

### **Team Members:**

**T P MURLETHAAR (20BLC1090) ([murlethaar.tp2020@vitstudent.ac.in](mailto:murlethaar.tp2020@vitstudent.ac.in))**

**UTKARSH TIWARI(20BCE10284)([utkarsh.tiwari2020@vitbhopal.ac.in](mailto:utkarsh.tiwari2020@vitbhopal.ac.in))**

**ISHAN SATI(20BCT0201)([ishan.sati2020@vitstudent.ac.in](mailto:ishan.sati2020@vitstudent.ac.in))**

**Vemasani Varshini(20BCE1240)([vemasani.varshini2020@vitstudent.ac.in](mailto:vemasani.varshini2020@vitstudent.ac.in))**

## **Time Banking**

### **Project Description :**

Time banking is a system based on the concept of reciprocal service exchange where individuals or communities exchange services with each other using time as the currency. It is an alternative economic model that values the skills, knowledge, and time of individuals equally. In a time banking system, participants earn and spend "time credits" instead of traditional money. When someone provides a service or assistance to another member, they earn time credits for the hours they spent. These time credits can then be used to receive services from other members of the time bank. Basically, the motive of the product is time banking. And what our project does is we exchange services in the place of currency. That is, if we need some service, that has to be done for us. We will make the user do some other service as a return of favor for the service that has to be needed for them. That has to be done for them. So basically our product is using a barter system instead of the currency system so users can provide service and take service based on their credit system. And the credit system adds up when they do services. I mean, the credits add up when they do services, and it declines when they decrease when they take services from other users. So basically, if one of our users is a tax agent and the other person is a carpenter, and if this Carpenter person needs his taxes filed, then he can go to the tax agent and get the work done. So this tax agent will have positive score credits and he will have negative credits and this negative credits will become positive if he does equivalent amount of job service or job to some other person who needs carpentry. So this is how the system works and here is the code. And basically, this product hasn't been put to work completely yet, but here is the glimpse.

### **Technologies and Requirements**

**IDE : My Eclipse**

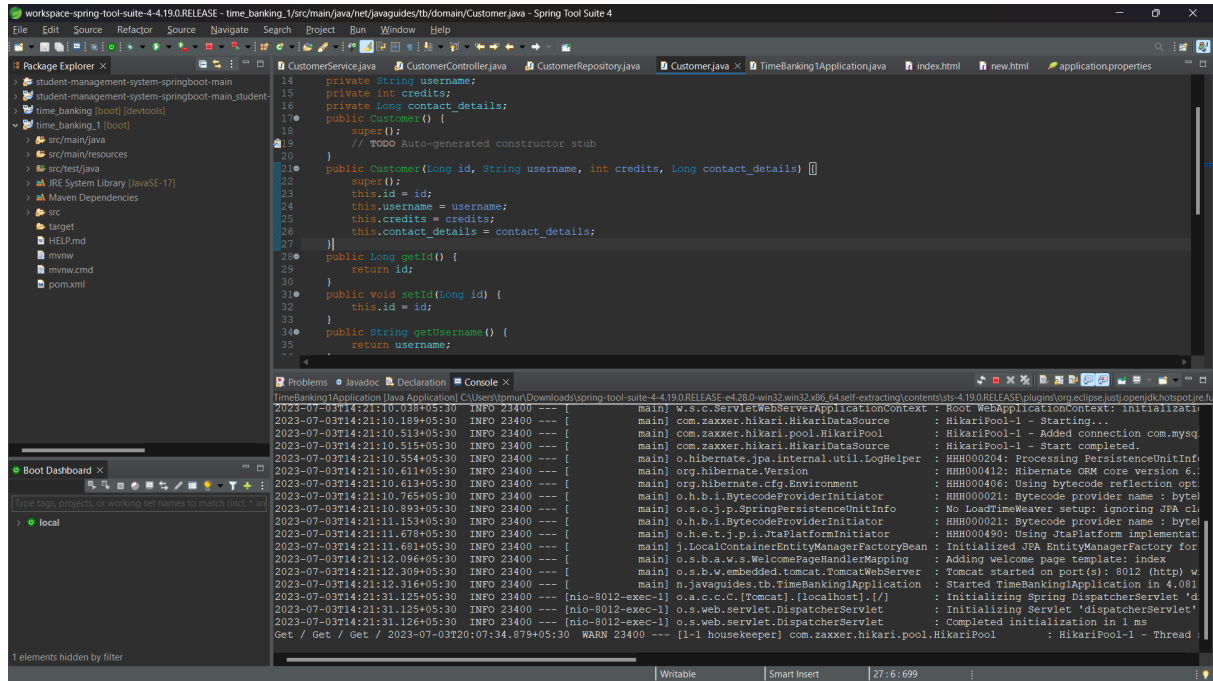
**Front End : HTML, CSS , Javascript**

**Programming Language : JAVA**

## Back End : MySQL DB

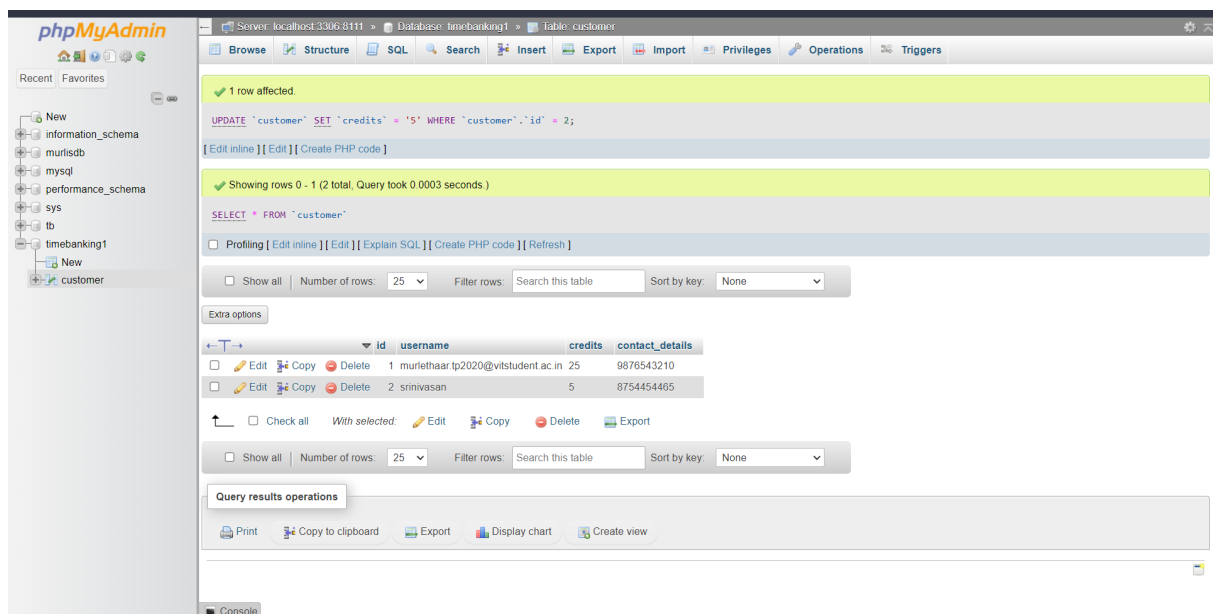
### Project Structure :

### Project folder Structure Image



### Project folder Structure explanation





### Database Table:



## Transaction Details :



### Transaction Details

Id	Name	Hours Taken	Phone No.	Email	Address	Edit	Delete
1	MURLETHAAR TP	25	9876543210	murlethaar.tp2020@vitstudent.ac.in	guru keerthanam apartments, sabari salai, madipakkam, chennai -91		
2	srinivasan TP	5	8754454462	murlethaar.tp2020@vitstudent.ac.in	guru keerthanam apartments, sabari salai, madipakkam, chennai -91		

## Pre-requisites :

Here are the prerequisites for our Java Spring Boot project with MySQL , along with relevant links for download and installation:

**1-Java Development Kit (JDK):**JDK is required to compile and run Java applications, providing the necessary tools and libraries. Download and install the latest JDK version from Oracle's website.

- Download JDK: <https://www.oracle.com/java/technologies/javase-jdk11-downloads.html>

**2-Integrated Development Environment (IDE):** An IDE offers a comprehensive development environment for writing, debugging, and managing code. IntelliJ IDEA, Eclipse, or Visual Studio Code are popular choices for Java development.

- Eclipse: <https://www.eclipse.org/downloads/>

**3-Spring Boot:** Spring Boot simplifies Java application development by providing predefined configurations, automatic dependency management, and a streamlined development experience. Use Spring Initializr or Spring Tools for your IDE to create a Spring Boot project.

- Spring Initializr (Online): <https://start.spring.io/>

- Spring Tools 4 for Eclipse: <https://spring.io/tools>
- Spring Tools for Visual Studio Code: Install via Extensions in Visual Studio Code

**4-MySQL Database:** MySQL is a popular relational database management system. Install MySQL Community Server and optionally MySQL Workbench, a graphical tool for managing MySQL databases.

- MySQL Community Server: <https://dev.mysql.com/downloads/installer/>
- MySQL Workbench: <https://dev.mysql.com/downloads/workbench/>

**5-MySQL Connector/J:** MySQL Connector/J is the official JDBC driver for connecting Java applications to MySQL databases. Include this dependency in your project to enable connectivity and interaction with MySQL.

- Maven:
- Add the following dependency to your project's pom.xml:

xml

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-data-jpa</artifactId>

    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-thymeleaf</artifactId>

    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

    <dependency>

        <groupId>com.mysql</groupId>

```
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
```

- Maven Repository: <https://mvnrepository.com/artifact/mysql/mysql-connector-java>

#### - Gradle:

- Add the following dependency to your project's build.gradle:

```
implementation 'mysql:mysql-connector-java:8.0.27'
```

- Gradle Repository:

<https://search.maven.org/artifact/mysql/mysql-connector-java/8.0.27/jar>

Make sure to download and install the appropriate versions based on your system and project requirements. With these prerequisites in place, you'll be ready to start building your Java Spring Boot project with both MySQL or MongoDB as the database.

## **CONTROLLER CODE:**

### **Controller Annotation:**

This annotation marks the class as a Spring MVC controller.

### **Autowired field:**

This annotation injects the `CustomerService` dependency into the controller. It allows the controller to use the methods provided by the `CustomerService` class.

### **Request Mapping and GetMapping annotations:**

This annotation maps the URL `/` to the `viewHomePage()` method. It handles the GET request for the home page.

#### **Method:** `viewHomePage()`

This method retrieves a list of all customers from the `CustomerService` and adds it to the model as an attribute named `"listcustomer"`. The method then prints `"Get / "` to the console and returns the string `"index"`. This indicates that the `"index"` template should be rendered for the home page.

#### **Method:** `add()`

This method adds a new `Customer` object to the model with the attribute name `"customer"`. It then returns the string `"new"`, indicating that the `"new"` template should be rendered for adding a new customer.

### **Request Mapping and RequestMethod annotations:**

This annotation maps the URL `/save` to the `saveCustomer()` method. It handles the POST request for saving a customer.

#### **Method:** `showEditStudentPage()`

This method receives the customer ID from the URL path variable using the `@PathVariable` annotation.

### **Request Mapping annotation:**

This annotation maps the URL `"/edit/{id}"` to the `showEditStudentPage()` method. It handles the request for editing a customer with the specified ID.

### **CUSTOMER DETAILS CODE:**

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
```

These import statements bring in the necessary annotations from the Jakarta Persistence API, which is used for Object-Relational Mapping (ORM) in Java.

### **Entity Annotation:**

This annotation marks the Customer class as an entity, indicating that it corresponds to a table in a database.

### **Constructors:**

```
public Customer() {
    super();
    // TODO Auto-generated constructor stub
}
```

```
public Customer(Long id, String username, int credits, Long contact_details) {
    super();
    this.id = id;
    this.username = username;
    this.credits = credits;
    this.contact_details = contact_details;
}
```

The first constructor is a default constructor with no parameters.

The second constructor takes the values for all the fields as parameters and initializes the corresponding instance variables.

Getter and Setter Methods:

```
public Long getId() {  
    return id;  
}
```

```
public void setId(Long id) {  
    this.id = id;  
}
```

```
public String getUsername() {  
    return username;  
}
```

```
public void setUsername(String username) {  
    this.username = username;  
}
```

```
public int getCredits() {  
    return credits;  
}
```

```
public void setCredits(int credits) {  
    this.credits = credits;  
}
```

```
public Long getContact_details() {  
    return contact_details;  
}
```



```
}
```

```
public void setContact_details(Long contact_details) {  
    this.contact_details = contact_details;  
}
```

These methods are getter and setter methods that provide access to the private fields of the Customer class. They allow getting and setting the values of the fields.

Overall, this code defines the Customer class with its fields, constructors, and getter/setter methods, which can be used to represent customer data in a database or any other application.

### **index.html :**

```
<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml"  
    xmlns:th="http://www.thymeleaf.org">  
  
    <head>  
  
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />  
  
        <script  
            src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"  
        ></script>  
  
    </head>  
  
    <body>  
  
        <div>  
  
            <h2>Spring Boot Crud System -Customer Registration</h2>  
  
            <tr>  
  
                <div align = "left" >  
  
                    <h3><a th:href="@{/new}">Add new</a></h3>  
  
                </div>  
  
            </tr>  
  
        </div>  
  
    </body>  
  
</html>
```

```

</tr>
<tr>

<div class="col-sm-5" align = "center">
    <div class="panel-body" align = "center" >
        <table class="table">
<thead class="thead-dark">
<tr>
    <th>user_id</th>
    <th>username</th>
    <th>credits</th>
    <th>contact_details</th>
    <th>Edit</th>
    <th>Delete</th>
</tr>
</thead>
<tbody>
<tr th:each="customer : ${list customer }">
    <td th:text="${customer.id}">user_id</td>
    <td th:text="${customer.username}">username</td>
    <td th:text="${customer.credits}">credits</td>
    <td th:text="${customer.contact_details}">Contact_details</td>
    <td>
        <a th:href="@{/edit/ + ${customer.id}}">Edit</a>
    </td>
    <td>
        <a th:href="@{/delete/ + ${customer.id}}">Delete</a>
    </td>
</tr>

</tbody>
</table>

```

```
        </div>

    </div>

</tr>

</tbody>
</table>

<div>
</body>
</html>
```

### Role Based Access:-

Roles of Admin, User can be defined for book a service application:

#### 1-Admin Role:

- **Responsibilities:** The admin role has full control and administrative privileges over the system.
- **Permissions:**
  - **Manage transactions:** Admins can add, edit, and delete information.
  - **Generate reports:** Admins have access to generate reports and analytics on revenue, and other system statistics.

#### 2-User Role:

- **Responsibilities:** Users are the customers of the service booking application who can search for multiple kinds of services, make bookings..
- **Permissions:**
  - **View available users:** Users can search for services, view their schedules, fares, and availability.
  - **Book service:** Users can select service and complete the booking process.
  - **Manage bookings:** Users can view their own bookings, modify service details, and cancel their bookings if permitted by the system rules.
  - **View personal profile:** Users can view and edit their own profile information, such as contact details or frequent flyer numbers.

**CODE:**

```
package net.javaguides.tb.controller;

import org.hibernate.mapping.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import net.javaguides.tb.domain.Customer;
import net.javaguides.tb.service.CustomerService;

@Controller

public class CustomerController {

    @Autowired
    private CustomerService service;

    @GetMapping("/")
    public String viewHomePage(Model model) {
        java.util.List<Customer> listcustomer = service.listAll();
        model.addAttribute("listcustomer", listcustomer);

        System.out.print("Get / ");

        return "index";
    }
}
```

```

    }

    @GetMapping("/new")
    public String add(Model model) {
        model.addAttribute("customer", new Customer());
        return "new";
    }

    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String saveCustomer(@ModelAttribute("customer") Customer std) {
        service.save(std);
        return "redirect:/";
    }

    @RequestMapping("/edit/{id}")
    public ModelAndView showEditStudentPage(@PathVariable(name = "id") int id) {
        ModelAndView mav = new ModelAndView("new");
        Customer std = service.get(id);
        mav.addObject("customer", std);
        return mav;
    }

    @RequestMapping("/delete/{id}")
    public String deletecustomer(@PathVariable(name = "id") int id) {
        service.delete(id);
        return "redirect:/";
    }
}

```

## Project Flow:

- Frontend Development
- Backend Development
- Integration
- Containerization of the Application
- Deployment to Kubernetes Cluster

## Milestone 1: Frontend Development:

Frontend development involves building the user interface (UI) and implementing the visual elements of the application. It focuses on creating an intuitive and engaging user experience that allows users to interact with the application seamlessly. The following activities are part of the frontend development process:

User   Display Transactions   Add Transaction

## Timebanking System

### Activity 1: UI Design and Layout

- Design the overall user interface (UI) for the time banking application.
- Create wireframes and mockups to visualize the layout and structure of the application.
- Determine the color scheme, typography, and overall visual style.
- Implement responsive design to ensure the application is compatible with different devices.
- 

### Activity 2: Creating Transaction Display and Edit

- Start by creating a user page for our timebanking application.
- Implement and create a different page for adding transactions to our server with all details stored in the sql database when added. It takes different parameters like hours to be transacted.

### Activity 3: User Authentication and Displaying Transactions

- Implement the authentication part in the form based where the user enters transaction details.
- Create a Display Transactions page which shows every transaction that is added with all details.
- Add edit transaction button here that lets users edit all these transactions for the user.

### Milestone 2: Backend Development:

Backend development involves building the server-side components and logic of the service booking application. It focuses on handling the business logic, processing requests from the frontend, and interacting with the database. The following activities are part of the backend development process:

#### Activity 1: Transaction details

- Set up the server environment and choose a suitable backend framework such as Java Spring Boot.
- Develop the RESTful APIs for service search, booking management, user authentication, and profile management.
- Implement server-side validation and error handling for API requests and responses.
- Integrate with external services such as payment gateways or service data providers, if required.

#### CODE:

```
package net.javaguides.tb.domain;
```

```
import jakarta.persistence.Entity;
```

```
import jakarta.persistence.GeneratedValue;
```

```
import jakarta.persistence.GenerationType;
```

```
import jakarta.persistence.Id;
```

```
@Entity
```

```
public class Customer {
```

```
    @Id
```

```
    @GeneratedValue(strategy= GenerationType.IDENTITY)
```

```
private Long id;

private String username;

private int credits;

private Long contact_details;

    public Customer() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Customer(Long id, String username, int credits, Long contact_details) {
        super();
        this.id = id;
        this.username = username;
        this.credits = credits;
        this.contact_details = contact_details;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public int getCredits() {
        return credits;
    }
```



```
}  
  
public void setCredits(int credits) {  
    this.credits = credits;  
}  
  
public Long getContact_details() {  
    return contact_details;  
}  
  
public void setContact_details(Long contact_details) {  
    this.contact_details = contact_details;  
}  
  
}
```

username

credits

contact\_details

Save

## Activity 2: Database Integration and ORM

- Set up the database server (MySQL) and establish the necessary database connections.
- Design the database schema based on the application requirements, including tables/entities and their relationships.
- Implement object-relational mapping (ORM) techniques (e.g., Hibernate or Spring Data) to interact with the database.
- Develop database queries and CRUD operations for services available data, user information, and bookings.

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/timebanking1?useUnicode=true&useJDBCCompliantTimezone
2
3 server.error.whitelabel.enabled=false
4 server.port=8012
5 spring.datasource.username=root
6 spring.datasource.password=1234
7
8 spring.jpa.open-in-view=false
9 spring.thymeleaf.cache=false
```

## Milestone 3: Integration:

Integration is the process of combining and connecting the frontend and backend components of the time banking application to create a unified and fully functional system. It involves establishing communication channels, exchanging data, and ensuring seamless interaction between the frontend UI and backend APIs. The following activities are part of the integration process:

### Activity 1: Frontend-Backend Integration

(adding transaction details - storing the details)

[User](#) [Display Transactions](#) [Add Transaction](#)

## Add Transaction Details

First name	Last name	Phone No.
<input type="text"/>	<input type="text"/>	<input type="text" value="+91"/>
Address	Email	Hours Needed
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="submit" value="Submit"/>		



## Transaction Details

Id	Name	Hours Taken	Phone No.	Email	Address	Edit	Delete
1	MURLETHAAR TP	25	9876543210	murlethaar.tp2020@vitstudent.ac.in	guru keerthanam apartments, sabari salai, madipakkam, chennai -91		
2	srinivasan TP	5	8754454462	murlethaar.tp2020@vitstudent.ac.in	guru keerthanam apartments, sabari salai, madipakkam, chennai -91		

- Integrate the frontend UI components with the backend, ensuring proper communication and data exchange.
- Handle data validation and error responses between the frontend and backend components.
- Conduct thorough testing to ensure seamless integration and compatibility between frontend and backend.

### Milestone 4: Containerization of the Application

Containerization is the process of packaging an application and its dependencies into a standardized unit called a container. Containers offer a lightweight and consistent runtime environment that can be easily deployed across various platforms and environments.

#### Activity 1: Dockerfile Creation

- Create a Dockerfile that defines the necessary steps to build the application image.
- Specify the base image, install dependencies, and configure the container environment.

#### Activity 2: Building the Docker Image

- Build the Docker image using the Dockerfile.
- Include all the required application files and dependencies in the image.

#### Activity 3: Container Testing

- Run and test the container locally to ensure it functions as expected.
- Verify that the application runs within the container environment without issues.

#### Activity 4: Publishing the Docker Image

- Push the built Docker image to a container registry (e.g., Docker Hub) to make it accessible to the Kubernetes cluster.
- Ensure proper tagging and versioning of the image for easy identification.

### **Milestone 5: Deployment to Kubernetes Cluster**

Kubernetes provides a platform for automating the deployment, scaling, and management of containerized applications. Deploying the application to a Kubernetes cluster enables efficient orchestration and scalability.

#### **Activity 1: Kubernetes Manifest**

- Create Kubernetes manifest files (YAML or JSON) to define the deployment specifications.
- Specify details such as the container image, resource requirements, and desired replicas.

#### **Activity 2: Deploying the Application**

- Use the Kubernetes command-line interface (kubectl) or deployment tools to apply the manifest files and deploy the application.
- Verify that the application pods are running and healthy within the cluster.

#### **Activity 3: Exposing the Application**

- Configure a Kubernetes service to expose the application internally within the cluster.
- Define appropriate service type (ClusterIP, NodePort, LoadBalancer) based on requirements.

#### **Activity 4: Verification**

- Test the deployed application to ensure it functions correctly in the Kubernetes environment.

### **Screenshots (Frontend and Backend):**

# Add Transaction Details

First name

Last name

Phone No.

+91

Address

Email

Hours Needed

Submit



username

username

credits

0

contact\_details

contact\_details

Save

Transaction Details

id	Name	Hours Taken	Phone No.	Email	Address	Edit	Delete
1	MURLETHAAR TP	25	9876543210	murlethaar.tp2020@vitstudent.ac.in	guru keerthanam apartments, sabari salai, madipakkam, chennai -91		
2	srinivasan TP	5	8754454462	murlethaar.tp2020@vitstudent.ac.in	guru keerthanam apartments, sabari salai, madipakkam, chennai -91		

Spring Boot System - Time Banking Customer Registration

[Add new](#)

user_id	username	credits	contact_details	Edit	Delete
1	murlethaar.tp2020@vitstudent.ac.in	25	9876543210	<a href="#">Edit</a>	<a href="#">Delete</a>
2	20blc1090	5000	8754454465	<a href="#">Edit</a>	<a href="#">Delete</a>

Timebanking System

phpMyAdmin

Server: localhost:3306 [111] Database: timebanking1 Table: customer

Recent Favorites

New information\_schema murisdb mysql performance\_schema sys tb timebanking1 New customer

1 row affected.

```
UPDATE `customer` SET `credits` = '5' WHERE `customer`.`id` = 2;
```

[ Edit inline ] [ Edit ] [ Create PHP code ]

Showing rows 0 - 1 (2 total, Query took 0.0003 seconds)

```
SELECT * FROM `customer`
```

[ Profiling ] [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	id	username	credits	contact_details
<input type="checkbox"/>	1	muriethaar tp2020@vilstudent.ac.in	25	9876543210
<input type="checkbox"/>	2	srinivasan	5	8754454465

Check all With selected: Edit Copy Delete Export

Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

Console

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- murisdb
- sys
- tb
- timebanking1
  - Tables
    - customer
      - Columns
        - id
        - username
        - credits
        - contact\_details
      - Indexes
      - Foreign Keys
      - Triggers
    - Views
    - Stored Procedures
    - Functions

Administration Schemas

Information

Table: customer

Columns:

Column	Type	Attributes
id	int	AI PK
username	varchar(50)	
credits	varchar(30)	
contact_details	varchar(30)	

Object Info Session

QL Editor Opened.

Query 1

Limit to 1000 rows

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
---	------	--------	---------	------------------