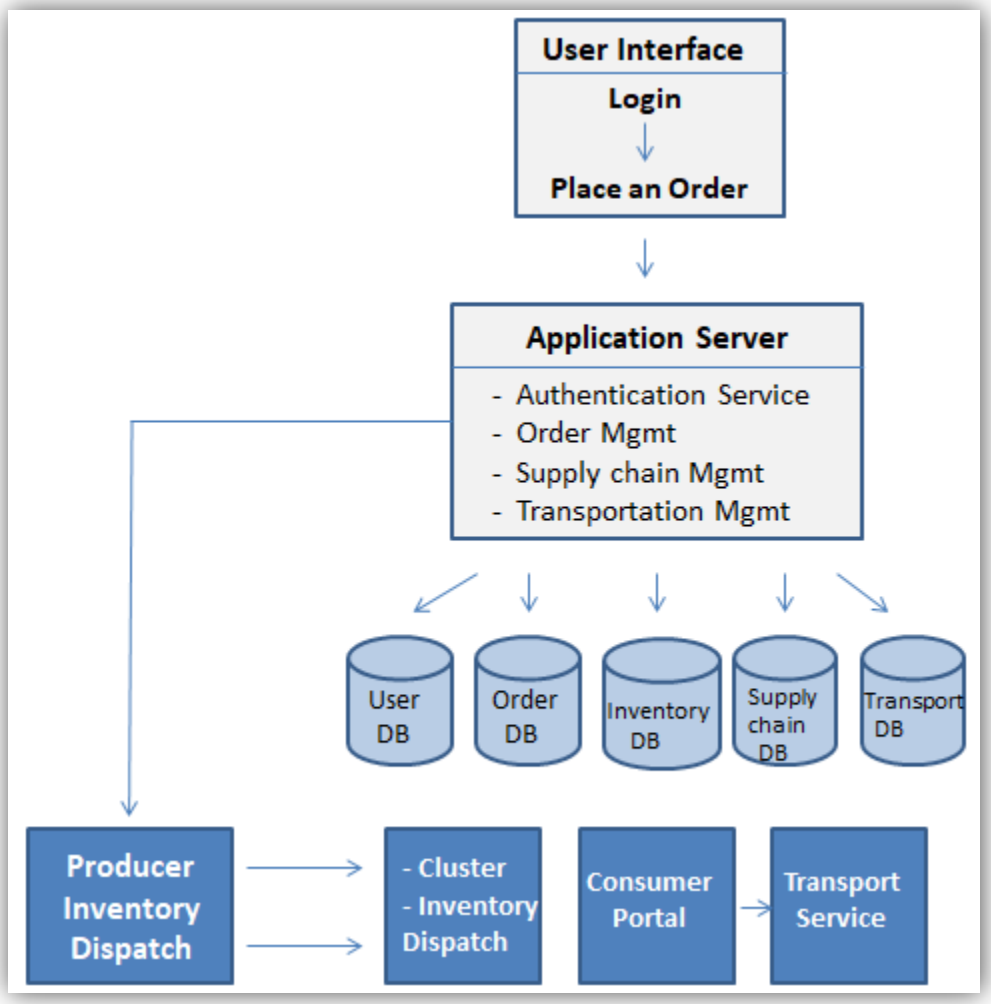
INDEX

Explanation of Full Stack Project

Objectives

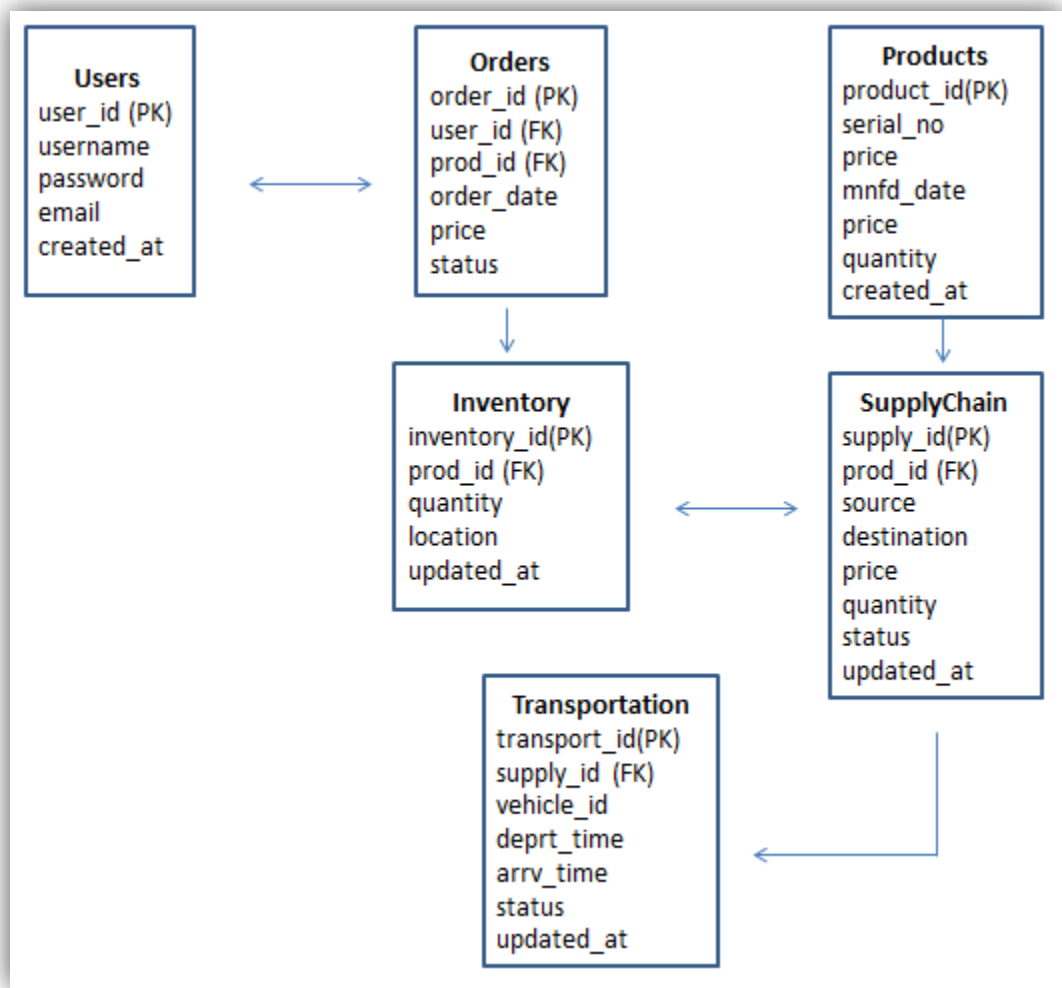
It's an web application developed for farmers that connects farmers and consumers directly. The main aim of this project is to encourage customer to reach the farmer directly and buy produce from the farmer directly and to prevent at least one farmer suicide. The presence of too many middlemen results in the exploitation of both farmers and consumers who offers lower prices for farmers and higher for consumers.

Backend



1. **User Interface (UI)**
    - **User Login -** Users log in and place orders via the User Interface.
    - **Place an Order –** User places an order.
2. **Application Server**
    - **Authentication Service:** Handles user login and authentication.
    - **Order Management Service:** Manages order placements.
    - **Supply Chain Management Service:** Manages consumer-initiated product supply.
    - **Transportation Management Service:** Handles transportation of products from producer to consumer.
3. **Database**
    - **User Database:** Stores user information and credentials.
    - **Order Database:** Stores order details and status.
    - **Inventory Database:** Tracks product inventory.
    - **Supply Chain Database:** Manages supply chain information.
    - **Transportation Database:** Tracks transportation details.
4. **Producer Inventory:** Manages product inventory at the producer end.
    - **Dispatch Service:** Handles dispatch of products from producer to consumer.
5. **Consumer Portal:** Interface for consumers to place orders and track deliveries.
6. **Delivery Service:** Manages delivery of products to the consumer.
7. **Cluster Inventory:** Centralized inventory for managing supply to multiple consumers.
    - **Cluster Dispatch Service:** Manages dispatch from the cluster to consumers.

Database Design



Explanation

1. **Users Table:** Contains user information.
2. **Products Table:** Contains product information.
3. **Orders Table:** Links users and products, recording each order's details.
4. **Inventory Table:** Tracks the quantity and location of products.
5. **Supply Chain Table:** Manages the supply chain information, linking products to their source and destination.
6. **Transportation Table:** Manages transportation details, linking to the supply chain.

Relationships

1. **Users to Orders:** One-to-Many (a user can place multiple orders).
2. **Products to Orders:** One-to-Many (a product can appear in multiple orders).
3. **Products to Inventory:** One-to-Many (a product can be in multiple inventory locations).
4. **Products to SupplyChain:** One-to-Many (a product can be part of multiple supply chains).
5. **SupplyChain to Transportation:** One-to-One (each supply chain entry has a corresponding transportation record).

Server side Logic

We'll use the gorilla/mux package for routing and gorm for ORM (Object-Relational Mapping).

Initial Setup

```
go mod init dealership
go get -u github.com/gorilla/mux
go get -u gorm.io/gorm
go get -u gorm.io/driver/mysql
```

Create a main.go file.

```go
import (
    "dealership/handlers"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

func main() {
    r := mux.NewRouter()

    // User routes
    r.HandleFunc("/register", handlers.Register).Methods("POST")
    r.HandleFunc("/login", handlers.Login).Methods("POST")

    // Order routes
    r.HandleFunc("/orders", handlers.PlaceOrder).Methods("POST")

    // Inventory routes
    r.HandleFunc("/inventory", handlers.UpdateInventory).Methods("PUT")

    // Supply chain routes
    r.HandleFunc("/supply_chain", handlers.ManageSupplyChain).Methods("POST")

    // Transportation routes
    r.HandleFunc("/transportation", handlers.ManageTransportation).Methods("POST")

    log.Fatal(http.ListenAndServe(":8000", r))
```

Database Connection and Models

Create a db package for database connection and models:

db.go

```go
package db

import (
    "gorm.io/driver/mysql"
    "gorm.io/gorm"
    "log"
)

var DB *gorm.DB

func Init() {
    dsn := "user:password@tcp(127.0.0.1:3306)/dealership?charset=utf8mb4&parseTime=True&lo
    var err error
    DB, err = gorm.Open(mysql.Open(dsn), &gorm.Config{})
    if err != nil {
        log.Fatal("Failed to connect to database:", err)
    }

    DB.AutoMigrate(&User{}, &Product{}, &Order{}, &Inventory{}, &SupplyChain{}, &Transport
}
```

Db.models.go

```go
package db

import "time"

type User struct {
    ID        uint      `gorm:"primaryKey"`
    Username  string    `gorm:"unique;not null"`
    Password  string    `gorm:"not null"`
    Email     string    `gorm:"unique;not null"`
    CreatedAt time.Time
}

type Product struct {
    ID          uint      `gorm:"primaryKey"`
    SerialNo    string    `gorm:"unique;not null"`
    Model       string    `gorm:"not null"`
    Manufacturer string   `gorm:"not null"`
    Price       float64   `gorm:"not null"`
    CreatedAt   time.Time
}

type Order struct {
    ID        uint      `gorm:"primaryKey"`
    UserID    uint      `gorm:"not null"`
    ProductID uint      `gorm:"not null"`
    OrderDate time.Time `gorm:"default:CURRENT_TIMESTAMP"`
    SalePrice float64   `gorm:"not null"`
    Status    string    `gorm:"default:Pending"`
}
```

```go
type Inventory struct {
    ID        uint      `gorm:"primaryKey"`
    ProductID uint      `gorm:"not null"`
    Quantity  int       `gorm:"not null"`
    Location  string    `gorm:"not null"`
    UpdatedAt time.Time `gorm:"default:CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP"`
}

type SupplyChain struct {
    ID         uint      `gorm:"primaryKey"`
    ProductID  uint      `gorm:"not null"`
    Source     string    `gorm:"not null"`
    Destination string   `gorm:"not null"`
    Quantity   int       `gorm:"not null"`
    Status     string    `gorm:"default:In Transit"`
    UpdatedAt  time.Time `gorm:"default:CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP"`
}

type Transportation struct {
    ID           uint      `gorm:"primaryKey"`
    SupplyID     uint      `gorm:"not null"`
    VehicleID    string
    DriverName   string
    DepartureTime time.Time `gorm:"default:CURRENT_TIMESTAMP"`
    ArrivalTime  time.Time
    Status       string    `gorm:"default:On the way"`
}
```

Handlers

Create a handlers package for handling requests:

handlers/auth.go

```go
package handlers

import (
    "dealership/db"
    "encoding/json"
    "net/http"

    "golang.org/x/crypto/bcrypt"
    "gorm.io/gorm"
)

func Register(w http.ResponseWriter, r *http.Request) {
    var user db.User
    json.NewDecoder(r.Body).Decode(&user)

    hashedPassword, err := bcrypt.GenerateFromPassword([]byte(user.Password), bcrypt.Defau
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    user.Password = string(hashedPassword)

    result := db.DB.Create(&user)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusInternalServerError)
        return
    }

    w.WriteHeader(http.StatusCreated)
    json.NewEncoder(w).Encode(user)
}
```

```go
func Login(w http.ResponseWriter, r *http.Request) {
    var user db.User
    json.NewDecoder(r.Body).Decode(&user)

    var storedUser db.User
    result := db.DB.Where("username = ?", user.Username).First(&storedUser)
    if result.Error == gorm.ErrRecordNotFound {
        http.Error(w, "Invalid credentials", http.StatusUnauthorized)
        return
    }

    err := bcrypt.CompareHashAndPassword([]byte(storedUser.Password), []byte(user.Passwor
    if err != nil {
        http.Error(w, "Invalid credentials", http.StatusUnauthorized)
        return
    }

    // In a real application, you would generate a JWT token here
    w.WriteHeader(http.StatusOK)
    json.NewEncoder(w).Encode(storedUser)
}
```

Technologies

1. Golang
2. MySQL
3. HTML
4. CSS
5. Bootstrap
6. JavaScript
7. JQuery
8. Beego Framework
9. Robot Framework

1. Golang

Golang, or Go, is a statically typed, compiled programming language designed at Google. It is known for its simplicity, concurrency support, and performance. Go is particularly popular for building web servers, network tools, and other server-side applications. Key features include garbage collection, memory safety, and the ability to run multiple processes concurrently through go routines.

2. MySQL

MySQL is an open-source relational database management system (RDBMS) based on Structured Query Language (SQL). It is widely used for storing and managing data in web applications. MySQL supports data manipulation (CRUD operations), complex queries, transactions, and multiple storage engines, making it a versatile choice for various applications from small-scale to enterprise-level systems.

3. HTML

HTML (Hyper Text Markup Language) is the standard markup language for creating web pages. It provides the structure of a webpage, allowing developers to define elements such as headings, paragraphs, links, images, and other content. HTML uses tags and attributes to specify the appearance and functionality of elements on a webpage.

4. CSS

CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation of a document written in HTML or XML. CSS defines how HTML elements should be displayed, including layout, colors, fonts, and responsiveness. It separates content from design, allowing developers to change the look and feel of a website without altering its structure.

5. Bootstrap

Bootstrap is a popular open-source front-end framework for developing responsive and mobile-first websites. It includes HTML, CSS, and JavaScript components for common interface elements such as navigation bars, forms, buttons, and grids. Bootstrap's grid system and pre-built components help developers create consistent and aesthetically pleasing designs quickly.

6. JavaScript

JavaScript is a high-level, dynamic programming language primarily used for adding interactivity to web pages. It runs in the browser and enables client-side

scripting to create dynamic content, control multimedia, animate images, and handle events. JavaScript is an essential technology for modern web development, often used in combination with HTML and CSS.

7. JQuery

jQuery is a fast, small, and feature-rich JavaScript library designed to simplify HTML document traversal and manipulation, event handling, animation, and Ajax interactions. It provides an easy-to-use API that works across a multitude of browsers, allowing developers to write less code and achieve more functionality compared to plain JavaScript.

8. Beego Framework

Beego is a full-featured web application framework for Go. It is inspired by frameworks like Django (Python) and can be used for developing web applications, microservices, and APIs. Beego provides a robust set of features, including an MVC architecture, ORM for database management, built-in tools for handling sessions, caching, logging, and more. It simplifies the development of scalable web applications in Go.

9. Robot Framework

Robot Framework is an open-source automation framework used for test automation and robotic process automation (RPA). It uses a keyword-driven testing approach, making it readable and easy to use for both technical and non-technical users. Robot Framework supports external libraries (such as Selenium for web testing), and its capabilities can be extended with custom libraries implemented in Python or Java. It is ideal for acceptance testing, regression testing, and automated RPA tasks.

Hosting Environment and Deployment process

GitLab is a popular DevOps platform that provides a comprehensive suite of tools for software development, including version control, CI/CD (Continuous Integration/Continuous Deployment), and project management. Here's how you can use GitLab for hosting your code and managing the deployment process:

1. Setting Up GitLab for Hosting

Creating a Repository

Sign Up/In: Create an account on GitLab or sign in to your existing account.

New Project: Click on "New Project" and choose between creating a new repository from scratch, importing a project, or using a template.

Repository Details: Provide a name, description, visibility level (public, internal, or private), and initialize with a README if desired.

Cloning the Repository

Clone URL: Navigate to your newly created repository and copy the HTTPS or SSH clone URL.

Clone Locally

2. Setting Up CI/CD Pipeline

GitLab CI/CD is used to automate the process of testing, building, and deploying your code. This involves creating a .gitlab-ci.yml file in your repository.

Example .gitlab-ci.yml for a Go Application

3. Setting Up Deployment

Deploying your application involves moving the built application to your server and starting it. GitLab CI/CD can handle this through the deployment stage in your .gitlab-ci.yml.

SSH Keys for Deployment

Generate SSH Key Pair

ssh-keygen -t rsa -b 4096 -C your_email@example.com

Add SSH Key to Server:

Copy the public key (~/.ssh/id_rsa.pub) and add it to the ~/.ssh/authorized_keys file on your server.

Add SSH Key to GitLab:

Go to your GitLab project settings -> CI/CD -> Variables.

Add a new variable SSH_PRIVATE_KEY with the content of your private key (~/.ssh/id_rsa).

Securing Deployment

Ensure you follow security best practices:

- **Use strong passwords and key-based authentication.**
- **Limit the scope and permissions of the deployment keys.**
- **Use environment variables for sensitive data.**

4. Monitoring and Maintenance

After deploying your application, you should set up monitoring and maintenance routines.

Monitoring Tools

Prometheus/Grafana: For monitoring metrics and setting up dashboards.

ELK Stack: Elasticsearch, Logstash, and Kibana for log management.

AlertManager: For setting up alerts based on predefined conditions.

Scheduled Tasks

Cron Jobs: Use cron jobs to schedule regular maintenance tasks like backups, clean-ups, and updates.

5. Rolling Back Deployments

In case of deployment failures, ensure you have a rollback strategy.

Rollback Strategies

i. **Versioned Releases:** Maintain previous versions of your releases to roll back easily.
ii. **Automated Rollback:** Implement automated rollback scripts in your CI/CD pipeline.

Example Rollback Script in .gitlab-ci.yml