

Software Documentation

Software Name: HUNGRY

Version: 1.0.0

Date: 📅 Apr 30, 2024

TABLE OF CONTENTS

- Software Summary
 - Requirements
 - How-to set up
 - Step 1
 - Step 2
 - Step 3
 - Step 4
 - Step 5
 - Step 6
 - Step 7
 - Step 8
-

Software Summary

A sophisticated culinary marketplace where vendors showcase and sell their delectable creations, providing customers with the opportunity to indulge in a wide array of culinary delights.

FEATURES

- Robust and encrypted authentication ensures secure access.
- Sellers can add, modify prices, or remove items from their menu.
- Sellers can conveniently view feedback and order details provided by customers.
- Users can add, adjust quantities, or remove items from their orders and shopping carts.

- Users can provide feedback to sellers and access their order history.
- Users can make payments seamlessly through the Stripe payment gateway.

BUG FIXES

- Image loading bug

Requirements

- Any Code Editor
- Node.js
- MySQL
- Git

How-to set up

Step 1

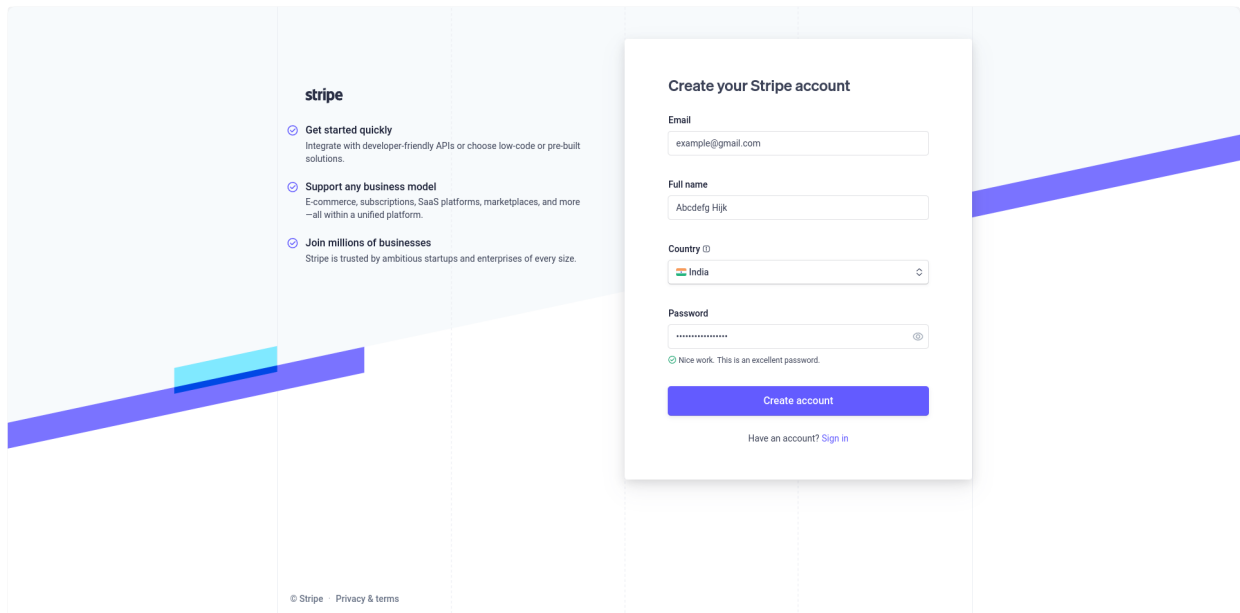
Create a database named **food_ordering_site** in your MySQL.

```
MariaDB [(none)]> create database food_ordering_site
-> ;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| filestore |
| food_ordering_site |
| information_schema |
| mysql |
| performance_schema |
| sys |
| test |
+-----+
7 rows in set (0.001 sec)
```

Step 2

Create an account in Stripe for payments.



Step 3

Clone into the repository using git.

```
[@ programs]$ git clone https://github.com/Code-Challengers-hacks/Hungry.git
Cloning into 'Hungry'...
remote: Enumerating objects: 142, done.
remote: Counting objects: 100% (142/142), done.
remote: Compressing objects: 100% (95/95), done.
remote: Total 142 (delta 44), reused 122 (delta 38), pack-reused 0
Receiving objects: 100% (142/142), 629.68 KiB | 763.00 KiB/s, done.
Resolving deltas: 100% (44/44), done.
```

Step 4

Fill in the .env file according to the following specifications.

`MYSQL_USERNAME` : Enter your MySQL username

`MYSQL_PASSWORD` : Enter your MySQL password

`STRIPE_PUBLIC_KEY` : Get your stripe public key from Stripe

`STRIPE_PRIVATE_KEY` : Get your stripe public key from Stripe

`SECRET` : Type any random string of characters for JWT

Step 5

Cd into the backend directory and run npm install

```
1 cd backend
2 npm install
```

```

[@ Hungry]$ cd backend/
[@ backend]$ npm install

added 164 packages, and audited 165 packages in 7s

19 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 10.5.0 -> 10.6.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.6.0
npm notice Run npm install -g npm@10.6.0 to update!
npm notice

```

Step 6

Run the backend development server.

```
1 npm run dev
```

```

[@ backend]$ npm run dev
> backend@1.0.0 dev
> node server.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting node server.js
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Sellers' AND TABLE_SCHEMA = 'food_ordering_system'
Executing (default): CREATE TABLE IF NOT EXISTS `Sellers` (`id` INTEGER NOT NULL auto_increment , `name` VARCHAR(255), `email` TEXT, `password` TEXT, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `Sellers` FROM `food_ordering_system`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'MenuItems' AND TABLE_SCHEMA = 'food_ordering_system'
Executing (default): CREATE TABLE IF NOT EXISTS `MenuItems` (`id` INTEGER NOT NULL auto_increment , `name` VARCHAR(255), `price` FLOAT, `description` TEXT, `image` VARCHAR(255), `sellerName` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `deletedAt` DATETIME, `SellerId` INTEGER, PRIMARY KEY (`id`), FOREIGN KEY (`SellerId`) REFERENCES `Sellers` (`id`) ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `MenuItems` FROM `food_ordering_system`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = `Users` AND TABLE_SCHEMA = 'food_ordering_system'
Executing (default): CREATE TABLE IF NOT EXISTS `Users` (`id` INTEGER NOT NULL auto_increment , `name` VARCHAR(255), `email` TEXT, `password` TEXT, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `Users` FROM `food_ordering_system`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Feedbacks' AND TABLE_SCHEMA = 'food_ordering_system'
Executing (default): CREATE TABLE IF NOT EXISTS `Feedbacks` (`id` INTEGER NOT NULL auto_increment , `feedback` TEXT, `username` VARCHAR(255), `adminName` VARCHAR(255), `rating` FLOAT, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `UserId` INTEGER, PRIMARY KEY (`id`), FOREIGN KEY (`UserId`) REFERENCES `Users` (`id`) ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `Feedbacks` FROM `food_ordering_system`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Carts' AND TABLE_SCHEMA = 'food_ordering_system'
Executing (default): CREATE TABLE IF NOT EXISTS `Carts` (`id` INTEGER NOT NULL auto_increment , `total` FLOAT, `username` VARCHAR(255), `adminName` VARCHAR(255), `isPlaced` INTEGER DEFAULT 0, `isFinished` INTEGER DEFAULT 0, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `deletedAt` DATETIME, `UserId` INTEGER, `SellerId` INTEGER, PRIMARY KEY (`id`), FOREIGN KEY (`UserId`) REFERENCES `Users` (`id`) ON DELETE SET NULL ON UPDATE CASCADE, FOREIGN KEY (`SellerId`) REFERENCES `Sellers` (`id`) ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `Carts` FROM `food_ordering_system`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = `UserItems` AND TABLE_SCHEMA = 'food_ordering_system'
Executing (default): CREATE TABLE IF NOT EXISTS `UserItems` (`id` INTEGER NOT NULL auto_increment , `name` VARCHAR(255), `price` FLOAT, `quantity` INTEGER, `adminName` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `deletedAt` DATETIME, `CartId` INTEGER, PRIMARY KEY (`id`), FOREIGN KEY (`CartId`) REFERENCES `Carts` (`id`) ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `UserItems` FROM `food_ordering_system`
Database synchronized successfully.
server is running on port 4000

```

Step 7

Cd into the backend directory and run npm install

```

1 cd frontend
2 npm install

```

```
[@ Hungry]$ cd frontend/  
[@ frontend]$ npm install  
  
added 735 packages, and audited 736 packages in 2m  
  
180 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

Step 8

Run the frontend development server

```
1  npm run dev
```

```
[@ frontend]$ npm run dev  
  
> food_ordering_site@0.1.0 dev  
> next dev  
  
▲ Next.js 14.2.3  
- Local:      http://localhost:3000  
  
✓ Starting...  
✓ Ready in 21.6s  
  ○ Compiling / ...  
✓ Compiled / in 41.8s (2289 modules)  
AuthContext state: { name: null, mode: null }  
MenuContext state: { menu: [] }  
CartsContext state: { cart: [] }  
FeedbackContext state: { feedback: [] }  
OrderedContext state: { ordered: [] }  
SellerOrdersContext state: { sellerOrders: [] }  
AuthContext state: { name: null, mode: null }  
MenuContext state: { menu: [] }  
CartsContext state: { cart: [] }  
FeedbackContext state: { feedback: [] }  
OrderedContext state: { ordered: [] }  
SellerOrdersContext state: { sellerOrders: [] }
```

The backend and frontend are available at <http://localhost:4000> and <http://localhost:3000> respectively.

Changes made in the code are reflected automatically in the development servers.