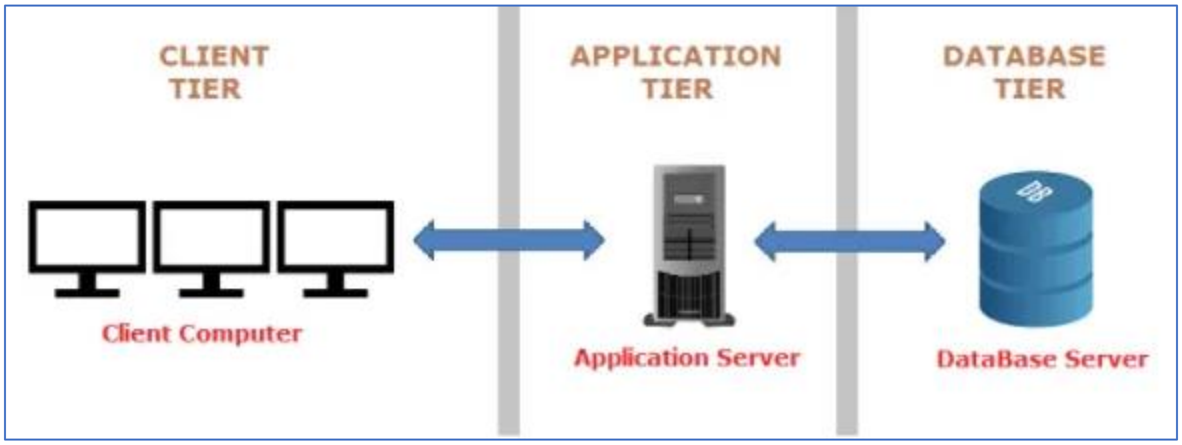
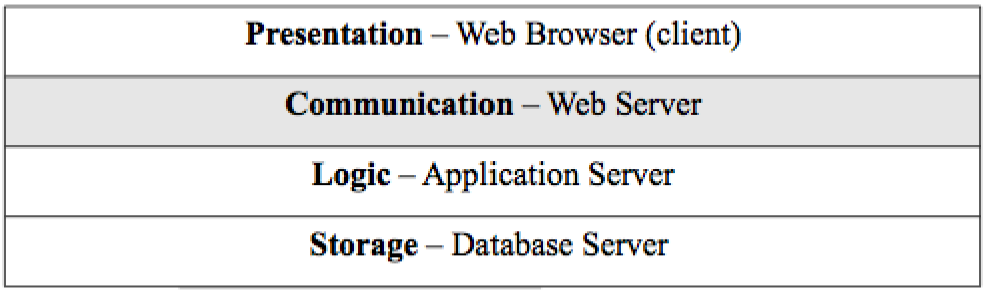


Figure 4: Silva, H.S. (2021). Architecture of WWW. medium. https://silvahansini.medium.com/evolution-of-client-computing-architecture-70e9097d9b96

In the 1990s, APRANET was replaced by the cheaper Internet (as opposed to the past WANs). The presence of the Internet allowed everyone to access the WWW. WWW is a collection of documents linked by URLs, the connection between Web server and computers is based on HTTP protocol thus the contents can be presented to the users using HTML. The N-tier architecture is an architecture that consists of more than 3 layers. *Figure 5* shows the N-tier architecture for web-based systems, where the names on the left are the logical representation of the tiers while the names on the right are the physical representation of the tiers.



Presentation – Web Browser (client)
Communication – Web Server
Logic – Application Server
Storage – Database Server

Figure 5: Wang, J.W. (n.d.). N-tier architecture for web-based systems. Umbc.edu. https://userpages.umbc.edu/~jianwu/is651/651book/is651-strapdown.php?f=is651-Chapter02.md

In the 2000s, service-oriented architecture (SOA) came out and it is a set of design principles. The eight principles of SOA are standard service contracts, loose coupling, abstraction, reusability, autonomy, statelessness, discoverability, and composability (Park, 2022). By adopting SOA, a set of services can be integrated into either one or multiple computers to form a distributed system (Fernando, 2018). SOA allows applications to be assembled by using services in the network by the service consumers as shown in *Figure 6*, and those services can be gotten through network calls over the internet (SarthakG, 2023). *Figure 7* shows the components and subcomponents of the SOA. XML web services were implemented based on SOA, and XML interfaces can be implemented in any programming framework due to the SOA principles being adhered to in XML web services. Other than that, cloud computing, mobile computing, and IoT are also based on SOA (Lindsay, Gill, Smirnova, & Garraghan, 2020). Cloud computing uses SOA to allows multiple service entities to share information through a single system, and easier to deploy Software as a Service (SaaS) in a cloud platform (hitechnectar, n.d.).

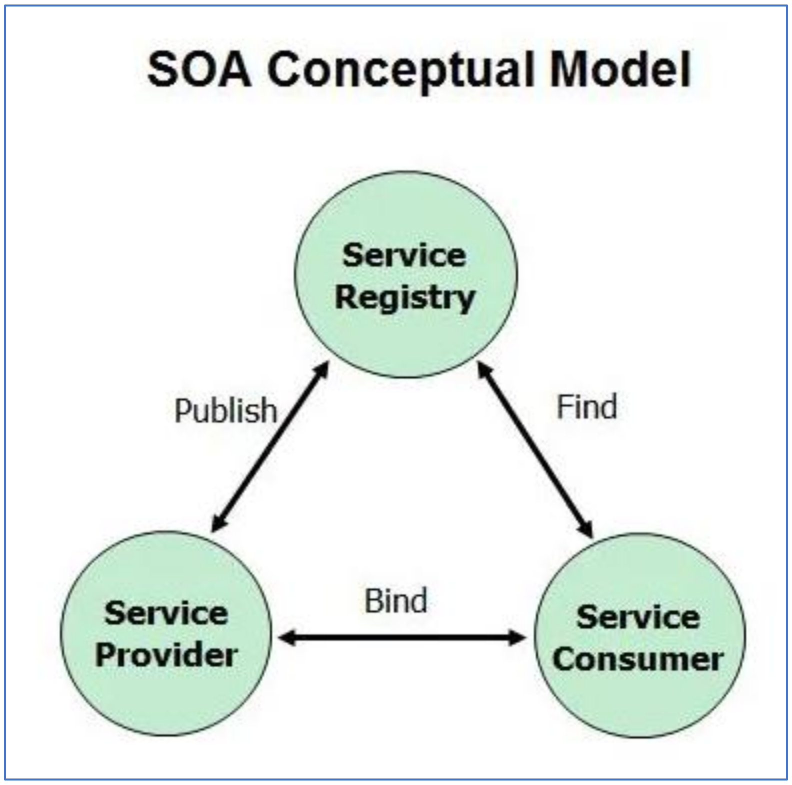


Figure 6: Fernando, C. (2018). SOA Conceptual Model. medium. https://medium.com/microservices-learning/the-evolution-of-distributed-systems-fec4d35beffd

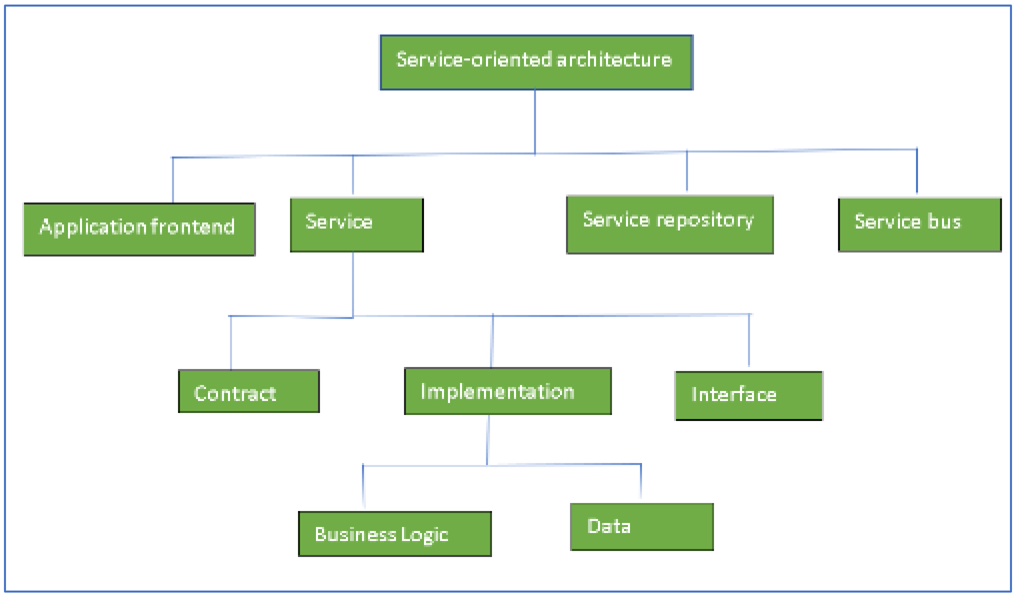
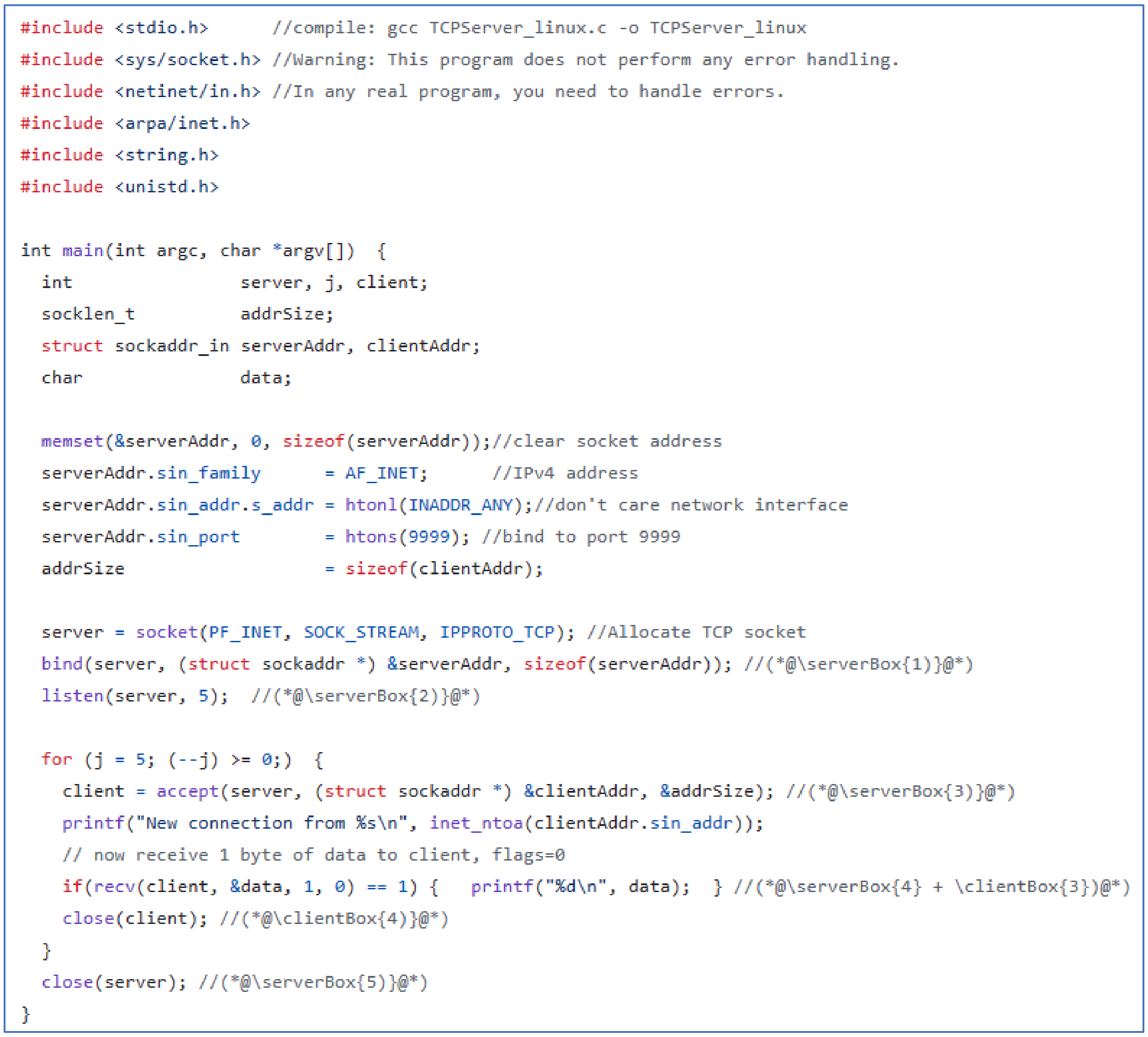


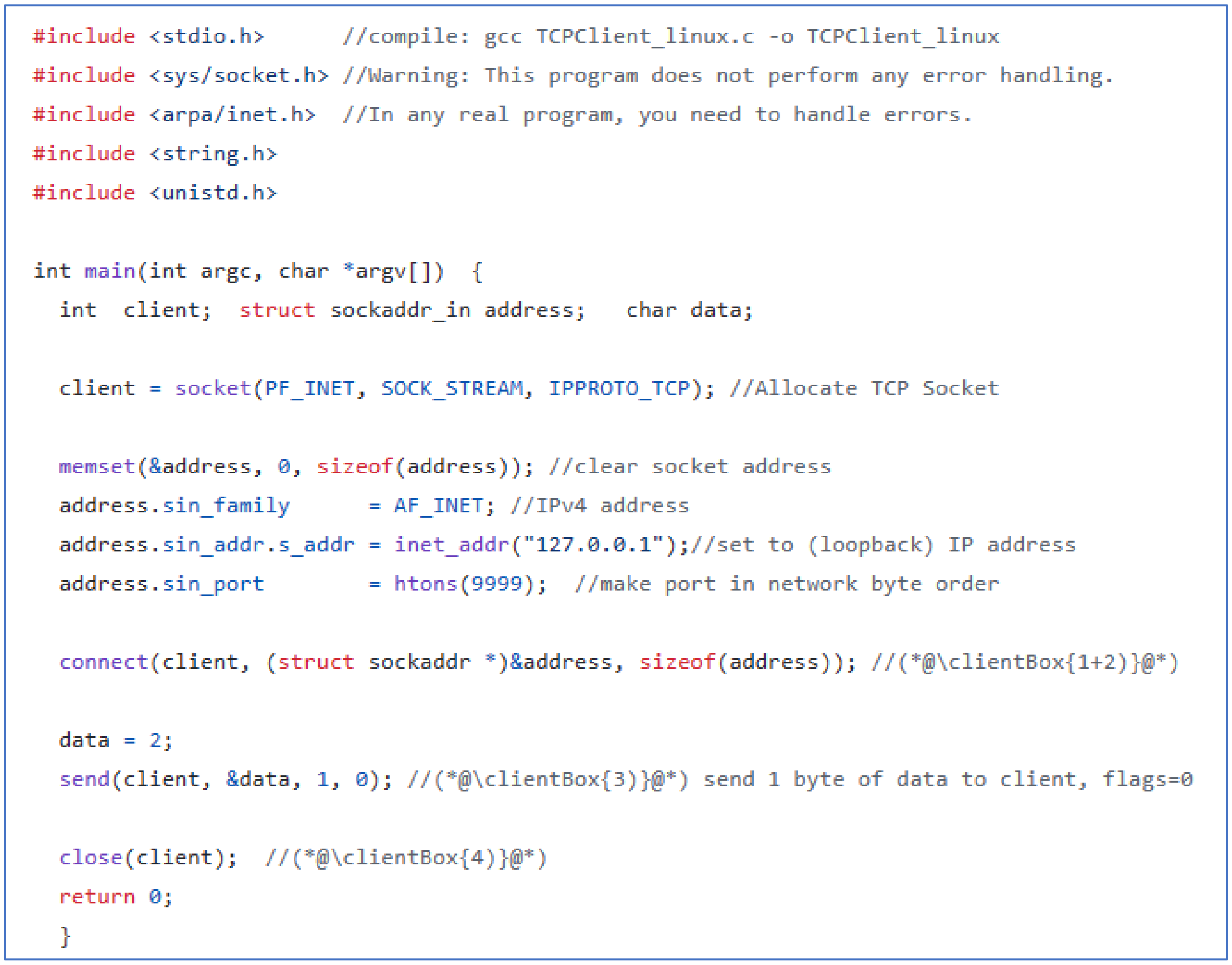
Figure 7: SarthakG (2023). Components and subcomponents of SOA. GeeksforGeeks. https://www.geeksforgeeks.org/service-oriented-architecture/

Distributed computing is everywhere, and the most straightforward example is the Internet hosting the World Wide Web (thomasWeise, 2016). Other than that, a lot of enterprise computing systems are also distributed, connecting different devices from different departments and ensuring efficient communication among them. The concepts of distributed computing can be implemented by using various high-level programming languages such as C, Java, Python, etc. The figures below show the sample codes about how to create TCP Server and TCP Client using C programming language. The TCP server listens on port 9999 of the local host, then it accepts 5 incoming connections, eventually terminates. The TCP client sends one byte with value 2 to the TCP server, eventually terminates.



```c
#include <stdio.h>      //compile: gcc TCPServer_linux.c -o TCPServer_linux
#include <sys/socket.h> //Warning: This program does not perform any error handling.
#include <netinet/in.h> //In any real program, you need to handle errors.
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[])  {
  int                server, j, client;
  socklen_t          addrSize;
  struct sockaddr_in serverAddr, clientAddr;
  char               data;

  memset(&serverAddr, 0, sizeof(serverAddr));//clear socket address
  serverAddr.sin_family      = AF_INET;      //IPv4 address
  serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);//don't care network interface
  serverAddr.sin_port        = htons(9999); //bind to port 9999
  addrSize                   = sizeof(clientAddr);

  server = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); //Allocate TCP socket
  bind(server, (struct sockaddr *) &serverAddr, sizeof(serverAddr)); //(*@\serverBox{1}@*)
  listen(server, 5);  //(*@\serverBox{2}@*)

  for (j = 5; (--j) >= 0;)  {
    client = accept(server, (struct sockaddr *) &clientAddr, &addrSize); //(*@\serverBox{3}@*)
    printf("New connection from %s\n", inet_ntoa(clientAddr.sin_addr));
    // now receive 1 byte of data to client, flags=0
    if(recv(client, &data, 1, 0) == 1) {   printf("%d\n", data);  } //(*@\serverBox{4} + \clientBox{3})@*)
    close(client); //(*@\clientBox{4}@*)
  }
  close(server); //(*@\serverBox{5}@*)
}
```

Figure 8: thomasWeise (2016). Sample codes for creating TCP Server. github. https://github.com/thomasWeise/distributedComputingExamples/blob/master/sockets/c/TCPServer_linux.c



```c
#include <stdio.h>      //compile: gcc TCPClient_linux.c -o TCPClient_linux
#include <sys/socket.h> //Warning: This program does not perform any error handling.
#include <arpa/inet.h>  //In any real program, you need to handle errors.
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[])  {
  int  client;  struct sockaddr_in address;   char data;

  client = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); //Allocate TCP Socket

  memset(&address, 0, sizeof(address)); //clear socket address
  address.sin_family      = AF_INET; //IPv4 address
  address.sin_addr.s_addr = inet_addr("127.0.0.1");//set to (loopback) IP address
  address.sin_port        = htons(9999);  //make port in network byte order

  connect(client, (struct sockaddr *)&address, sizeof(address)); //(*@\clientBox{1+2)}@*)

  data = 2;
  send(client, &data, 1, 0); //(*@\clientBox{3)}@*) send 1 byte of data to client, flags=0

  close(client);  //(*@\clientBox{4)}@*)
  return 0;
  }
```

Figure 9: thomasWeise (2016). Sample codes for creating TCP Client. github. https://github.com/thomasWeise/distributedComputingExamples/blob/master/sockets/c/TCPClient_linux.c

(b) Brief overview of various types of enterprise application and architecture.

An enterprise application (EA) is a software application developed to simplify business processes by including some common features such as invoicing systems, workflow automation, and finance management (JACINTO, 2023).

Enterprise application architecture is a group of strategies and principles used by businesses to define the way of constructing programs (Sharma, n.d.). The figure below shows 7 types of enterprise application architecture:

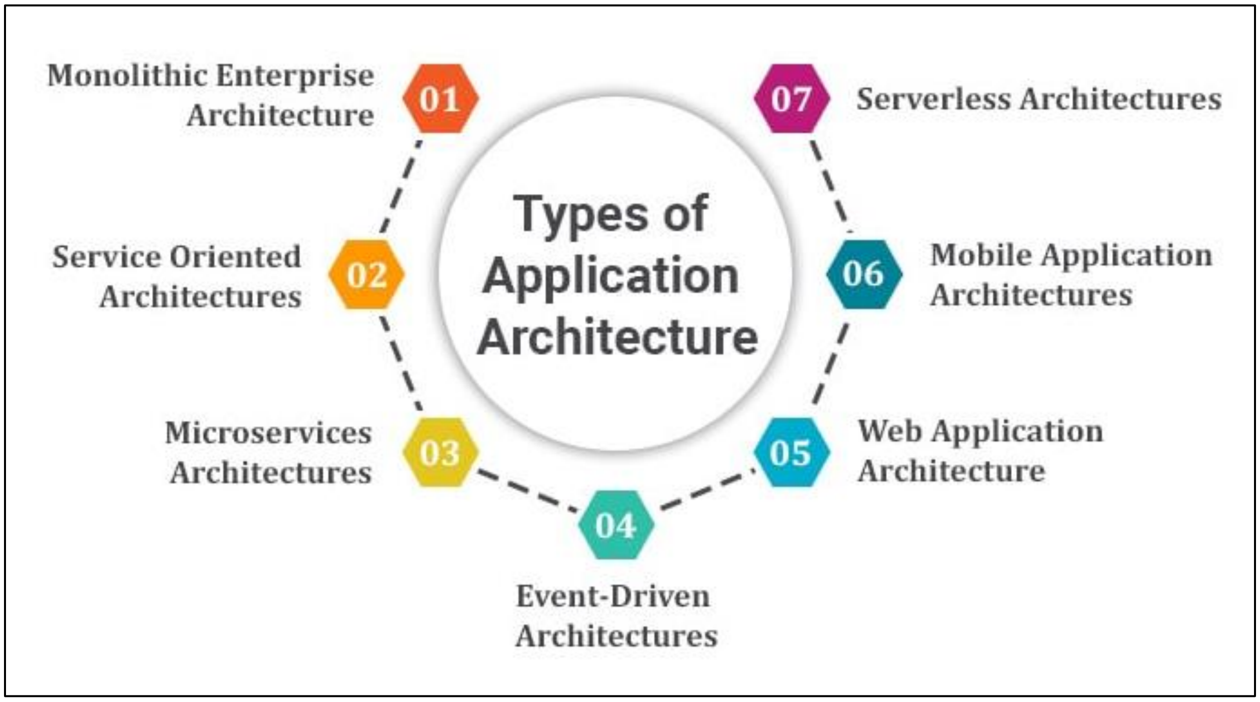

Types of Application Architecture
01 Monolithic Enterprise Architecture
02 Service Oriented Architectures
03 Microservices Architectures
04 Event-Driven Architectures
05 Web Application Architecture
06 Mobile Application Architectures
07 Serverless Architectures

Figure 10: Hernandez, R.D. (2021). Types of Enterprise Application Architecture. tatvasoft. https://www.tatvasoft.com/outsourcing/2022/10/enterprise-application-architecture.html

Monolithic Enterprise Architecture is an architecture which each component must be compiled and run with its associated components, in other words, the program's components are tightly coupled (Awati & Wigmore, 2022). As shown in the figure below, the components user interface, business layer, and data interface are interconnected and stored on the same codebase. Monolithic architecture is suitable for those simple and lightweight applications. The benefit of using monolithic architecture is that it is easier to debug and test due to little of elements are there.

Its drawback is that programmers find it difficult to understand and modify the large codebase, and it is hard to scale the application.

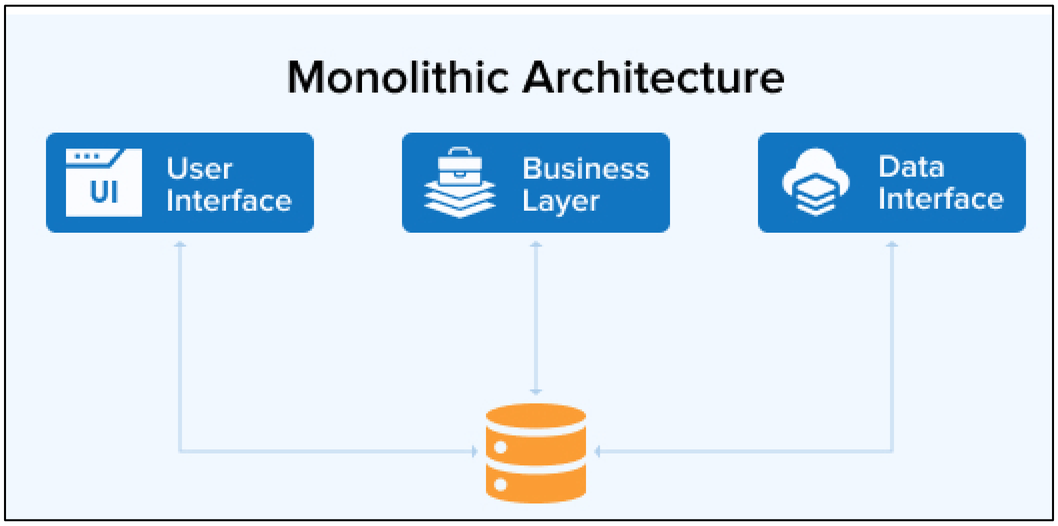


Figure 11: Hernandez, R.D. (2021). Monolithic Architecture. tatvasoft. https://www.tatvasoft.com/outsourcing/2022/10/enterprise-application-architecture.html

Service-Oriented Architecture is to make software components reusable and interoperable via service interfaces (IBM, n.d.). Each service in such architecture contains the code and data needed to execute a business function, and services are loosely coupled. The advantage of adopting such architecture is it is faster to market an application due to the reusability of the services, the developers are not required to rewrite the codes for every new project.

Microservices Architecture is an architecture which makes services to be loosely coupled, so that they can be developed, deployed, and maintained independently (Singh, 2018). The services can communicate with each other through simple APIs, working together to solve a complex business problem. The figure below shows how each service is separate from each other, so that the entire system won't go down if a single service fails.

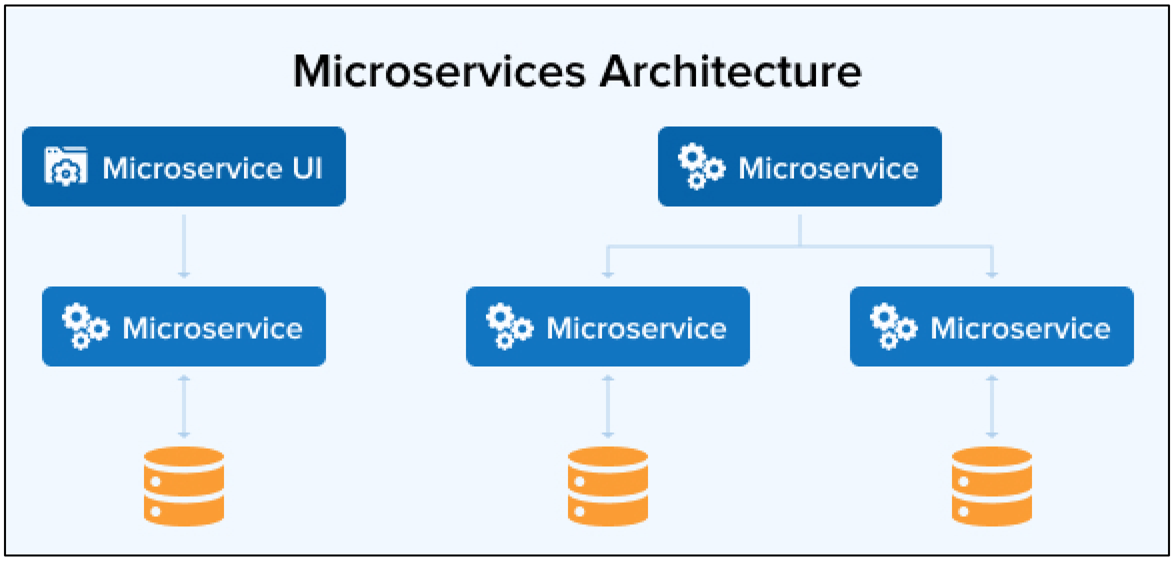


Figure 12: Hernandez, R.D. (2021). Microservices Architecture. tatvasoft. https://www.tatvasoft.com/outsourcing/2022/10/enterprise-application-architecture.html

Event-Driven Architecture responds to events, such as touch of a button, the measurement of a device's temperature, or the scan of a bank card in real-time computing and self-service settings (Awati & Wigmore, 2022). The events are used in communication between services. There are 3 components in event-driven architectures: event producers, event brokers, and event consumers (aws, n.d.). A producer publishes an event to the event broker, then it pushes the events to the event consumers (aws, n.d.).

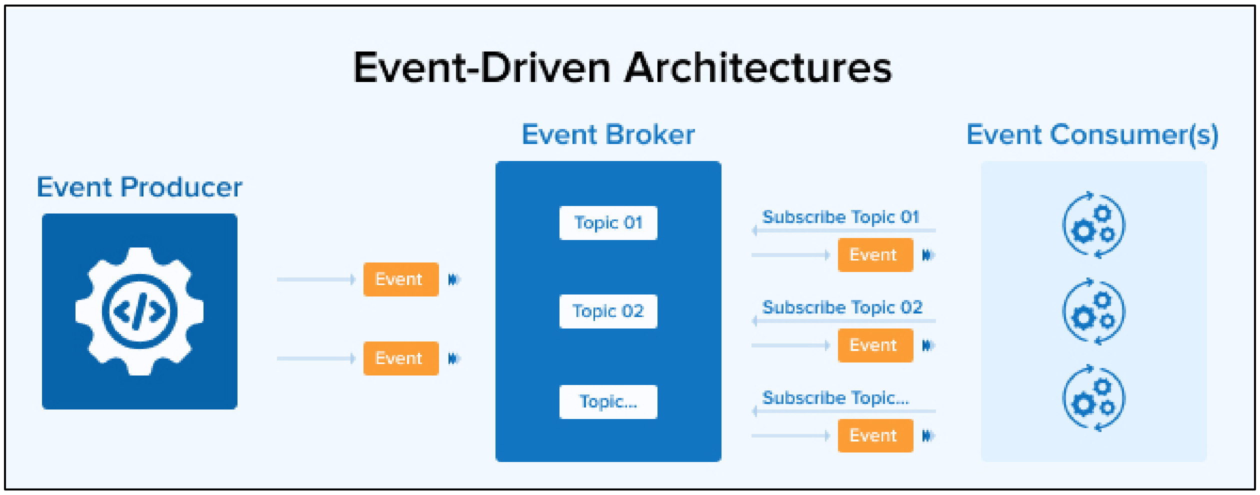


Figure 13: Hernandez, R.D. (2021). Event-Driven Architecture. tatvasoft. https://www.tatvasoft.com/outsourcing/2022/10/enterprise-application-architecture.html

Figure below shows the web application architecture. The users are at front-end, which means they will interact with the web browser to provide inputs, the HTML, CSS, and JavaScript are the programming languages for managing the presentation logic and validating user inputs (William, 2022). Back-end consists of Web Server and database server, a Web server contains all the application/business logic to handle the requests sent from the front-end and send responses to the front-end. Other than that, when the Web server is handling the requests from the front-end, it might want to query the database where all the application data resides at.

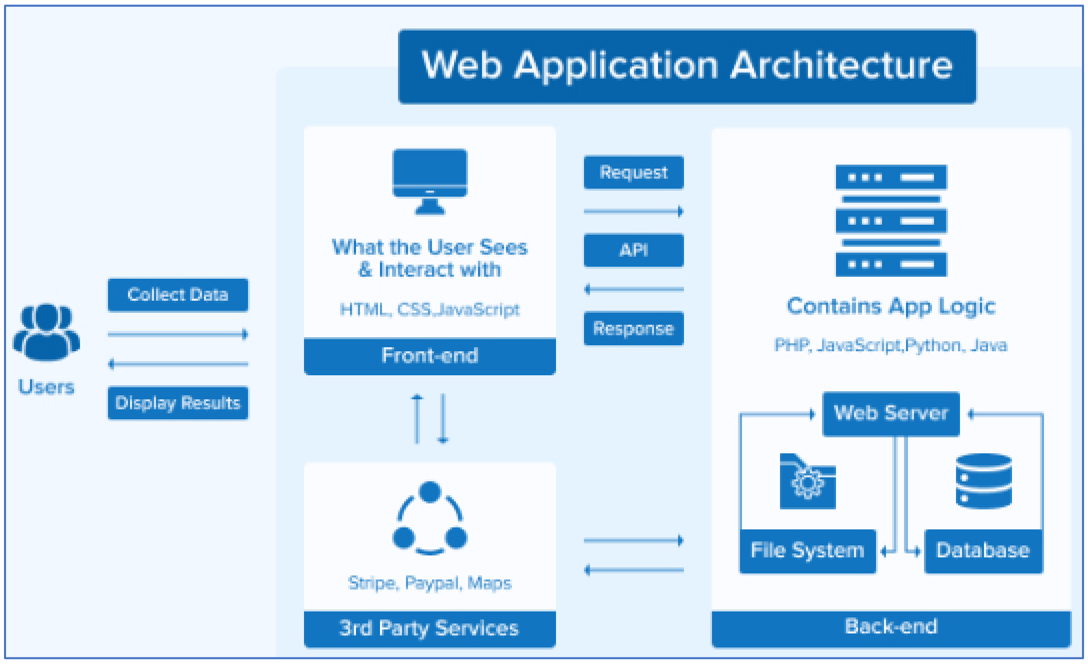


Figure 14: Hernandez, R.D. (2021). Web Application Architecture. tatvasoft. https://www.tatvasoft.com/outsourcing/2022/10/enterprise-application-architecture.html

The figure below shows the architecture of mobile applications. The presentation layer controls how the end users can see the application and interact with the application through user interfaces, the business layer provides a set of rules (business logic) to manage the flow of data, and the data layer is to ensure every data transaction is secure and efficient (Shah, 2023).

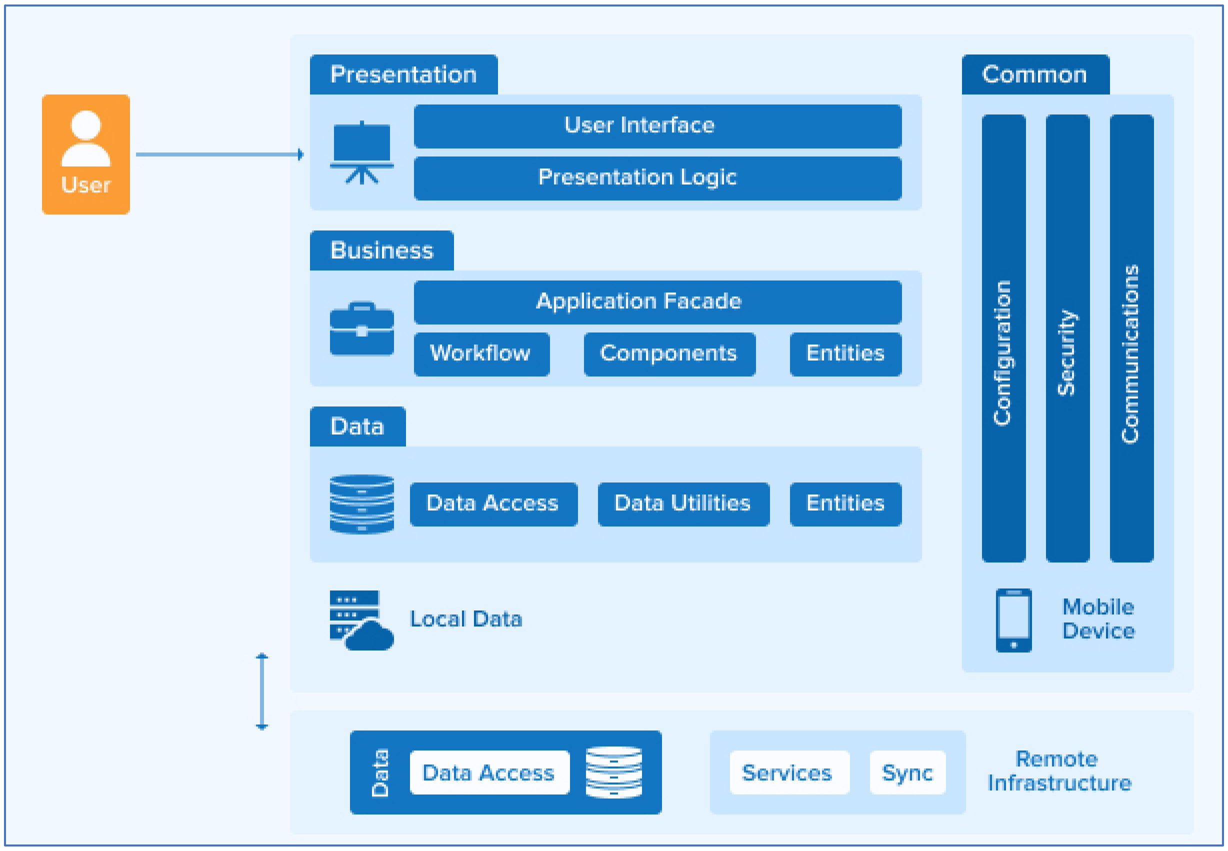


Figure 15: Hernandez, R.D. (2021). Mobile Application Architecture. tatvasoft. https://www.tatvasoft.com/outsourcing/2022/10/enterprise-application-architecture.html

The figure below shows the serverless architecture. By adopting such architecture, the scaling and maintenance of the servers will be done by the cloud-service providers to run the applications, databases, and storage systems. The developers can focus more on developing the products instead of managing and operating servers (amazon, n.d.).

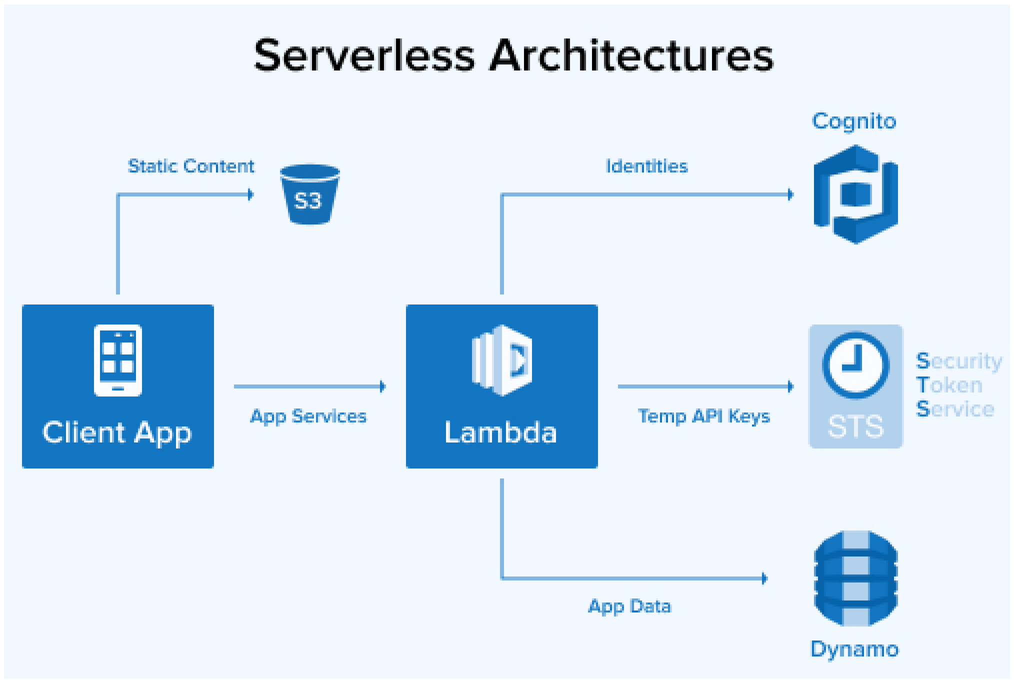
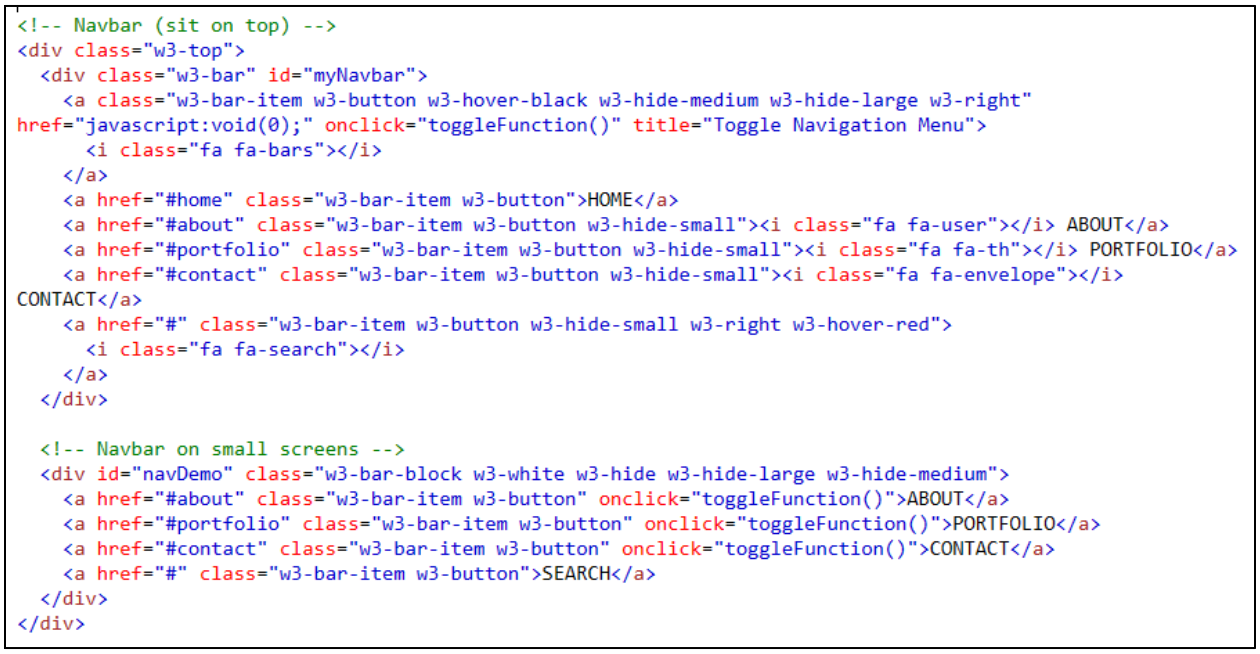


Figure 16: Hernandez, R.D. (2021). Serverless Architecture. tatvasoft. https://www.tatvasoft.com/outsourcing/2022/10/enterprise-application-architecture.html

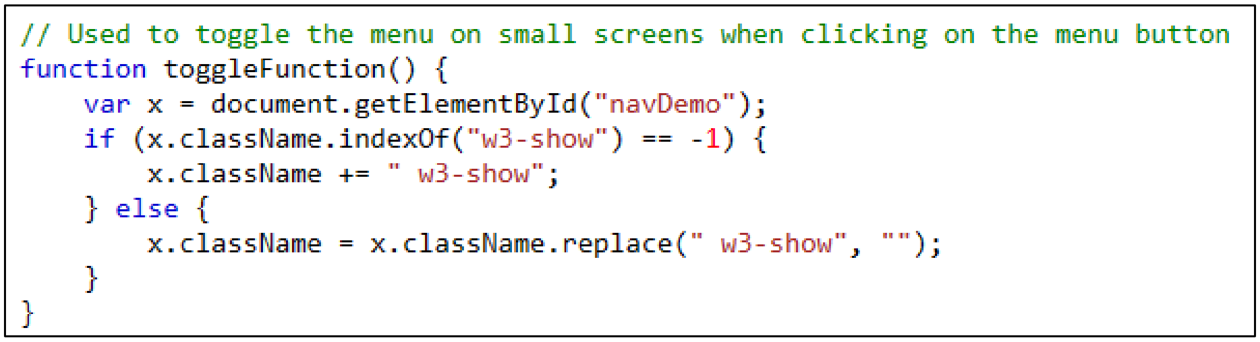
One of the most common types of enterprise application is web application. The front-end is responsible for presentation of user interfaces to the users and allowing users to interact with the user interfaces. The common programming languages to implement the front-end are HTML, CSS, JavaScript, as shown in the figure below:



```html
<!-- Navbar (sit on top) -->
<div class="w3-top">
  <div class="w3-bar" id="myNavbar">
    <a class="w3-bar-item w3-button w3-hover-black w3-hide-medium w3-hide-large w3-right"
href="javascript:void(0);" onclick="toggleFunction()" title="Toggle Navigation Menu">
      <i class="fa fa-bars"></i>
    </a>
    <a href="#home" class="w3-bar-item w3-button">HOME</a>
    <a href="#about" class="w3-bar-item w3-button w3-hide-small"><i class="fa fa-user"></i> ABOUT</a>
    <a href="#portfolio" class="w3-bar-item w3-button w3-hide-small"><i class="fa fa-th"></i> PORTFOLIO</a>
    <a href="#contact" class="w3-bar-item w3-button w3-hide-small"><i class="fa fa-envelope"></i>
CONTACT</a>
    <a href="#" class="w3-bar-item w3-button w3-hide-small w3-right w3-hover-red">
      <i class="fa fa-search"></i>
    </a>
  </div>

  <!-- Navbar on small screens -->
  <div id="navDemo" class="w3-bar-block w3-white w3-hide w3-hide-large w3-hide-medium">
    <a href="#about" class="w3-bar-item w3-button" onclick="toggleFunction()">ABOUT</a>
    <a href="#portfolio" class="w3-bar-item w3-button" onclick="toggleFunction()">PORTFOLIO</a>
    <a href="#contact" class="w3-bar-item w3-button" onclick="toggleFunction()">CONTACT</a>
    <a href="#" class="w3-bar-item w3-button">SEARCH</a>
  </div>
</div>
```

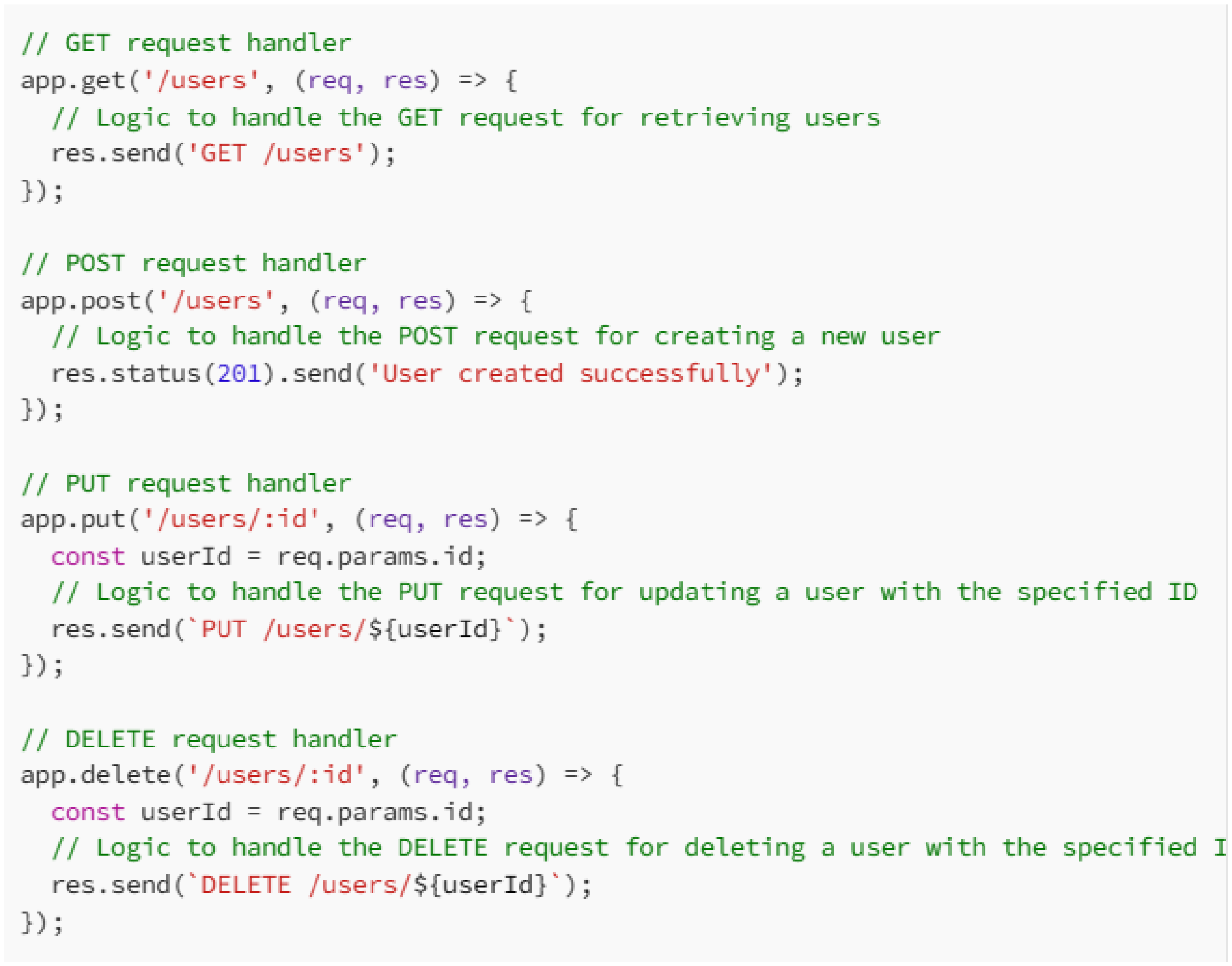
Figure 17: w3schools (n.d.). HTML codes for a navigation bar menu. W3schools. https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_templates_parallax&stacked=h

The figure above shows the HTML codes for creating a navigation bar menu, consisting of HOME, ABOUT, PORTFOLIO, and CONTACT items for users to get redirected to the corresponding section of the webpage. The figure below shows the JavaScript function to redirect the users to the corresponding section of the webpage after the users click on any item of the navigation menu bar.



```javascript
// Used to toggle the menu on small screens when clicking on the menu button
function toggleFunction() {
    var x = document.getElementById("navDemo");
    if (x.className.indexOf("w3-show") == -1) {
        x.className += " w3-show";
    } else {
        x.className = x.className.replace(" w3-show", "");
    }
}
```

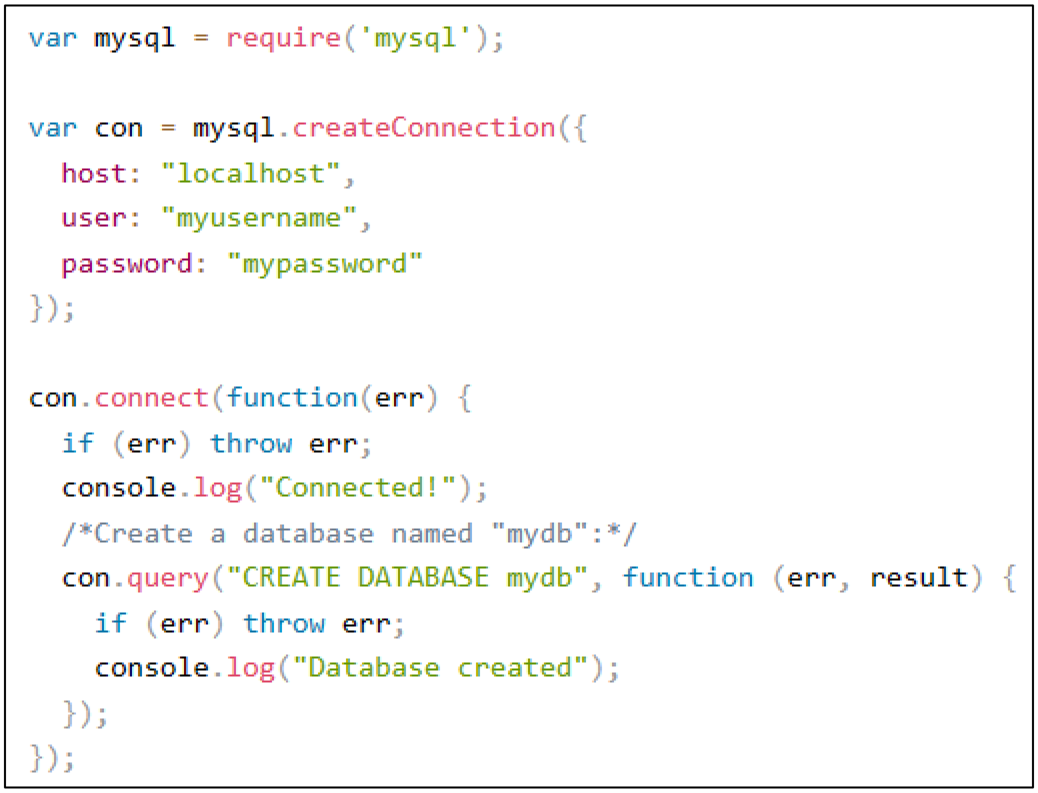
Figure 18: w3schools (n.d.). JavaScript codes for handling users clicking on any item of the navigation bar menu. W3schools. https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_templates_parallax&stacked=h



```
// GET request handler
app.get('/users', (req, res) => {
  // Logic to handle the GET request for retrieving users
  res.send('GET /users');
});

// POST request handler
app.post('/users', (req, res) => {
  // Logic to handle the POST request for creating a new user
  res.status(201).send('User created successfully');
});

// PUT request handler
app.put('/users/:id', (req, res) => {
  const userId = req.params.id;
  // Logic to handle the PUT request for updating a user with the specified ID
  res.send(`PUT /users/${userId}`);
});

// DELETE request handler
app.delete('/users/:id', (req, res) => {
  const userId = req.params.id;
  // Logic to handle the DELETE request for deleting a user with the specified I
  res.send(`DELETE /users/${userId}`);
});
```

Figure 19: Korzhenko, V. (2023). JavaScript codes using NodeJS framework to handle HTTP methods. Medium. https://medium.com/@vitaliykorzenkoua/handling-requests-and-responses-in-node-js-36ed725f6944

The figure above shows how to use NodeJS framework in JavaScript to handle various HTTP methods, such as GET, POST, PUT, and DELETE. This is useful when the front-end programming languages have to send some HTTP requests to the back end for some purposes, such as sending GET request for retrieving some desired information, then NodeJS has to handle the GET request and query the database for relevant information, eventually send the information to frontend for presentation to the users. The figure below shows how to use NodeJS to connect to the MySQL database and execute a SQL query to create a database.



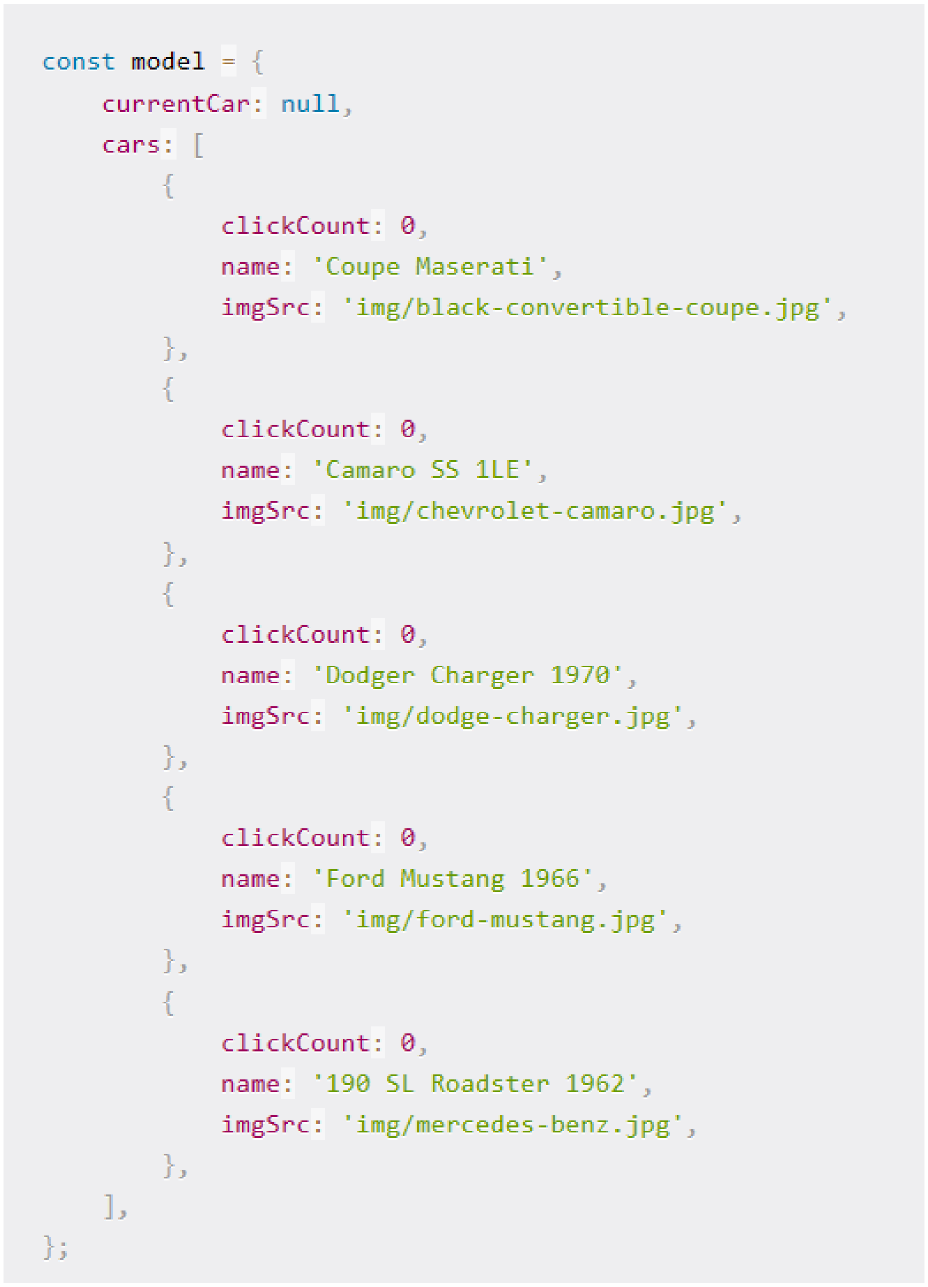
```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "myusername",
  password: "mypassword"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  /*Create a database named "mydb":*/
  con.query("CREATE DATABASE mydb", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

Figure 20: w3schools (n.d.). JavaScript codes using NodeJS framework to query the MySQL database. W3schools. https://www.w3schools.com/nodejs/nodejs_mysql_create_db.asp

(c) Study Model-View-Controller (MVC) pattern by critically analysing its role in the Distributed Web Application.

MVC consists of three components, that are Model, View, and Controller. The MVC was proposed by Trygve Reenskaug, which was initially for development of desktop application GUIs (Hernandez, 2021). Nowadays MVC is widely applied in developing web applications due to its maintainability and scalability (Hernandez, 2021).

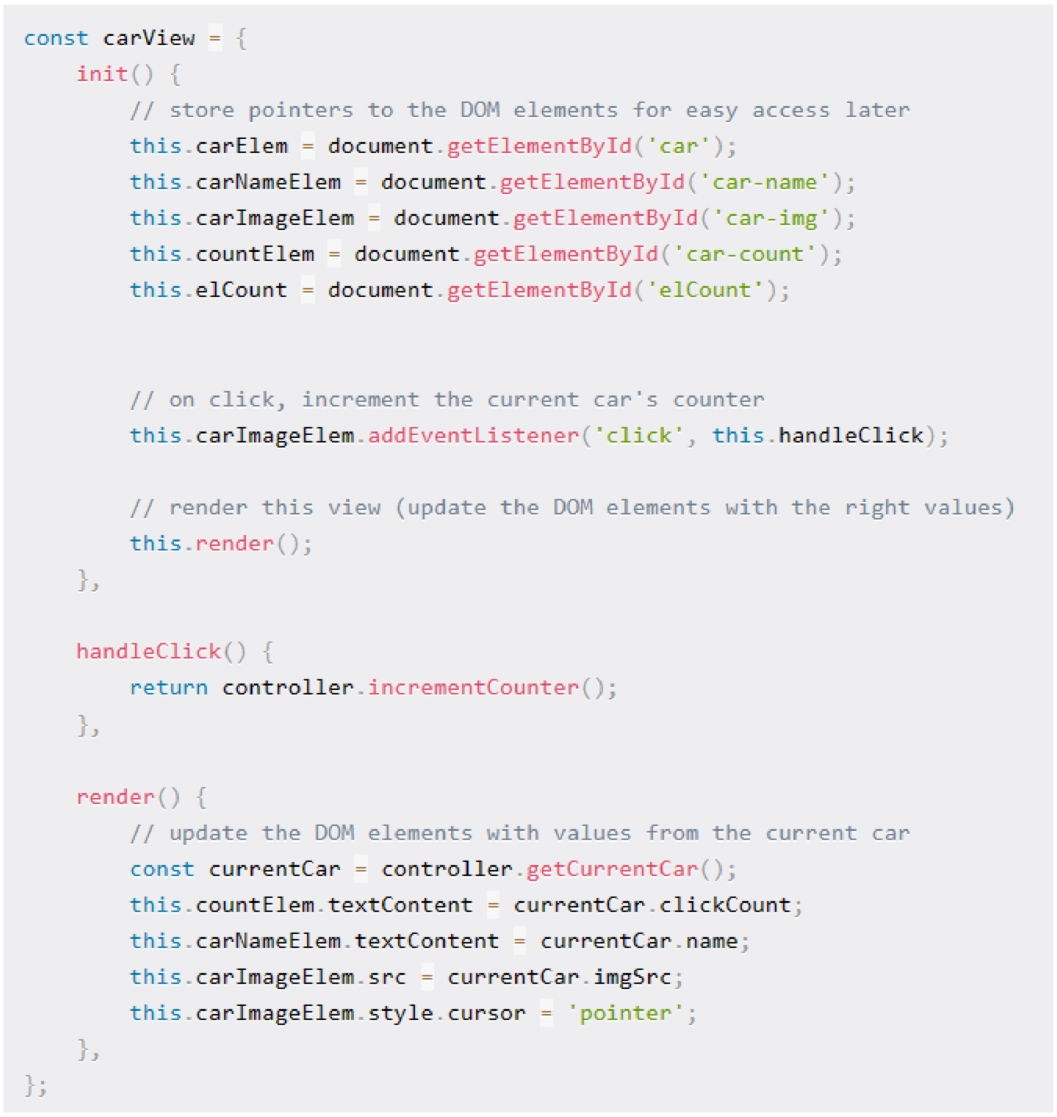
Model is the backend that contains all the data logic, such as what are the attributes of the objects and how many objects should be defined in a model (Hernandez, 2021). In distributed web applications, the data could originate from an API, a JSON object, or a database, and the Model is to manage the data in terms of objects. The data objects can be easily understood by other components. In a distributed web application, a model would interact with the database server to retrieve and store the data in terms of data objects. The figure below shows the sample code for defining a model for Car Clicker web application, the *currentCar* stores the reference of the car being displayed to the users, the *cars* stores an array of cars where each car has the count of click, its name, and its picture.



```
const model = {
    currentCar: null,
    cars: [
        {
            clickCount: 0,
            name: 'Coupe Maserati',
            imgSrc: 'img/black-convertible-coupe.jpg',
        },
        {
            clickCount: 0,
            name: 'Camaro SS 1LE',
            imgSrc: 'img/chevrolet-camaro.jpg',
        },
        {
            clickCount: 0,
            name: 'Dodger Charger 1970',
            imgSrc: 'img/dodge-charger.jpg',
        },
        {
            clickCount: 0,
            name: 'Ford Mustang 1966',
            imgSrc: 'img/ford-mustang.jpg',
        },
        {
            clickCount: 0,
            name: '190 SL Roadster 1962',
            imgSrc: 'img/mercedes-benz.jpg',
        },
    ],
};
```

Figure 21: Hernandez, R.D. (2021). Sample Code for a Model. freeCodeCamp. https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

View is to decide all those GUI components in a page, such as how many buttons and drop-down menus in a page to present the data to the users. In a distributed web application, the View would be the HTML, CSS, and JavaScript that runs in the user's browser. The figure below shows the sample code for *carView*, which handles what happens when the click event happens on the car image (that is incrementing the counter of the car), and also what to render to the user if the user has switched the car:



```
const carView = {
    init() {
        // store pointers to the DOM elements for easy access later
        this.carElem = document.getElementById('car');
        this.carNameElem = document.getElementById('car-name');
        this.carImageElem = document.getElementById('car-img');
        this.countElem = document.getElementById('car-count');
        this.elCount = document.getElementById('elCount');


        // on click, increment the current car's counter
        this.carImageElem.addEventListener('click', this.handleClick);

        // render this view (update the DOM elements with the right values)
        this.render();
    },

    handleClick() {
        return controller.incrementCounter();
    },

    render() {
        // update the DOM elements with values from the current car
        const currentCar = controller.getCurrentCar();
        this.countElem.textContent = currentCar.clickCount;
        this.carNameElem.textContent = currentCar.name;
        this.carImageElem.src = currentCar.imgSrc;
        this.carImageElem.style.cursor = 'pointer';
    },
};
```

Figure 22: Hernandez, R.D. (2021). Sample Code for a View. freeCodeCamp. https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

Controller is to pull, modify and provide data to the users, more specifically it controls what happens after users clicking the GUI components (Hernandez, 2021). Controller is like the brain of the application, tying the Model and the View together closely. In a distributed web application, the controller would be the server-side code that handles HTTP requests, interacts with the model, and sends back the HTTP responses. The figure below shows the how each component interacts with each other:

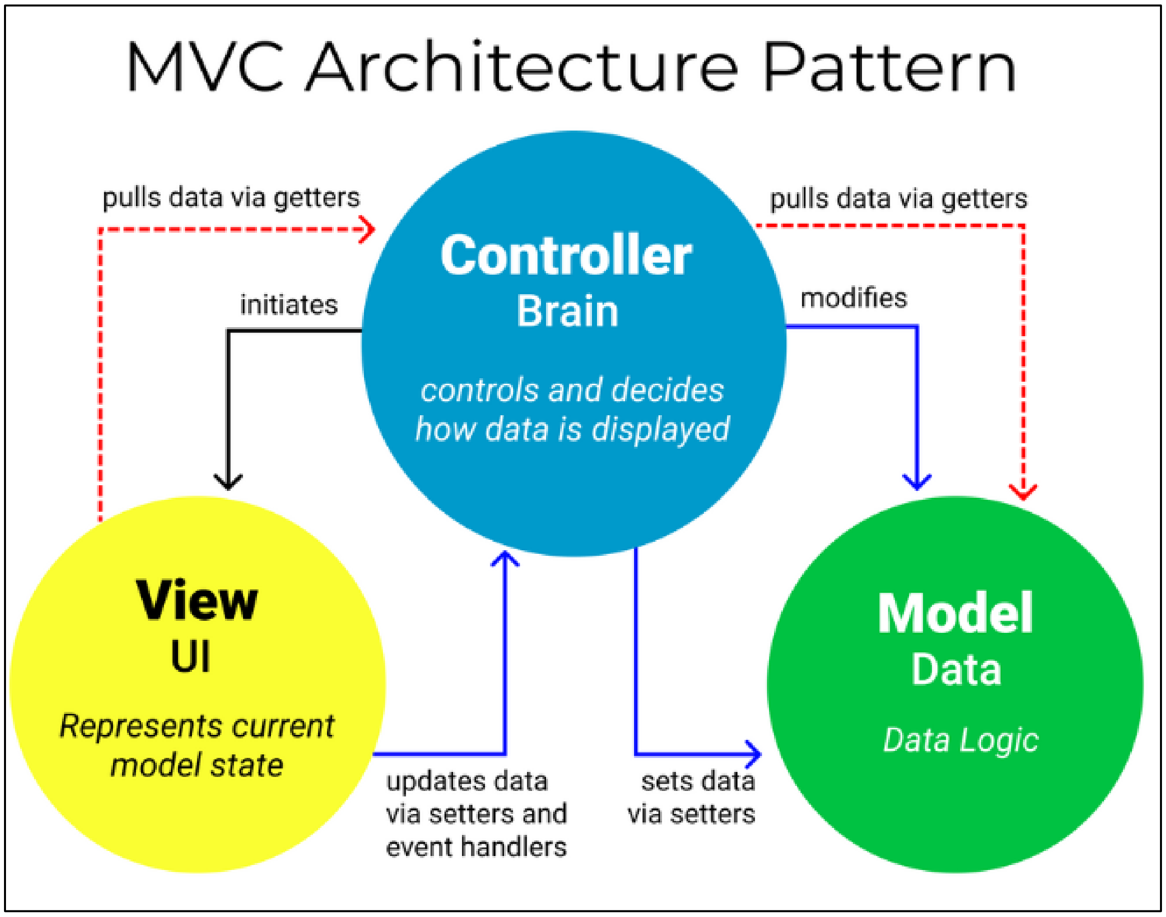


Figure 23: Hernandez, R.D. (2021). MVC Architecture Pattern. freeCodeCamp. https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

(d) AngularJS based on JavaScript & Zend Framework based on PHP and Rails based on Ruby.

AngularJS is a JavaScript framework written in JavaScript (w3schools, n.d.), dedicated for front-end development. AngularJS was developed by Google, and it is used for developing dynamic Single Page Application (SPA) (seo@ebuilderz.site, 2023). Thea architecture behind AngularJS is MVC design pattern:

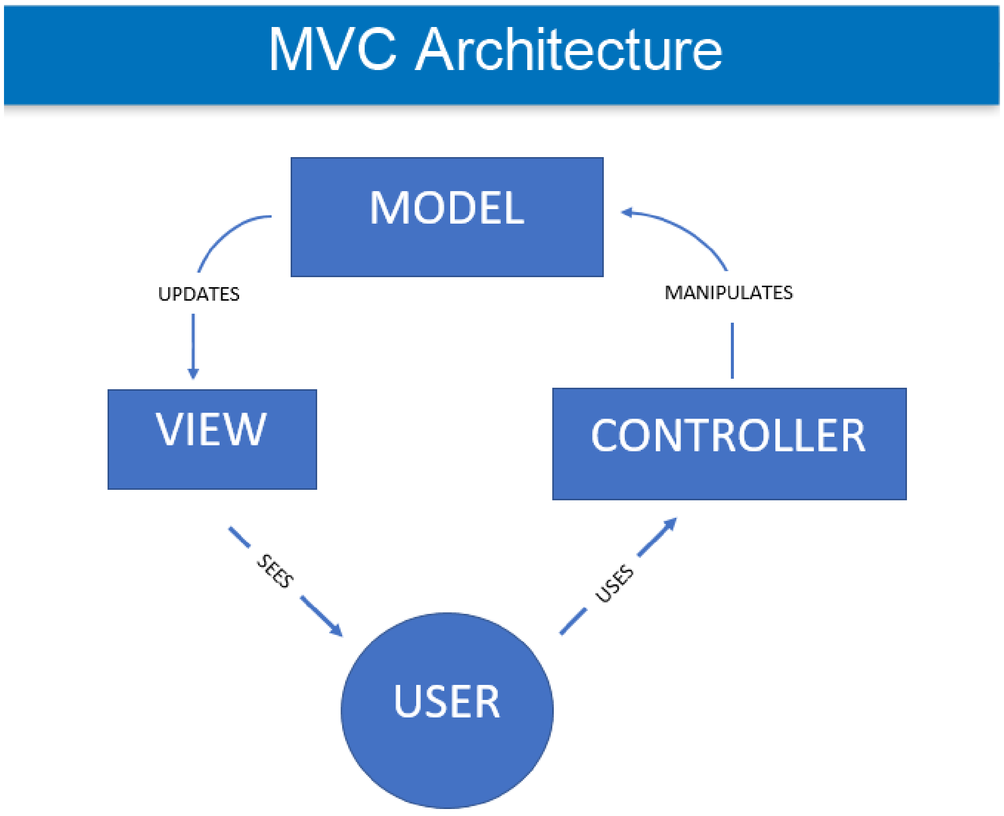
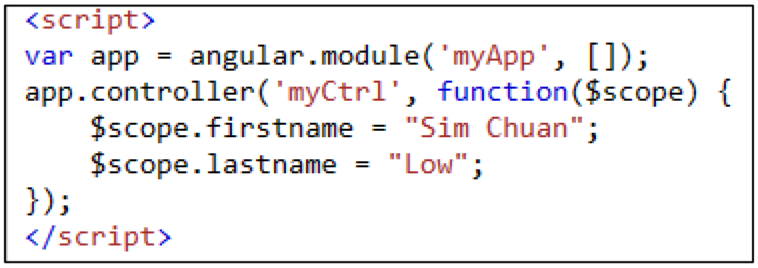


Figure 24: Pedamkar P. (2023). MVC Architecture. educba. https://www.educba.com/angularjs-architecture/

But different terminologies are used in the AngularJS for representing the MVC architecture, that are **Scope**, **HTML with Data Binding**, and **ngController** (Pedamkar, 2023):

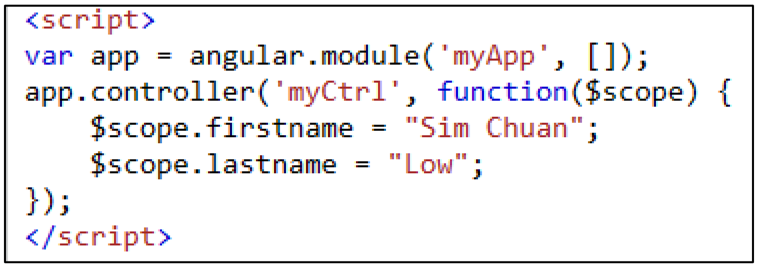
- Scope – **Scope** is the model that holds the application data (variables in AngularJS application), corresponding to the M inside MVC pattern. The Scope is passed as an argument *$scope* (in terms of objects), and its properties are the variables inside the AngularJS application:



```
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstname = "Sim Chuan";
    $scope.lastname = "Low";
});
</script>
```

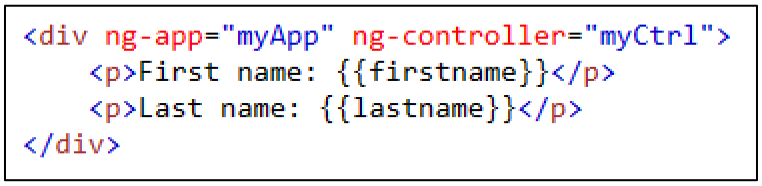
Figure 25: Using Scope in AngularJS Application

- ngController – **ngController** controls the AngularJS application, corresponding to the C inside MVC pattern. The controller can be created by naming it (naming it as *myCtrl* in the figure below), and passing a function with the argument **Scope**:



```
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstname = "Sim Chuan";
    $scope.lastname = "Low";
});
</script>
```

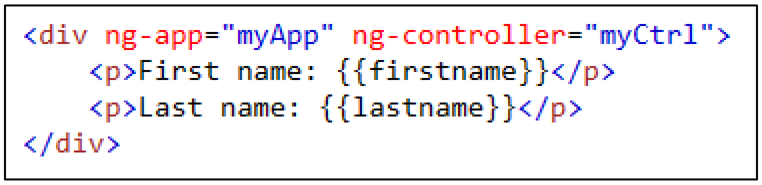
Figure 26: Defining a Controller in AngularJS Application



```
<div ng-app="myApp" ng-controller="myCtrl">
    <p>First name: {{firstname}}</p>
    <p>Last name: {{lastname}}</p>
</div>
```

Figure 27: Using a Controller in AngularJS Application

- HTML with Data Binding – presents the data to the users, corresponding to the V inside MVC pattern. Data Binding can be done through enclosing the variable name using {{}} as shown in the figure below.

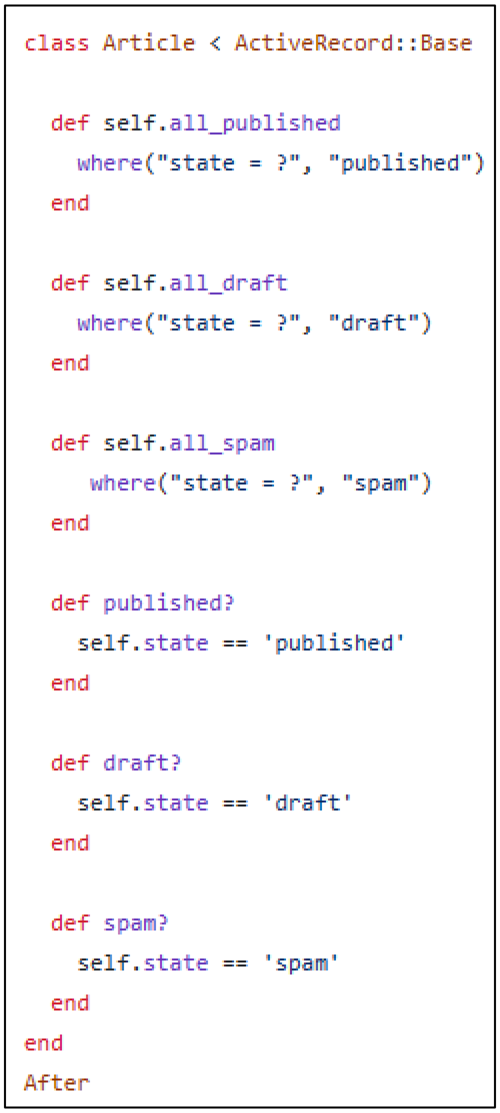


```
<div ng-app="myApp" ng-controller="myCtrl">
    <p>First name: {{firstname}}</p>
    <p>Last name: {{lastname}}</p>
</div>
```

Figure 28: Data Bindings in AngularJS Application

Zend is one of the most popular PHP frameworks to develop a web application, and it has several loosely coupled components (tutorialspoint, n.d.), and it is dedicated for full-stack development. The components are for solving different kinds of tasks such as authentication, database access, caching etc (development, 2023). It is object-oriented and based on the MVC design pattern, the MVC architecture allows programmers to create clean, scalable, and well-structured code, leading to enhancing the performance (Emily, 2022). Other than that, it is lightweight and due to its components being loosely coupled, programs can be compiled with necessary components instead of the whole library (Base, 2023).

Ruby on Rails is a Ruby framework to develop web applications, and it is also based on the MVC architecture (Shut, 2023), dedicated for full-stack development as well. Ruby on Rails is heavily used in writing backend codes, but it can also be used to write frontend codes. To be more specific, Ruby on Rails provides a default structure for databases, web pages, and web services (Pareek, 2021). Other than that, it supports the use of HTML, CSS, and JavaScript for the user interfaces, and the JSON or XML for data transfer between frontend and backend (Pareek, 2021). Ruby on Rails signifies the importance of Convention over Configuration (COC) paradigm and the Don't Repeat Yourself (DRY) principle for boosting the development speed (Bonisteel, 2023). This framework is excellent when programmers want to build web applications with less boilerplate code as shown in the two figures below. The two figures below show how to apply DRY principle using the Ruby on Rails framework, that is when you find some methods whose definitions are similar:



```
class Article < ActiveRecord::Base

  def self.all_published
    where("state = ?", "published")
  end

  def self.all_draft
    where("state = ?", "draft")
  end

  def self.all_spam
     where("state = ?", "spam")
  end

  def published?
    self.state == 'published'
  end

  def draft?
    self.state == 'draft'
  end

  def spam?
    self.state == 'spam'
  end
end
After
```

Figure 29: harsh-sparkway, (2014). Before Applying DRY, Similar Definitions of the Methods . Github. https://gist.github.com/harssh-sparkway/8707634

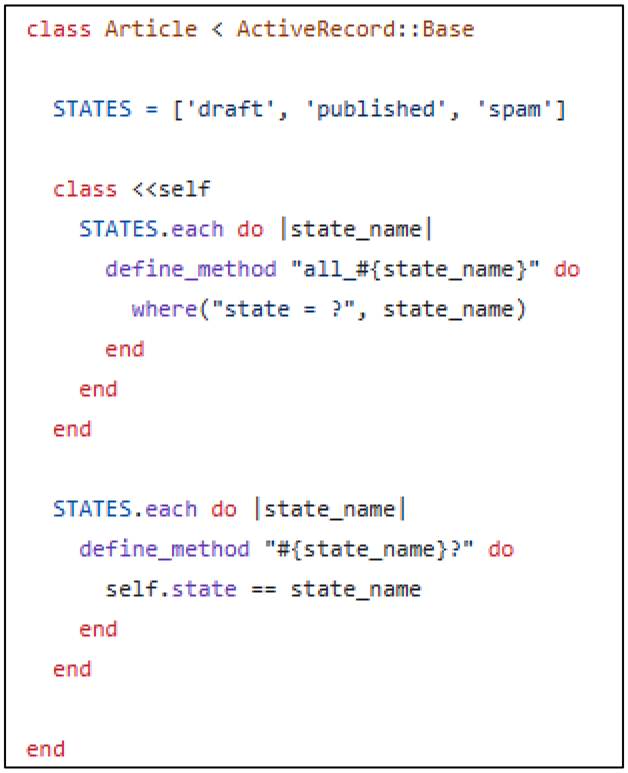
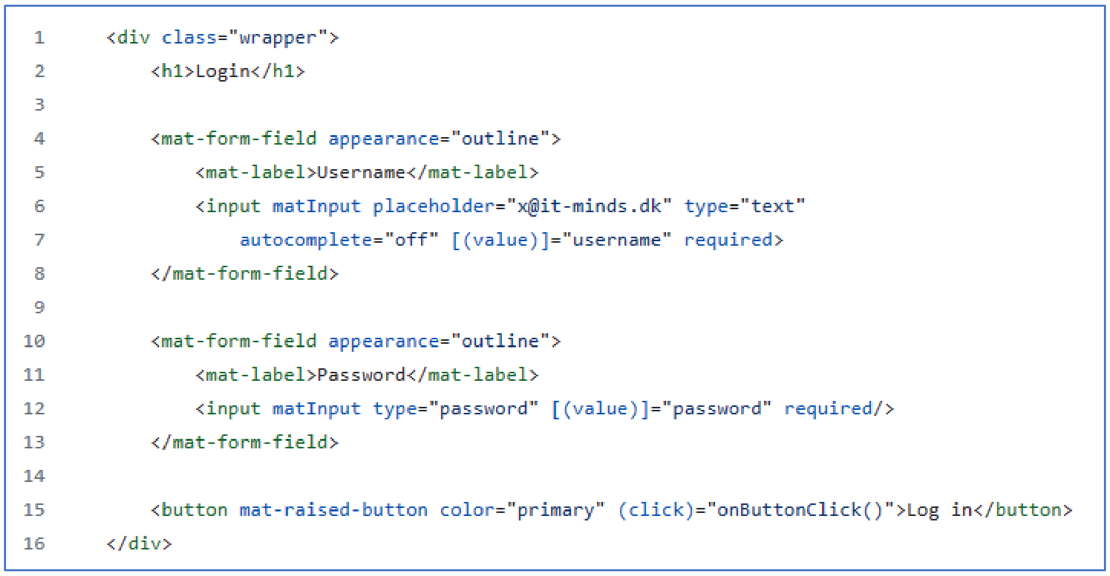

```
class Article < ActiveRecord::Base

  STATES = ['draft', 'published', 'spam']

  class <<self
    STATES.each do |state_name|
      define_method "all_#{state_name}" do
        where("state = ?", state_name)
      end
    end
  end

  STATES.each do |state_name|
    define_method "#{state_name}?" do
      self.state == state_name
    end
  end

end
```

Figure 30: harsh-sparkway, (2014). After Applying DRY. Github. https://gist.github.com/harssh-sparkway/8707634

(e) Evaluate whether client-side only MVC sufficient or server-side only MVC sufficient or need Hybrid of both MVC to make Web Application efficient.

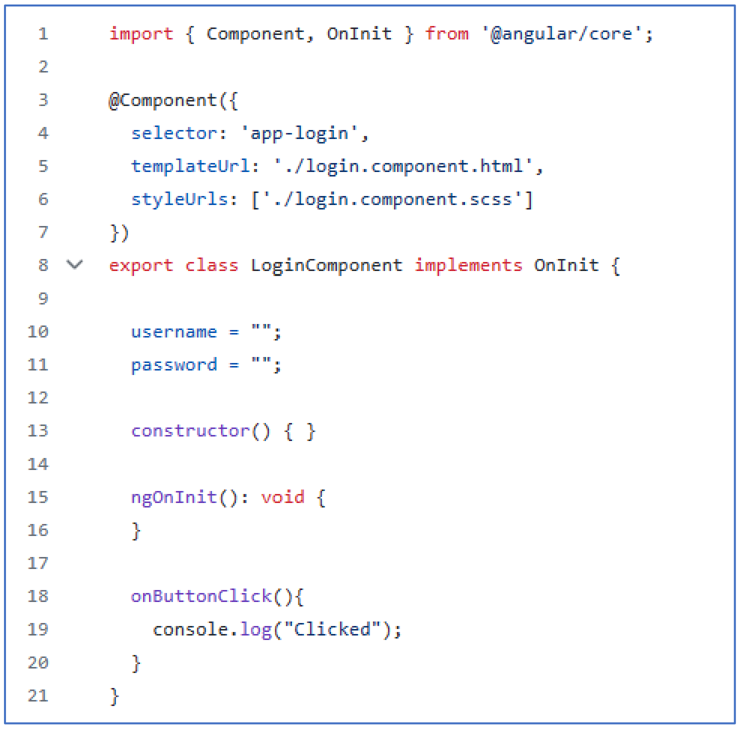
Client-side MVC means that the framework is built on the client side (Tom, 2013). In the application adopting client-side MVC, most of the presentation logics run in the user's web browser. The presentation logics could be happening in the sequence such that rendering webpages, waiting for and processing the user interactions, eventually updating the webpages, then repeat the processes (Kumar A. , 2023). To implement such client-side MVC, a lot of JavaScript frameworks are available out there such as React, Angular, and Vue.js.



```html
<div class="wrapper">
    <h1>Login</h1>

    <mat-form-field appearance="outline">
        <mat-label>Username</mat-label>
        <input matInput placeholder="x@it-minds.dk" type="text"
            autocomplete="off" [(value)]="username" required>
    </mat-form-field>

    <mat-form-field appearance="outline">
        <mat-label>Password</mat-label>
        <input matInput type="password" [(value)]="password" required/>
    </mat-form-field>

    <button mat-raised-button color="primary" (click)="onButtonClick()">Log in</button>
</div>
```

Figure 31: PeterOeClausen (2020). login.component.html using Angular framework. Github. https://github.com/PeterOeClausen/angular-material-design-components-example-app/blob/master/example-app/src/app/login/login.component.html

The figure above shows how to use Angular framework to define a HTML template, *login.component.html*, to be loaded by an Angular application. The figure below shows the file *login.component.ts* loading the HTML template and a CSS file, then define a method called *onButtonClick()* to handle what happens afterwards when the user clicks the Login button.



```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {

  username = "";
  password = "";

  constructor() { }

  ngOnInit(): void {
  }

  onButtonClick(){
    console.log("Clicked");
  }
}
```

Figure 32: PeterOeClausen (2020). login.component.ts using Angular framework to load the HTML template. Github. https://github.com/PeterOeClausen/angular-material-design-components-example-app/blob/master/example-app/src/app/login/login.component.ts

The figure below shows the interactive and dynamic login page created through Angular framework.

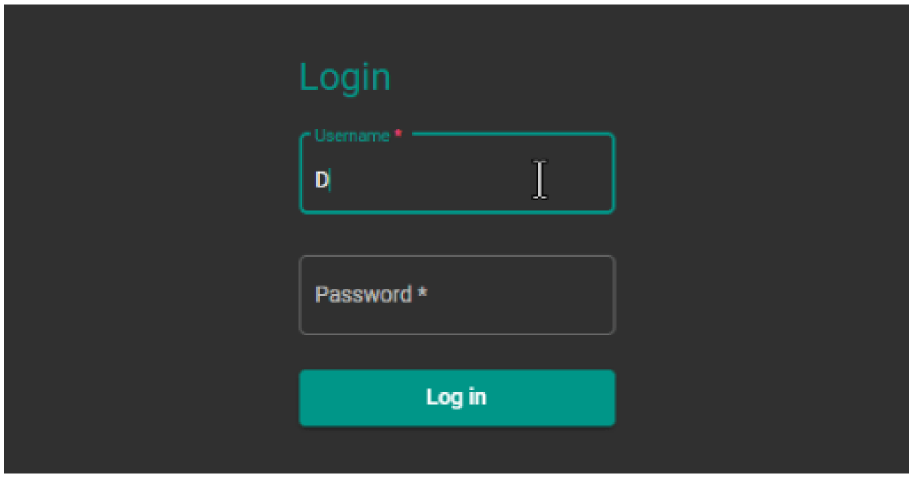
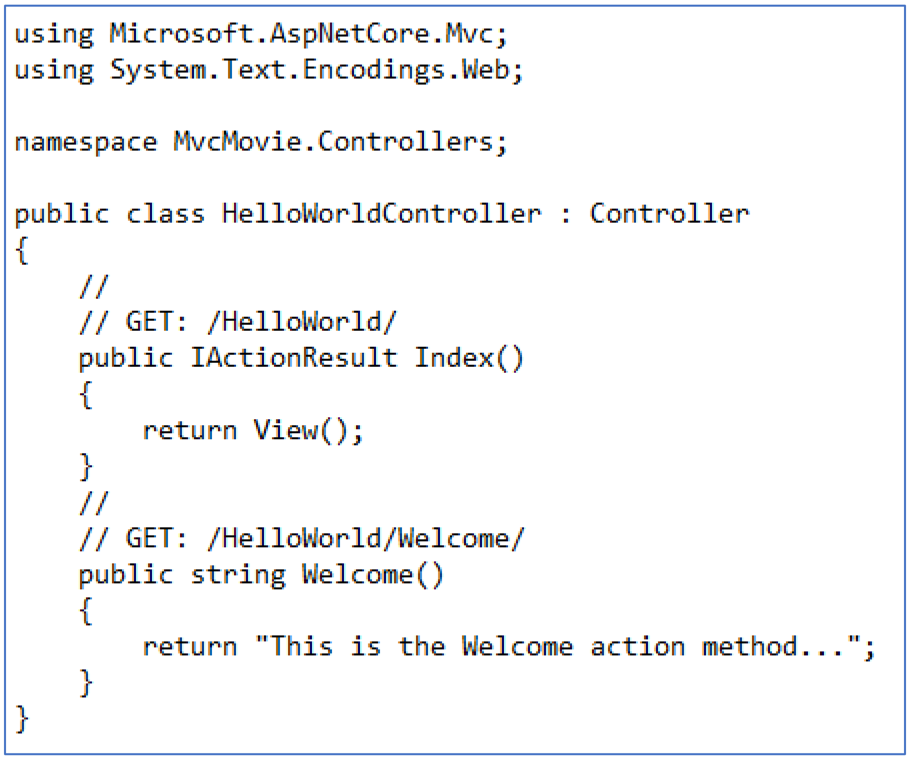


Figure 33: PeterOeClausen (2020). Interactive and dynamic login page using Angular framework. Github. https://github.com/PeterOeClausen/angular-material-design-components-example-app

Server-side MVC means that the framework is built on the server side (Kenneth, 2014). In the application adopting server-side MVC, most of the presentation logics run in the server (Kumar

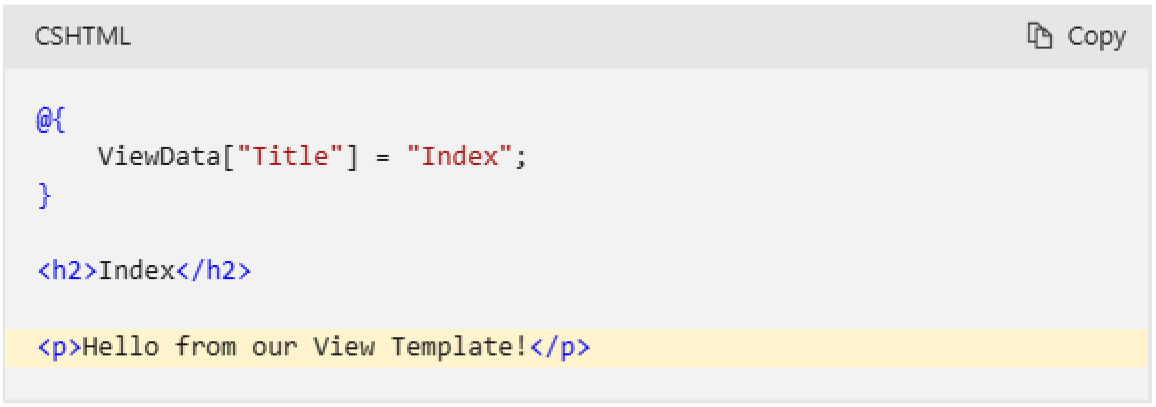
A. , 2023). Server-side MVC is adhering to the traditional web application architecture, where each user interaction with the webpage can lead to full or partial reloading of the webpage, meaning frequent HTTP requests will be sent to the server if the webpages are designed to be dynamic and interactive. The server is responsible for generating dynamic webpages. Other than that, the server-side MVC can include complex server-side logic when returning webpages to the users, such as authentication, authorization, and database access. The idea of server-side MVC can be implemented using ASP.NET framework in C# programming language:



```
using Microsoft.AspNetCore.Mvc;
using System.Text.Encodings.Web;

namespace MvcMovie.Controllers;

public class HelloWorldController : Controller
{
    //
    // GET: /HelloWorld/
    public IActionResult Index()
    {
        return View();
    }
    //
    // GET: /HelloWorld/Welcome/
    public string Welcome()
    {
        return "This is the Welcome action method...";
    }
}
```

Figure 34: Anderson, R. (2023). Defining a controller using ASP.NET framework. Learn Microsoft. https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-view?view=aspnetcore-8.0&tabs=visual-studio

The figure above shows two methods handling HTTP GET request from the users. The *Index()* method is responsible for returning a predefined webpage (as shown in the figure below) to the users. More webpages can be defined as such to be returned to users after the users send HTTP requests to the server.


CSHTML Copy

```
@{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

<p>Hello from our View Template!</p>
```

Figure 35: Anderson, R. (2023). Writing CSHTML file to be returned to users. Learn Microsoft. https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-view?view=aspnetcore-8.0&tabs=visual-studio

As the name suggests, Hybrid MVC means that the framework is built partially on both client side and server side. In practice, several JavaScript frameworks are used to ensure that the webpages are interactive and dynamic, while most of the application's logic are kept on the server (Kumar A. , 2023).

The web applications can be efficient if the correct MVC has been adopted. But before adopting any MVC, the performance goals, preferences, and application requirements of the web application has to be determined at the beginning.

If the web application is focusing more on rendering dynamic and interactive webpages, client-side MVC definitely suits the web application. Client-side MVC is good at creating interactive and awesome user interfaces in terms of their visual appearance (Singhal, 2016). Other than that, applications created by client-side MVC is also more responsive, which means that the reloading of pages due to minor changes doesn't happen frequently (Kenneth, 2014). So, client-side MVC is efficient if the web application requires a lot of dynamic content updates. But this also requires more processing power on the client's device if persist using client-side MVC, while the server-side MVC can offload the processing processes to the server instead of burdening the client's devices leading to customer dissatisfaction. Lastly, if the web application needs to support a large number of users, client-side MCV is more efficient because it can reduce the load on the servers.

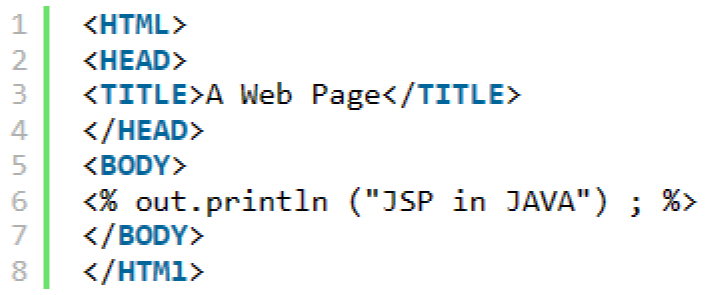
If the web application is focusing more on complex business logic when rendering the webpages and the webpages are not dynamic, then server-side MVC suits the web application well. Server-side MVC is good when the web application has complex business logic and security requirements (psr, 2011). But if the web application contains a lot of dynamic content, server-side MVC will be less efficient because frequent server roundtrips for updates are required. So, server-side MVC is efficient if the web application is mostly about static content.

Hybrid MVC enables a responsive user interface with efficient server-side processing for business logic and security (guardex, 2023). Hybrid MVC is also efficient when the web application needs a balance between interactivity and security and privacy concerns.

(f) Study how to create a web application using Java technologies and provide an overview for it.

The most common way to create a web application is to have static HTML pages and style them using CSS. If there is a need for dynamic websites, server-side technologies are required to ensure that the website is dynamic, for examples e-commerce website having slightly different home page for each user based on their various characteristics (Juviler, 2022). Java can be used to create web applications that are dynamic, and Java server-side technologies to do that are Servlets and JSPs (Pankaj, 2022). To create a fully working web application, we need a database (Oracle or Mysql), a server (Tomcat), and an IDE (Eclipse or Netbeans) (javaTpoint, n.d.).

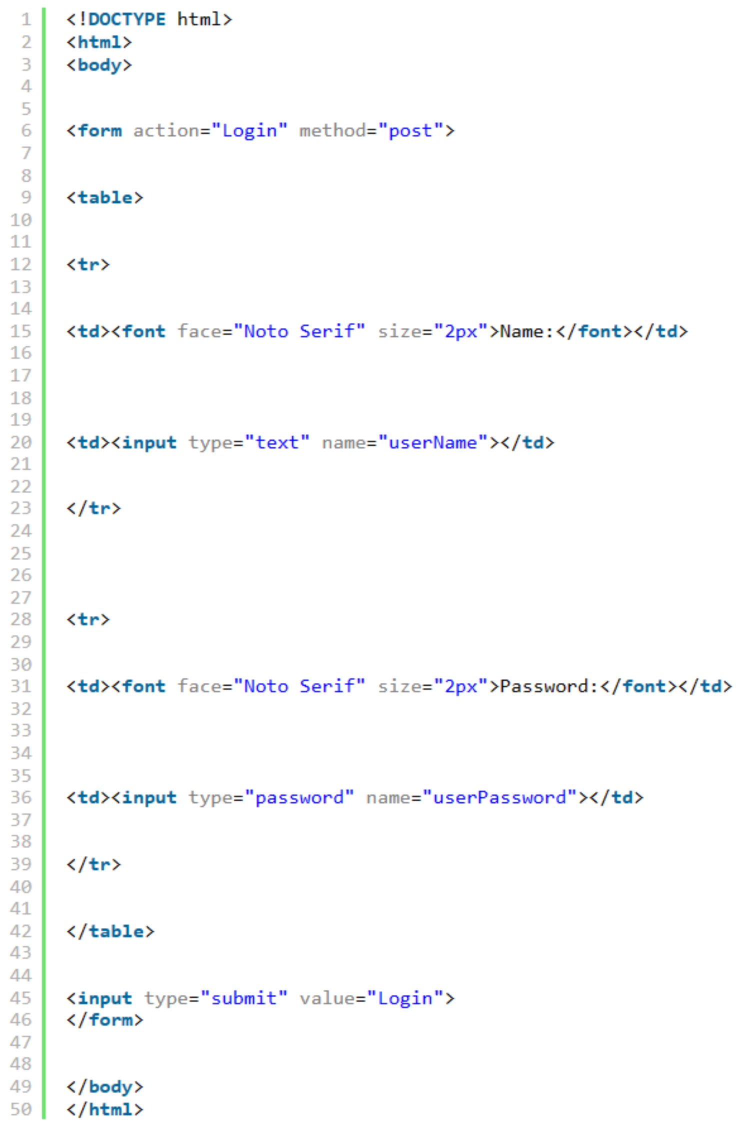
JSP technology allows users to create JSP pages, which consist of static content and dynamic content. Static content can be expressed in the usual way that is HTML, while the dynamic content is created through Java codes (Kalwan, 2023). JSP is well known for its ease of coding, ability to create interactive web pages, and also ease of connecting to a database. The following figure shows a simple JSP page for printing out "JSP in JAVA" on the webpage by executing the Java codes being enclosed by the <% and %>.



```
<HTML>
<HEAD>
<TITLE>A Web Page</TITLE>
</HEAD>
<BODY>
<% out.println ("JSP in JAVA") ; %>
</BODY>
</HTMl>
```

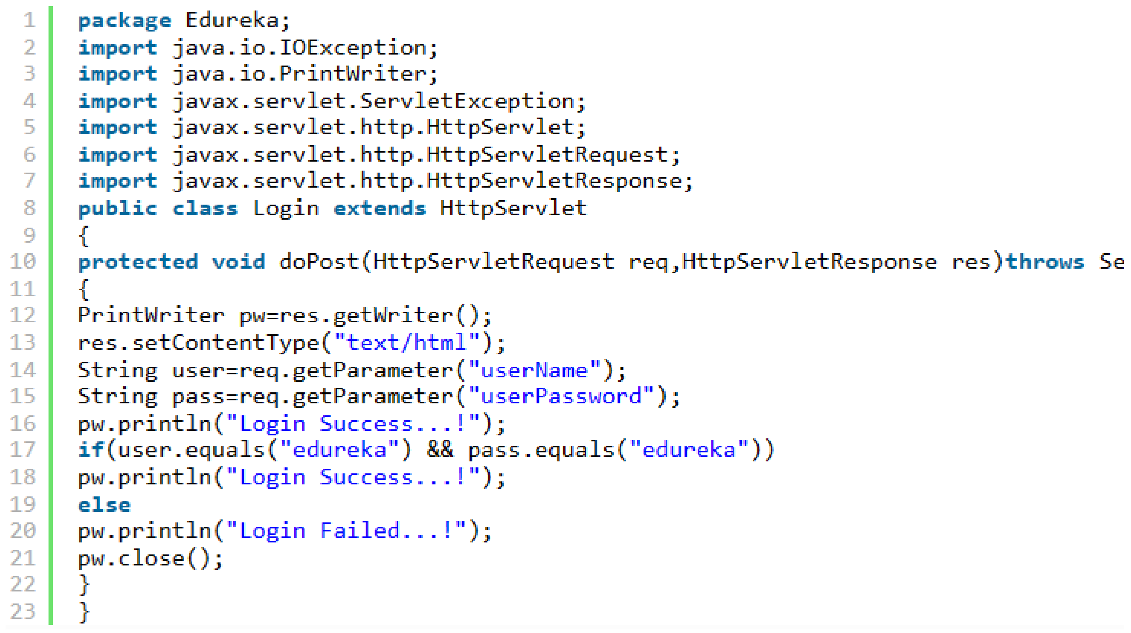
Figure 36: Anamika, K. (2023). Simple JSP Page including Java codes. edureka. https://www.edureka.co/blog/jsp-in-java/

Servlet technology is actually a class in Java programming language, and it can enhance servers that host applications accessed by a request-response model (Vaidya, 2021). The following figure shows a HTML file for creating a login page for users to key in their usernames and password, and after keying in there is a Servlet file validating the usernames and passwords.



```
<!DOCTYPE html>
<html>
<body>


<form action="Login" method="post">


<table>


<tr>


<td><font face="Noto Serif" size="2px">Name:</font></td>




<td><input type="text" name="userName"></td>


</tr>




<tr>


<td><font face="Noto Serif" size="2px">Password:</font></td>




<td><input type="password" name="userPassword"></td>


</tr>


</table>


<input type="submit" value="Login">
</form>


</body>
</html>
```

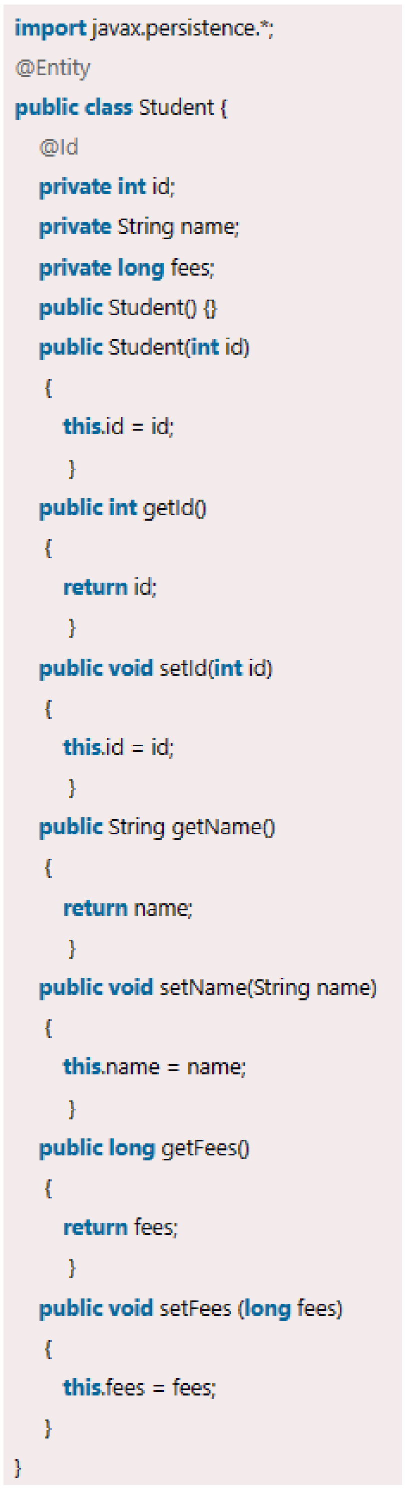
Figure 37: Neha, V. (2021). HTML for Login Page. edureka. https://www.edureka.co/blog/java-servlets



```java
package Edureka;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Login extends HttpServlet
{
protected void doPost(HttpServletRequest req,HttpServletResponse res)throws Se
{
PrintWriter pw=res.getWriter();
res.setContentType("text/html");
String user=req.getParameter("userName");
String pass=req.getParameter("userPassword");
pw.println("Login Success...!");
if(user.equals("edureka") && pass.equals("edureka"))
pw.println("Login Success...!");
else
pw.println("Login Failed...!");
pw.close();
}
}
```

Figure 38: Neha, V. (2021). Servlet File for Validating Usernames and Passwords. edureka. https://www.edureka.co/blog/java-servlets

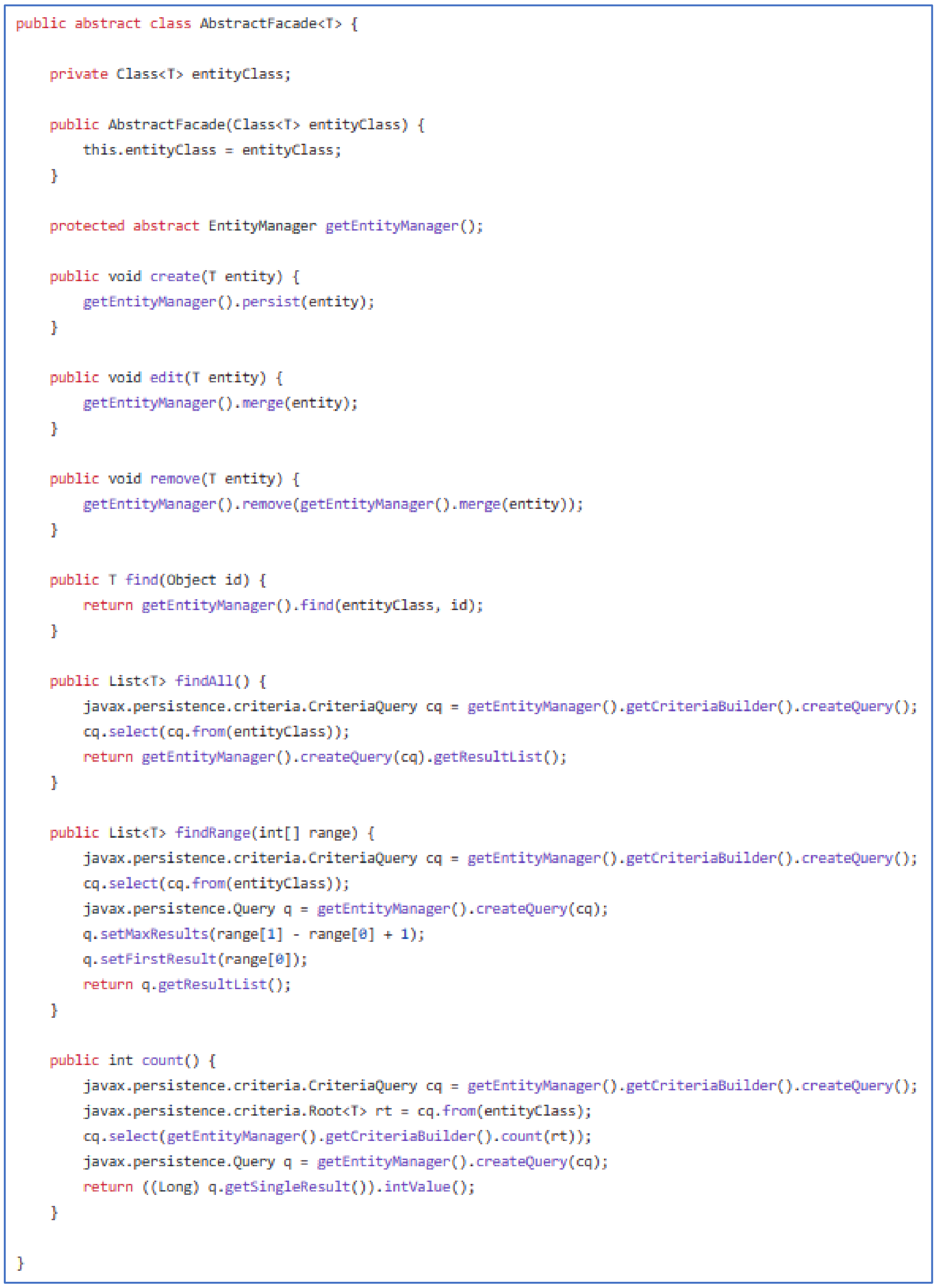
The technologies that have been discussed so far are rendering some GUI components through JSP file or HTML file allowing users to give inputs, then validations are done through the Servlet file (the credentials are hardcoded). In the real world, validations should be done based on the results queried from the database, and there are two Java technologies for that called Entity class and Session Beans. An entity class is a class for creating a table, and the class is with the annotation **@Entity**, while the primary key is labelled with the annotation **@Id**. The following figure shows an entity class for the table called ***Student***, and the columns of the table are defined by the attributes of the entity class, that is ***id***, ***name***, and ***fees***.



```
import javax.persistence.*;
@Entity
public class Student {
  @Id
  private int id;
  private String name;
  private long fees;
  public Student() {}
  public Student(int id)
   {
     this.id = id;
     }
  public int getId()
   {
     return id;
     }
  public void setId(int id)
   {
     this.id = id;
     }
  public String getName()
   {
     return name;
     }
  public void setName(String name)
   {
     this.name = name;
     }
  public long getFees()
   {
     return fees;
     }
  public void setFees (long fees)
   {
     this.fees = fees;
   }
}
```

Figure 39: JavaTpoint (n.d.). JPA Creating an Entity Class. javaTpoint. https://www.javatpoint.com/jpa-creating-an-entity

The session bean contains business logic that can invoked by local, remote or webservice client, and the session beans create an EJB session façade for each entity class with basic access methods (Netbeans, n.d.). The following figure shows an abstract façade that is to be inherited by facades of all entity classes, and it provides some of the basic access methods such as ***create***, ***edit***, ***remove***, ***find***, ***findAll***, ***findRange***, and ***count***. All these methods allows programmers to interact with tables created by the entity classes, on top of that programmers can add more customized access methods in respective façade of the entity classes.



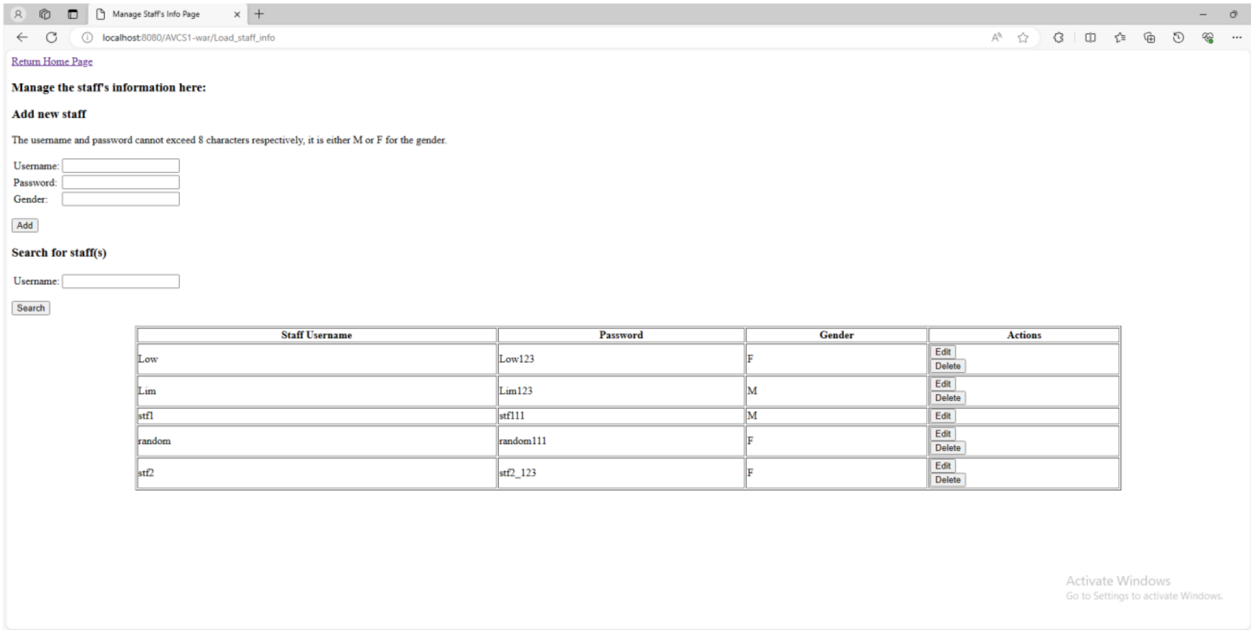
```java
public abstract class AbstractFacade<T> {

    private Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) {
        getEntityManager().remove(getEntityManager().merge(entity));
    }

    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    public List<T> findRange(int[] range) {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        q.setMaxResults(range[1] - range[0] + 1);
        q.setFirstResult(range[0]);
        return q.getResultList();
    }

    public int count() {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        javax.persistence.criteria.Root<T> rt = cq.from(entityClass);
        cq.select(getEntityManager().getCriteriaBuilder().count(rt));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    }

}
```

Figure 40: HouariZegai (2019). Abstract Façade to be inherited by facades of all entity classes. github. https://github.com/HouariZegai/HotelReservationSystem/blob/master/HotelReservation-ejb/src/java/sessionbeans/AbstractFacade.java

Part B1

Design of web components

In this assignment, JSP files are used as a template of a webpage, while the Servlet files are used to execute all the business logic (such as validating the user inputs and interact with the database) and add some extra webpage contents to the associated JSP files such as warnings, system messages, and updated list of existing records. For example, say that a webpage is designed for a staff performing CRUD operations on other staffs' information, the webpage should look like this:



Manage the staff's information here:

Add new staff

The username and password cannot exceed 8 characters respectively, it is either M or F for the gender.

Username:	
Password:	
Gender:	

Search for staff(s)

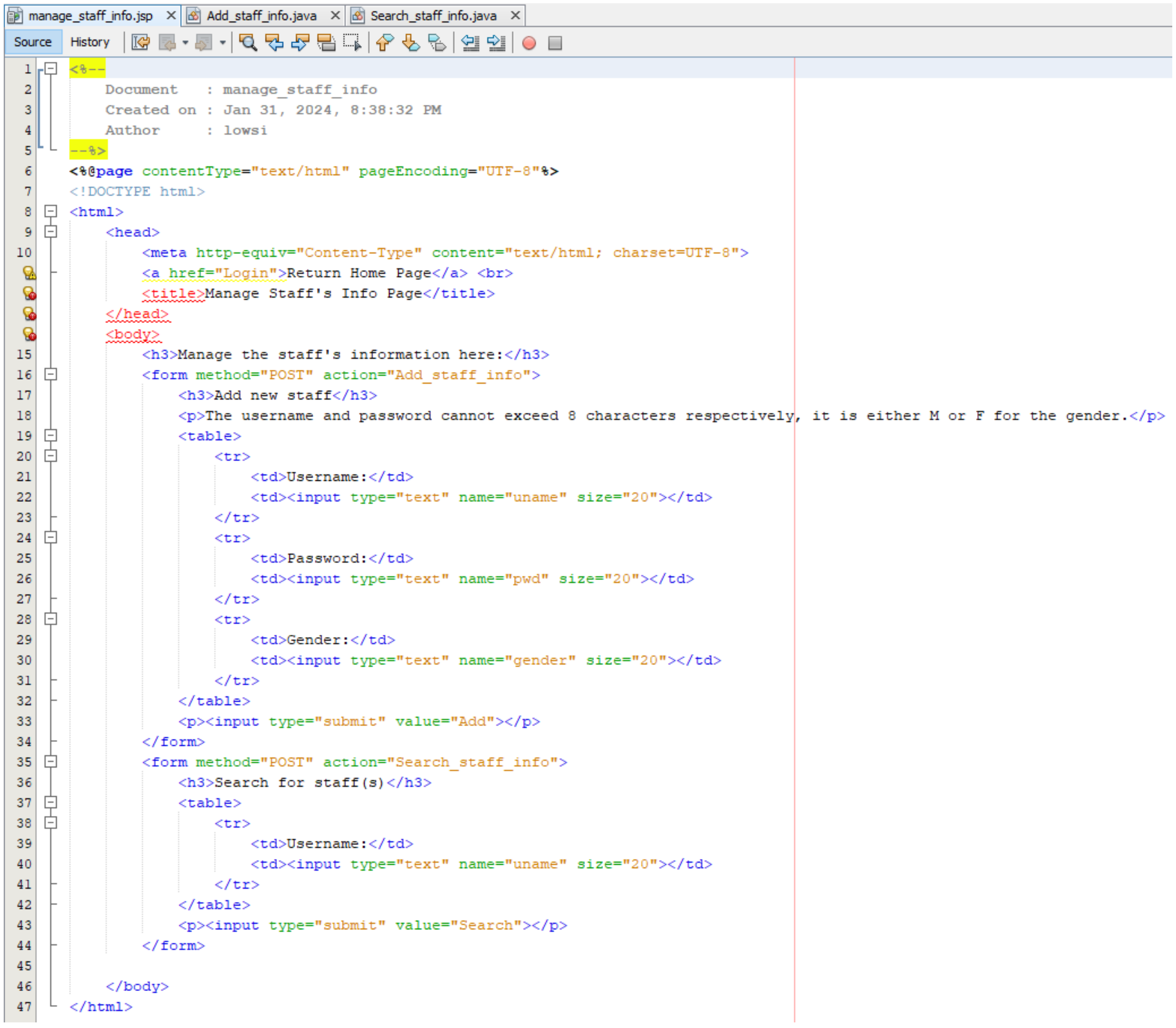
Username:	

Staff Username	Password	Gender	Actions
Low	Low123	F	Edit Delete
Lim	Lim123	M	Edit Delete
stf1	stf111	M	Edit
random	random111	F	Edit Delete
stf2	stf2_123	F	Edit Delete

Figure 41: A webpage design for a staff to perform CRUD operations on other staffs' information

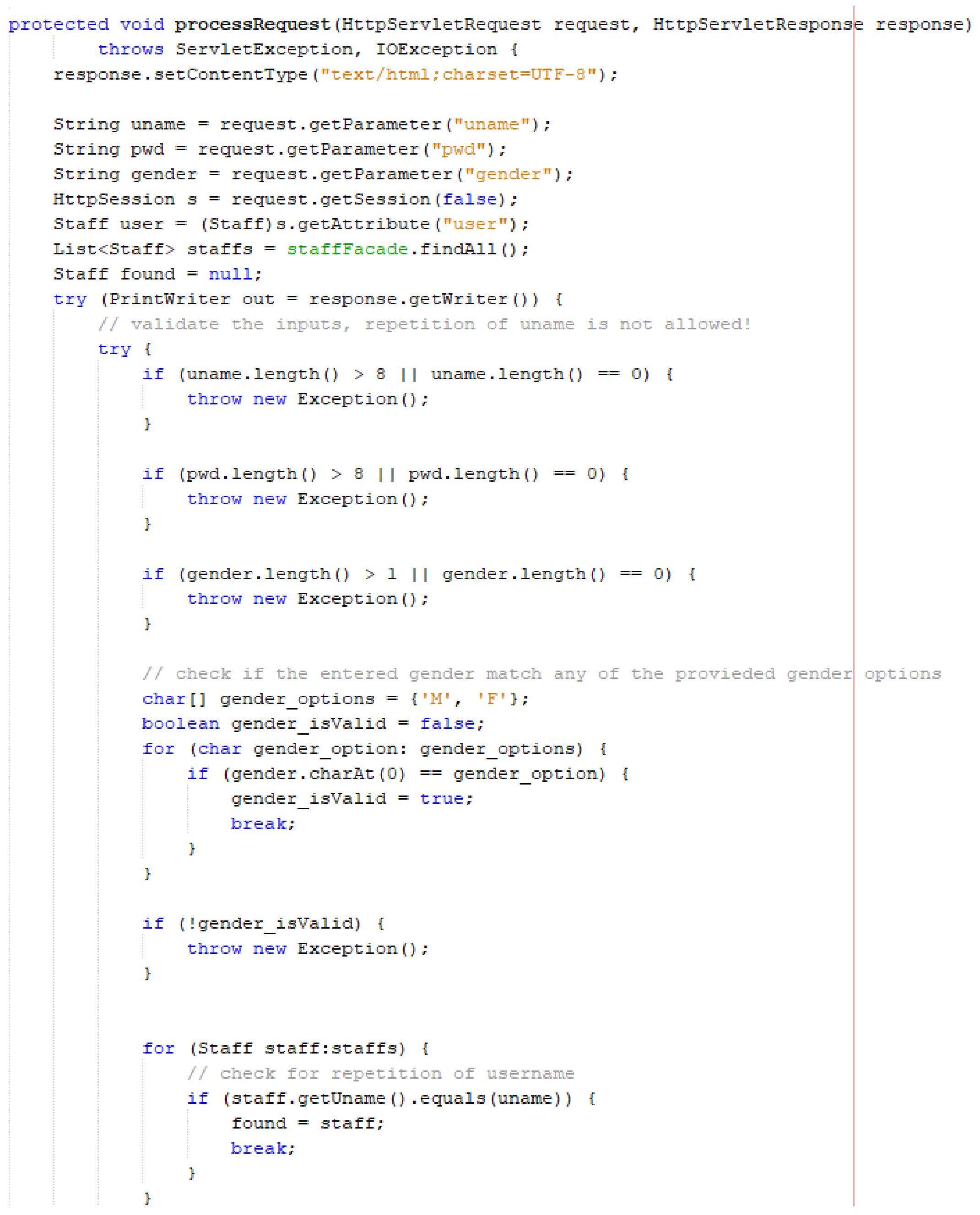
The webpage provides a section for a staff to add a new staff, a section for a staff to search for a staff based on the username, and a section for displaying a list of existing records of staffs' information which updates immediately once any operation of the CRUD operation happens. Technically speaking, to implement such webpage, a JSP file called ***manage_staff_info.jsp*** is created for rendering the template of the webpage, which is the section for a staff to add a new staff and a section for a staff to search for a staff based on username, through two individual HTML

form elements (one form sends HTTP POST request to the ***Add_staff_info*** Servlet file and one form sends HTTP POST request to the ***Search_staff_info*** Servlet file).



```
<%--
    Document   : manage_staff_info
    Created on : Jan 31, 2024, 8:38:32 PM
    Author     : lowsi
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <a href="Login">Return Home Page</a> <br>
        <title>Manage Staff's Info Page</title>
    </head>
    <body>
        <h3>Manage the staff's information here:</h3>
        <form method="POST" action="Add_staff_info">
            <h3>Add new staff</h3>
            <p>The username and password cannot exceed 8 characters respectively, it is either M or F for the gender.</p>
            <table>
                <tr>
                    <td>Username:</td>
                    <td><input type="text" name="uname" size="20"></td>
                </tr>
                <tr>
                    <td>Password:</td>
                    <td><input type="text" name="pwd" size="20"></td>
                </tr>
                <tr>
                    <td>Gender:</td>
                    <td><input type="text" name="gender" size="20"></td>
                </tr>
            </table>
            <p><input type="submit" value="Add"></p>
        </form>
        <form method="POST" action="Search_staff_info">
            <h3>Search for staff(s)</h3>
            <table>
                <tr>
                    <td>Username:</td>
                    <td><input type="text" name="uname" size="20"></td>
                </tr>
            </table>
            <p><input type="submit" value="Search"></p>
        </form>

    </body>
</html>
```

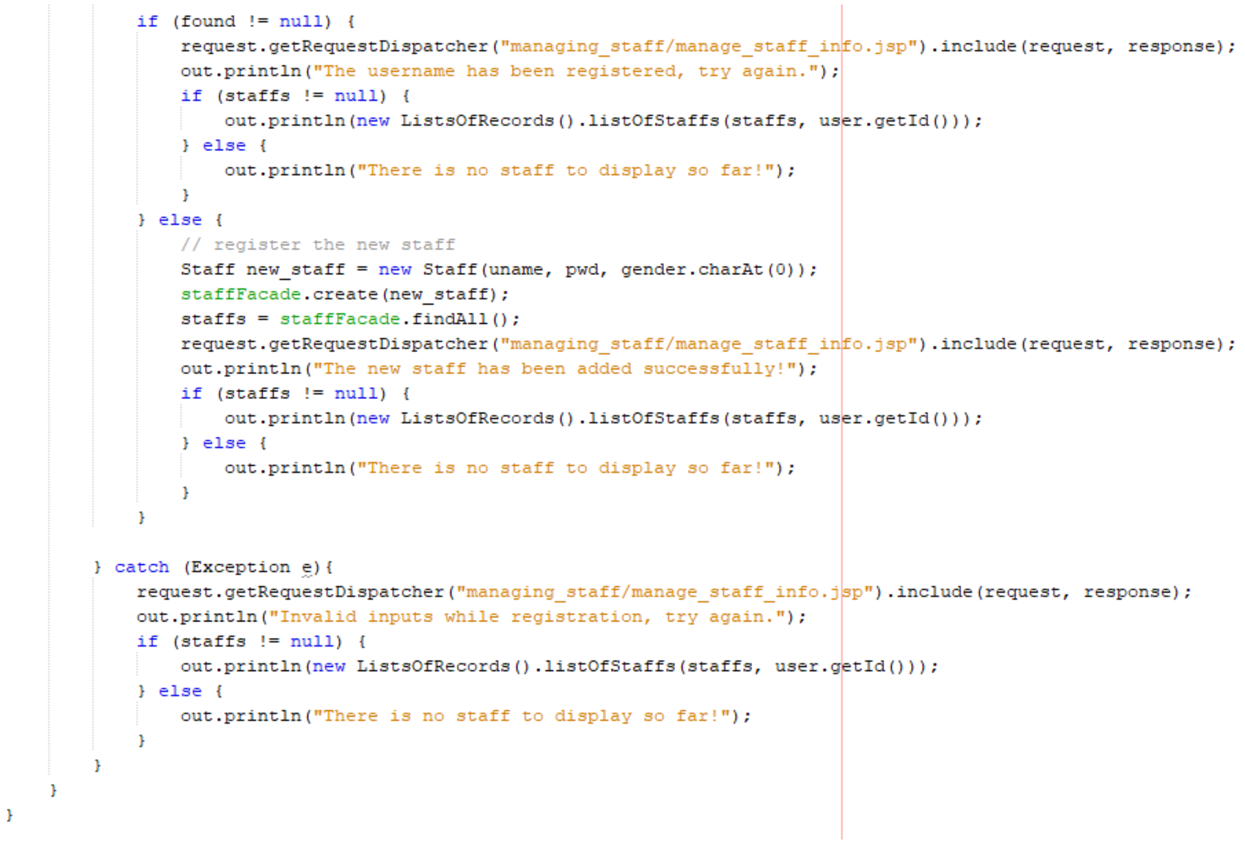
Figure 42: manage_staff_info.jsp



```java
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");

    String uname = request.getParameter("uname");
    String pwd = request.getParameter("pwd");
    String gender = request.getParameter("gender");
    HttpSession s = request.getSession(false);
    Staff user = (Staff)s.getAttribute("user");
    List<Staff> staffs = staffFacade.findAll();
    Staff found = null;
    try (PrintWriter out = response.getWriter()) {
        // validate the inputs, repetition of uname is not allowed!
        try {
            if (uname.length() > 8 || uname.length() == 0) {
                throw new Exception();
            }

            if (pwd.length() > 8 || pwd.length() == 0) {
                throw new Exception();
            }

            if (gender.length() > 1 || gender.length() == 0) {
                throw new Exception();
            }

            // check if the entered gender match any of the provieded gender options
            char[] gender_options = {'M', 'F'};
            boolean gender_isValid = false;
            for (char gender_option: gender_options) {
                if (gender.charAt(0) == gender_option) {
                    gender_isValid = true;
                    break;
                }
            }

            if (!gender_isValid) {
                throw new Exception();
            }


            for (Staff staff:staffs) {
                // check for repetition of username
                if (staff.getUname().equals(uname)) {
                    found = staff;
                    break;
                }
            }
```

Figure 43: Add_staff_info Servlet file – Part 1

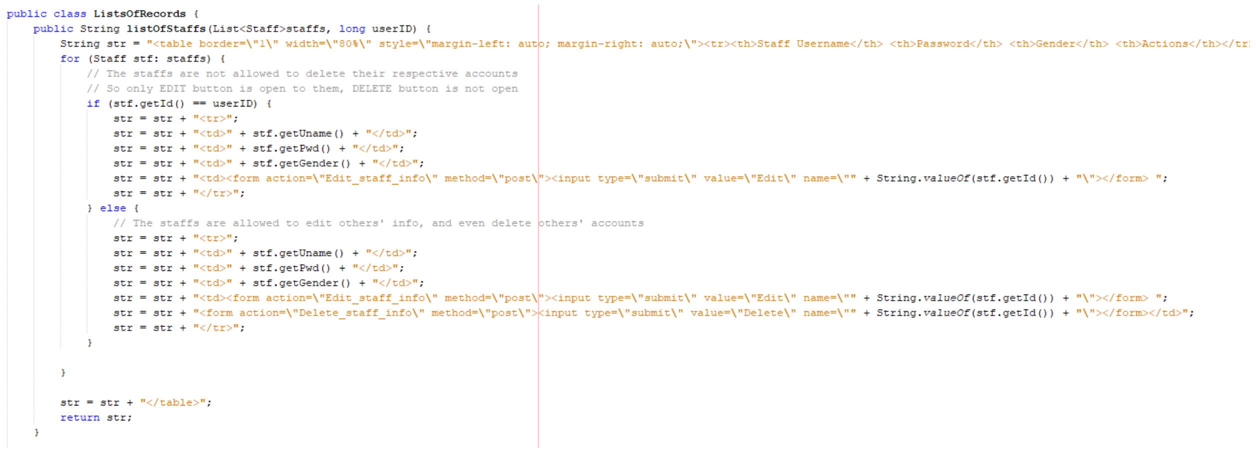
The ***Add_staff_info*** Servlet file is about processing the inputs sent by the HTTP POST request from the JSP file. At here, all the inputs are validated to see if they are illogical: if they are illogical, the JSP page will be rendered again with a warning message and list of existing records (using ***request.getRequestDispatcher().include(request, response)*** and ***out.println()***); otherwise the new staff will be added to the database, then the JSP page will be rendered again with a successful message and updated list of records (using ***request.getRequestDispatcher().include(request, response)*** and ***out.println()***.



```
            if (found != null) {
                request.getRequestDispatcher("managing_staff/manage_staff_info.jsp").include(request, response);
                out.println("The username has been registered, try again.");
                if (staffs != null) {
                    out.println(new ListsOfRecords().listOfStaffs(staffs, user.getId()));
                } else {
                    out.println("There is no staff to display so far!");
                }
            } else {
                // register the new staff
                Staff new_staff = new Staff(uname, pwd, gender.charAt(0));
                staffFacade.create(new_staff);
                staffs = staffFacade.findAll();
                request.getRequestDispatcher("managing_staff/manage_staff_info.jsp").include(request, response);
                out.println("The new staff has been added successfully!");
                if (staffs != null) {
                    out.println(new ListsOfRecords().listOfStaffs(staffs, user.getId()));
                } else {
                    out.println("There is no staff to display so far!");
                }
            }

        } catch (Exception e){
            request.getRequestDispatcher("managing_staff/manage_staff_info.jsp").include(request, response);
            out.println("Invalid inputs while registration, try again.");
            if (staffs != null) {
                out.println(new ListsOfRecords().listOfStaffs(staffs, user.getId()));
            } else {
                out.println("There is no staff to display so far!");
            }
        }
    }
}
```

Figure 44: Add_staff_info Servlet file – Part 2

In the figure above, an object ***ListOfRecords()*** is created and its method ***listOfStaffs()*** has been called. They are just some HTML codes for rendering a list of existing staffs' information using HTML ***table*** element, as shown in the figure below:



```
public class ListsOfRecords {
    public String listOfStaffs(List<Staff>staffs, long userID) {
        String str = "<table border=\"1\" width=\"80%\" style=\"margin-left: auto; margin-right: auto;\"><tr><th>Staff Username</th> <th>Password</th> <th>Gender</th> <th>Actions</th></tr
        for (Staff stf: staffs) {
            // The staffs are not allowed to delete their respective accounts
            // So only EDIT button is open to them, DELETE button is not open
            if (stf.getId() == userID) {
                str = str + "<tr>";
                str = str + "<td>" + stf.getUname() + "</td>";
                str = str + "<td>" + stf.getPwd() + "</td>";
                str = str + "<td>" + stf.getGender() + "</td>";
                str = str + "<td><form action=\"Edit_staff_info\" method=\"post\"><input type=\"submit\" value=\"Edit\" name=\"" + String.valueOf(stf.getId()) + "\"></form> ";
                str = str + "</tr>";
            } else {
                // The staffs are allowed to edit others' info, and even delete others' accounts
                str = str + "<tr>";
                str = str + "<td>" + stf.getUname() + "</td>";
                str = str + "<td>" + stf.getPwd() + "</td>";
                str = str + "<td>" + stf.getGender() + "</td>";
                str = str + "<td><form action=\"Edit_staff_info\" method=\"post\"><input type=\"submit\" value=\"Edit\" name=\"" + String.valueOf(stf.getId()) + "\"></form> ";
                str = str + "<form action=\"Delete_staff_info\" method=\"post\"><input type=\"submit\" value=\"Delete\" name=\"" + String.valueOf(stf.getId()) + "\"></form></td>";
                str = str + "</tr>";
            }

        }

        str = str + "</table>";
        return str;
    }
```

Figure 45: ListsOfRecords().listOfStaffs for rendering a list of staff's records

From the figure above, the HTML codes shows that an Edit button is rendered for all staffs' records (sending HTTP POST request to ***Edit_staff_info*** Servlet file), and a Delete button is rendered for all staffs' records except for the staff who is using the system (sending HTTP POST request to ***Delete_staff_info*** Servlet file). Such a way of connecting JSP files with Servlet files is applicable to develop other functionalities of this assignment.

Since the functionalities of this assignment are all about CRUD, the naming convention of the Servlet files follows the CRUD operations. Generally, the Servlet files that start with the word "Load" stands for initializing the list of existing records in the webpage (R in CRUD), the word "Create" stands for creating a new record then render the webpage with system message and a list of existing records, the word "Edit" stands for rendering the webpage with modifiable existing records, and the word “Delete” stands for rendering the webpage with system message and a list of existing records.

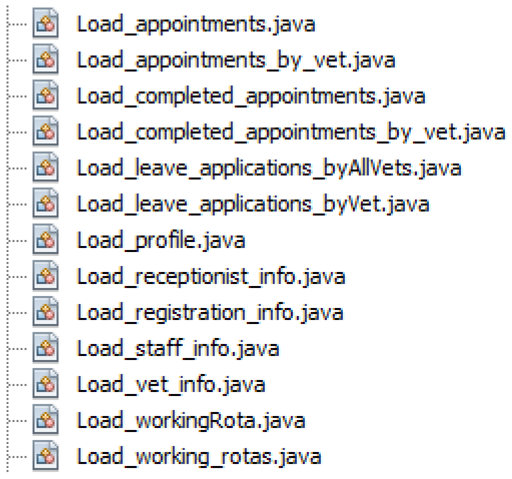

Load_appointments.java
Load_appointments_by_vet.java
Load_completed_appointments.java
Load_completed_appointments_by_vet.java
Load_leave_applications_byAllVets.java
Load_leave_applications_byVet.java
Load_profile.java
Load_receptionist_info.java
Load_registration_info.java
Load_staff_info.java
Load_vet_info.java
Load_workingRota.java
Load_working_rotas.java

Figure 46: Servlet files to load data

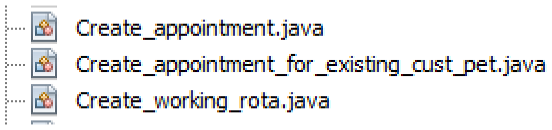

Create_appointment.java
Create_appointment_for_existing_cust_pet.java
Create_working_rota.java

Figure 47: Servlet files to create data

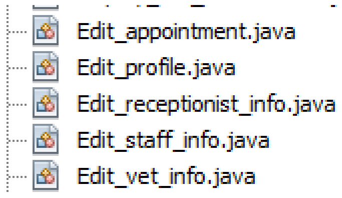

Edit_appointment.java
Edit_profile.java
Edit_receptionist_info.java
Edit_staff_info.java
Edit_vet_info.java

Figure 48: Servlet files to edit data

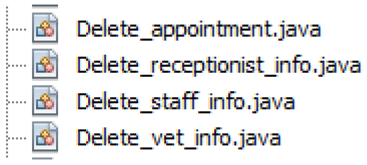

Delete_appointment.java
Delete_receptionist_info.java
Delete_staff_info.java
Delete_vet_info.java

Figure 49: Servlet files to delete data

User Manual – Webpage design and Sitemap

This section is about webpage design and sitemap. First, the figure below is the design for the login page, and it can be used by all user types: Staff, Vet, and Receptionist. The users are required to select the user type, provide credentials then click the Login button. Other than that, there is a hyperlink "New User Registration" for redirecting the users to a registration page, and the page is for registration of vets' or receptionists' accounts.

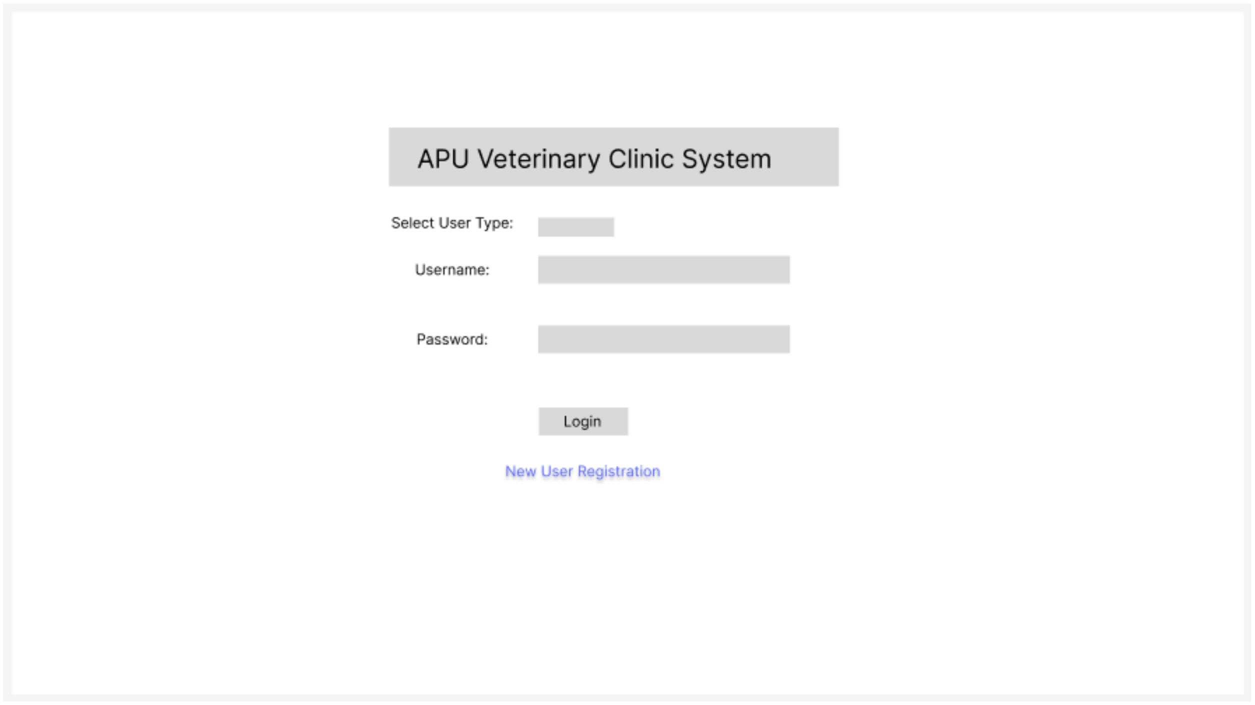


Figure 50: Login Page for All User Types

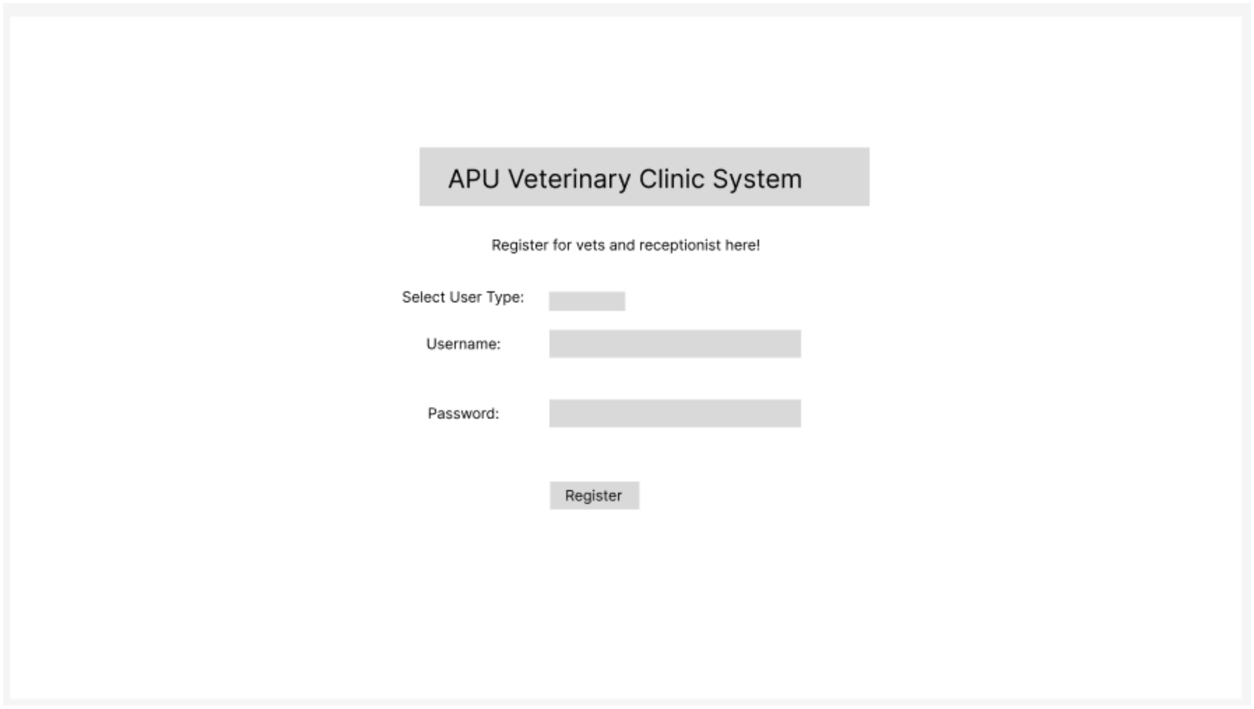

APU Veterinary Clinic System
Register for vets and receptionist here!
Select User Type:
Username:
Password:
Register

Figure 51: Registration Page for Vets and Receptionists

The figure below shows the home page for all user types. The welcome message is specific until the user type and the username are also displayed. This webpage consists of various hyperlinks, each redirecting to corresponding webpage for further action. Then, the last action must be logging out.

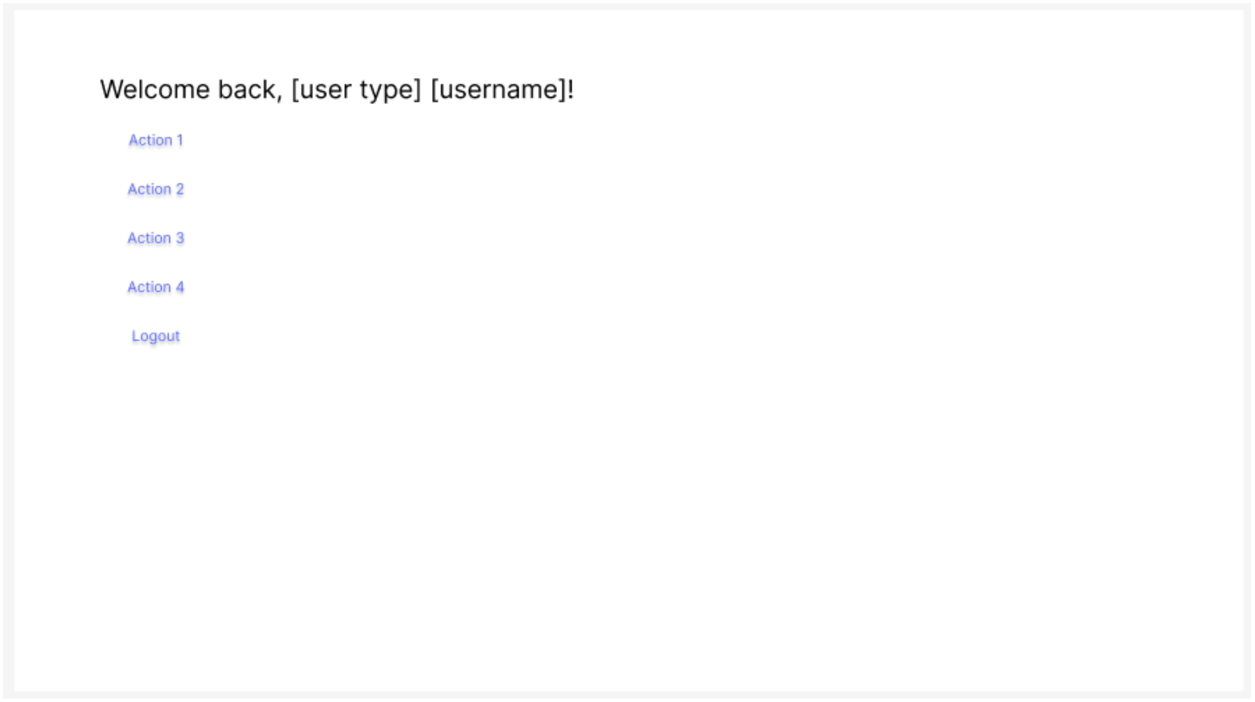
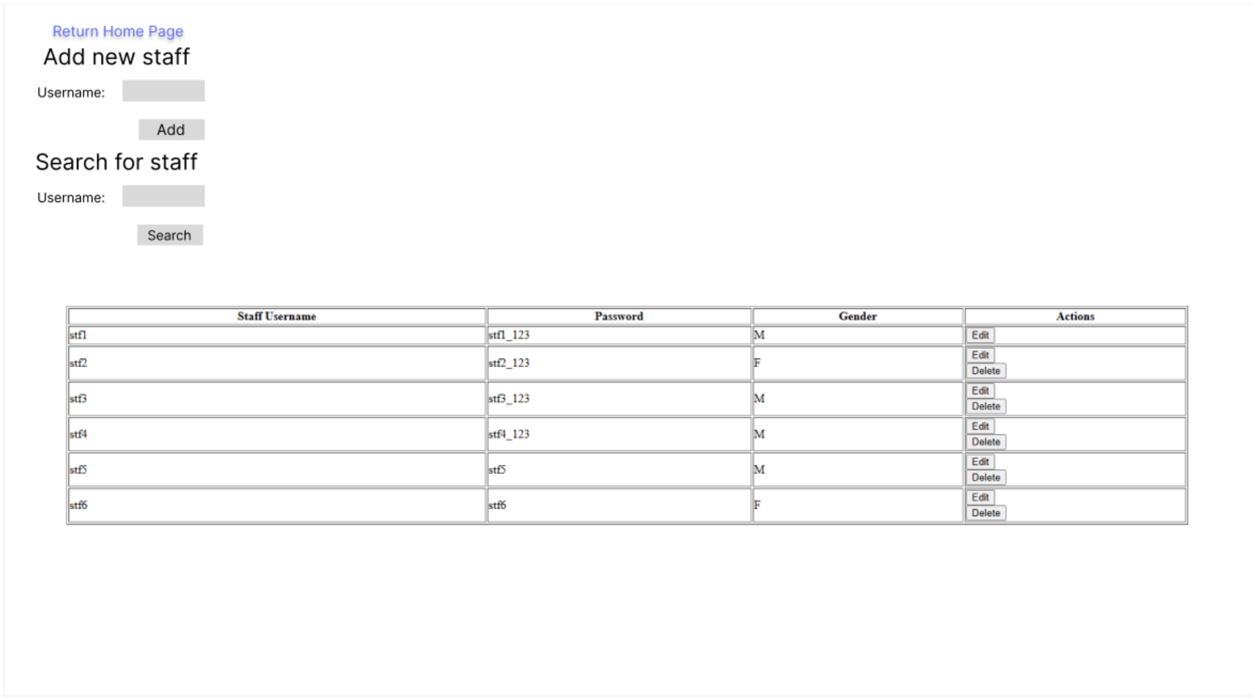

Welcome back, [user type] [username]!

Figure 52: Home Page for All User Types

Then, after redirecting from the home page, the figure below shows a template for all further actions. This is because all actions can be generally recognized as CRUD, and the design can be done as shown in the figure below. In the figure below, say that the CRUD webpage is designed for a staff to manage staffs' information, at the top of the webpage, the staff is allowed to return to the home page. Then, adding new staff by entering username is also allowed. Then, searching for a staff based on username is also allowed. Lastly, a table displaying all existing records (or partial existing records due to the searching functionality) allows the staff to edit and delete the staffs. This kind of design can be applied to other functionalities since all of them are about CRUD.


Return Home Page

Add new staff

Username:

Add

Search for staff

Username:

Search

Staff Username	Password	Gender	Actions
stf1	stf1_123	M	Edit
stf2	stf2_123	F	Edit Delete
stf3	stf3_123	M	Edit Delete
stf4	stf4_123	M	Edit Delete
stf5	stf5	M	Edit Delete
stf6	stf6	F	Edit Delete

Figure 53: CRUD Webpage Design - Managing Staffs' Information

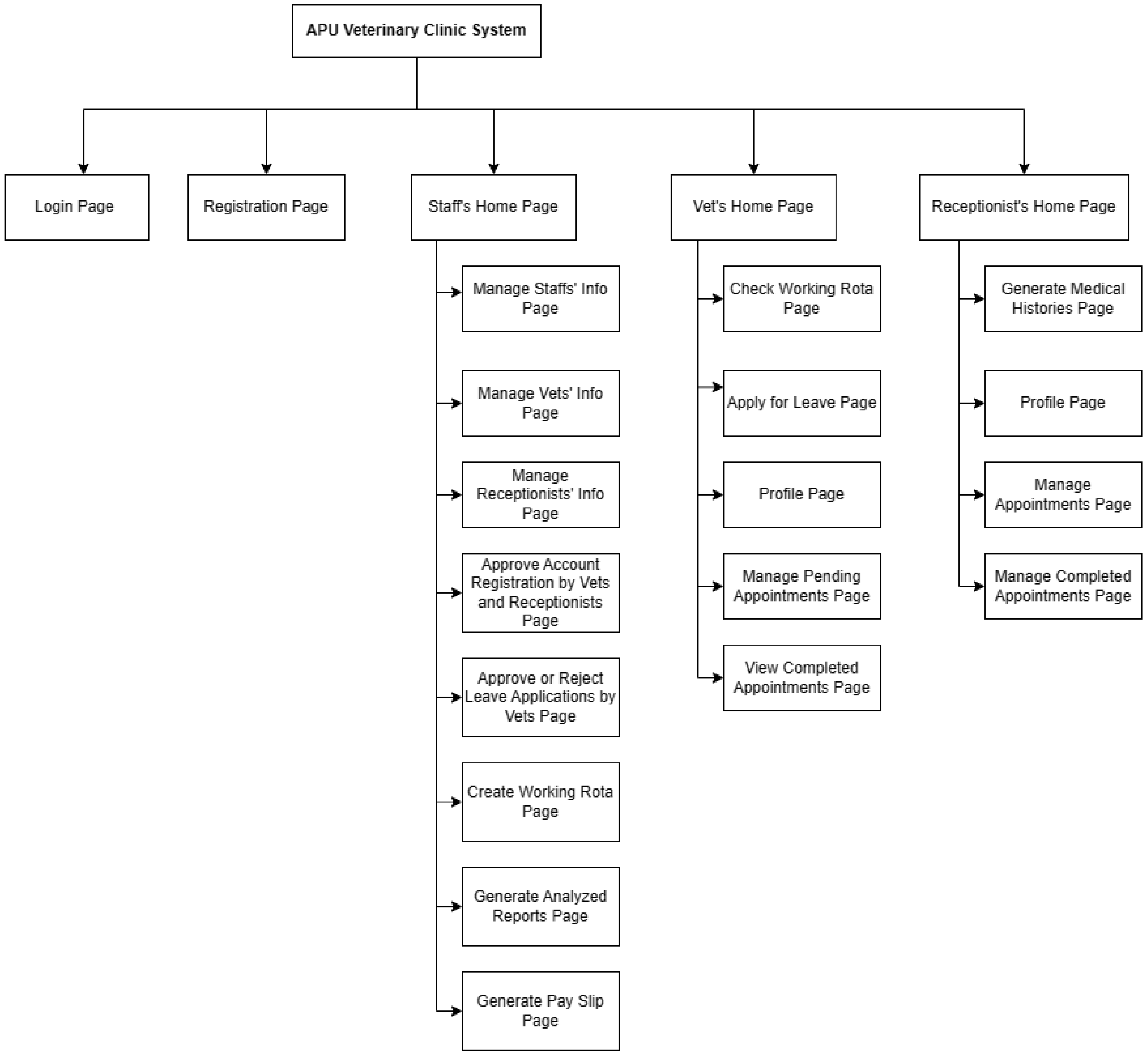

APU Veterinary Clinic System
Login Page
Registration Page
Staff's Home Page
Vet's Home Page
Receptionist's Home Page
Manage Staffs' Info Page
Manage Vets' Info Page
Manage Receptionists' Info Page
Approve Account Registration by Vets and Receptionists Page
Approve or Reject Leave Applications by Vets Page
Create Working Rota Page
Generate Analyzed Reports Page
Generate Pay Slip Page
Check Working Rota Page
Apply for Leave Page
Profile Page
Manage Pending Appointments Page
View Completed Appointments Page
Generate Medical Histories Page
Profile Page
Manage Appointments Page
Manage Completed Appointments Page

Figure 54: Sitemap

Part B2

Overview of the application

The application was developed using a multi-tier architecture: the presentation tier, the business tier, and the database tier.

The presentation tier of the application is based on the JSP technology and Servlet technology. In this application, JSP technology is used to render the user interfaces through HTML tags. For example, in the case of adding a new record of a staff through the user interface, the JSP can be used to generate some necessary HTML tags for generating texts, text fields, and buttons for prompting the users. In each of the text fields, a parameter name is given so that the input text can be sent as its parameter value to the specified Servlet file through HTTP POST request. Other than that, Servlet is also used to generate some parts of the user interface as it can include some HTML codes based on a specified JSP file. For example, in the case of updating the list of records once a new record is added, the Servlet file is used to query the database to retrieve a new list of records, then the updated list of records is included based on the specified JSP file.

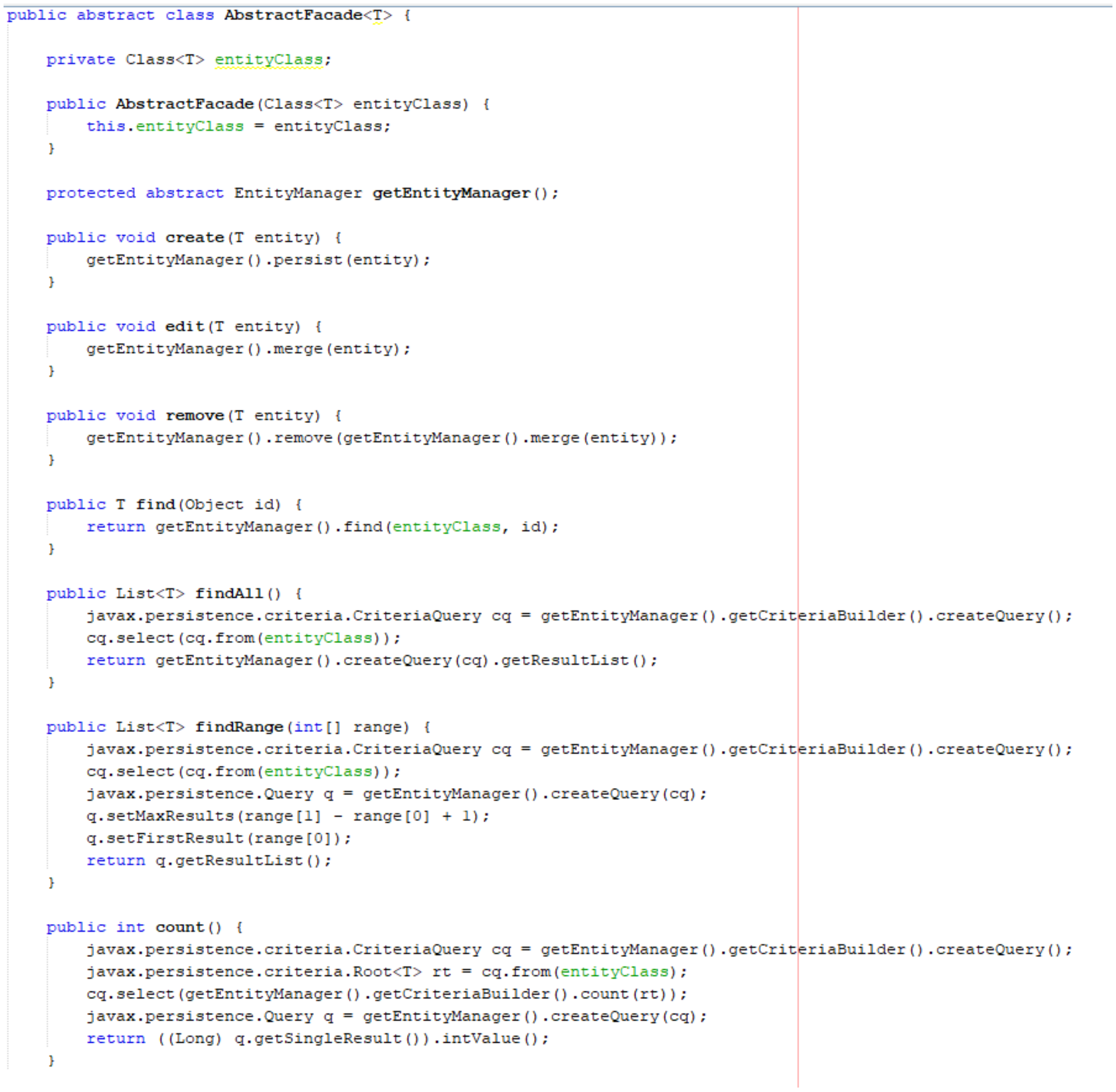
The business tier of the application is based on the Servlet technology, Persistence Entity technology, and Persistence Session Beans for Entity Classes technology. The Persistence Entity is a Java Class, and it is used to create a table in the database, by specifying all the attributes. The Persistence Session Beans for Entity Classes is used to create a façade for each Persistence Entity, and the façade comes with default methods for performing CRUD operations on the database. Other than the default methods provided by the façade, some customized methods for manipulating the database can be written by inserting a SQL statement into them. The Servlet here is to call the façades and their methods to interact with the database, then return some content to the user interface.

The database tier of the application is based on the Java DB (Derby) Database. The Derby database is used to store the data generated by users through the user interface. Without the database, all the data generated by the users will disappear if the application is shut down or rebooted.

Speaking of interconnections among the tiers, the presentation tier is to render the user interface to the users, so that the users can input data into the application. Then, the input data is sent to the business tier through the HTTP POST request, business logic is executed so that it will return some messages to the presentation tier, or it will interact with the database tier to perform common CRUD operations on the database. The database tier is to store the data generated through the presentation tier.

Design of the Business Tier

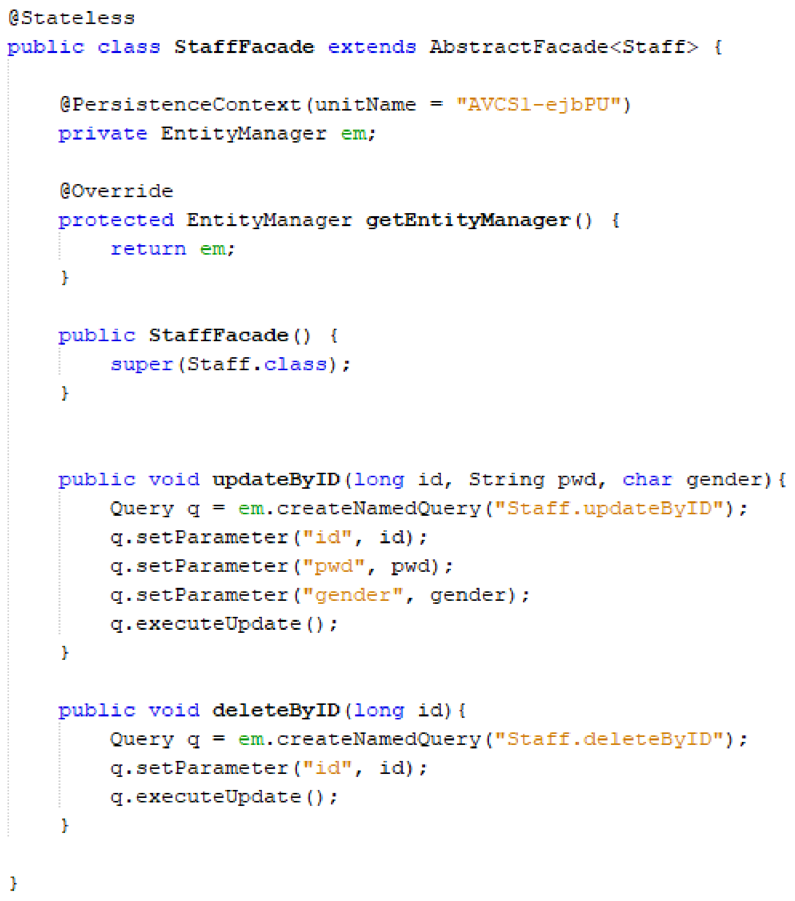
In the business tier, EJB technologies are used to create tables in database and interact with the database. To be specific, Persistence Entity Classes are used to create tables in database, while Persistence Session Beans are used to create façades for interacting with the database. An abstract façade is an abstract class that contains some basic database access methods:



```java
public abstract class AbstractFacade<T> {

    private Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) {
        getEntityManager().remove(getEntityManager().merge(entity));
    }

    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    public List<T> findRange(int[] range) {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        q.setMaxResults(range[1] - range[0] + 1);
        q.setFirstResult(range[0]);
        return q.getResultList();
    }

    public int count() {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        javax.persistence.criteria.Root<T> rt = cq.from(entityClass);
        cq.select(getEntityManager().getCriteriaBuilder().count(rt));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    }
```

Figure 55: Abstract Façade providing some basic database access methods

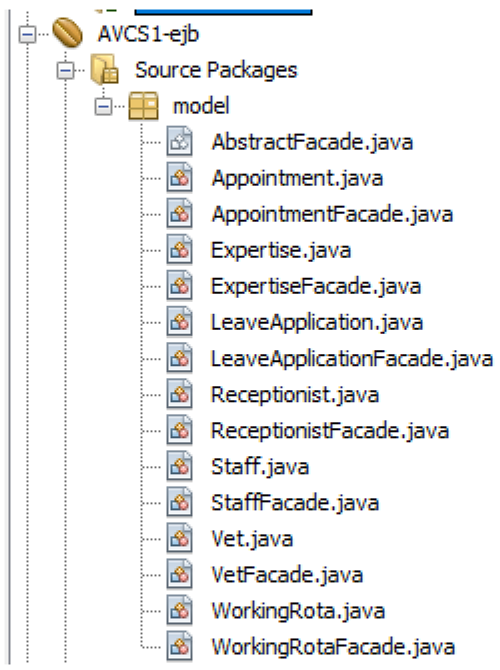
The abstract façade will be inherited by all the façade by each table in the database, on top of the basic database access methods, programmers can customize their database access methods for each table using SQL statements. For example, to create a table called ***Staff*** in database, an entity class named as ***Staff*** is created. Inside the entity class ***Staff***, the instance attributes mean the columns of the table, constructors are defined so that an object that represents a record in table ***Staff*** can be created (the object can be used with the basic database access methods in Abstract Façade to perform CRUD operations), several getters and setters are also defined under the entity class. Then a façade called ***StaffFacade*** is created for interacting with the database, it provides the basic database access methods inherited from the Abstract Façade, then programmers can customize any database access method they want using SQL statements, such as the method to update all staff's information based on the staff's ID, and the method to delete a staff's account based on the staff's ID:



```
@Stateless
public class StaffFacade extends AbstractFacade<Staff> {

    @PersistenceContext(unitName = "AVCS1-ejbPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public StaffFacade() {
        super(Staff.class);
    }


    public void updateByID(long id, String pwd, char gender){
        Query q = em.createNamedQuery("Staff.updateByID");
        q.setParameter("id", id);
        q.setParameter("pwd", pwd);
        q.setParameter("gender", gender);
        q.executeUpdate();
    }

    public void deleteByID(long id){
        Query q = em.createNamedQuery("Staff.deleteByID");
        q.setParameter("id", id);
        q.executeUpdate();
    }

}
```

Figure 56: Customizing database access methods in StaffFacade

The design of the business tier is shown in the figure below:



```
AVCS1-ejb
  Source Packages
    model
      AbstractFacade.java
      Appointment.java
      AppointmentFacade.java
      Expertise.java
      ExpertiseFacade.java
      LeaveApplication.java
      LeaveApplicationFacade.java
      Receptionist.java
      ReceptionistFacade.java
      Staff.java
      StaffFacade.java
      Vet.java
      VetFacade.java
      WorkingRota.java
      WorkingRotaFacade.java
```

Figure 57: Design of the business tier

The basic requirements of the assignment mention that there will be three types of users, hence entity classes ***Staff***, ***Vet***, and ***Receptionist*** are defined. Other than that, receptionists are required to create appointments for customers and pets and assign a vet to the appointment, hence an entity class ***Appointment*** to contain all the necessary information. Moreover, the staffs can create week's working rota for vets, so an entity class ***WorkingRota*** is created to contain the starting date of the week's working rota (date of Monday), and the usernames of the remaining 21 vets. The vets can also apply for leave and wait for approvals or to be rejected by the staffs, hence an entity class called ***LeaveApplication*** is defined to contain the date of leave, vet's username, reason, and status of the leave application. Lastly, since the system is for a clinic, hence there should be expertise of each vet, an entity class called ***Expertise*** is defined to contain the name of the expertise. For more details about the defined database access methods in each Façade, please refer to Part B3.

UML diagram (use case and class)

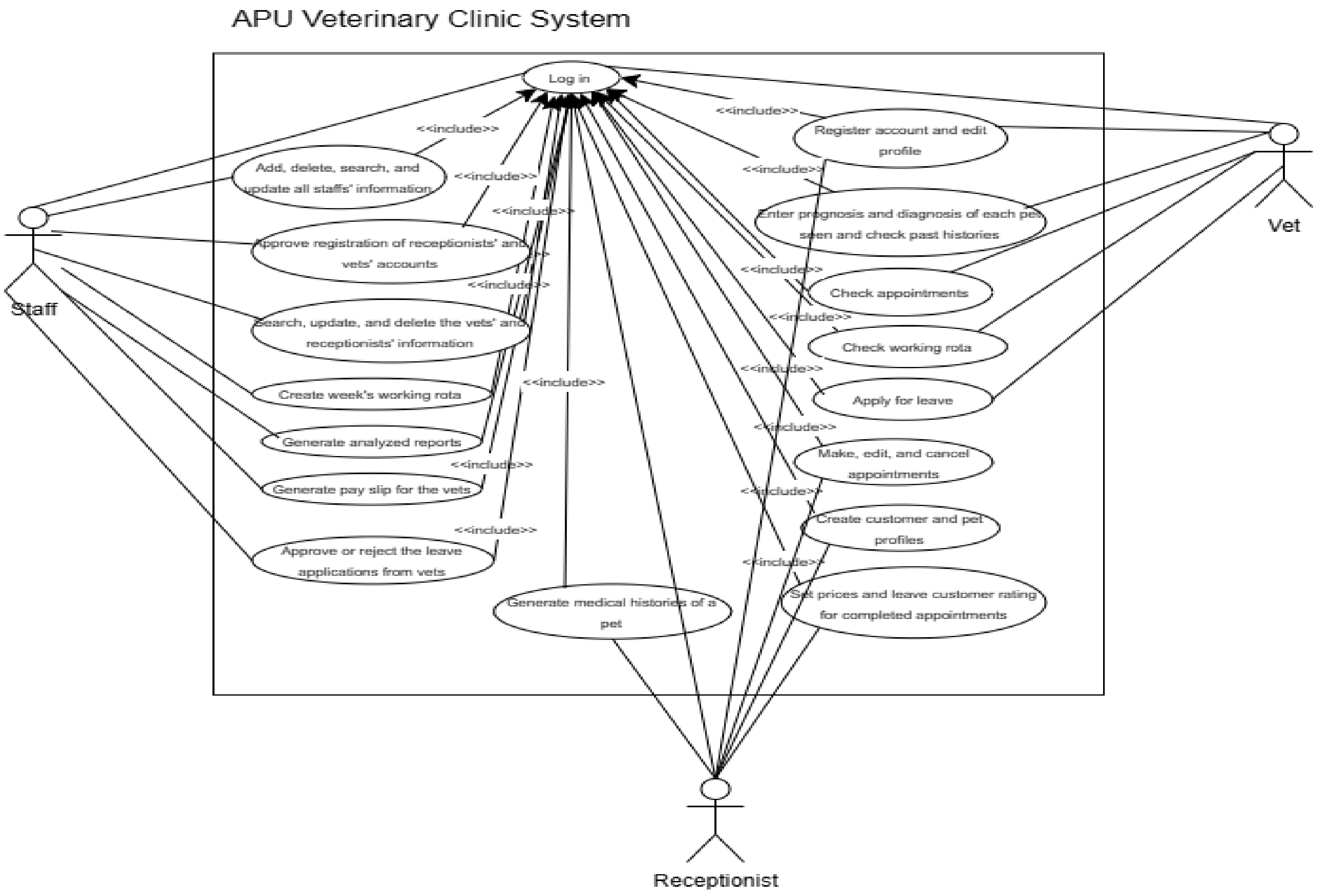

APU Veterinary Clinic System
Log in
<<include>>
Add, delete, search, and update all staffs' information
Approve registration of receptionists' and vets' accounts
Search, update, and delete the vets' and receptionists' information
Create week's working rota
Generate analyzed reports
Generate pay slip for the vets
Approve or reject the leave applications from vets
Generate medical histories of a pet
Register account and edit profile
Enter prognosis and diagnosis of each pet seen and check past histories
Check appointments
Check working rota
Apply for leave
Make, edit, and cancel appointments
Create customer and pet profiles
Set prices and leave customer rating for completed appointments
Staff
Vet
Receptionist

Figure 58: UML Use Case Diagram

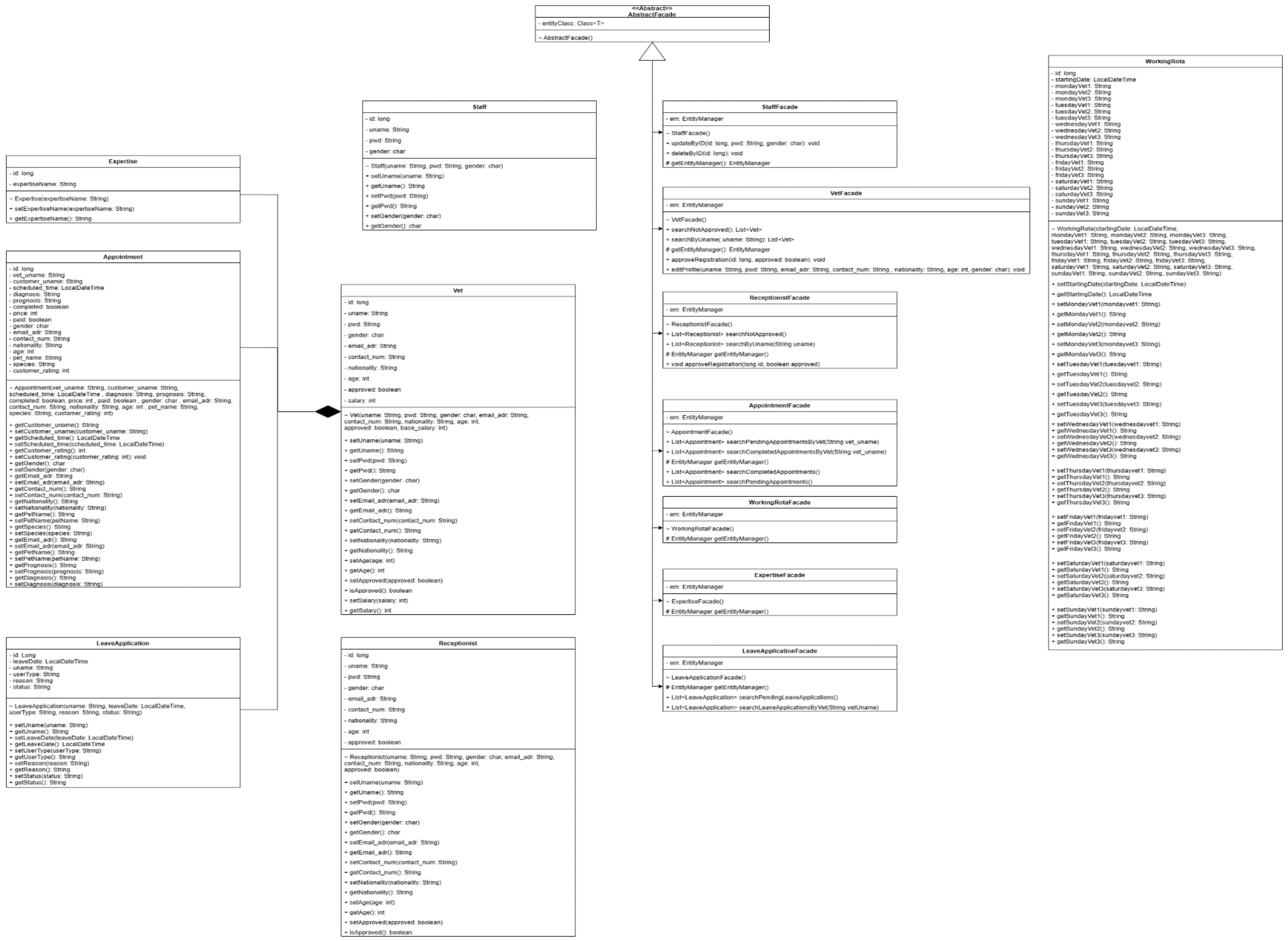

<<Abstract>>
AbstractFacade
- entityClass: Class<T>
~ AbstractFacade()
Staff
- id: long
- uname: String
- pwd: String
- gender: char
~ Staff(uname: String, pwd: String, gender: char)
+ setUname(uname: String)
+ getUname(): String
+ setPwd(pwd: String)
+ getPwd(): String
+ setGender(gender: char)
+ getGender(): char
Expertise
- id: long
- expertiseName: String
~ Expertise(expertiseName: String)
+ setExpertiseName(expertiseName: String)
+ getExpertiseName(): String
Appointment
- id: long
- vet_uname: String
- customer_uname: String
- scheduled_time: LocalDateTime
- diagnosis: String
- prognosis: String
- completed: boolean
- price: int
- paid: boolean
- gender: char
- email_adr: String
- contact_num: String
- nationality: String
- age: int
- pet_name: String
- species: String
- customer_rating: int
~ Appointment(vet_uname: String, customer_uname: String, scheduled_time: LocalDateTime , diagnosis: String, prognosis: String, completed: boolean, price: int , paid: boolean , gender: char , email_adr: String, contact_num: String, nationality: String, age: int , pet_name: String, species: String, customer_rating: int)
+ getCustomer_uname(): String
+ setCustomer_uname(customer_uname: String)
+ getScheduled_time(): LocalDateTime
+ setScheduled_time(scheduled_time: LocalDateTime)
+ getCustomer_rating(): int
+ setCustomer_rating(customer_rating: int): void
+ getGender(): char
+ setGender(gender: char)
+ getEmail_adr: String
+ setEmail_adr(email_adr: String)
+ getContact_num(): String
+ setContact_num(contact_num: String)
+ getNationality(): String
+ setNationality(nationality: String)
+ getPetName(): String
+ setPetName(petName: String)
+ getSpecies(): String
+ setSpecies(species: String)
+ getEmail_adr(): String
+ setEmail_adr(email_adr: String)
+ getPetName(): String
+ setPetName(petName: String)
+ getPrognosis(): String
+ setPrognosis(prognosis: String)
+ getDiagnosis(): String
+ setDiagnosis(diagnosis: String)
Vet
- id: long
- uname: String
- pwd: String
- gender: char
- email_adr: String
- contact_num: String
- nationality: String
- age: int
- approved: boolean
- salary: int
~ Vet(uname: String, pwd: String, gender: char, email_adr: String, contact_num: String, nationality: String, age: int, approved: boolean, base_salary: int)
+ setUname(uname: String)
+ getUname(): String
+ setPwd(pwd: String)
+ getPwd(): String
+ setGender(gender: char)
+ getGender(): char
+ setEmail_adr(email_adr: String)
+ getEmail_adr(): String
+ setContact_num(contact_num: String)
+ getContact_num(): String
+ setNationality(nationality: String)
+ getNationality(): String
+ setAge(age: int)
+ getAge(): int
+ setApproved(approved: boolean)
+ isApproved(): boolean
+ setSalary(salary: int)
+ getSalary(): int
LeaveApplication
- id: Long
- leaveDate: LocalDateTime
- uname: String
- userType: String
- reason: String
- status: String
~ LeaveApplication(uname: String, leaveDate: LocalDateTime, userType: String, reason: String, status: String)
+ setUname(uname: String)
+ getUname(): String
+ setLeaveDate(leaveDate: LocalDateTime)
+ getLeaveDate(): LocalDateTime
+ setUserType(userType: String)
+ getUserType(): String
+ setReason(reason: String)
+ getReason(): String
+ setStatus(status: String)
+ getStatus(): String
Receptionist
- id: long
- uname: String
- pwd: String
- gender: char
- email_adr: String
- contact_num: String
- nationality: String
- age: int
- approved: boolean
~ Receptionist(uname: String, pwd: String, gender: char, email_adr: String, contact_num: String, nationality: String, age: int, approved: boolean)
+ setUname(uname: String)
+ getUname(): String
+ setPwd(pwd: String)
+ getPwd(): String
+ setGender(gender: char)
+ getGender(): char
+ setEmail_adr(email_adr: String)
+ getEmail_adr(): String
+ setContact_num(contact_num: String)
+ getContact_num(): String
+ setNationality(nationality: String)
+ getNationality(): String
+ setAge(age: int)
+ getAge(): int
+ setApproved(approved: boolean)
+ isApproved(): boolean
StaffFacade
- em: EntityManager
~ StaffFacade()
+ updateByID(id: long, pwd: String, gender: char): void
+ deleteByID(id: long): void
# getEntityManager(): EntityManager
VetFacade
- em: EntityManager
~ VetFacade()
+ searchNotApproved(): List<Vet>
+ searchByUname(uname: String): List<Vet>
# getEntityManager(): EntityManager
+ approveRegistration(id: long, approved: boolean): void
+ editProfile(uname: String, pwd: String, email_adr: String, contact_num: String , nationality: String, age: int, gender: char): void
ReceptionistFacade
- em: EntityManager
~ ReceptionistFacade()
+ List<Receptionist> searchNotApproved()
+ List<Receptionist> searchByUname(String uname)
# EntityManager getEntityManager()
+ void approveRegistration(long id, boolean approved)
AppointmentFacade
- em: EntityManager
~ AppointmentFacade()
+ List<Appointment> searchPendingAppointmentsByVet(String vet_uname)
+ List<Appointment> searchCompletedAppointmentsByVet(String vet_uname)
# EntityManager getEntityManager()
+ List<Appointment> searchCompletedAppointments()
+ List<Appointment> searchPendingAppointments()
WorkingRotaFacade
- em: EntityManager
~ WorkingRotaFacade()
# EntityManager getEntityManager()
ExpertiseFacade
- em: EntityManager
~ ExpertiseFacade()
# EntityManager getEntityManager()
LeaveApplicationFacade
- em: EntityManager
~ LeaveApplicationFacade()
# EntityManager getEntityManager()
+ List<LeaveApplication> searchPendingLeaveApplications()
+ List<LeaveApplication> searchLeaveApplicationsByVet(String vetUname)
WorkingRota
- id: long
- startingDate: LocalDateTime
- mondayVet1: String
- mondayVet2: String
- mondayVet3: String
- tuesdayVet1: String
- tuesdayVet2: String
- tuesdayVet3: String
- wednesdayVet1: String
- wednesdayVet2: String
- wednesdayVet3: String
- thursdayVet1: String
- thursdayVet2: String
- thursdayVet3: String
- fridayVet1: String
- fridayVet2: String
- fridayVet3: String
- saturdayVet1: String
- saturdayVet2: String
- saturdayVet3: String
- sundayVet1: String
- sundayVet2: String
- sundayVet3: String
~ WorkingRota(startingDate: LocalDateTime, mondayVet1: String, mondayVet2: String, mondayVet3: String, tuesdayVet1: String, tuesdayVet2: String, tuesdayVet3: String, wednesdayVet1: String, wednesdayVet2: String, wednesdayVet3: String, thursdayVet1: String, thursdayVet2: String, thursdayVet3: String, fridayVet1: String, fridayVet2: String, fridayVet3: String, saturdayVet1: String, saturdayVet2: String, saturdayVet3: String, sundayVet1: String, sundayVet2: String, sundayVet3: String)
+ setStartingDate(startingDate: LocalDateTime)
+ getStartingDate(): LocalDateTime
+ setMondayVet1(mondayvet1: String)
+ getMondayVet1(): String
+ setMondayVet2(mondayvet2: String)
+ getMondayVet2(): String
+ setMondayVet3(mondayvet3: String)
+ getMondayVet3(): String
+ setTuesdayVet1(tuesdayvet1: String)
+ getTuesdayVet1(): String
+ setTuesdayVet2(tuesdayvet2: String)
+ getTuesdayVet2(): String
+ setTuesdayVet3(tuesdayvet3: String)
+ getTuesdayVet3(): String
+ setWednesdayVet1(wednesdayvet1: String)
+ getWednesdayVet1(): String
+ setWednesdayVet2(wednesdayvet2: String)
+ getWednesdayVet2(): String
+ setWednesdayVet3(wednesdayvet3: String)
+ getWednesdayVet3(): String
+ setThursdayVet1(thursdayvet1: String)
+ getThursdayVet1(): String
+ setThursdayVet2(thursdayvet2: String)
+ getThursdayVet2(): String
+ setThursdayVet3(thursdayvet3: String)
+ getThursdayVet3(): String
+ setFridayVet1(fridayvet1: String)
+ getFridayVet1(): String
+ setFridayVet2(fridayvet2: String)
+ getFridayVet2(): String
+ setFridayVet3(fridayvet3: String)
+ getFridayVet3(): String
+ setSaturdayVet1(saturdayvet1: String)
+ getSaturdayVet1(): String
+ setSaturdayVet2(saturdayvet2: String)
+ getSaturdayVet2(): String
+ setSaturdayVet3(saturdayvet3: String)
+ getSaturdayVet3(): String
+ setSundayVet1(sundayvet1: String)
+ getSundayVet1(): String
+ setSundayVet2(sundayvet2: String)
+ getSundayVet2(): String
+ setSundayVet3(sundayvet3: String)
+ getSundayVet3(): String

Figure 59: UML Class Diagram

Part B3

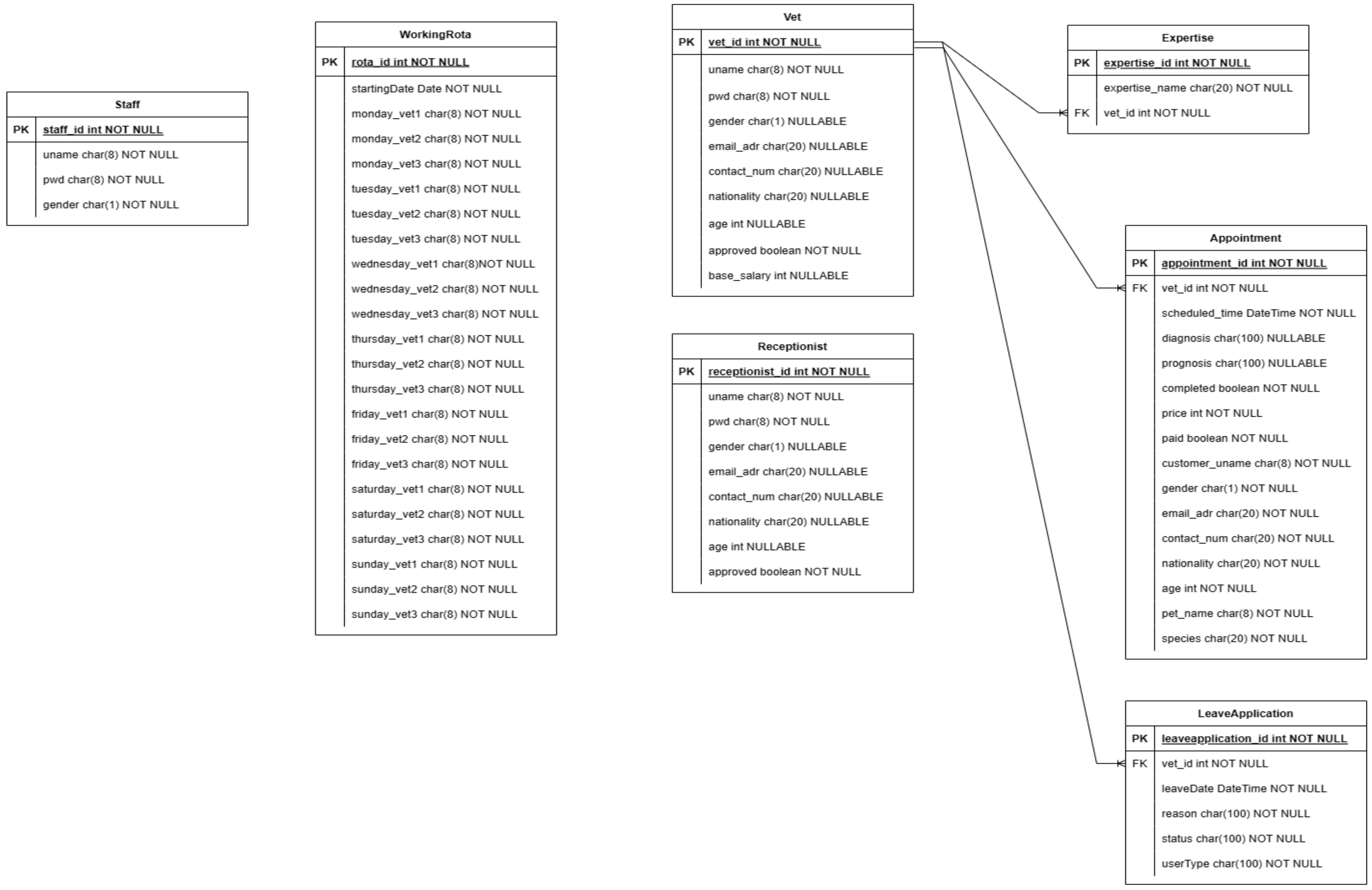
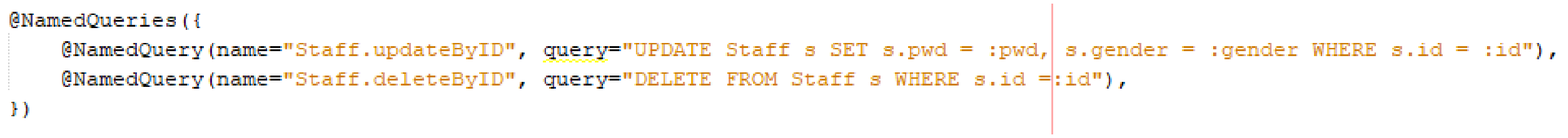

Staff
PK staff_id int NOT NULL
uname char(8) NOT NULL
pwd char(8) NOT NULL
gender char(1) NOT NULL
WorkingRota
PK rota_id int NOT NULL
startingDate Date NOT NULL
monday_vet1 char(8) NOT NULL
monday_vet2 char(8) NOT NULL
monday_vet3 char(8) NOT NULL
tuesday_vet1 char(8) NOT NULL
tuesday_vet2 char(8) NOT NULL
tuesday_vet3 char(8) NOT NULL
wednesday_vet1 char(8)NOT NULL
wednesday_vet2 char(8) NOT NULL
wednesday_vet3 char(8) NOT NULL
thursday_vet1 char(8) NOT NULL
thursday_vet2 char(8) NOT NULL
thursday_vet3 char(8) NOT NULL
friday_vet1 char(8) NOT NULL
friday_vet2 char(8) NOT NULL
friday_vet3 char(8) NOT NULL
saturday_vet1 char(8) NOT NULL
saturday_vet2 char(8) NOT NULL
saturday_vet3 char(8) NOT NULL
sunday_vet1 char(8) NOT NULL
sunday_vet2 char(8) NOT NULL
sunday_vet3 char(8) NOT NULL
Vet
PK vet_id int NOT NULL
uname char(8) NOT NULL
pwd char(8) NOT NULL
gender char(1) NULLABLE
email_adr char(20) NULLABLE
contact_num char(20) NULLABLE
nationality char(20) NULLABLE
age int NULLABLE
approved boolean NOT NULL
base_salary int NULLABLE
Receptionist
PK receptionist_id int NOT NULL
uname char(8) NOT NULL
pwd char(8) NOT NULL
gender char(1) NULLABLE
email_adr char(20) NULLABLE
contact_num char(20) NULLABLE
nationality char(20) NULLABLE
age int NULLABLE
approved boolean NOT NULL
Expertise
PK expertise_id int NOT NULL
expertise_name char(20) NOT NULL
FK vet_id int NOT NULL
Appointment
PK appointment_id int NOT NULL
FK vet_id int NOT NULL
scheduled_time DateTime NOT NULL
diagnosis char(100) NULLABLE
prognosis char(100) NULLABLE
completed boolean NOT NULL
price int NOT NULL
paid boolean NOT NULL
customer_uname char(8) NOT NULL
gender char(1) NOT NULL
email_adr char(20) NOT NULL
contact_num char(20) NOT NULL
nationality char(20) NOT NULL
age int NOT NULL
pet_name char(8) NOT NULL
species char(20) NOT NULL
LeaveApplication
PK leaveapplication_id int NOT NULL
FK vet_id int NOT NULL
leaveDate DateTime NOT NULL
reason char(100) NOT NULL
status char(100) NOT NULL
userType char(100) NOT NULL

Figure 60: E-R Diagram for Database Design

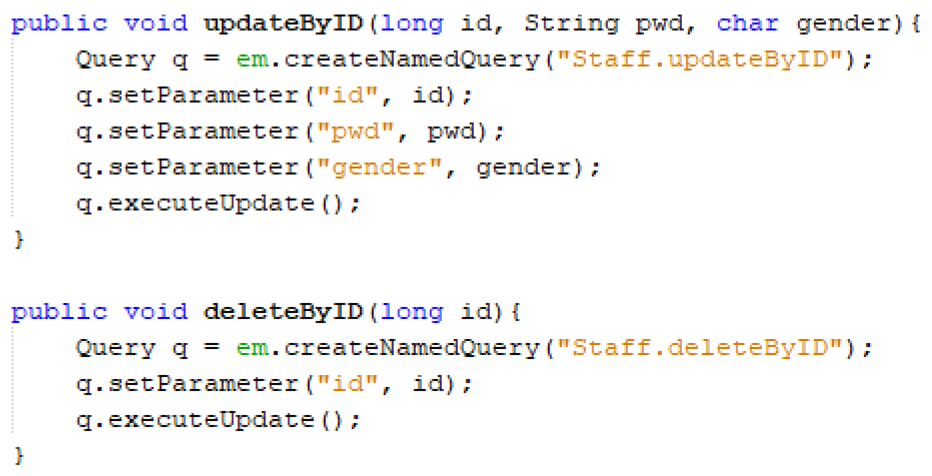
For the database design, there will be 7 tables for storing the data generated by the system, that is: ***Staff***, ***Vet***, ***Receptionist***, ***WorkingRota***, ***Expertise***, ***Appointment***, and ***LeaveApplication***.

The table ***Staff*** is for storing relevant information of the managing staff, such as credentials and their genders. The same story goes to the table ***Vet*** and ***Receptionist***, but more information needs to be stored for supporting the other system functionalities. For example, the column *approved* in both tables ***Vet*** and ***Receptionist***, is to indicate whether the registration of the account has been approved by the managing staff. Other than that, the column *base_salary* of the table ***Vet*** is for managing staffs to generate pay slip for vets. For the remaining columns of both tables ***Vet*** and ***Receptionist***, are for storing necessary personal information as they are working for the clinic. The two figures below show the design of the database access APIs of the table ***Staff***: The first API is for updating all information of the staff based on the staff ID, while the second API is for deleting the staff record from the databased based on the staff ID.



```
@NamedQueries({
    @NamedQuery(name="Staff.updateByID", query="UPDATE Staff s SET s.pwd = :pwd, s.gender = :gender WHERE s.id = :id"),
    @NamedQuery(name="Staff.deleteByID", query="DELETE FROM Staff s WHERE s.id =:id"),
})
```

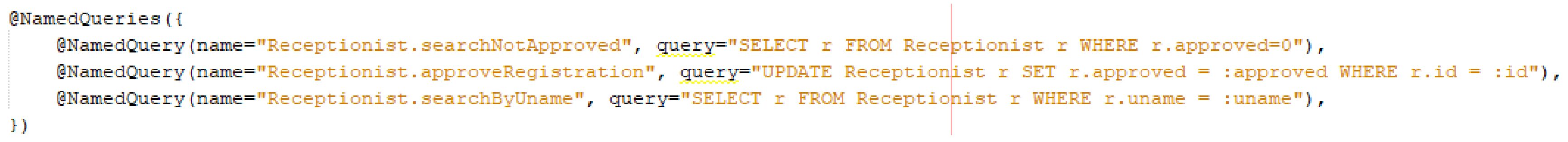
Figure 61: SQL Queries of the table Staff



```
public void updateByID(long id, String pwd, char gender){
    Query q = em.createNamedQuery("Staff.updateByID");
    q.setParameter("id", id);
    q.setParameter("pwd", pwd);
    q.setParameter("gender", gender);
    q.executeUpdate();
}

public void deleteByID(long id){
    Query q = em.createNamedQuery("Staff.deleteByID");
    q.setParameter("id", id);
    q.executeUpdate();
}
```

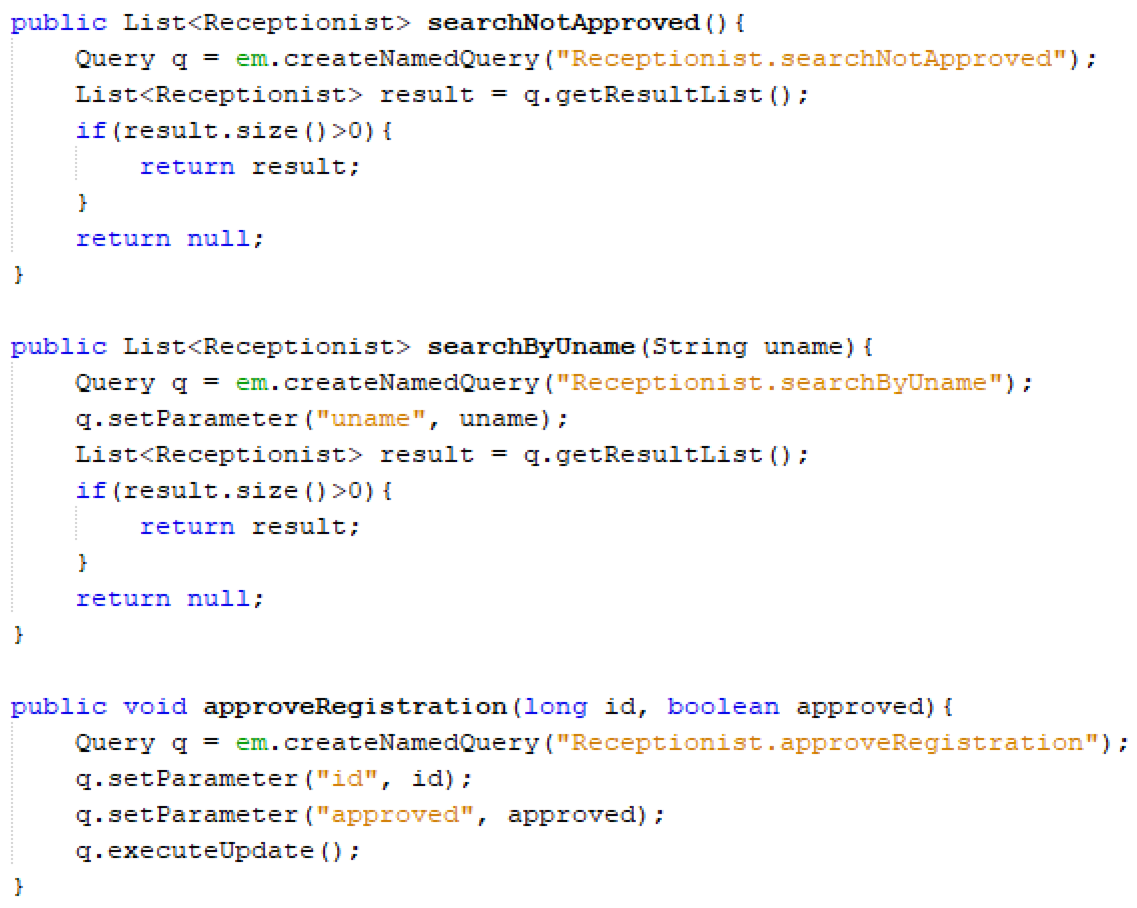
Figure 62: Database Access APIs of the table Staff

The two figures below show the database access APIs for the table ***Receptionist***: The first API is for retrieving all the receptionist accounts that have not been approved, the second API is for retrieving a receptionist account based on the receptionist's username, and the third API is for approve the registration of a receptionist account based on the receptionist's ID.



```
@NamedQueries({
    @NamedQuery(name="Receptionist.searchNotApproved", query="SELECT r FROM Receptionist r WHERE r.approved=0"),
    @NamedQuery(name="Receptionist.approveRegistration", query="UPDATE Receptionist r SET r.approved = :approved WHERE r.id = :id"),
    @NamedQuery(name="Receptionist.searchByUname", query="SELECT r FROM Receptionist r WHERE r.uname = :uname"),
})
```

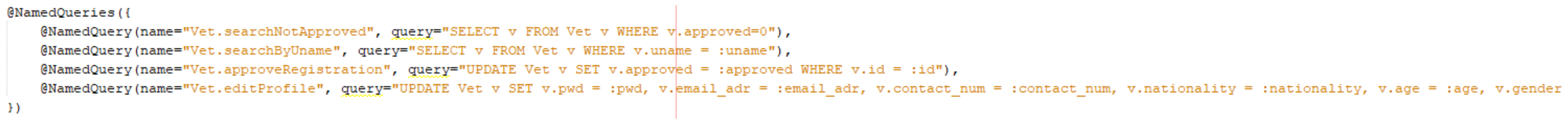
Figure 63: SQL Queries of the table Receptionist



```java
public List<Receptionist> searchNotApproved(){
    Query q = em.createNamedQuery("Receptionist.searchNotApproved");
    List<Receptionist> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}

public List<Receptionist> searchByUname(String uname){
    Query q = em.createNamedQuery("Receptionist.searchByUname");
    q.setParameter("uname", uname);
    List<Receptionist> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}

public void approveRegistration(long id, boolean approved){
    Query q = em.createNamedQuery("Receptionist.approveRegistration");
    q.setParameter("id", id);
    q.setParameter("approved", approved);
    q.executeUpdate();
}
```

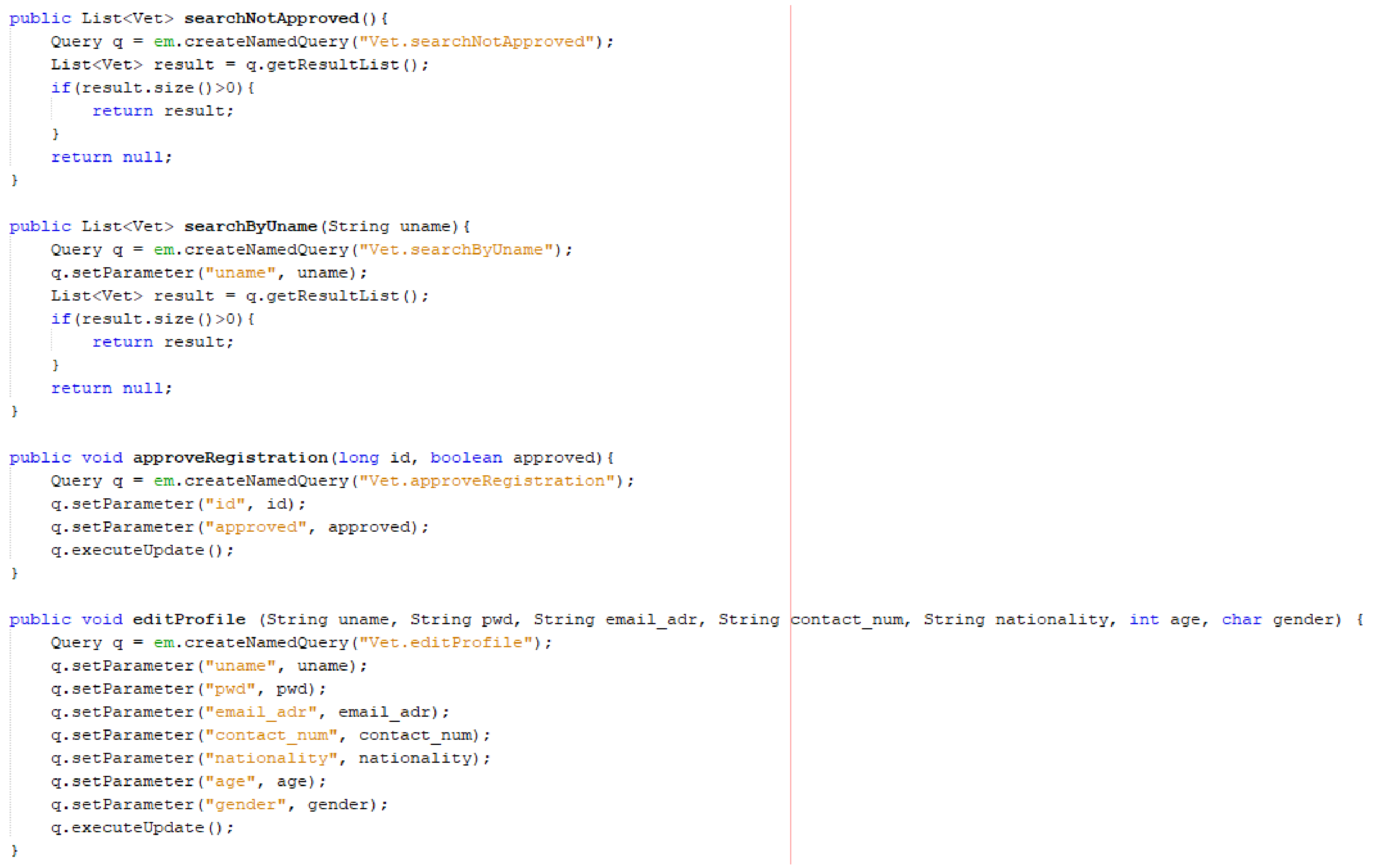
Figure 64: Database Access APIs Designs of the table Receptionist

The two figures below are the database access APIs: the first API is for retrieving all the vet accounts that have not been approved, the second API is for retrieving a vet account based on the vet's username, the third API is for approving the registration of the vet account, and the fourth API is for updating the information of a vet account based on the vet's ID.



```
@NamedQueries({
    @NamedQuery(name="Vet.searchNotApproved", query="SELECT v FROM Vet v WHERE v.approved=0"),
    @NamedQuery(name="Vet.searchByUname", query="SELECT v FROM Vet v WHERE v.uname = :uname"),
    @NamedQuery(name="Vet.approveRegistration", query="UPDATE Vet v SET v.approved = :approved WHERE v.id = :id"),
    @NamedQuery(name="Vet.editProfile", query="UPDATE Vet v SET v.pwd = :pwd, v.email_adr = :email_adr, v.contact_num = :contact_num, v.nationality = :nationality, v.age = :age, v.gender
})
```

Figure 65: SQL Queries of the table Vet

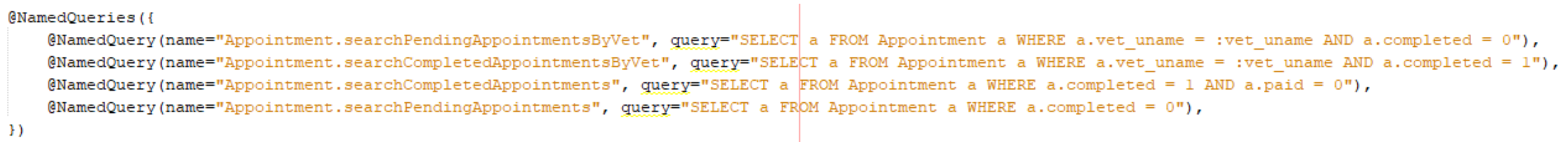


```java
public List<Vet> searchNotApproved(){
    Query q = em.createNamedQuery("Vet.searchNotApproved");
    List<Vet> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}

public List<Vet> searchByUname(String uname){
    Query q = em.createNamedQuery("Vet.searchByUname");
    q.setParameter("uname", uname);
    List<Vet> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}

public void approveRegistration(long id, boolean approved){
    Query q = em.createNamedQuery("Vet.approveRegistration");
    q.setParameter("id", id);
    q.setParameter("approved", approved);
    q.executeUpdate();
}

public void editProfile (String uname, String pwd, String email_adr, String contact_num, String nationality, int age, char gender) {
    Query q = em.createNamedQuery("Vet.editProfile");
    q.setParameter("uname", uname);
    q.setParameter("pwd", pwd);
    q.setParameter("email_adr", email_adr);
    q.setParameter("contact_num", contact_num);
    q.setParameter("nationality", nationality);
    q.setParameter("age", age);
    q.setParameter("gender", gender);
    q.executeUpdate();
}
```

Figure 66: Database Access Design APIs of the table Vet

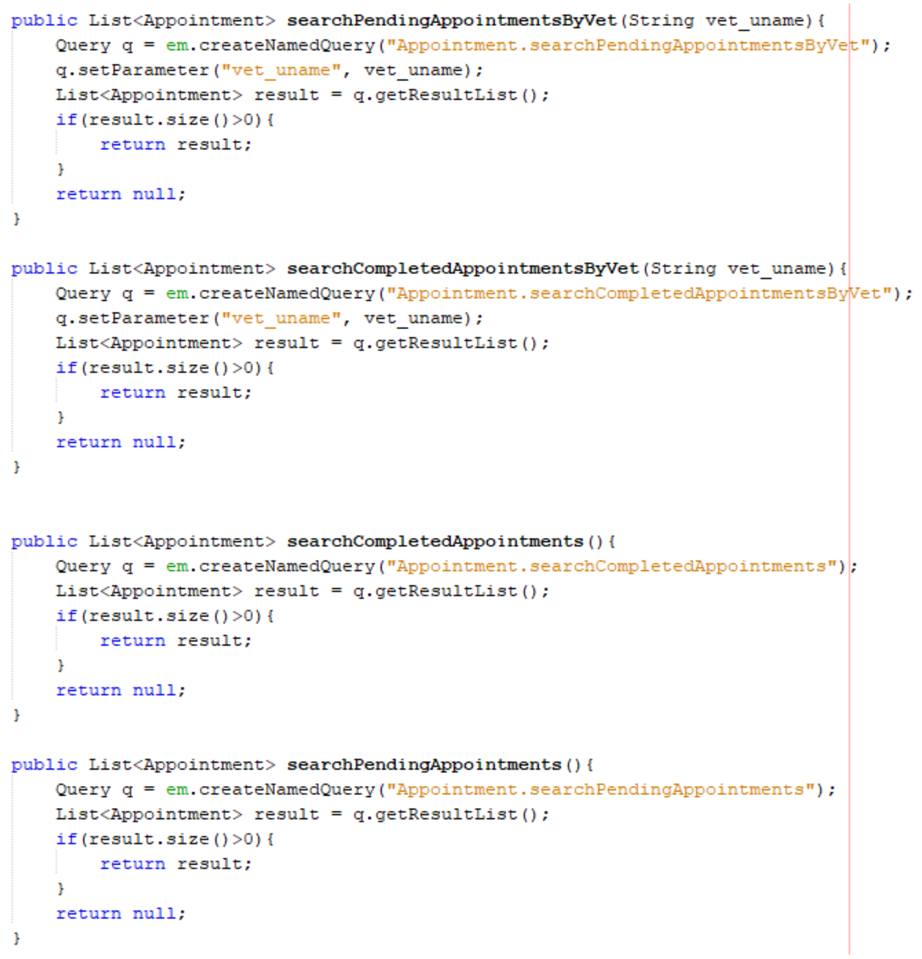
The table ***Expertise*** is for storing all the expertise available in this clinic, and it consists of a column *expertise_name* for stating the name of the expertise. The relationship between the table ***Vet*** and ***Expertise*** is one-to-many, which means that a vet can have many expertise at the same time.

The table ***Appointment*** is for storing all the appointments made by the receptionists. The relationship between the table ***Vet*** and ***Appointment*** is one-to-many, which means that a vet can have many appointments at the same time. In each appointment record, it consists of the scheduled datetime, the ID of the assigned vet, diagnosis and prognosis entered by the assigned vet, a Boolean value to indicate whether the appointment has been completed, price, a Boolean value to indicate whether the appointment has been paid by the customer, customer's username, customer's gender, customer's email address, customer's contact number, customer's nationality, customer's age, pet's name, and pet's species (must fulfill any one of the available expertise provided by the clinic). The two figures below show the database access APIs of the table ***Appointment***: the first API is for retrieving all pending appointments by a vet, the second API is for retrieving all completed appointment by a vet, the third API is for retrieving all the completed appointments, and the fourth API is for retrieving all pending appointments.



```
@NamedQueries({
    @NamedQuery(name="Appointment.searchPendingAppointmentsByVet", query="SELECT a FROM Appointment a WHERE a.vet_uname = :vet_uname AND a.completed = 0"),
    @NamedQuery(name="Appointment.searchCompletedAppointmentsByVet", query="SELECT a FROM Appointment a WHERE a.vet_uname = :vet_uname AND a.completed = 1"),
    @NamedQuery(name="Appointment.searchCompletedAppointments", query="SELECT a FROM Appointment a WHERE a.completed = 1 AND a.paid = 0"),
    @NamedQuery(name="Appointment.searchPendingAppointments", query="SELECT a FROM Appointment a WHERE a.completed = 0"),
})
```

Figure 67: SQL Queries of the table Appointment

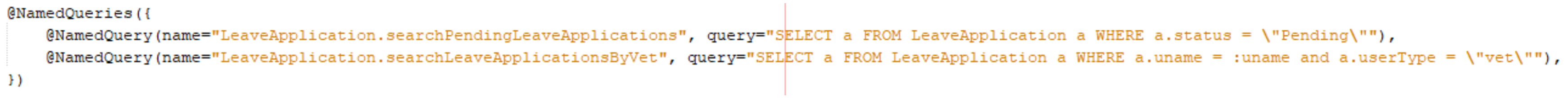


```
public List<Appointment> searchPendingAppointmentsByVet(String vet_uname){
    Query q = em.createNamedQuery("Appointment.searchPendingAppointmentsByVet");
    q.setParameter("vet_uname", vet_uname);
    List<Appointment> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}

public List<Appointment> searchCompletedAppointmentsByVet(String vet_uname){
    Query q = em.createNamedQuery("Appointment.searchCompletedAppointmentsByVet");
    q.setParameter("vet_uname", vet_uname);
    List<Appointment> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}


public List<Appointment> searchCompletedAppointments(){
    Query q = em.createNamedQuery("Appointment.searchCompletedAppointments");
    List<Appointment> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}

public List<Appointment> searchPendingAppointments(){
    Query q = em.createNamedQuery("Appointment.searchPendingAppointments");
    List<Appointment> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}
```

Figure 68: Database Access APIs of the table Appointment

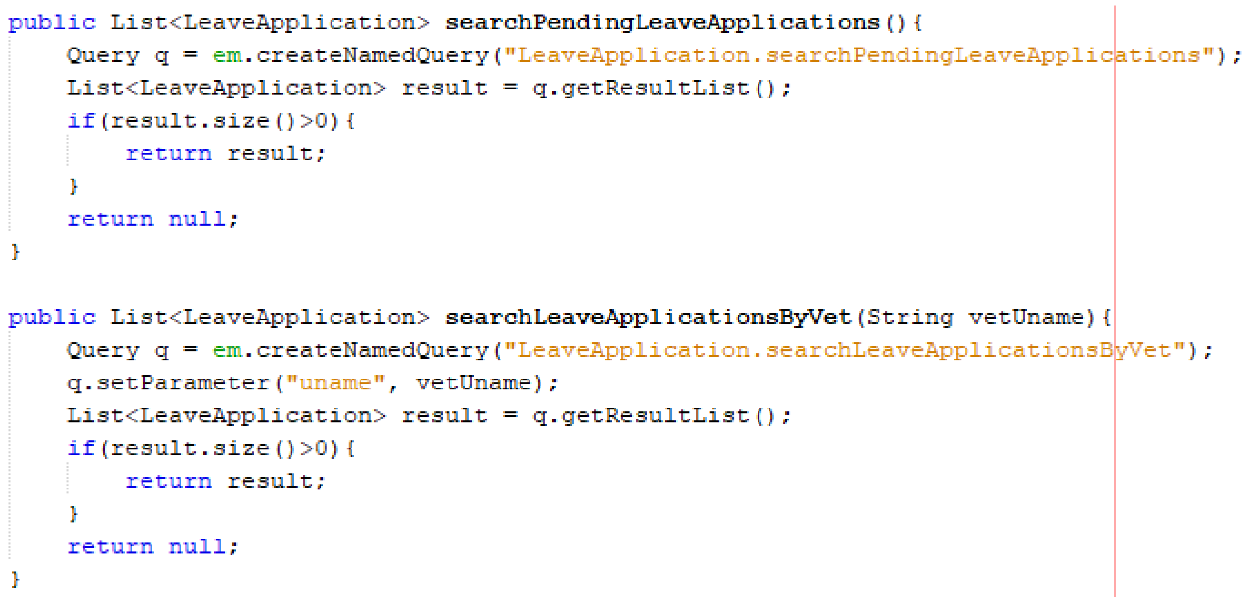
The table ***LeaveApplication*** is for storing all the leave applications submitted by the vets to managing staffs. The relationship between the table ***Vet*** and ***LeaveApplication*** is one-to-many, which means that a vet can have many leave applications at the same time.

For each record of leave applications, it consists of the ID of the vet who applied for leave, the date of the leave, the reason provided by the vet, the status of the leave application (it could be “Pending”, “Approved”, or “Rejected”), and the user type (this could be extended to any type of users so that all of them can apply for leave). The two figures show the database access APIs of the table ***LeaveApplication***: the first API is for retrieving all the pending leave applications, while the second API is for retrieving all leave applications made by a vet.



```
@NamedQueries({
    @NamedQuery(name="LeaveApplication.searchPendingLeaveApplications", query="SELECT a FROM LeaveApplication a WHERE a.status = \"Pending\""),
    @NamedQuery(name="LeaveApplication.searchLeaveApplicationsByVet", query="SELECT a FROM LeaveApplication a WHERE a.uname = :uname and a.userType = \"vet\""),
})
```

Figure 69: SQL Queries of the table LeaveApplication



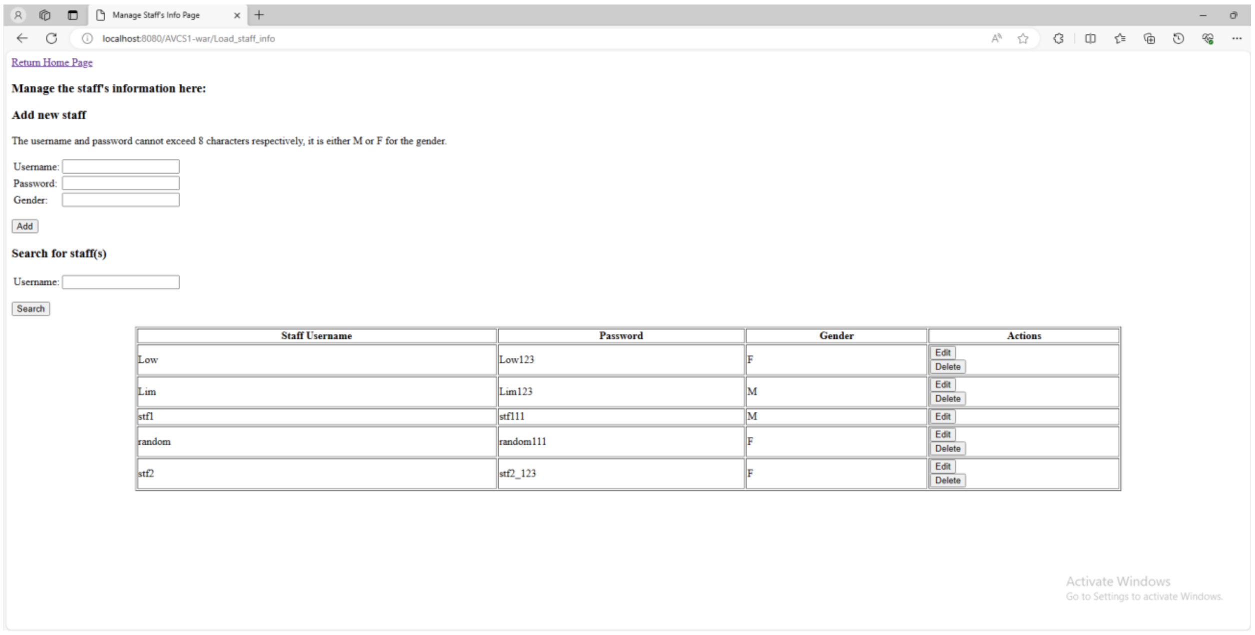
```
public List<LeaveApplication> searchPendingLeaveApplications(){
    Query q = em.createNamedQuery("LeaveApplication.searchPendingLeaveApplications");
    List<LeaveApplication> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}

public List<LeaveApplication> searchLeaveApplicationsByVet(String vetUname){
    Query q = em.createNamedQuery("LeaveApplication.searchLeaveApplicationsByVet");
    q.setParameter("uname", vetUname);
    List<LeaveApplication> result = q.getResultList();
    if(result.size()>0){
        return result;
    }
    return null;
}
```

Figure 70: Database Access APIs of the table LeaveApplication

The table ***WorkingRota*** is for storing all the week’s working rota for the vets. For each record of the working rota, it consists of the date of the first day of the working rota (which is Monday), and the usernames of the 21 vets (there are 7 days in a week, and each day has 3 vets to be on duty).

Description of at least 2 additional features

Display a list of existing staffs' information


Manage Staff's Info Page
localhost:8080/AVCS1-war/Load_staff_info
Return Home Page
Add
Search
Activate Windows
Go to Settings to activate Windows.

Manage the staff's information here:

Add new staff

The username and password cannot exceed 8 characters respectively, it is either M or F for the gender.

Username:
Password:
Gender:

Search for staff(s)

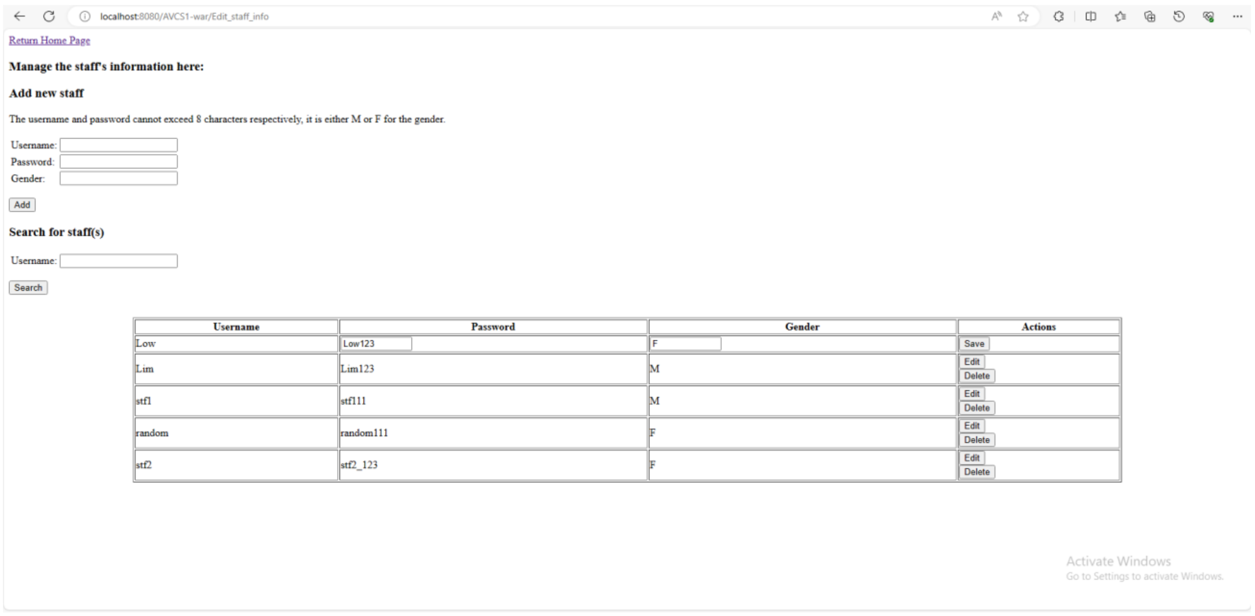
Username:

Staff Username	Password	Gender	Actions
Low	Low123	F	Edit Delete
Lim	Lim123	M	Edit Delete
stf1	stf111	M	Edit
random	random111	F	Edit Delete
stf2	stf2_123	F	Edit Delete

Figure 71: Displaying a list of existing staffs

As the basic requirements are to add, delete, search, and update all managing staffs' information, it would be too troublesome if a staff has to keep searching for a staff, then perform the operations of updating and deleting. Hence the first additional feature which is displaying a list of existing staffs' information to facilitate the process mentioned above.

In the list, a staff can see the username, password, gender of all existing staff. For each existing staff, the staff can edit the password and gender, and delete the staff account from the system. Note that in this system, a staff cannot delete their own account, hence in the figure above, the *stf1* can only edit his/her account information as the ***Delete*** button is not available.



localhost:8080/AVCS1-war/Edit_staff_info

Return Home Page

Manage the staff's information here:

Add new staff

The username and password cannot exceed 8 characters respectively, it is either M or F for the gender.

Username:
Password:
Gender:

Add

Search for staff(s)

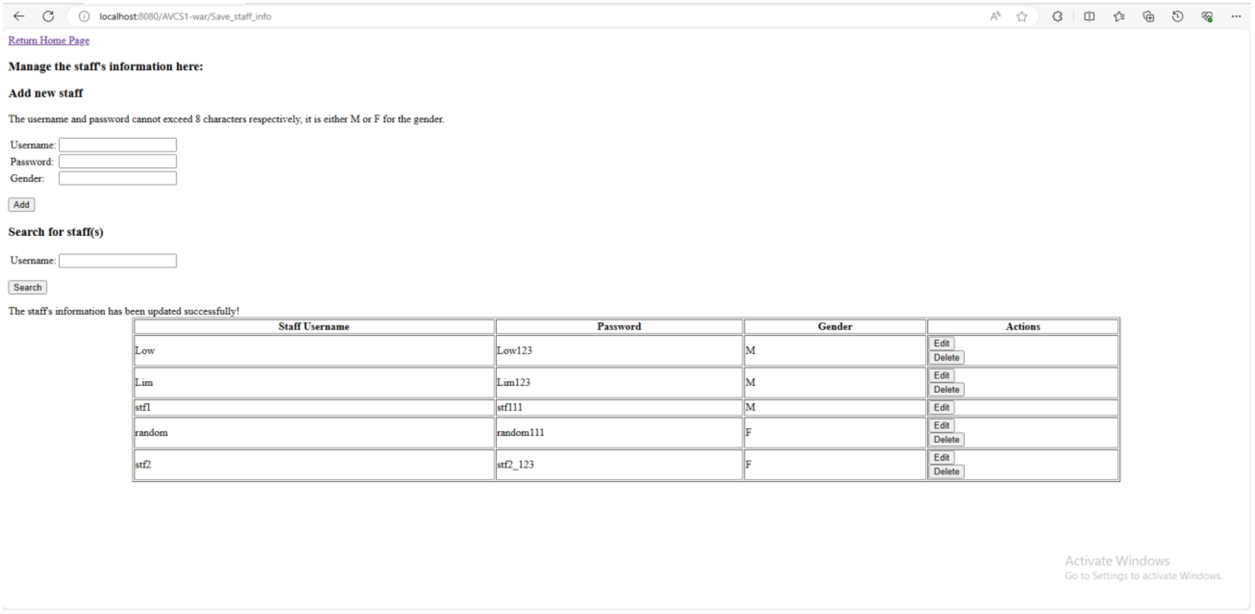
Username:

Search

Username	Password	Gender	Actions
Low	Low123	F	Save
Lim	Lim123	M	Edit Delete
stf1	stf111	M	Edit Delete
random	random111	F	Edit Delete
stf2	stf2_123	F	Edit Delete

Figure 72: After pressing Edit button on staff "Low"

The figure above shows what happens after pressing the ***Edit*** button on the staff "Low", the password and gender becomes modifiable and with original values as their default values. At here, a staff can modify the password and gender, then save the updated information. In this case, say that the staff modified the gender of staff "Low" from "F" to "M".



localhost:8080/AVCS1-war/Save_staff_info

Return Home Page

Manage the staff's information here:

Add new staff

The username and password cannot exceed 8 characters respectively, it is either M or F for the gender.

Username:
Password:
Gender:

Add

Search for staff(s)

Username:

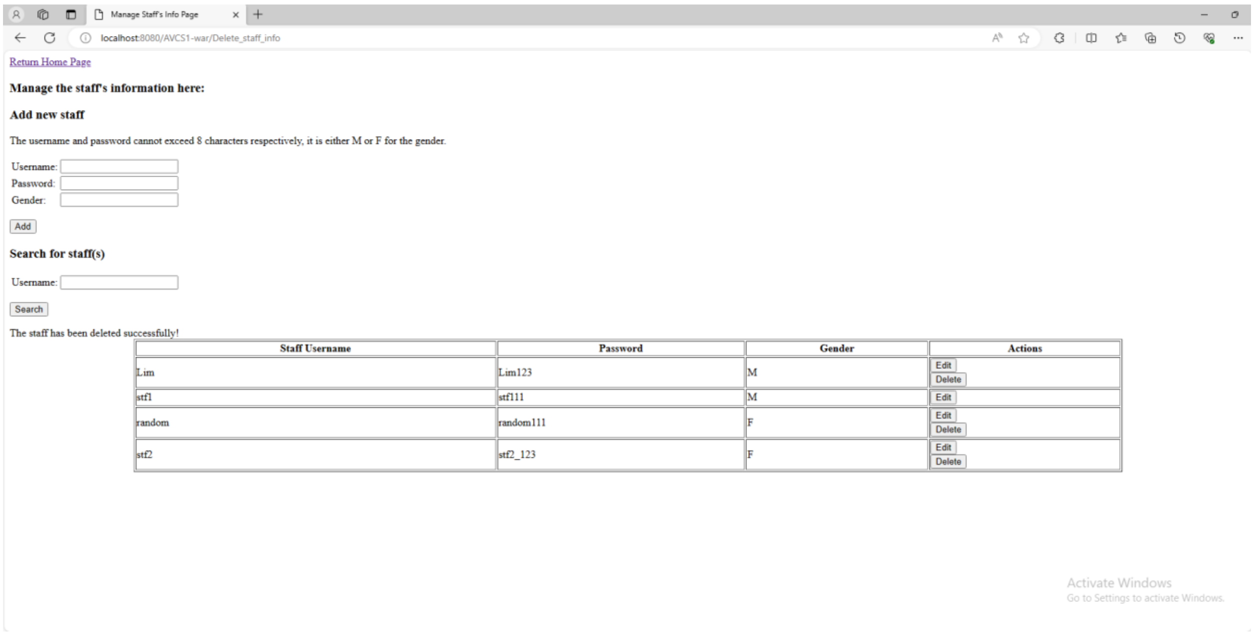
Search

The staff's information has been updated successfully!

Staff Username	Password	Gender	Actions
Low	Low123	M	Edit Delete
Lim	Lim123	M	Edit Delete
stf1	stf111	M	Edit
random	random111	F	Edit Delete
stf2	stf2_123	F	Edit Delete

Figure 73: After pressing Save button on staff "Low" with updated gender

The figure above shows what happens after modifying the gender of staff "Low". From the figure above, it shows the message "The staff's information has been updated successfully", then the gender is also updated.


Manage Staff's Info Page
localhost:8080/AVCS1-war/Delete_staff_info
Activate Windows
Go to Settings to activate Windows.

Return Home Page

Manage the staff's information here:

Add new staff

The username and password cannot exceed 8 characters respectively, it is either M or F for the gender.

Username:
Password:
Gender:

Add

Search for staff(s)

Username:

Search

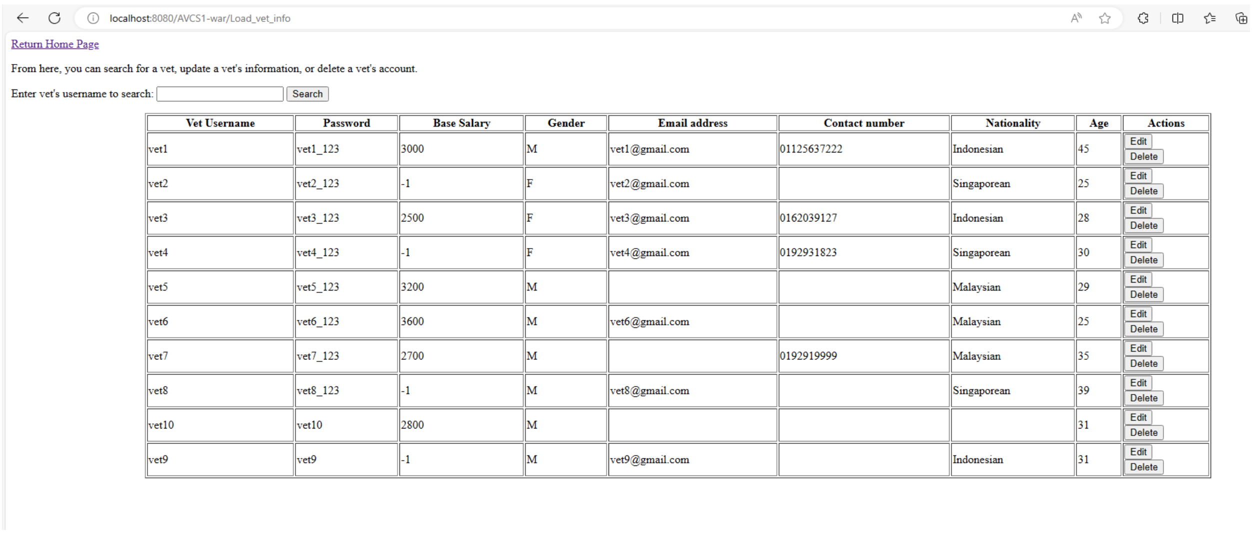
The staff has been deleted successfully!

Staff Username	Password	Gender	Actions
Lim	Lim123	M	Edit Delete
stf1	stf111	M	Edit
random	random111	F	Edit Delete
stf2	stf2_123	F	Edit Delete

Figure 74: After pressing Delete button on the staff "Low"

The figure above shows what happens after pressing the ***Delete*** button on the staff "Low". The system will remove the account from the database, and the front end here shows a message "The staff has been deleted successfully" and a list of updated existing staffs' information.

Display a list of existing vets' information



localhost:8080/AVCS1-war/Load_vet_info

Return Home Page

From here, you can search for a vet, update a vet's information, or delete a vet's account.

Enter vet's username to search: Search

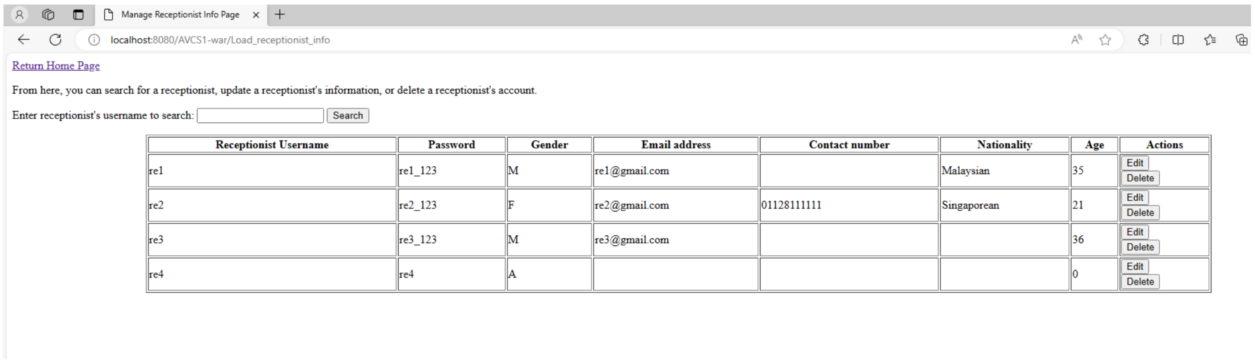
Vet Username	Password	Base Salary	Gender	Email address	Contact number	Nationality	Age	Actions
vet1	vet1_123	3000	M	vet1@gmail.com	01125637222	Indonesian	45	Edit Delete
vet2	vet2_123	-1	F	vet2@gmail.com		Singaporean	25	Edit Delete
vet3	vet3_123	2500	F	vet3@gmail.com	0162039127	Indonesian	28	Edit Delete
vet4	vet4_123	-1	F	vet4@gmail.com	0192931823	Singaporean	30	Edit Delete
vet5	vet5_123	3200	M			Malaysian	29	Edit Delete
vet6	vet6_123	3600	M	vet6@gmail.com		Malaysian	25	Edit Delete
vet7	vet7_123	2700	M		0192919999	Malaysian	35	Edit Delete
vet8	vet8_123	-1	M	vet8@gmail.com		Singaporean	39	Edit Delete
vet10	vet10	2800	M				31	Edit Delete
vet9	vet9	-1	M	vet9@gmail.com		Indonesian	31	Edit Delete

Figure 75: Displaying a list of existing vets

As the basic requirements are to delete, search, and update all vets' information, it would be too troublesome if a staff has to keep searching for a vet, then perform the operations of updating and deleting. Hence the second additional feature is displaying a list of existing vets' information to facilitate the process mentioned above.

In the list, the username, password, base salary, gender, email address, contact number, nationality, and age of the vets are displayed. Upon registration of the vets' accounts, the vets only provide the username and password, hence the other information will be initialized with some suitable values: the base salary is initialized to -1, gender is initialized to "A", age is initialized to 0 etc. All these initialized values can be modified later by the staff through the user interface. For the ***Edit*** button and ***Delete*** button in each record of vets' information, they work exactly the same as the section above.

Display a list of existing receptionist' information


Manage Receptionist Info Page
localhost:8080/AVCS1-war/Load_receptionist_info

Return Home Page

From here, you can search for a receptionist, update a receptionist's information, or delete a receptionist's account.

Enter receptionist's username to search: Search

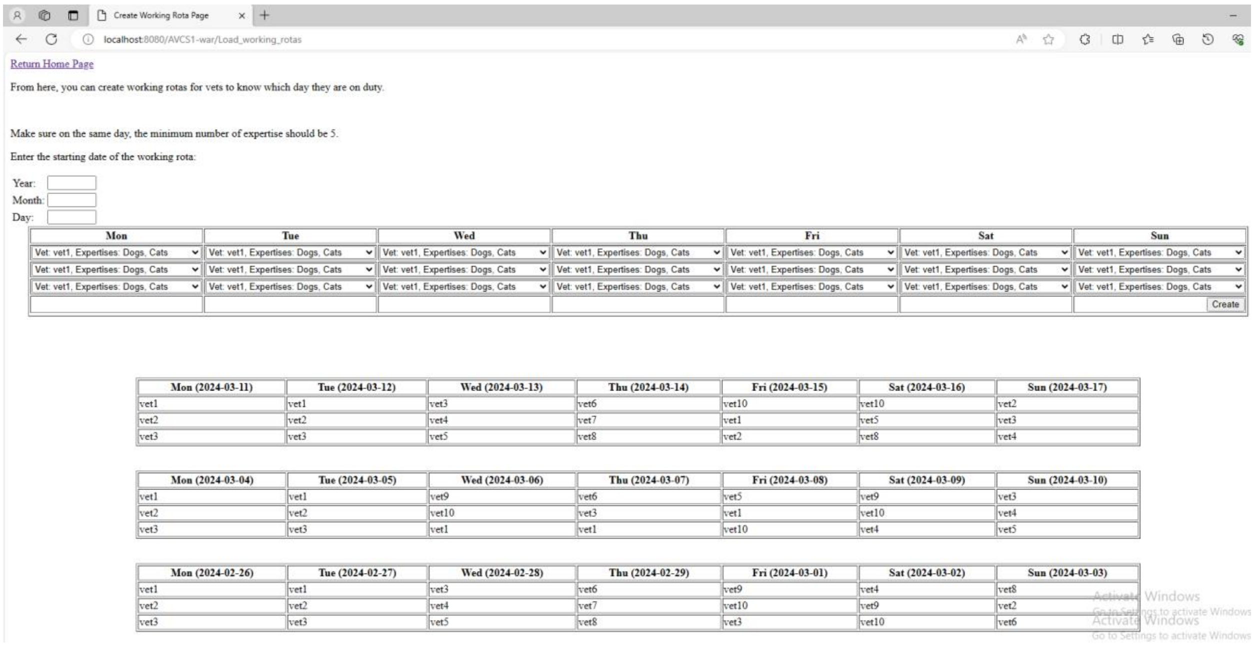
Receptionist Username	Password	Gender	Email address	Contact number	Nationality	Age	Actions
re1	re1_123	M	re1@gmail.com		Malaysian	35	Edit Delete
re2	re2_123	F	re2@gmail.com	01128111111	Singaporean	21	Edit Delete
re3	re3_123	M	re3@gmail.com			36	Edit Delete
re4	re4	A				0	Edit Delete

Figure 76: Displaying a list of existing receptionists

As the basic requirements are to delete, search, and update all receptionists' information, it would be too troublesome if a staff has to keep searching for a receptionist, then perform the operations of updating and deleting. Hence the third additional feature is displaying a list of existing receptionists' information to facilitate the process mentioned above.

In the list, the username, password, gender, email address, contact number, nationality, and age of the receptionists are displayed. Upon registration of the receptionists' accounts, the receptionists only provide the username and password, hence the other information will be initialized with some suitable values: gender is initialized to "A", age is initialized to 0 etc. All these initialized values can be modified later by the staff through the user interface. For the ***Edit*** button and ***Delete*** button in each record of receptionists' information, they work exactly the same as this section.

Display a list of existing working rotas



Return Home Page

From here, you can create working rotas for vets to know which day they are on duty.

Make sure on the same day, the minimum number of expertise should be 5.

Enter the starting date of the working rota:

Year:
Month:
Day:

Mon	Tue	Wed	Thu	Fri	Sat	Sun
Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats
Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats
Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats
						Create

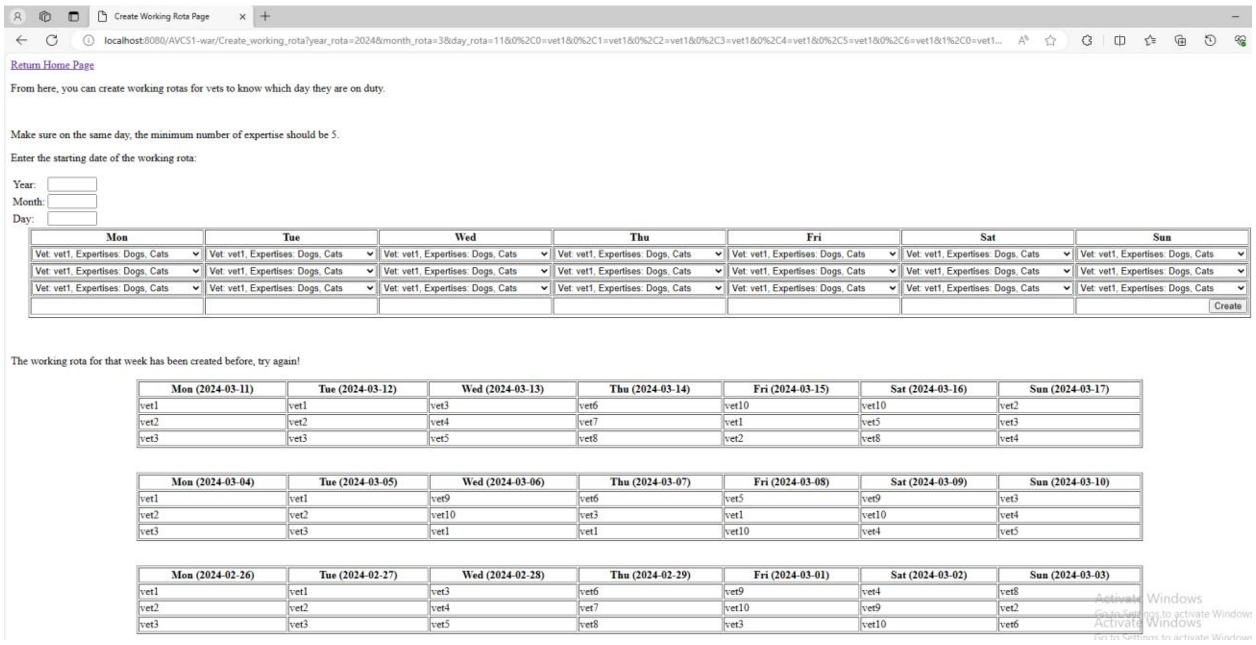
Mon (2024-03-11)	Tue (2024-03-12)	Wed (2024-03-13)	Thu (2024-03-14)	Fri (2024-03-15)	Sat (2024-03-16)	Sun (2024-03-17)
vet1	vet1	vet3	vet6	vet10	vet10	vet2
vet2	vet2	vet4	vet7	vet1	vet5	vet3
vet3	vet3	vet5	vet8	vet2	vet8	vet4

Mon (2024-03-04)	Tue (2024-03-05)	Wed (2024-03-06)	Thu (2024-03-07)	Fri (2024-03-08)	Sat (2024-03-09)	Sun (2024-03-10)
vet1	vet1	vet9	vet6	vet5	vet9	vet3
vet2	vet2	vet10	vet3	vet1	vet10	vet4
vet3	vet3	vet1	vet1	vet10	vet4	vet5

Mon (2024-02-26)	Tue (2024-02-27)	Wed (2024-02-28)	Thu (2024-02-29)	Fri (2024-03-01)	Sat (2024-03-02)	Sun (2024-03-03)
vet1	vet1	vet3	vet6	vet9	vet4	vet8
vet2	vet2	vet4	vet7	vet10	vet9	vet2
vet3	vet3	vet5	vet8	vet3	vet10	vet6

Figure 77: Displaying a list of existing working rotas

As the basic requirement of the system is a staff is allowed to create week's working rota for vets only, hence it would be a bit difficult for staff to do so if there isn't a list of existing working rotas. The list of existing working rotas is displayed to the staffs so that they can know which week's working rota has been scheduled before, otherwise they have to enter the date again and again to see which week's working rota has been scheduled by looking at the warnings. In the figure above, with the provided list of existing working rotas, it can be clearly known that the working rotas for 2024-03-11, 2024-03-04, and 2024-02-26 have been scheduled. Without the list, we have to enter the year, month, and day of the starting day (Monday) of the working rota and look at the returned warning to see if it has been created before as shown in the figure below:



Create Working Rota Page

localhost:8080/AVCS1-war/Create_working_rota?year_rota=2024&month_rota=3&day_rota=11&0%2C0=vet1&0%2C1=vet1&0%2C2=vet1&0%2C3=vet1&0%2C4=vet1&0%2C5=vet1&0%2C6=vet1&1%2C0=vet1...

Return Home Page

From here, you can create working rotas for vets to know which day they are on duty.

Make sure on the same day, the minimum number of expertise should be 5.

Enter the starting date of the working rota:

Year:
Month:
Day:

Mon	Tue	Wed	Thu	Fri	Sat	Sun
Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats
Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats
Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats	Vet: vet1, Expertises: Dogs, Cats
						Create

The working rota for that week has been created before, try again!

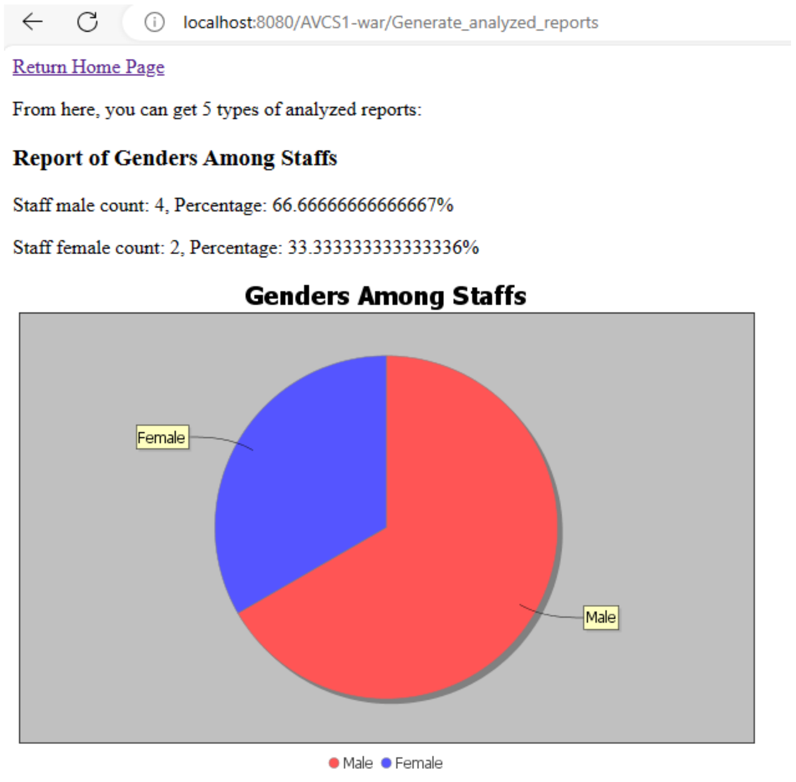
Mon (2024-03-11)	Tue (2024-03-12)	Wed (2024-03-13)	Thu (2024-03-14)	Fri (2024-03-15)	Sat (2024-03-16)	Sun (2024-03-17)
vet1	vet1	vet3	vet6	vet10	vet10	vet2
vet2	vet2	vet4	vet7	vet1	vet5	vet3
vet3	vet3	vet5	vet8	vet2	vet8	vet4

Mon (2024-03-04)	Tue (2024-03-05)	Wed (2024-03-06)	Thu (2024-03-07)	Fri (2024-03-08)	Sat (2024-03-09)	Sun (2024-03-10)
vet1	vet1	vet9	vet6	vet5	vet9	vet3
vet2	vet2	vet10	vet3	vet1	vet10	vet4
vet3	vet3	vet1	vet1	vet10	vet4	vet5

Mon (2024-02-26)	Tue (2024-02-27)	Wed (2024-02-28)	Thu (2024-02-29)	Fri (2024-03-01)	Sat (2024-03-02)	Sun (2024-03-03)
vet1	vet1	vet3	vet6	vet9	vet4	vet8
vet2	vet2	vet4	vet7	vet10	vet9	vet2
vet3	vet3	vet5	vet8	vet3	vet10	vet6

Figure 78: Warning that the weeking rota has been created before

Generate visualization charts


localhost:8080/AVCS1-war/Generate_analyzed_reports

Return Home Page

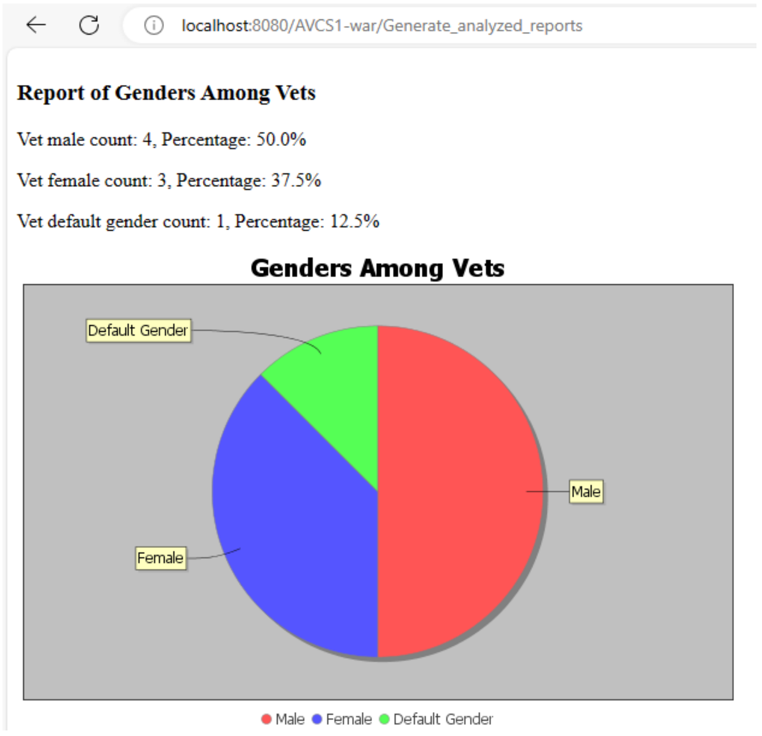
From here, you can get 5 types of analyzed reports:

Report of Genders Among Staffs

Staff male count: 4, Percentage: 66.66666666666667%

Staff female count: 2, Percentage: 33.333333333333336%

Figure 79: Report of Genders Among Staffs


localhost:8080/AVCS1-war/Generate_analyzed_reports

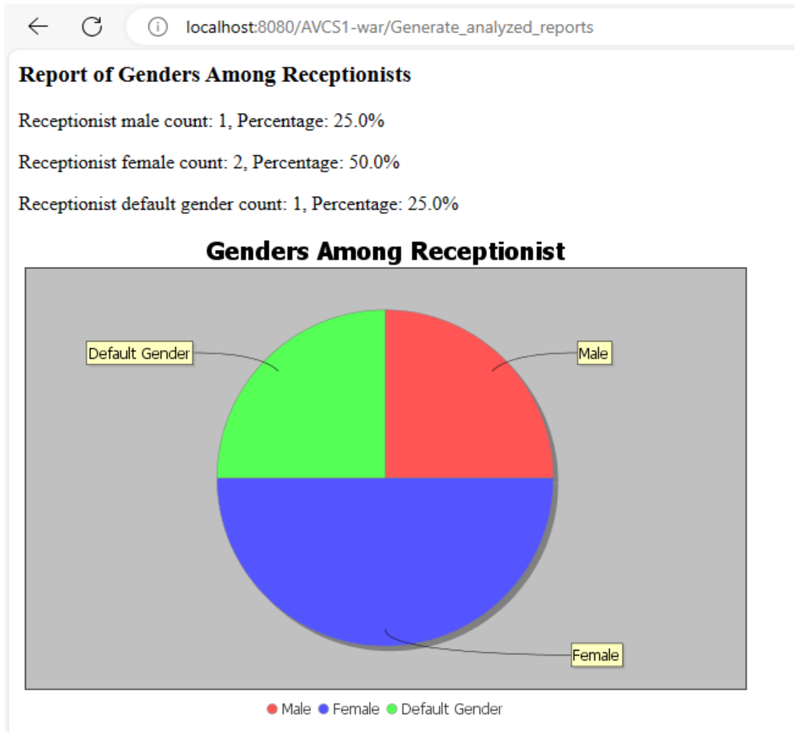
Report of Genders Among Vets

Vet male count: 4, Percentage: 50.0%

Vet female count: 3, Percentage: 37.5%

Vet default gender count: 1, Percentage: 12.5%

Figure 80: Report of Genders Among Vets


localhost:8080/AVCS1-war/Generate_analyzed_reports

Report of Genders Among Receptionists

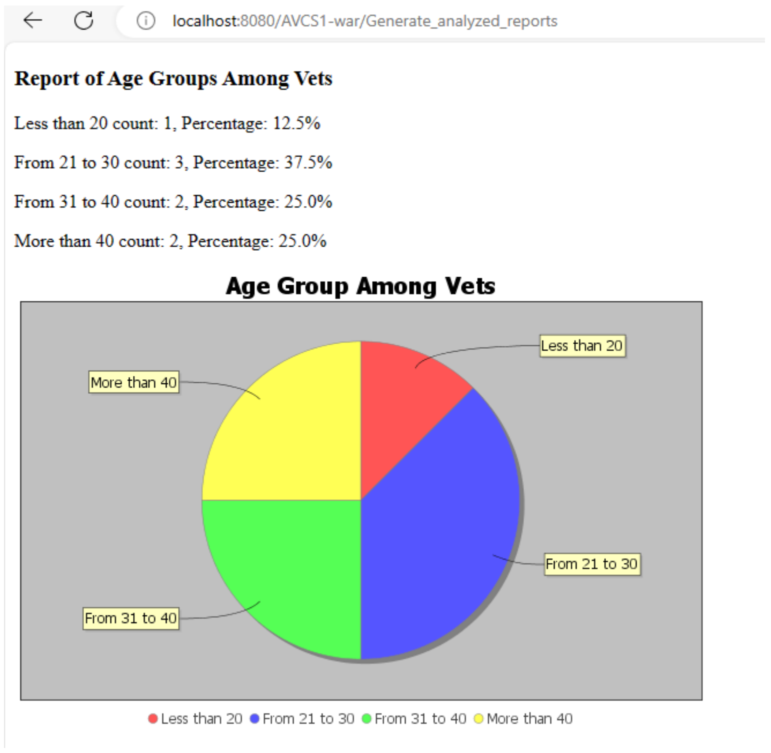
Receptionist male count: 1, Percentage: 25.0%

Receptionist female count: 2, Percentage: 50.0%

Receptionist default gender count: 1, Percentage: 25.0%

Genders Among Receptionist
Default Gender
Male
Female
● Male ● Female ● Default Gender

Figure 81:Report of Genders Among Receptionists


localhost:8080/AVCS1-war/Generate_analyzed_reports

Report of Age Groups Among Vets

Less than 20 count: 1, Percentage: 12.5%

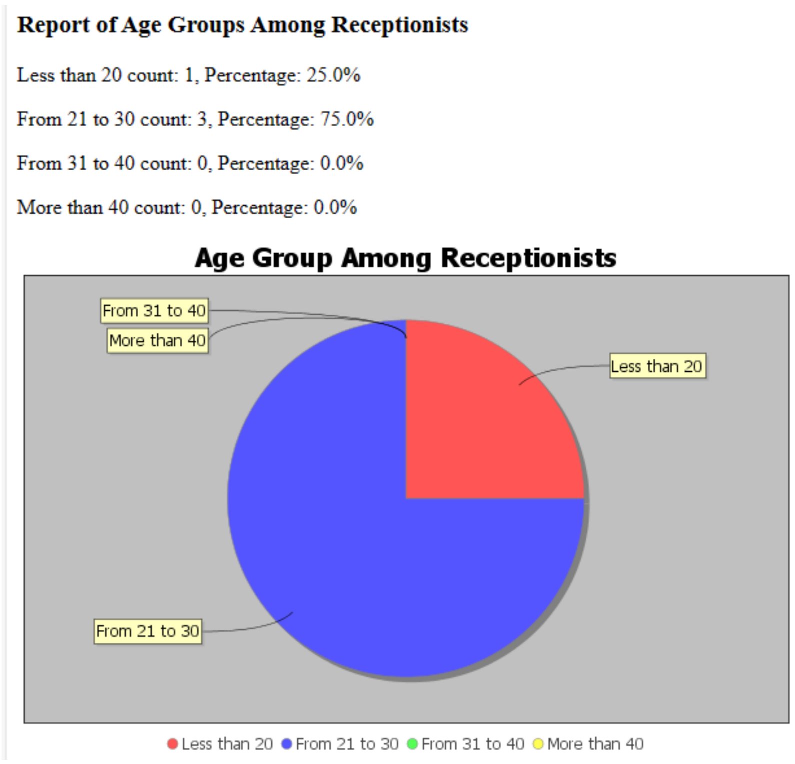
From 21 to 30 count: 3, Percentage: 37.5%

From 31 to 40 count: 2, Percentage: 25.0%

More than 40 count: 2, Percentage: 25.0%

Age Group Among Vets
Less than 20
More than 40
From 21 to 30
From 31 to 40
● Less than 20 ● From 21 to 30 ● From 31 to 40 ● More than 40

Figure 82: Report of Age Groups Among Vets



Report of Age Groups Among Receptionists

Less than 20 count: 1, Percentage: 25.0%

From 21 to 30 count: 3, Percentage: 75.0%

From 31 to 40 count: 0, Percentage: 0.0%

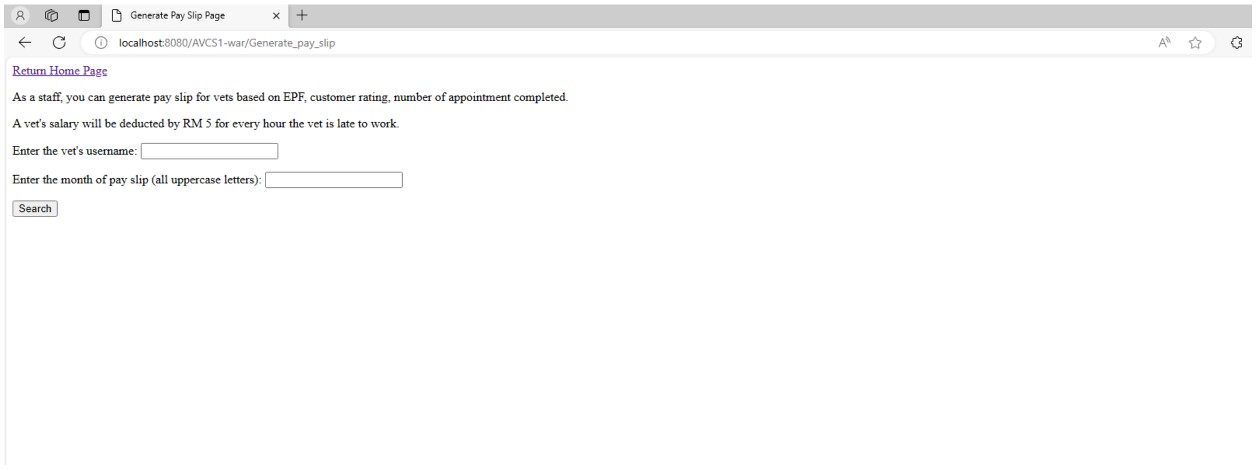
More than 40 count: 0, Percentage: 0.0%

Age Group Among Receptionists
From 31 to 40
More than 40
Less than 20
From 21 to 30
● Less than 20 ● From 21 to 30 ● From 31 to 40 ● More than 40

Figure 83: Report of Age Groups Among Receptionists

For the analyzed reports generated, each report comes with statistical information, indicating the exact count and percentage of each group. For this kind of statistical information, visualization of the data is done for providing more insight into the data because it is visual-appealing.

Generate pay slip for the vets



As a staff, you can generate pay slip for vets based on EPF, customer rating, number of appointment completed.

A vet's salary will be deducted by RM 5 for every hour the vet is late to work.

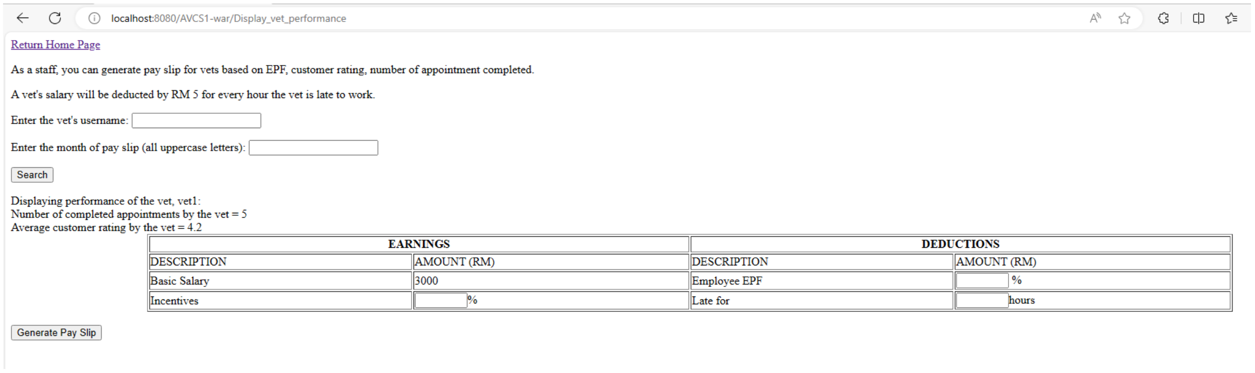
Enter the vet's username:

Enter the month of pay slip (all uppercase letters):

Search

Figure 84: Prompting vet's username and month of the pay slip

Every company needs a reliable tool for generating pay slips for their employees, so that they can keep them as a reference in future. Another additional feature of this system would be generating pay slips for the vets. First, the staff needs to enter the vet's username and the month of the pay slip. Say that the staff is interested in generating February's pay slip for the vet with username "vet1".



As a staff, you can generate pay slip for vets based on EPF, customer rating, number of appointment completed.

A vet's salary will be deducted by RM 5 for every hour the vet is late to work.

Enter the vet's username:

Enter the month of pay slip (all uppercase letters):

Search

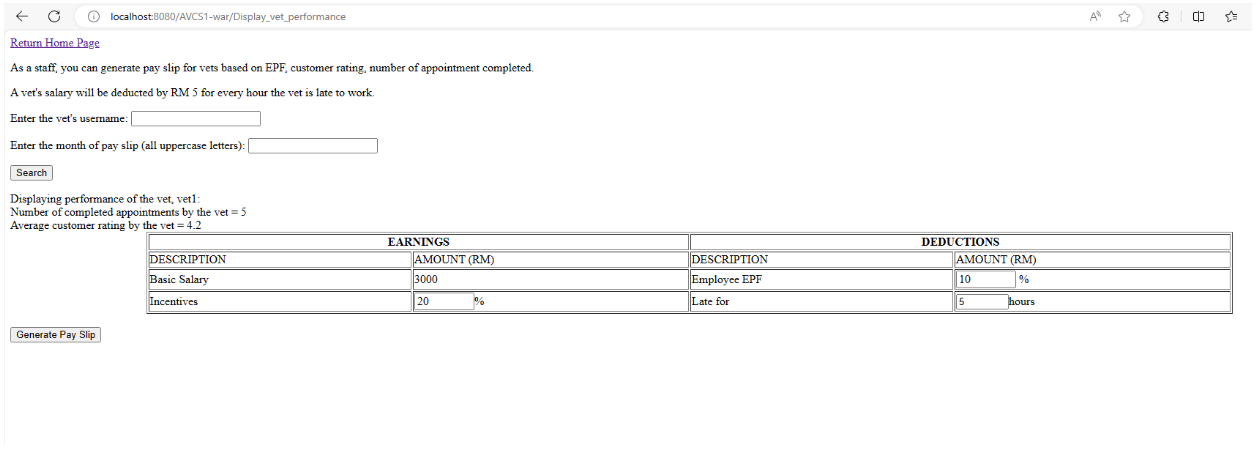
Displaying performance of the vet, vet1:
Number of completed appointments by the vet = 5
Average customer rating by the vet = 4.2

EARNINGS		DEDUCTIONS	
DESCRIPTION	AMOUNT (RM)	DESCRIPTION	AMOUNT (RM)
Basic Salary	3000	Employee EPF	%
Incentives	%	Late for	hours

Generate Pay Slip

Figure 85: After submitting "vet1" as the vet's username and FEBRUARY as the month of the pay slip

After submitting "vet1" as the vet's username and FEBRUARY as the month of the pay slip, the figure above shows what happens afterwards. It displays the performance of the vet, in terms of the number of appointments he/she has completed and the average customer rating. The staff can determine the incentives rate based on the vet's performance. The staff can also enter the employee EPF rate for the vet. Lastly, the staff can enter how many hours the vet is late to work for deduction of salary. The figure below shows all the rates and number of hours entered:



localhost:8080/AVCS1-war/Display_vet_performance

Return Home Page

As a staff, you can generate pay slip for vets based on EPF, customer rating, number of appointment completed.

A vet's salary will be deducted by RM 5 for every hour the vet is late to work.

Enter the vet's username:

Enter the month of pay slip (all uppercase letters):

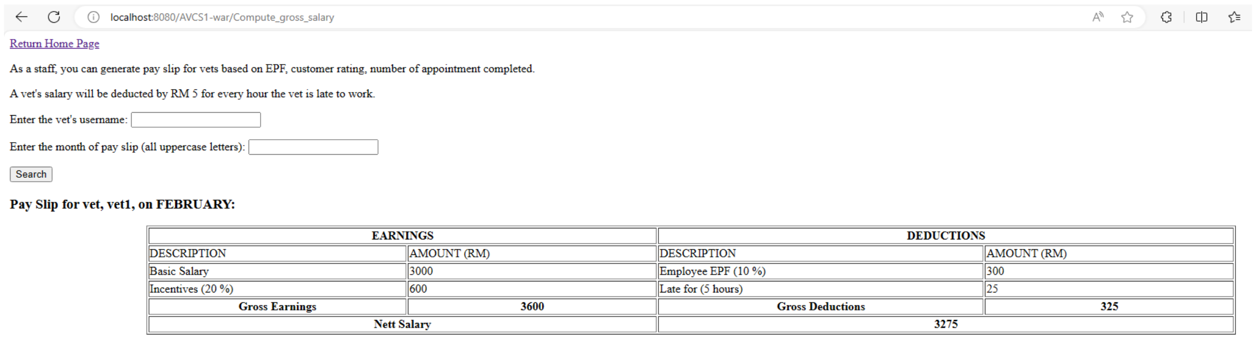
Search

Displaying performance of the vet, vet1:
Number of completed appointments by the vet = 5
Average customer rating by the vet = 4.2

EARNINGS		DEDUCTIONS	
DESCRIPTION	AMOUNT (RM)	DESCRIPTION	AMOUNT (RM)
Basic Salary	3000	Employee EPF	10 %
Incentives	20 %	Late for	5 hours

Generate Pay Slip

Figure 86: Display the performance of the vet "vet1"



localhost:8080/AVCS1-war/Compute_gross_salary

Return Home Page

As a staff, you can generate pay slip for vets based on EPF, customer rating, number of appointment completed.

A vet's salary will be deducted by RM 5 for every hour the vet is late to work.

Enter the vet's username:

Enter the month of pay slip (all uppercase letters):

Search

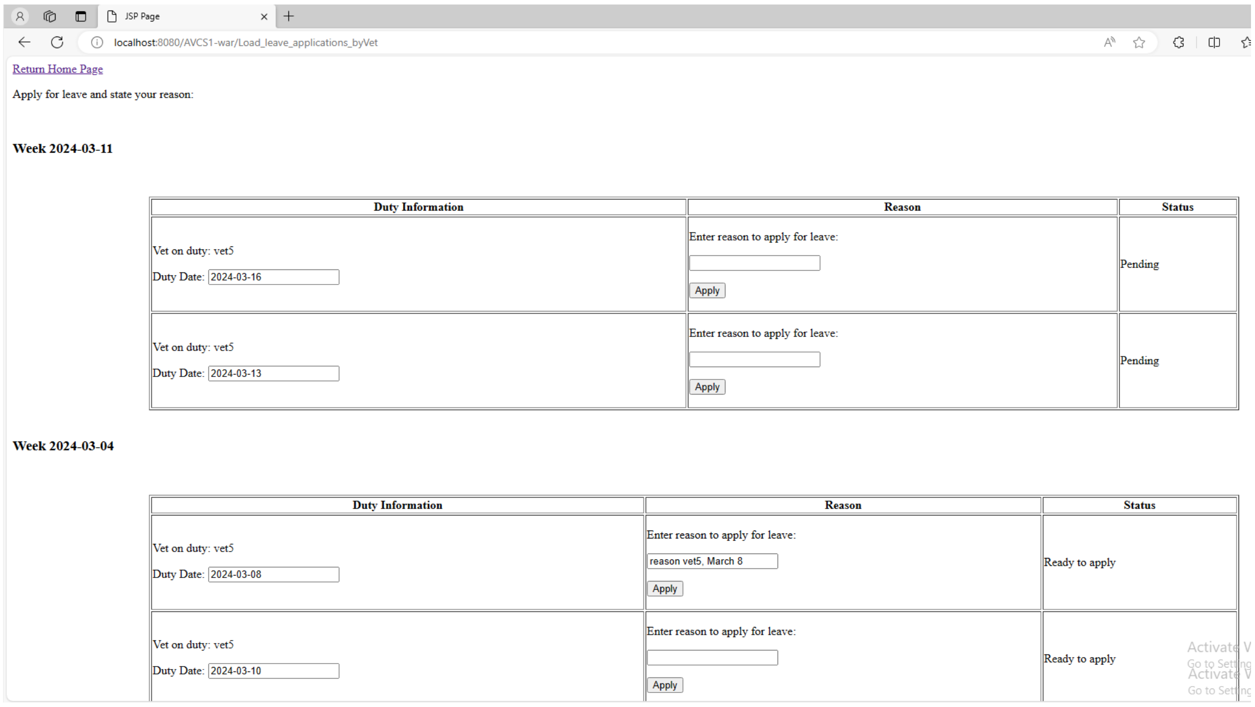
Pay Slip for vet, vet1, on FEBRUARY:

EARNINGS		DEDUCTIONS	
DESCRIPTION	AMOUNT (RM)	DESCRIPTION	AMOUNT (RM)
Basic Salary	3000	Employee EPF (10 %)	300
Incentives (20 %)	600	Late for (5 hours)	25
Gross Earnings	**3600**	**Gross Deductions**	**325**
Nett Salary		**3275**	

Figure 87: Final Pay Slip for the vet "vet1"

Then, the final pay slip is generated. All the gross earnings and gross deductions are calculated, to come up with the net salary.

Apply for leave on the working days



localhost:8080/AVCS1-war/Load_leave_applications_byVet

Return Home Page

Apply for leave and state your reason:

Week 2024-03-11

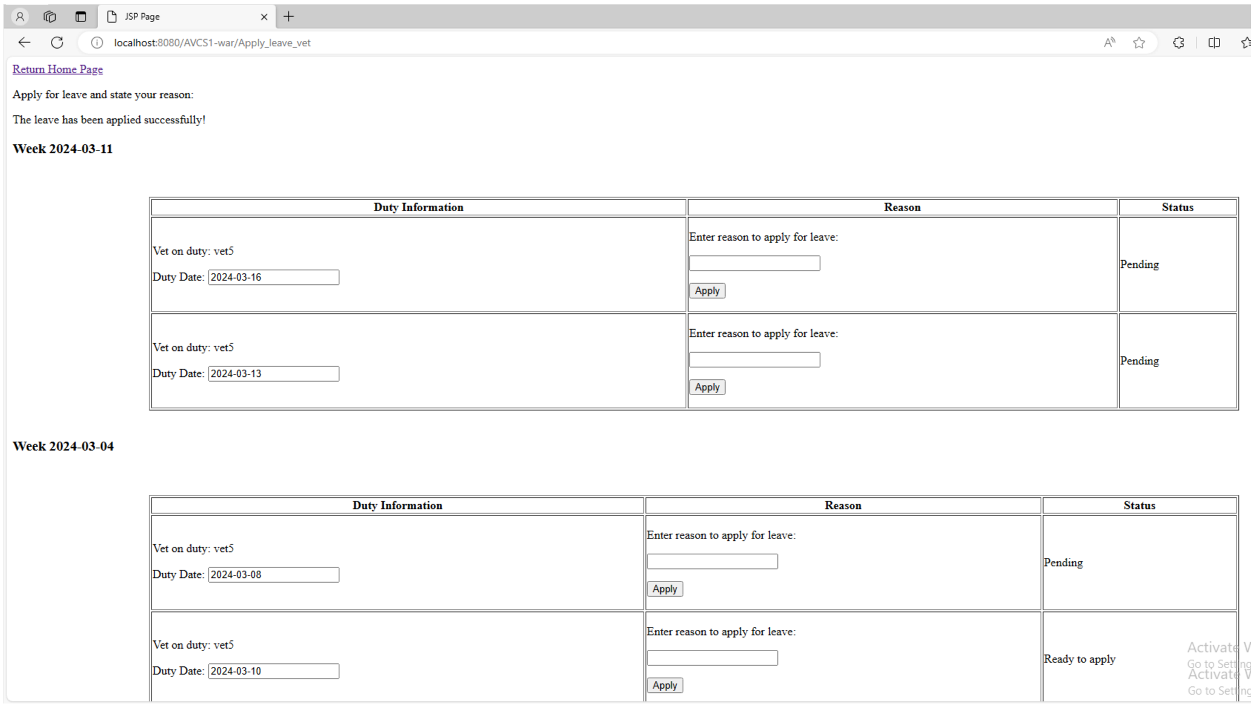
Duty Information	Reason	Status
Vet on duty: vet5 Duty Date: 2024-03-16	Enter reason to apply for leave: Apply	Pending
Vet on duty: vet5 Duty Date: 2024-03-13	Enter reason to apply for leave: Apply	Pending

Week 2024-03-04

Duty Information	Reason	Status
Vet on duty: vet5 Duty Date: 2024-03-08	Enter reason to apply for leave: reason vet5, March 8 Apply	Ready to apply
Vet on duty: vet5 Duty Date: 2024-03-10	Enter reason to apply for leave: Apply	Ready to apply

Figure 88: Apply for leave and check the status of leave applications

The figure above shows the working days for the vet “vet5”, for each working day, the duty date and the status are displayed. The status of the working days could be “Ready to apply”, “Pending”, or “Rejected”: “Ready to apply” means the vet can apply for leave on that day by entering the reason, “Rejected” means that the leave application has been rejected but the vet is still allowed to submit the reason again and again until it is approved, and “Approved” simply means that the leave application has been approved. The figure above shows pending leave applications by vet5 on 2024-03-16 and 2024-03-13, and the vet “vet5” is going to submit a reason for applying for leave on 2024-03-08.



Return Home Page

Apply for leave and state your reason:

The leave has been applied successfully!

Week 2024-03-11

Duty Information	Reason	Status
Vet on duty: vet5 Duty Date: 2024-03-16	Enter reason to apply for leave: Apply	Pending
Vet on duty: vet5 Duty Date: 2024-03-13	Enter reason to apply for leave: Apply	Pending

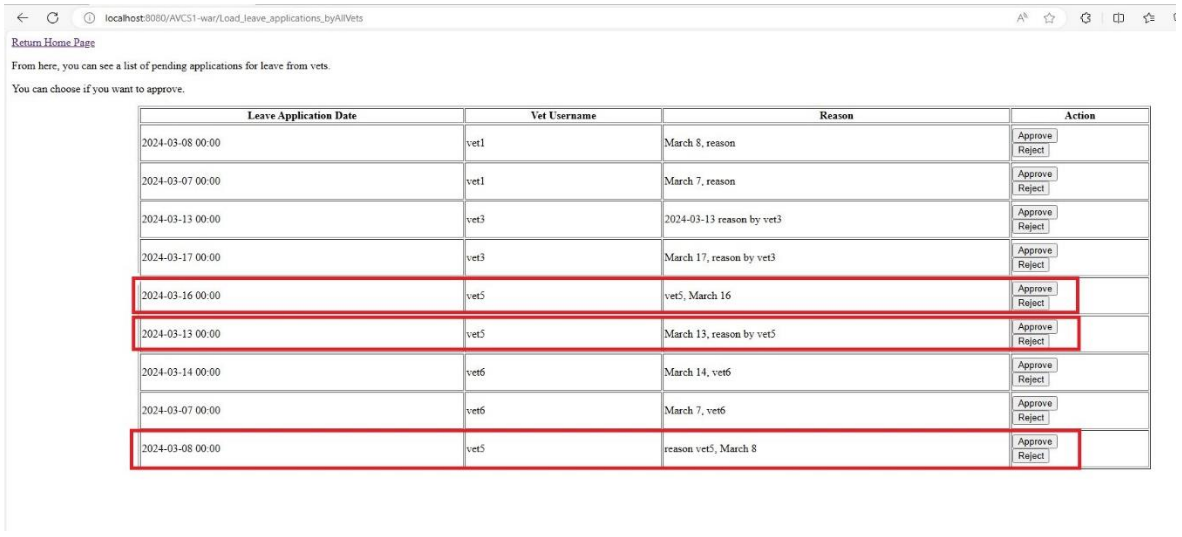
Week 2024-03-04

Duty Information	Reason	Status
Vet on duty: vet5 Duty Date: 2024-03-08	Enter reason to apply for leave: Apply	Pending
Vet on duty: vet5 Duty Date: 2024-03-10	Enter reason to apply for leave: Apply	Ready to apply

Figure 89: Updated status of leave application on 2024-03-08

After the vet “vet5” submitted the reason for 2024-03-08, the status changed from “Ready to applied” to “Pending”, and there is a message “The leave has been applied successfully!”.

Accept or reject the leave applications submitted by vets


localhost:8080/AVCS1-war/Load_leave_applications_byAllVets

Return Home Page

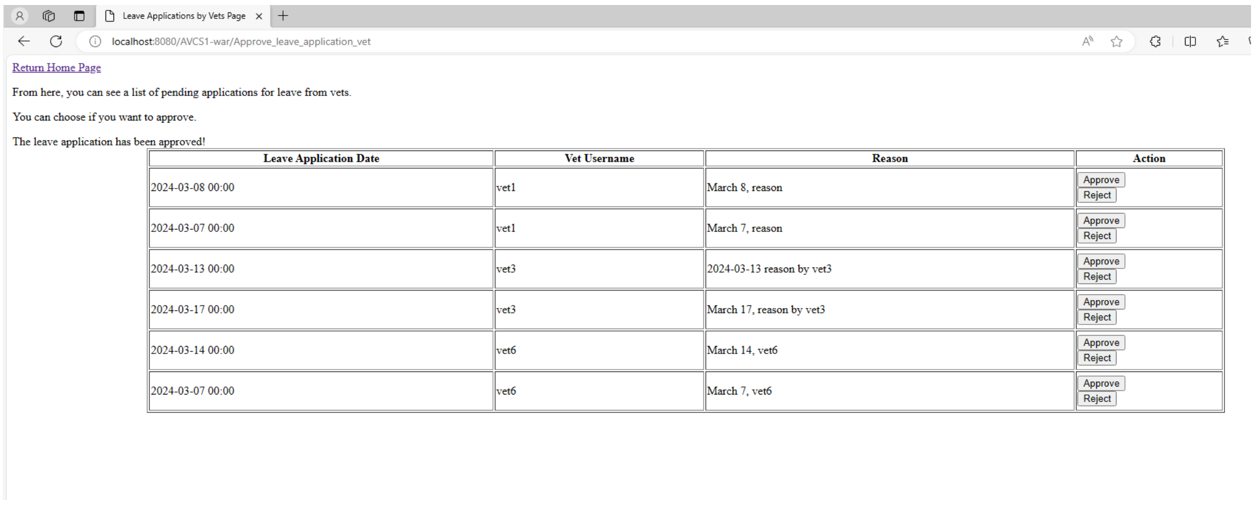
From here, you can see a list of pending applications for leave from vets.

You can choose if you want to approve.

Leave Application Date	Vet Username	Reason	Action
2024-03-08 00:00	vet1	March 8, reason	Approve Reject
2024-03-07 00:00	vet1	March 7, reason	Approve Reject
2024-03-13 00:00	vet3	2024-03-13 reason by vet3	Approve Reject
2024-03-17 00:00	vet3	March 17, reason by vet3	Approve Reject
2024-03-16 00:00	vet5	vet5, March 16	Approve Reject
2024-03-13 00:00	vet5	March 13, reason by vet5	Approve Reject
2024-03-14 00:00	vet6	March 14, vet6	Approve Reject
2024-03-07 00:00	vet6	March 7, vet6	Approve Reject
2024-03-08 00:00	vet5	reason vet5, March 8	Approve Reject

Figure 90: Approve and reject the leave applications from the vet "vet5"

A staff can approve or reject the leave applications from vets. From the figure above, it shows a list of pending leave applications from all the vets. The leave applications bounded by red rectangles are from the vet "vet5", say that the staff plans to reject the leave applications on 2024-03-16 and 2024-03-13, then approve the leave application on 2024-03-08.


Leave Applications by Vets Page

localhost:8080/AVCS1-war/Approve_leave_application_vet

Return Home Page

From here, you can see a list of pending applications for leave from vets.

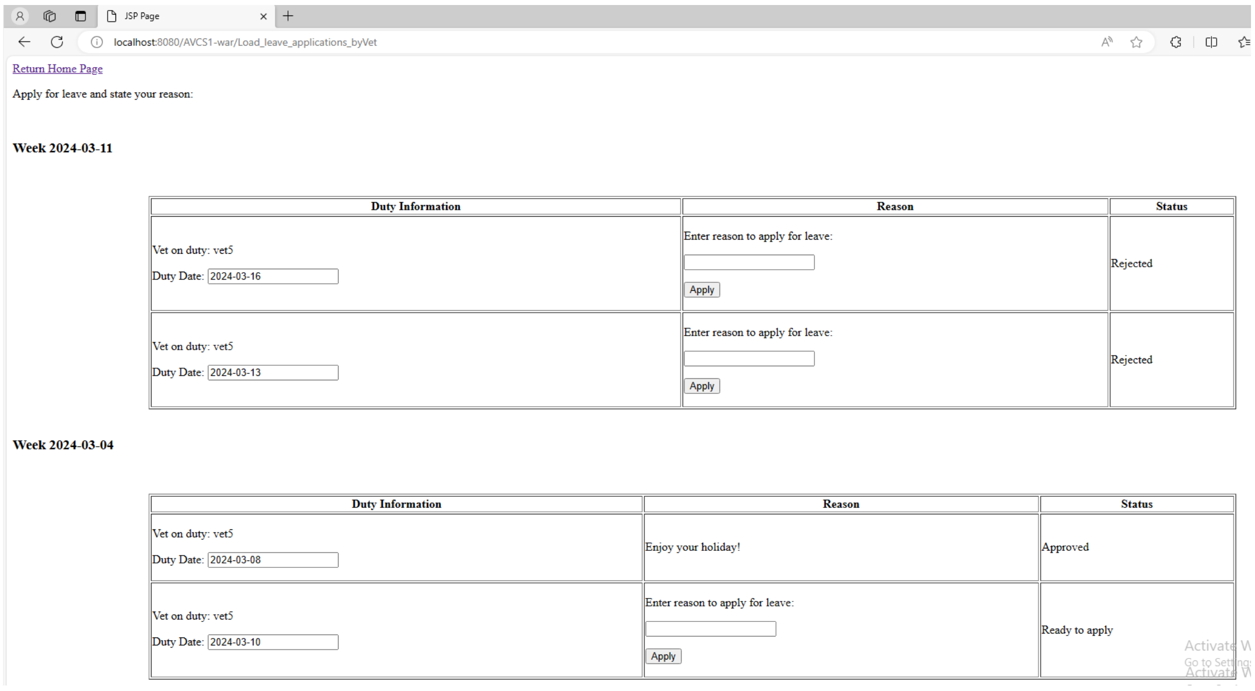
You can choose if you want to approve.

The leave application has been approved!

Leave Application Date	Vet Username	Reason	Action
2024-03-08 00:00	vet1	March 8, reason	Approve Reject
2024-03-07 00:00	vet1	March 7, reason	Approve Reject
2024-03-13 00:00	vet3	2024-03-13 reason by vet3	Approve Reject
2024-03-17 00:00	vet3	March 17, reason by vet3	Approve Reject
2024-03-14 00:00	vet6	March 14, vet6	Approve Reject
2024-03-07 00:00	vet6	March 7, vet6	Approve Reject

Figure 91: After rejecting the 2 leave applications and approving 1 leave application from the vet "vet5"

The figure above shows the updated list of pending leave applications and the message "the leave application" has been approved, as the last action done by the staff is to approve the leave application on 2024-03-08.



localhost:8080/AVCS1-war/Load_leave_applications_byVet

Return Home Page

Apply for leave and state your reason:

Week 2024-03-11

Duty Information	Reason	Status
Vet on duty: vet5 Duty Date: 2024-03-16	Enter reason to apply for leave: Apply	Rejected
Vet on duty: vet5 Duty Date: 2024-03-13	Enter reason to apply for leave: Apply	Rejected

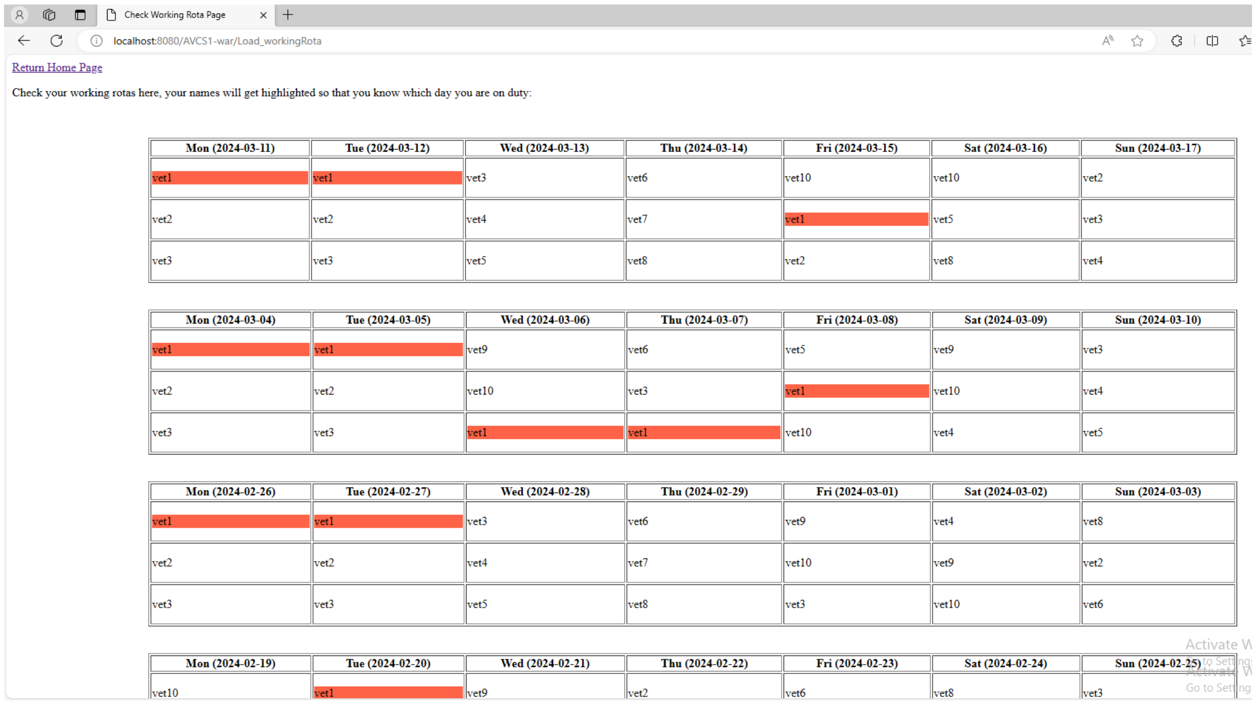
Week 2024-03-04

Duty Information	Reason	Status
Vet on duty: vet5 Duty Date: 2024-03-08	Enjoy your holiday!	Approved
Vet on duty: vet5 Duty Date: 2024-03-10	Enter reason to apply for leave: Apply	Ready to apply

Figure 92: The vet "vet5" checking the status of the leave applications

Then, the vet “vet5” may login again to check the status of the leave applications. As expected, the first two leave applications show the status “Rejected” and allow vets to re-submit the reason. Then, the leave application on 2024-03-08 shows the status “Approved” and the message “Enjoy your holiday!” as there is no need to re-submit the reason.

Check working rotas


Check Working Rota Page
localhost:8080/AVCS1-war/Load_workingRota

Return Home Page

Check your working rotas here, your names will get highlighted so that you know which day you are on duty:

Mon (2024-03-11)	Tue (2024-03-12)	Wed (2024-03-13)	Thu (2024-03-14)	Fri (2024-03-15)	Sat (2024-03-16)	Sun (2024-03-17)
vet1	vet1	vet3	vet6	vet10	vet10	vet2
vet2	vet2	vet4	vet7	vet1	vet5	vet3
vet3	vet3	vet5	vet8	vet2	vet8	vet4

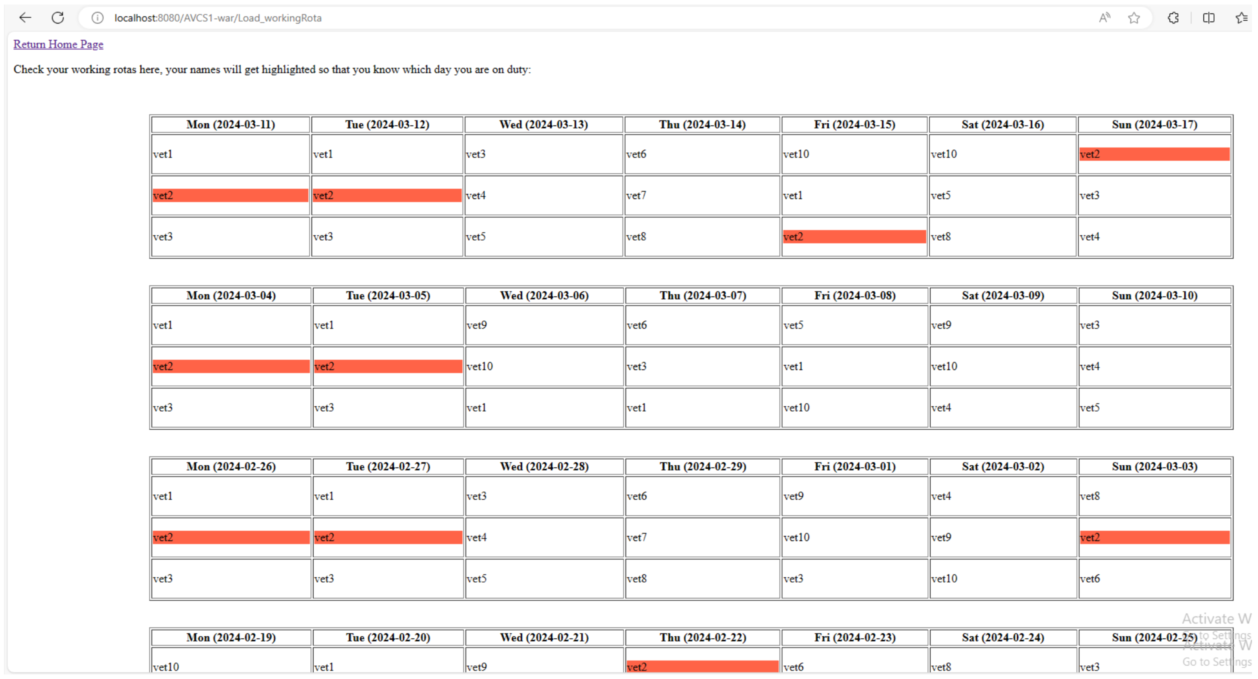
Mon (2024-03-04)	Tue (2024-03-05)	Wed (2024-03-06)	Thu (2024-03-07)	Fri (2024-03-08)	Sat (2024-03-09)	Sun (2024-03-10)
vet1	vet1	vet9	vet6	vet5	vet9	vet3
vet2	vet2	vet10	vet3	vet1	vet10	vet4
vet3	vet3	vet1	vet1	vet10	vet4	vet5

Mon (2024-02-26)	Tue (2024-02-27)	Wed (2024-02-28)	Thu (2024-02-29)	Fri (2024-03-01)	Sat (2024-03-02)	Sun (2024-03-03)
vet1	vet1	vet3	vet6	vet9	vet4	vet8
vet2	vet2	vet4	vet7	vet10	vet9	vet2
vet3	vet3	vet5	vet8	vet3	vet10	vet6

Mon (2024-02-19)	Tue (2024-02-20)	Wed (2024-02-21)	Thu (2024-02-22)	Fri (2024-02-23)	Sat (2024-02-24)	Sun (2024-02-25)
vet10	vet1	vet9	vet2	vet6	vet8	vet3

Figure 93: The vet "vet1" is checking which days he/she is on duty

A vet can use the system to view a list of working rotas, to see which day the vet is on duty. On the working rotas, the names of the vet are highlighted in red to tell which day the vet is on duty. From the figure above, the vet "vet1" is using the system to check which days he/she is on duty, and all the usernames "vet1" have been highlighted in red so that the vet needs not to find his/her usernames one by one. The figure below shows the perspective of the vet "vet2" to show that this additional feature is dynamic instead of hard-coded:



localhost:8080/AVCS1-war/Load_workingRota

Return Home Page

Check your working rotas here, your names will get highlighted so that you know which day you are on duty:

Mon (2024-03-11)	Tue (2024-03-12)	Wed (2024-03-13)	Thu (2024-03-14)	Fri (2024-03-15)	Sat (2024-03-16)	Sun (2024-03-17)
vet1	vet1	vet3	vet6	vet10	vet10	vet2
vet2	vet2	vet4	vet7	vet1	vet5	vet3
vet3	vet3	vet5	vet8	vet2	vet8	vet4

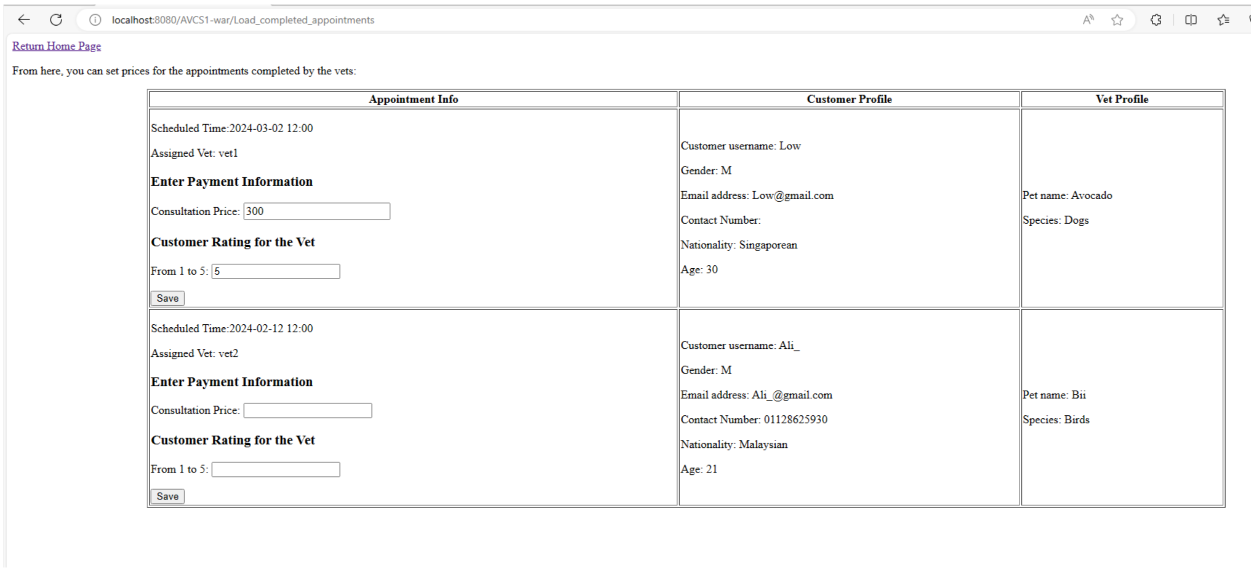
Mon (2024-03-04)	Tue (2024-03-05)	Wed (2024-03-06)	Thu (2024-03-07)	Fri (2024-03-08)	Sat (2024-03-09)	Sun (2024-03-10)
vet1	vet1	vet9	vet6	vet5	vet9	vet3
vet2	vet2	vet10	vet3	vet1	vet10	vet4
vet3	vet3	vet1	vet1	vet10	vet4	vet5

Mon (2024-02-26)	Tue (2024-02-27)	Wed (2024-02-28)	Thu (2024-02-29)	Fri (2024-03-01)	Sat (2024-03-02)	Sun (2024-03-03)
vet1	vet1	vet3	vet6	vet9	vet4	vet8
vet2	vet2	vet4	vet7	vet10	vet9	vet2
vet3	vet3	vet5	vet8	vet3	vet10	vet6

Mon (2024-02-19)	Tue (2024-02-20)	Wed (2024-02-21)	Thu (2024-02-22)	Fri (2024-02-23)	Sat (2024-02-24)	Sun (2024-02-25)
vet10	vet1	vet9	vet2	vet6	vet8	vet3

Figure 94: The vet "vet2" is checking which days he/she is on duty

Set price and leave customer rating for each completed appointment


localhost:8080/AVCS1-war/Load_completed_appointments

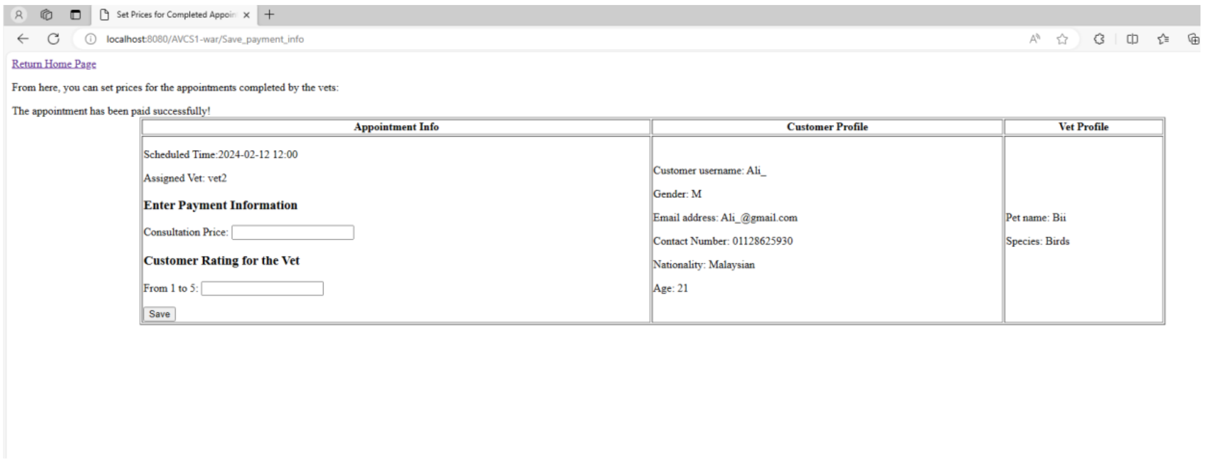
Return Home Page

From here, you can set prices for the appointments completed by the vets:

Appointment Info	Customer Profile	Vet Profile
Scheduled Time:2024-03-02 12:00 Assigned Vet: vet1 **Enter Payment Information** Consultation Price: 300 **Customer Rating for the Vet** From 1 to 5: 5 Save	Customer username: Low Gender: M Email address: Low@gmail.com Contact Number: Nationality: Singaporean Age: 30	Pet name: Avocado Species: Dogs
Scheduled Time:2024-02-12 12:00 Assigned Vet: vet2 **Enter Payment Information** Consultation Price: **Customer Rating for the Vet** From 1 to 5: Save	Customer username: Ali_ Gender: M Email address: Ali_@gmail.com Contact Number: 01128625930 Nationality: Malaysian Age: 21	Pet name: Bii Species: Birds

Figure 95: A receptionist handles completed appointments by vets

After the vet has entered the diagnosis and prognosis for an appointment, the appointment is considered as completed. Then, it will appear in the receptionists' user interface. The receptionists can set the price of the appointment and ask the customers to make payment. Other than that, the receptionist can ask the customer to rate the service provided by the vet on a scale of 1 to 5. The average customer rating of the vet is for staffs to determine the incentives rate for the vet while generating the pay slip. The figure above shows a list of completed appointments by all vets, then the receptionist sets the price for the appointment and asks the customer to leave a customer rating. The figure below shows the updated list of completed appointments:


Set Prices for Completed Appoin

localhost:8080/AVCS1-war/Save_payment_info

Return Home Page

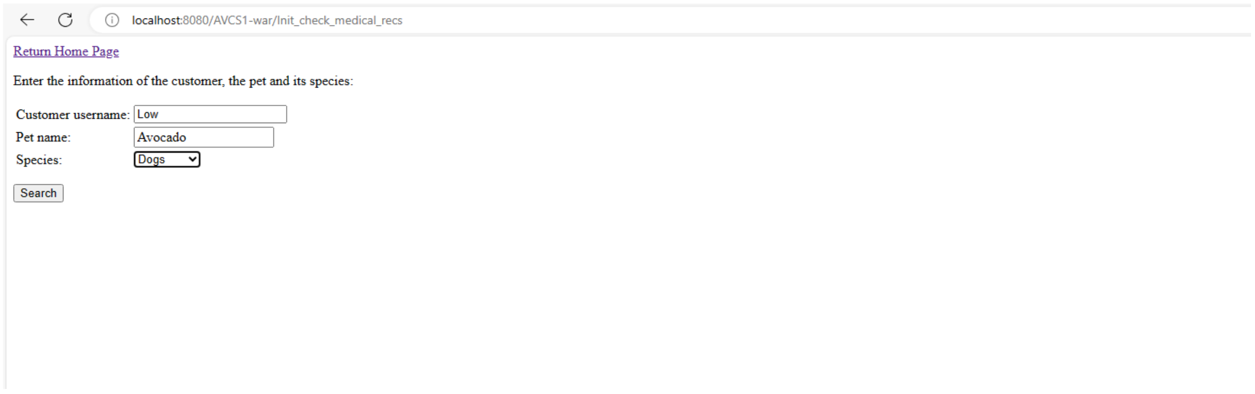
From here, you can set prices for the appointments completed by the vets:

The appointment has been paid successfully!

Appointment Info	Customer Profile	Vet Profile
Scheduled Time:2024-02-12 12:00 Assigned Vet: vet2 **Enter Payment Information** Consultation Price: **Customer Rating for the Vet** From 1 to 5: Save	Customer username: Ali_ Gender: M Email address: Ali_@gmail.com Contact Number: 01128625930 Nationality: Malaysian Age: 21	Pet name: Bii Species: Birds

Figure 96: Updated list of completed appointments by all vets

Generate a list of medical histories for a pet



localhost:8080/AVCS1-war/Init_check_medical_recs

Return Home Page

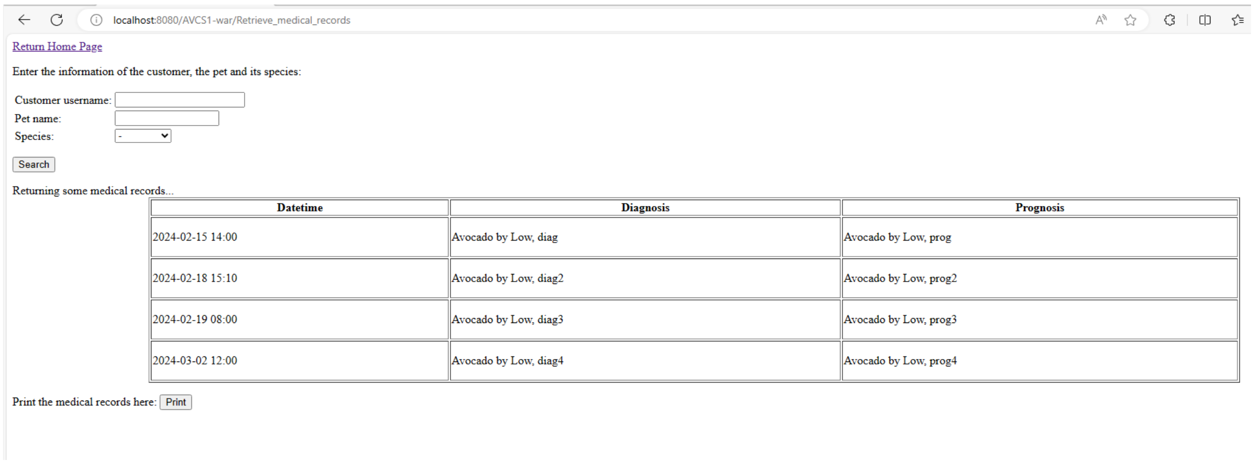
Enter the information of the customer, the pet and its species:

Customer username:	Low
Pet name:	Avocado
Species:	Dogs

Search

Figure 97: Prompting customer username, pet name, and species of the pet

It is important for a system to trace back the past medical histories of a pet, so that the trend in the health condition of the pet can be observed. The customer might approach any receptionist to ask for the medical histories so that they can send the pet to other clinics or even hospitals (when the disease becomes severe until the clinic cannot handle it anymore). Hence, the figure above shows how the system prompts the customer username, pet name, and pet species for generating medical histories.



localhost:8080/AVCS1-war/Retrieve_medical_records

Return Home Page

Enter the information of the customer, the pet and its species:

Customer username:	
Pet name:	
Species:	-

Search

Returning some medical records...

Datetime	Diagnosis	Prognosis
2024-02-15 14:00	Avocado by Low, diag	Avocado by Low, prog
2024-02-18 15:10	Avocado by Low, diag2	Avocado by Low, prog2
2024-02-19 08:00	Avocado by Low, diag3	Avocado by Low, prog3
2024-03-02 12:00	Avocado by Low, diag4	Avocado by Low, prog4

Print the medical records here: Print

Figure 98: Generate medical histories for the pet "Avocado"

All the medical records are sorted from earliest to latest, and this is for observing the trend in health conditions of the pet “Avocado” by the customer “Low”. Then, there is a button ***Print*** for the receptionist to print the medical histories.

References

amazon, a. (n.d.). *Building Applications with Serverless Architectures*. Retrieved from aws amazon: https://aws.amazon.com/lambda/serverless-architectures-learn-more/#:~:text=What%20is%20a%20serverless%20architecture,management%20is%20done%20by%20AWS.

Awati, R., & Wigmore, I. (2022, May). *monolithic architecture*. Retrieved from techtarget: https://www.techtarget.com/whatis/definition/monolithic-architecture

aws. (n.d.). *What is an Event-Driven Architecture?* Retrieved from aws: https://aws.amazon.com/event-driven-architecture/

Base, W. k. (2023, October 25). *10 Top PHP Frameworks For Robust Web Development in 2024*. Retrieved from linkedin: https://www.linkedin.com/pulse/10-top-php-frameworks-robust-web-development-rndexperts-l65cf/

Bonisteel, S. (2023, November 15). *Ruby vs Ruby on Rails: What's the Difference?* Retrieved from kinsta: https://kinsta.com/blog/ruby-vs-ruby-on-rails/

development, W. (2023, July 14). *Zend Framework*. Retrieved from ionos: https://www.ionos.com/digitalguide/websites/web-development/zend-framework-the-php-web-application-architecture/

Emily. (2022, February 7). *7 Good Reasons To Use Zend Framework For Your Website*. Retrieved from redalkemi: https://www.redalkemi.com/blog/7-good-reasons-to-use-zend-framework-for-your-website#:~:text=Being%20an%20object%2Doriented%20PHP,non%2Dobject%2Doriented%20frameworks.

Fernando, C. (2018, March 24). *The evolution of Distributed Systems*. Retrieved from medium: https://medium.com/microservices-learning/the-evolution-of-distributed-systems-fec4d35beffd

guardex. (2023, November 15). *ASP.NET Core Blazor*. Retrieved from learn.microsoft: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0

Hernandez, R. D. (2021, April 19). *The Model View Controller Pattern – MVC Architecture and Frameworks Explained*. Retrieved from freeCodeCamp: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

hitechnectar. (n.d.). *Role of Service-Oriented Architecture in Cloud Computing Explained*. Retrieved from hitechnectar: https://www.hitechnectar.com/blogs/service-oriented-architecture-cloud-computing/

IBM. (n.d.). *What is service-oriented architecture (SOA)?* Retrieved from IBM: https://www.ibm.com/topics/soa

JACINTO, A. (2023, September 14). *HOW ENTERPRISE APPS CAN SHAPE YOUR BUSINESS FOR SUCCESS*. Retrieved from STARTECHUP: https://www.startechup.com/blog/enterprise-apps/#What_Is_An_Enterprise_Application

javaTpoint. (n.d.). *How to build a Web Application Using Java*. Retrieved from javaTpoint: https://www.javatpoint.com/how-to-build-a-web-application-using-java

Juviler, J. (2022, March 30). *Static vs. Dynamic Websites: Here's the Difference*. Retrieved from hubspot: https://blog.hubspot.com/website/static-vs-dynamic-website#:~:text=Static%20vs.%20Dynamic%20Website,different%20information%20to%20different%20visitors.

Kalwan, A. (2023, Jun 19). *What Is JSP In Java? Know All About Java Web Applications*. Retrieved from edureka: https://www.edureka.co/blog/jsp-in-java/

Kanade, V. (2023, July 5). *What Is ARPANET? Definition, Features, and Importance*. Retrieved from spiceworks: https://www.spiceworks.com/tech/networking/articles/what-is-arpanet/

Kenneth. (2014, November 9). *client-MVC vs server-MVC*. Retrieved from stackoverflow: https://stackoverflow.com/questions/16447443/client-mvc-vs-server-mvc

Kumar, A. (2023, September 26). *What is the difference between client-side MVC and server-side MVC?* Retrieved from mindstick: https://www.mindstick.com/forum/159926/what-is-the-difference-between-client-side-mvc-and-server-side-mvc

Kumar, S. (2023, September 27). *Evolution of Distributed Computing Systems*. Retrieved from tutorialspoint: https://www.tutorialspoint.com/evolution-of-distributed-computing-systems#:~:text=The%20early%20days%20of%20distributed,sharing%20networks%20and%20email%20systems.

Landscape, B. D. (2023, September 25). *History of Distributed Computing*. Retrieved from medium: https://medium.com/@big_data_landscape/history-of-distributed-computing-8d64b71bad1d

Lindsay, D., Gill, S., Smirnova, D., & Garraghan, P. (2020). The Evolution of Distributed Computing Systems: From Fundamentals to New Frontiers. *Computing (Springer), 103*, 1859-1878. doi:https://doi.org/10.1007/s00607-020-00900-y

master. (2019, March 12). *1960's – 1970's Communication*. Retrieved from e-tutes: https://e-tutes.com/lesson1/1960-s-1970-s-communication/

Netbeans, A. (n.d.). *The NetBeans E-commerce Tutorial - Adding Entity Classes and Session Beans*. Retrieved from Apache Netbeans: https://netbeans.apache.org/tutorial/main/kb/docs/javaee/ecommerce/entity-session/

Pankaj. (2022, August 4). *Java Web Application Tutorial for Beginners*. Retrieved from DigitalOcean: https://www.digitalocean.com/community/tutorials/java-web-application-tutorial-for-beginners#servlets-jsps

Pareek, D. (2021, March 10). *Ruby on Rails Introduction*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/ruby-on-rails-introduction/

Park, M. (2022, October 10). *8 Principles of Service-Oriented Architecture: Is SOA Dead?* . Retrieved from MuleSoft: https://blogs.mulesoft.com/digital-transformation/soa-principles/

Pedamkar, P. (2023, July 3). *AngularJS Architecture*. Retrieved from educba: https://www.educba.com/angularjs-architecture/

psr. (2011, October 18). *Does business logic really belong on the server?* Retrieved from StackExchange: https://softwareengineering.stackexchange.com/questions/114794/does-business-logic-really-belong-on-the-server

SarthakG. (2023, January 10). *Service-Oriented Architecture*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/service-oriented-architecture/

seo@ebuilderz.site. (2023, February 17). *Angular vs AngularJS: Differences Between Angular and AngularJS*. Retrieved from stackify: https://stackify.com/angular-vs-angularjs-differences-between-angular-and-angularjs/

Shah, M. (2023, October 19). *Mobile App Architecture: Everything You Need to Know*. Retrieved from radixweb: https://radixweb.com/blog/guide-to-mobile-app-architecture

Sharma, I. (n.d.). *What is Enterprise Application Architecture and Its Types?* Retrieved from TatvaSoft: https://www.tatvasoft.com/outsourcing/2022/10/enterprise-application-architecture.html

Shut, A. (2023, August 3). *Ruby vs Ruby on Rails: Top 6 Differences*. Retrieved from netguru: https://www.netguru.com/blog/ruby-versus-ruby-on-rails

Silva, H. S. (2021, May 12). *Evolution of Client Computing Architecture*. Retrieved from medium: https://silvahansini.medium.com/evolution-of-client-computing-architecture-70e9097d9b96

Singh, J. (2018, Jun 7). *The What, Why, and How of a Microservices Architecture*. Retrieved from medium: https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9

Singhal, A. (2016, January 21). *Client-Side MVC Framework : Why you should use?* Retrieved from linkedin: https://www.linkedin.com/pulse/client-side-mvc-framework-why-you-should-use-aman-singhal/

thomasWeise. (2016). *distributedComputingExamples*. Retrieved from github: https://github.com/thomasWeise/distributedComputingExamples

Tom. (2013, September 6). *What is client side MVC and how is it implemented in JavaScript?* Retrieved from stackoverflow: https://stackoverflow.com/questions/18653014/what-is-client-side-mvc-and-how-is-it-implemented-in-javascript tutorialspoint. (n.d.). *Zend Framework- Introduction*. Retrieved from tutorialspoint: https://www.tutorialspoint.com/zend_framework/zend_framework_introduction.htm

Vaidya, N. (2021, June 17). *Introduction to Java Servlets – Servlets in a Nutshell*. Retrieved from edureka: https://www.edureka.co/blog/java-servlets#Servlet

w3schools. (n.d.). *AngularJS Introduction*. Retrieved from w3schools: https://www.w3schools.com/angular/angular_intro.asp#:~:text=AngularJS%20is%20a%20JavaScript%20framework,data%20to%20HTML%20with%20Expressions.

Wang, J. (n.d.). *Chapter 2 - The Evolution of Distributed Systems*. Retrieved from umbc edu: https://userpages.umbc.edu/~jianwu/is651/651book/is651-strapdown.php?f=is651-Chapter02.md

William. (2022, March 10). *Web Application Architecture: The Latest Guide 2024*. Retrieved from clickittech: https://www.clickittech.com/devops/web-application-architecture/

Wright, G. (2021, November). *APRANET*. Retrieved from techtarget: https://www.techtarget.com/searchnetworking/definition/ARPANET#:~:text=The%20U.S.%20Advanced%20Research%20Projects,for%20academic%20and%20research%20purposes.