CMPS 350 Web Development

Project Phase2- Get Glammed e-commerce platform.

<https://github.com/lama-la2103717/WebProject>

Team members:

Ekram Omer 202105434\_ Contribution:33.3%

Lujayn Alamir 201901343\_ Contribution:33.3%

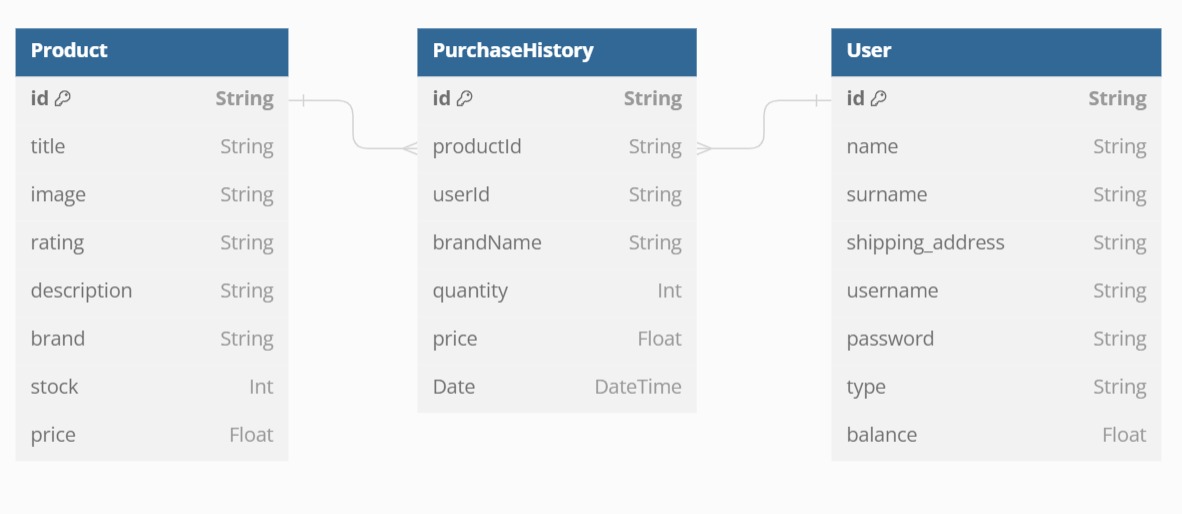
Lama Albanna 202103717\_ Contribution:33.3%

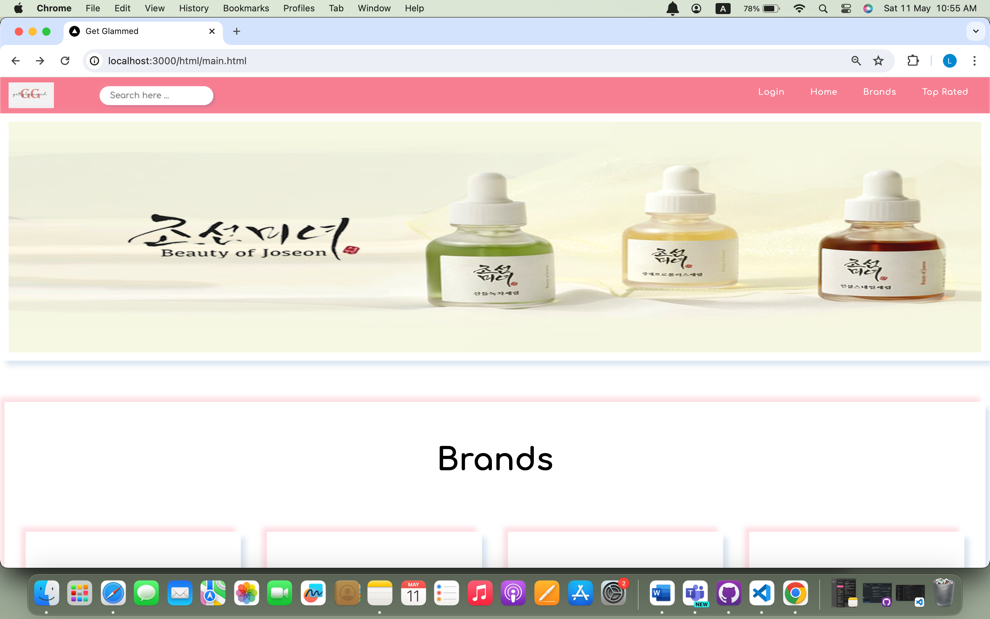
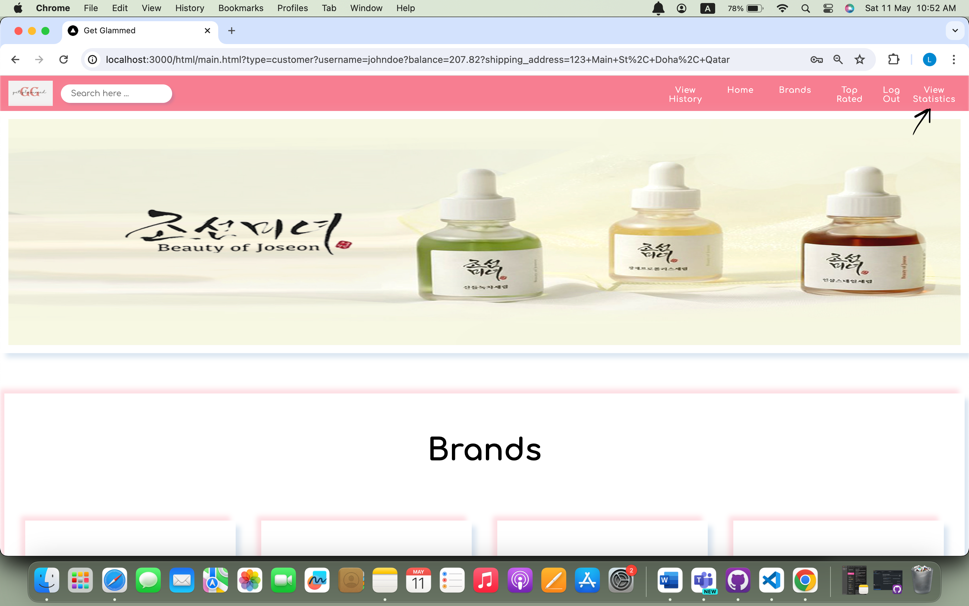
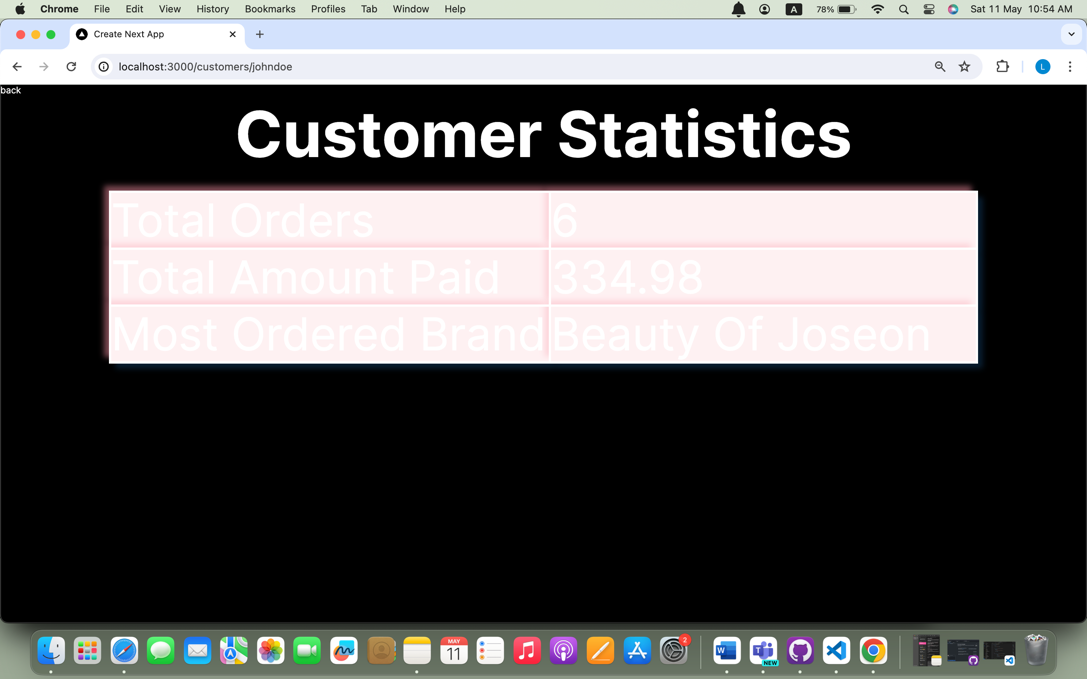
Instructor: Dr. Mucahid Kutlu

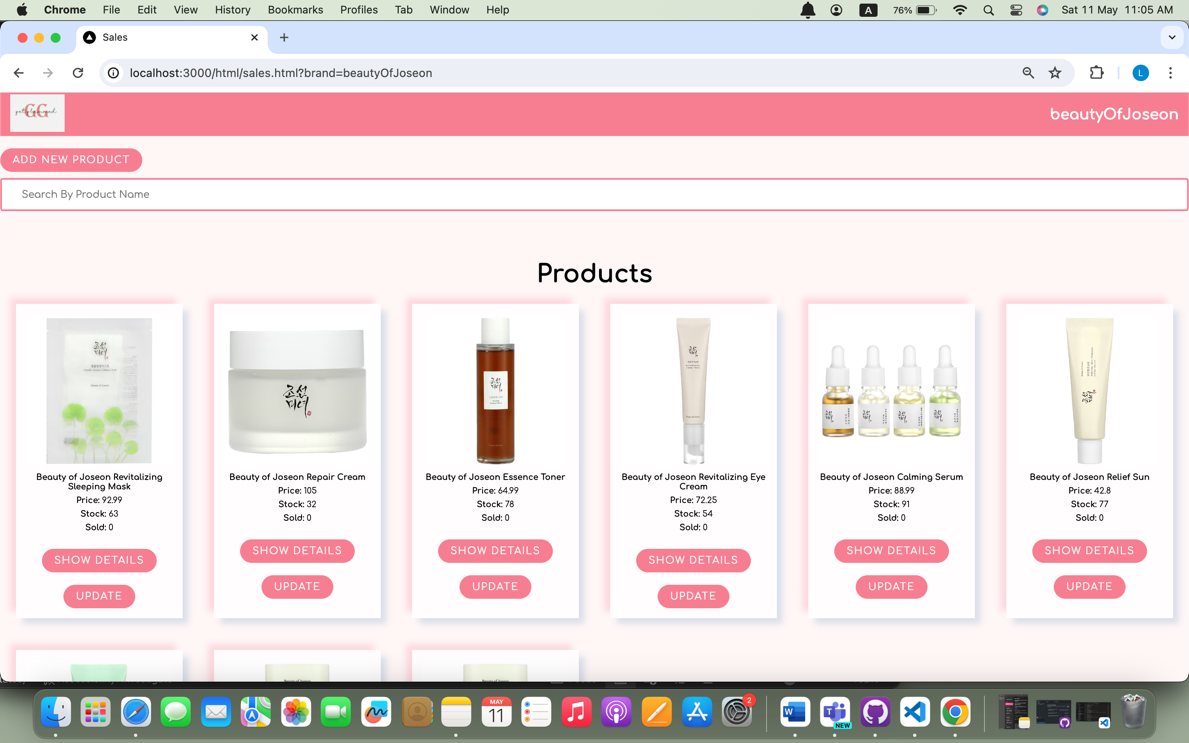
Spring 2024

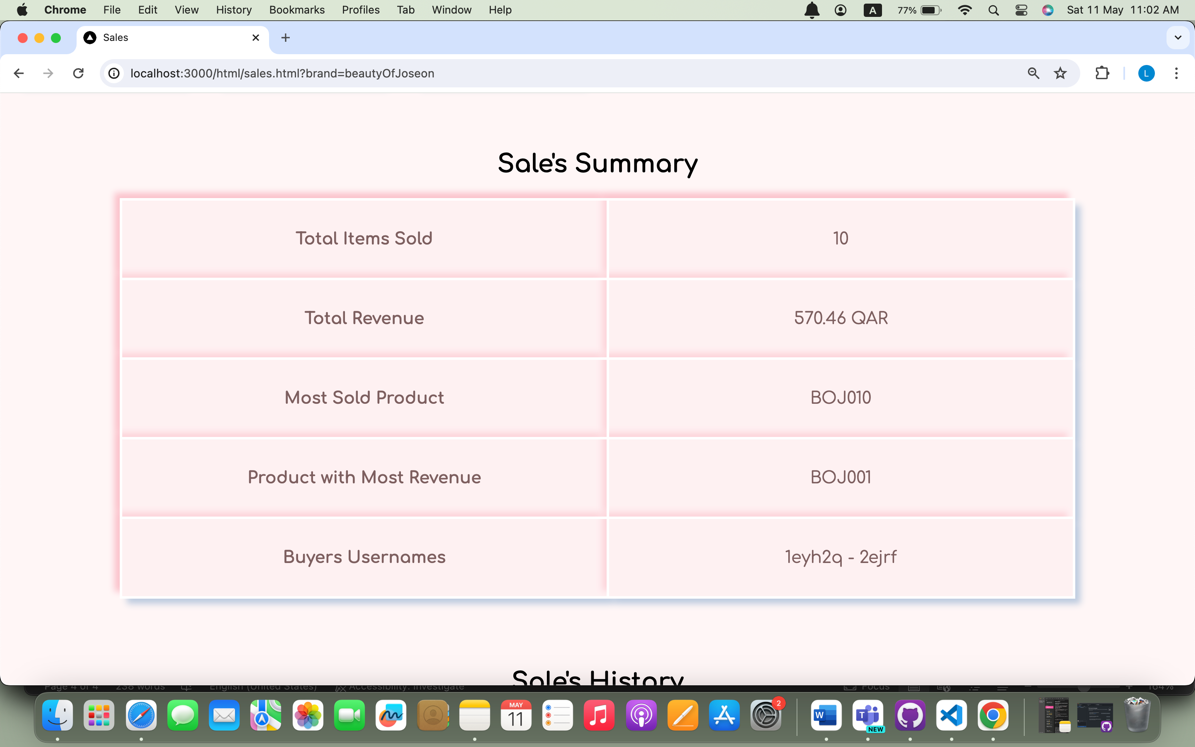
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **%** | **Functionali ty**\* | **Quality of the implementation** | **Grade** |
| Design and implement the Data Model. | 10 | 10 |  |  |
| Init DB: populate the database with the data from the json files in seed.js | 5 | 5 |  |  |
| APIs and Repository Implementation to read/write data from the database | 25 | 25 |  |  |
| Statistics use-case with NextJS | 40 | 40 |  |  |
| **Documentation**   * Data Model diagram. * UI Design with screenshots and description. * Database queries. * Conducted tests and evidence. * **Contribution** of each team member [-10pts if not done] | 20 | 20 |  |  |
| **Total** | 100 |  |  |  |
| Bonus - successful deployment of the app and the Database to a cloud hosting service such as <https://vercel.com/> | 5 |  |  |  |
| Copying and/or plagiarism or not being able to explain or answer questions about the implementation. | 0 |  |  |  |

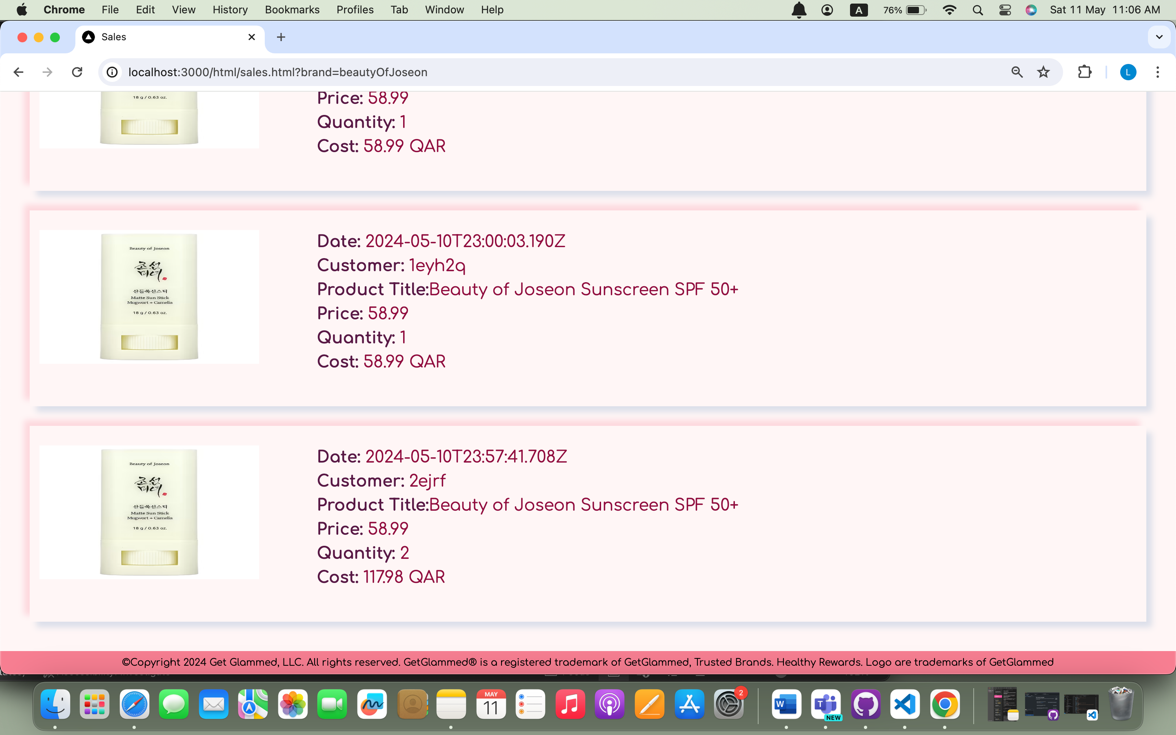
**Grading Rubric**

1. **Data Model diagram:**
2. **UI Design with screenshots and description:**

* Main page when no one is logged in:
* If the customer login in successfully it will direct the customer to the main page, where the customer can view statistics:
* Upon clicking on view statistics, the customer can view his own statistics:
* A screenshot of a computer

  Description automatically generatedIf the seller login in successfully it will direct the seller to the main page, where the seller can view statistics upon clicking view sales:
* Upon clicking on view sales, the seller will be directed to a page where he can view his products, statistics, and the sales:





1. **Database queries:**

* the database queries are added in folder app/repo, file name: ecomRepo.js:
* import { PrismaClient } from '@prisma/client'
* const prisma = new PrismaClient()
* class EcomRepo{
* async getUsers(){
* try {
* return prisma.user.findMany()
* } catch (error) {
* return { error: error.message }
* }
* }
* async getproductsbyBrand(brandName){
* try {
* return prisma.product.findMany({
* where: {brand: {contains: brandName.match(/[A-Z][a-z]+|[0-9]+/g).join(" ")}}
* ,include:{prodPurchases:true}
* })
* } catch (error) {
* return {error: error.message}
* }
* }
* async getProducts(){
* try {
* return prisma.product.findMany()
* } catch (error) {
* return { error: error.message }
* }
* }
* async getUser(userId){
* try{
* return prisma.user.findUnique(
* { where: { userId } }
* )
* } catch (error) {
* return { error: error.message }
* }
* }
* async getPurchases(userId) {
* try {
* return prisma.purchaseHistory.findMany(
* { where: {userId} }
* )
* } catch (error) {
* return { error: error.message }
* }
* }
* async updateUser(userId, user) {
* try {
* return await prisma.user.update({
* where: { userId },
* data: user
* });
* } catch (error) {
* return { error: error.message };
* }
* }
* async addPurchase(userId, purchaseHistory) {
* try {
* purchaseHistory.price = parseFloat(purchaseHistory.price);
* purchaseHistory.quantity = parseInt(purchaseHistory.quantity);
* const totalPurchasePrice = purchaseHistory.price \* purchaseHistory.quantity;
* const user = await this.getUser(userId);
* if (user.balance < totalPurchasePrice) {
* return { error: "Insufficient Balance" };
* }
* user.balance -= totalPurchasePrice;
* await this.updateUser(userId, user);
* const createdPurchase = await prisma.purchaseHistory.create({ data: purchaseHistory });
* return createdPurchase;
* } catch (error) {
* return { error: error.message };
* }
* }
* async updateProduct(id, product){
* try {
* return prisma.product.update(
* {
* where: {id},
* data: product
* }
* )
* } catch (error) {
* return {error: error.message}
* }
* }
* async addProduct(product){
* try {
* return prisma.product.create({
* data: product
* })
* } catch (error) {
* return {error: error.message}
* }
* }
* async getProductById(id){
* try {
* return prisma.product.findUnique({
* where: {id},
* include: {prodPurchases:true}
* })
* } catch (error) {
* return {error: error.message}
* }
* }
* async getSaleHistory(name){
* try {
* return prisma.purchaseHistory.findMany({
* where: {brandName: {contains: name.match(/[A-Z][a-z]+|[0-9]+/g).join(" ")}},
* include:{Product: true, User : true}
* })
* } catch (error) {
* return {error: error.message}
* }
* }
* async getSaleHistoryById(id){
* return prisma.product.findMany({
* where: {productId: id},
* include:{prodPurchases:true}
* })
* } catch (error) {
* return {error: error.message}
* }
* async getCustomerHistoryByName(name){
* return prisma.user.findFirst({
* where: {username: name},
* include:{userPurchases:true}
* })
* } catch (error) {
* return {error: error.message}
* }
* }
* export default new EcomRepo()

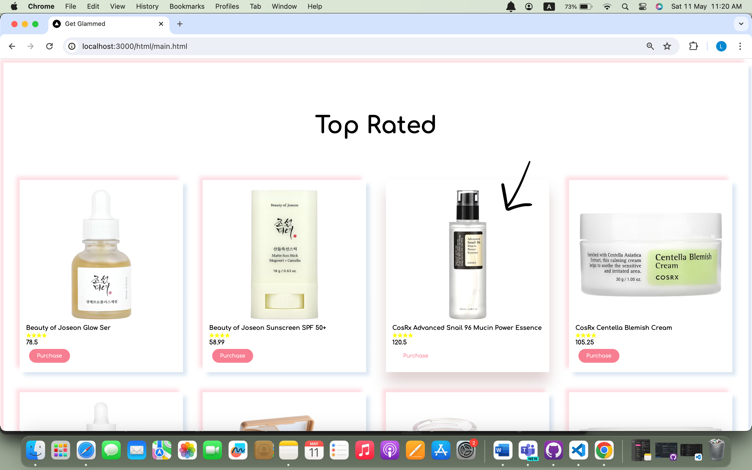
1. **Conducted tests and evidence:**

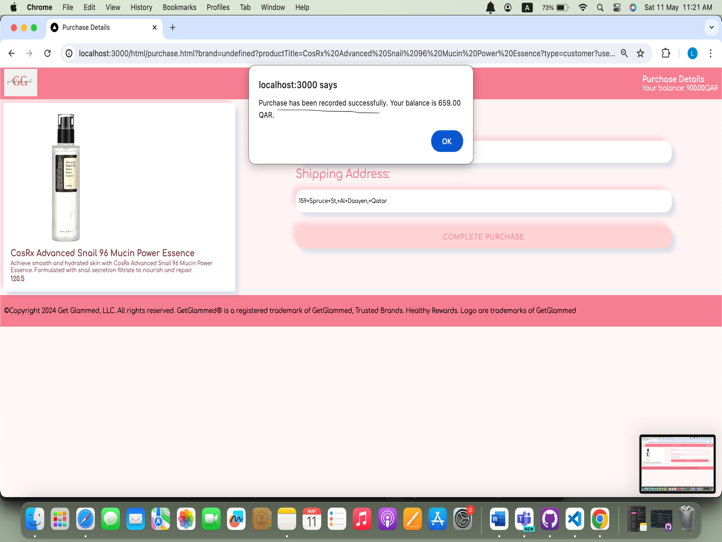
**Customer statistics:**

* A screenshot of a computer

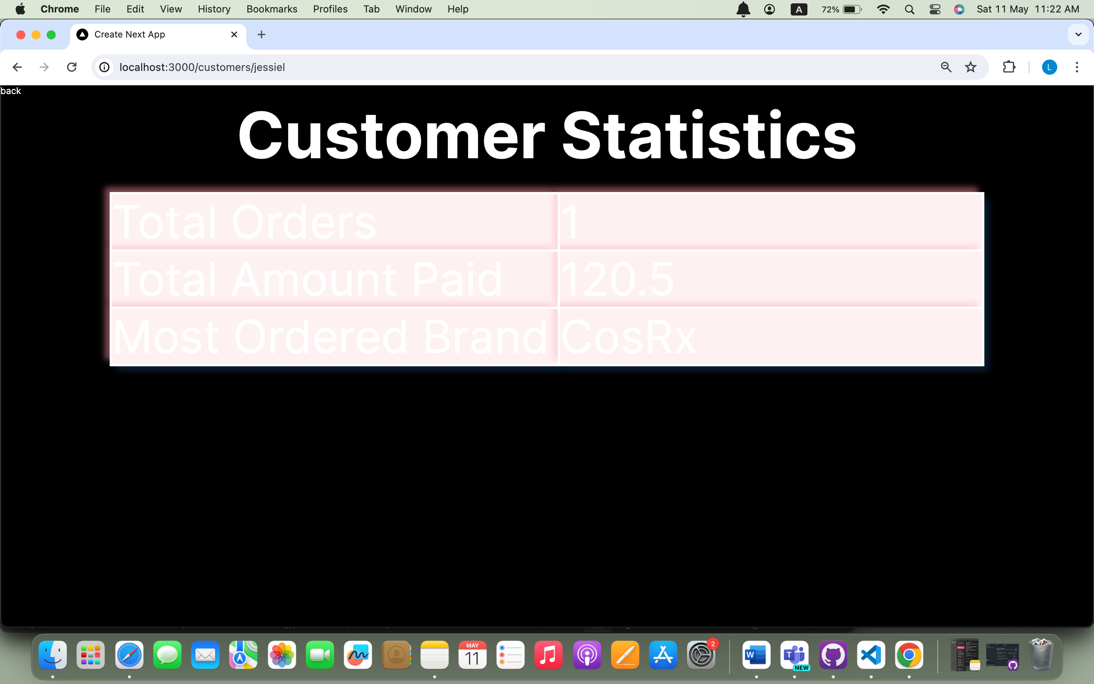
  Description automatically generatedA screenshot of a computer

  Description automatically generatedWhen a customer logged in but never buy product, upon clicking on view statistics, the statistic will be empty:
* When the same customer purchases an item, upon clicking on purchase the customer can add the quantity of the product and click on complete purchase.





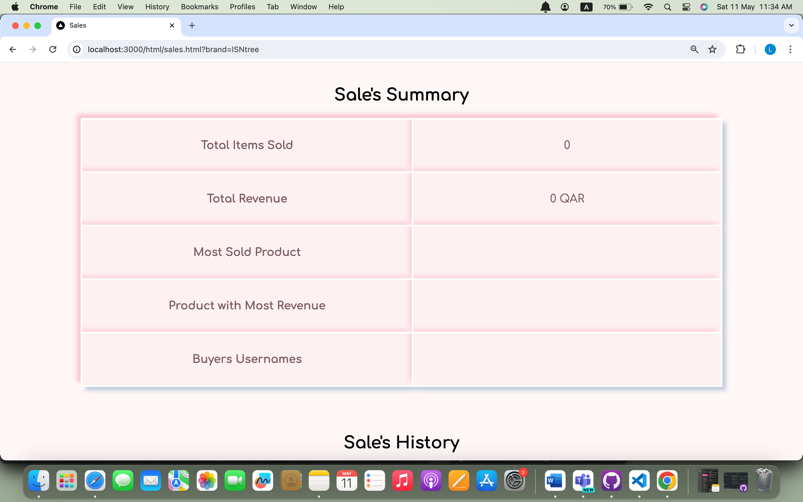
* When the customer now clicks on view statistics it will be updated:

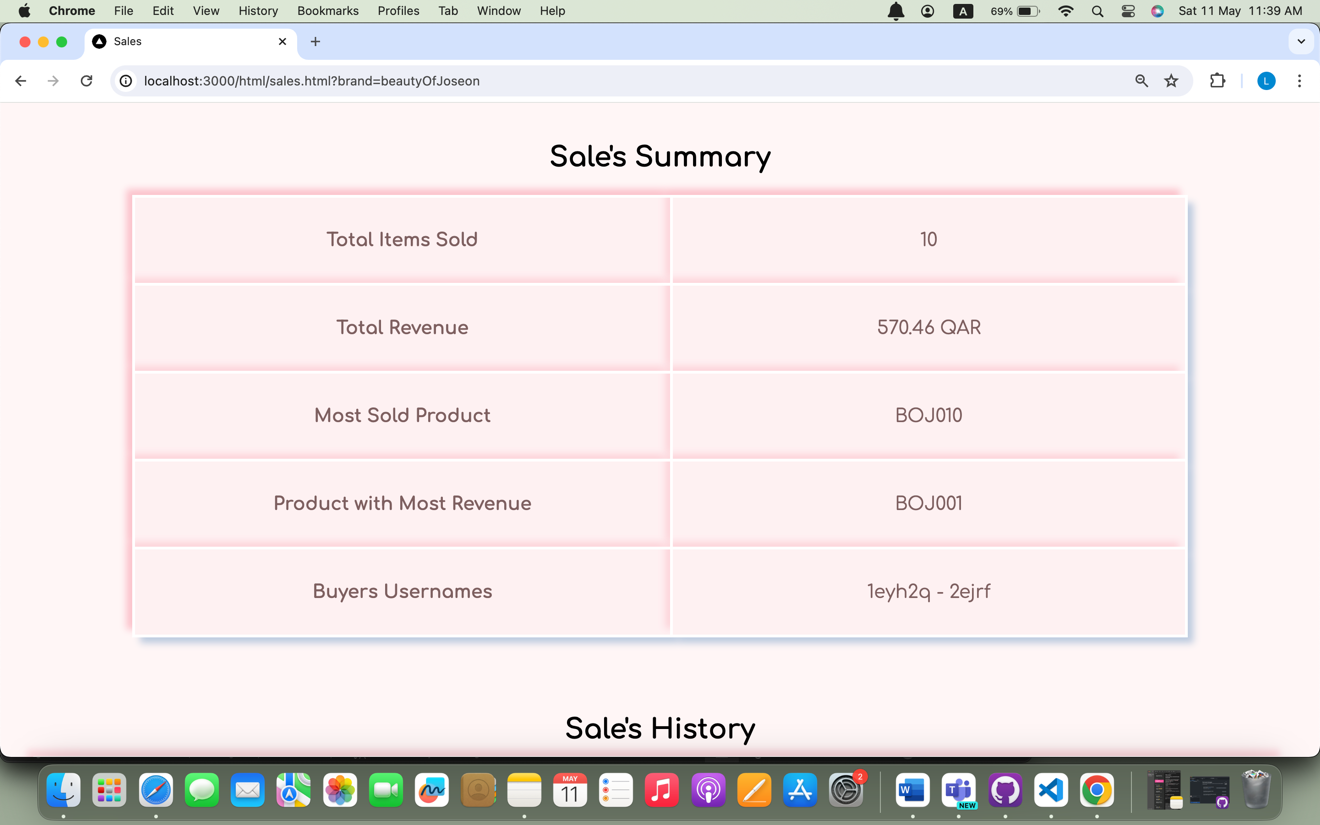


**Seller statistics:**

* When a seller logged in, but no one buy from their products, upon clicking on view sales, the statistic will be empty:

A screenshot of a computer

Description automatically generated

* If customers purchase from the seller products, when the seller now click on view sales, the statistics will be updated: