

Deliverable 5 Report

Team Loyalty

a. Updated Phase 1 Requirements Status

**Multiple requirements remaining from Phase 1 related to profile management, product display/search, and checkout functionality have transitioned to FINISHED status.*

Requirement Number	Status	Section Name	UI	API	Description
3.1.7.1.6.	DESCOPE	Birthday (optional field, month, and day)	✓		Enter account information.
3.1.7.1.7.	DESCOPE	Social login - Google, other [Conditional]		✓	Enter account information.
3.1.2.	FINISHED	Site visitors may browse products by category.		✓	Filter video games by genre. Displays item name in a card on-screen.
3.1.5.1.	FINISHED	Display product name.	✓		Displays item image in a card on-screen.
3.1.5.4.	FINISHED	Display product image.	✓		Displays item price in a card on-screen.
3.1.5.6.	FINISHED	Display product price (in Dollars).	✓		Shopping cart icon on the navigation bar.
3.1.6.1.	FINISHED	Site visitor can click icon to display.	✓		Project core functionality. Front-end of the project. Web store will sell Atari games.
3.1	FINISHED	eCommerce Storefront [Essential]	✓		Stylish CSS on the Navigation bar and pages.
3.1.1.	FINISHED	Web-based shopping site	✓	✓	Not chosen yet.
3.1.1.1.	FINISHED	Banner graphic	✓		Filter video games by name and price.
3.1.1.2.	FINISHED	Color theme	✓		Search video games by name. Displays item description in a card on-screen.
3.1.3.	DESCOPE	Site visitors may sort products by name or price.		✓	Displays item genre in a card on-screen.
3.1.4.	FINISHED	Site visitors may search for products by name.		✓	Displays unique ID in a card on-screen.
3.1.5.2.	FINISHED	Display product description.	✓		
3.1.5.3.	FINISHED	Display product category.	✓		
3.1.5.5.	FINISHED	Display product SKU number.	✓		
3.1.5.7.	FINISHED	Quantity entry and Add-to-Cart button.	✓		Send items to shopping cart. Shopping cart where selected items are stored.
3.1.6.	FINISHED	Shopping cart	✓	✓	

3.1.7.	FINISHED	Customer Profile	✓	✓	Page containing customer information.
3.1.7.1.	FINISHED	Site visitors can register and create a customer profile.		✓	Create an account.
3.1.7.1.1.	FINISHED	First and Last name (required fields)	✓		Enter account information.
3.1.7.1.2.	FINISHED	Email address (required field, must be unique)	✓		Enter account information.
3.1.7.1.3.	FINISHED	Secure password (required field)	✓		Enter account information.
3.1.7.2.	FINISHED	Customers can login with their registered user name and password.		✓	Customer can login with user/pass.
3.1.7.3.	DE-SCOPED	Customers can log out.	✓		Customer can logout.
3.1.7.4.	DE-SCOPED	Customer logged-in status is visible on the web site banner.	✓		Customer sees updated page when logged in.
3.1.1.3.	DE-SCOPED	Mobile responsive design [Conditional]	✓		Integrate mobile functionality.
3.1.5.	FINISHED	Product detail pages	✓		
3.1.6.2.	FINISHED	Display items added to cart.	✓		
3.1.6.2.1.	FINISHED	Display item quantity.	✓		
3.1.6.2.2.	FINISHED	Display item name.	✓		
3.1.6.2.3.	FINISHED	Display item (small) image.	✓		
3.1.6.2.4.	FINISHED	Display item price (in Dollars).	✓		
3.1.6.3.	FINISHED	Display calculated shopping cart subtotal.	✓		
3.1.6.4.	FINISHED	Calculate sales taxes (Texas 8.25%)	✓	✓	
3.1.6.5.	FINISHED	Calculate shipping (flat rate)	✓	✓	
3.1.6.6.	FINISHED	Calculate purchase total.	✓	✓	
3.1.6.7.	DE-SCOPED	Site visitor can change cart item quantity.		✓	
3.1.6.7.1.	DE-SCOPED	Update subtotal (in Dollars) when quantity changed.		✓	
3.1.6.7.2.	DE-SCOPED	Update sales taxes (Texas 8.25%)		✓	
3.1.6.7.3.	FINISHED	Update purchase total.	✓	✓	
3.1.6.8.	DE-SCOPED	Site visitor can remove item from cart.	✓	✓	
3.1.6.8.1.	DE-SCOPED	Update subtotal (in Dollars) when item removed.		✓	
3.1.6.8.2.	DE-SCOPED	Update sales taxes (Texas 8.25%)		✓	
3.1.6.8.3.	DE-SCOPED	Update purchase total.		✓	
3.1.6.9.	FINISHED	Proceed to Checkout	✓	✓	
3.1.6.9.1.	FINISHED	Site visitor shall be prompted to login if necessary.	✓	✓	

3.1.6.9.2.	FINISHED	Logged-in customer details are pre-populated:	✓	
3.1.6.9.2.1.	FINISHED	Populate name.	✓	
3.1.6.9.2.2.	FINISHED	Populate shipping address.	✓	
3.1.6.9.2.3.	DE-SCOPED	Populate email.	✓	
3.1.6.9.2.4.	FINISHED	Populate phone.	✓	
3.1.6.9.3.	FINISHED	Logged-in customer may update shipping details for order.	✓	✓
3.1.6.9.3.1.	FINISHED	Update name	✓	✓
3.1.6.9.3.2.	FINISHED	Update shipping address	✓	✓
3.1.6.9.3.3.	DE-SCOPED	Update email contact	✓	
3.1.6.9.3.4.	DE-SCOPED	Update phone	✓	
3.1.6.9.4.	FINISHED	Logged-in customer can enter payment details.	✓	
3.1.6.9.4.1.	FINISHED	Enter card type.	✓	
3.1.6.9.4.1.1.	DE-SCOPED	Multiple payment tender types supported.		✓
3.1.6.9.4.2.	FINISHED	Enter card number.	✓	
3.1.6.9.4.3.	DE-SCOPED	Enter card expiration.	✓	
3.1.6.9.4.4.	FINISHED	Enter name on card.	✓	
3.1.6.9.4.5.	FINISHED	Enter payment zip code.	✓	
3.1.6.9.5.	FINISHED	Logged-in customer may submit order.		✓
3.1.6.9.5.1.	DE-SCOPED	Validate shipping details complete.		✓
3.1.6.9.5.2.	DE-SCOPED	Validate payment details complete.	✓	
3.1.6.9.5.3.	FINISHED	Submit transaction details for processing.	✓	✓
3.1.6.9.5.4.	FINISHED	Authorize transaction payment.	✓	✓
3.1.6.9.5.4.1.	DE-SCOPED	Return message on failure (e.g., Payment declined)		✓
3.1.6.9.6.	FINISHED	System will record authorized transaction to customer account.		✓
3.1.6.9.6.1.	FINISHED	Transaction Header data is recorded.		✓
3.1.6.9.6.1.1.	FINISHED	Transaction Date		✓
3.1.6.9.6.1.2.	DE-SCOPED	Transaction Location (Web store)		✓
3.1.6.9.6.1.3.	FINISHED	Transaction Number (System generated)		✓
3.1.6.9.6.1.4.	FINISHED	Transaction Dollar Net Total		✓
3.1.6.9.6.1.5.	FINISHED	Transaction Dollar Gross Total		✓
3.1.6.9.6.2.	FINISHED	Transaction Line Detail data is recorded in Database.		✓
3.1.6.9.6.2.1.	FINISHED	Line number (DB)		✓

3.1.6.9.6.2.2.	FINISHED	SKU (DB)	✓	
3.1.6.9.6.2.3.	FINISHED	Quantity (DB)	✓	
3.1.6.9.6.2.4.	FINISHED	Unit price (DB)	✓	
3.1.6.9.6.2.5.	FINISHED	Net amount (DB)	✓	
3.1.6.9.6.2.6.	FINISHED	Tax amount (DB)	✓	
3.1.6.9.6.2.7.	FINISHED	Gross amount (DB)	✓	
3.1.6.9.6.2.8.	FINISHED	Discount amount applied.	✓	✓
3.1.6.9.6.3.	FINISHED	Transaction Tender data is recorded (single tender).	✓	
3.1.6.9.6.3.1.	FINISHED	Payment type	✓	✓
3.1.6.9.6.3.2.	FINISHED	Tender code	✓	✓
3.1.6.9.6.3.3.	FINISHED	Payment details	✓	✓
3.1.6.9.6.3.4.	FINISHED	Tender amount	✓	✓
3.1.6.9.6.4.	FINISHED	Charge payment method.	✓	
3.1.7.1.4.	DE-SCOPED	Phone number (optional field)	✓	Enter account information.
3.1.7.1.5.	FINISHED	Physical address - street, city, state, zip (optional, must be complete)	✓	Enter account information.
3.1.7.5.	FINISHED	Logged-in customer can update their profile data.	✓	✓
3.1.7.5.1.	FINISHED	Update first and last name.	✓	✓
3.1.7.5.2.	FINISHED	Update email address (must be unique).	✓	✓
3.1.7.5.3.	DE-SCOPED	Change secure password.	✓	
3.1.7.5.4.	FINISHED	Update phone number.	✓	✓
3.1.7.5.5.	FINISHED	Update physical address.	✓	✓
3.1.7.5.6.	DE-SCOPED	Update birthday.	✓	
3.1.7.6.	FINISHED	Customer can view purchase history.	✓	✓
3.1.7.6.1.	FINISHED	Summary list of transactions including data, transaction number, total dollar amount, status.	✓	✓

b. Phase 2 Requirements Status

Requirement Number	Status	Description	Page	UI	API
3.2	FINISHED	Loyalty Stack Point Management System [Essential]			
3.2.1.	FINISHED	Loyalty Enrollment.	Sign-Up		✓
3.2.1.1.	FINISHED	Customer Profile enhancement - new and existing registered customers can join the E-commerce Loyalty program.	Sign-Up		
3.2.1.1.1.	FINISHED	Checkbox on Customer Profile create/update page to join.	Sign-Up	✓	✓
3.2.1.1.2.	FINISHED	Accept T&C's	Sign-Up	✓	✓
3.2.1.1.3.	FINISHED	System will capture customer's active enrollment status and enrollment date.	Sign-Up		✓
3.2.1.1.4.	FINISHED	System shall issue a unique loyalty card/account number to newly joined loyalty customer.	Sign-Up		✓
3.2.1.1.5.	FINISHED	Customer account shall be initialized with a 0-point balance.	Sign-Up		✓
3.2.1.1.6.	DESCOPED	Customer will receive a Welcome bonus for joining loyalty [Optional]	Sign-Up		
3.2.2.	FINISHED	Loyalty Points Earn	Checkout		✓
3.2.2.1.	FINISHED	Loyalty Customers earn in a single currency: Points.	Checkout		✓
3.2.2.2.	FINISHED	Points do not expire while the customer is an active member of the loyalty program.	NA		✓
3.2.2.3.	FINISHED	Loyalty Customers can earn a program-defined Base points	Checkout		✓

		per dollar on eligible spend.			
3.2.2.4.	FINISHED	Base points are earned for net spend on eligible products, excluding taxes, shipping, or excluded products [rate TBD].	Checkout		✓
3.2.2.5.	FINISHED	Bonus points can be earned on business-defined promotional basis for spend on specific products, product categories or spend threshold.	Checkout		✓
3.2.2.6.	FINISHED	Bonus points can be additional points per dollar, fixed amount of bonus points, or point multipliers (2X, 3X, etc.).	Checkout		✓
3.2.2.7.	FINISHED	Points will be calculated by a rules-based points engine.	Checkout		✓
3.2.2.7.1.	FINISHED	Points can be earned on customer purchase transaction events.	Checkout		✓
3.2.2.7.1.1.	FINISHED	Rules are entered and stored in the system and dynamically executed by the points engine when evaluating customer transactions.	Checkout		✓
3.2.2.7.1.1.1.	FINISHED	Rule shall capture point eligibility criteria.	Checkout		✓
3.2.2.7.1.1.2.	FINISHED	Rule shall capture point outcome calculation formula.	Checkout		✓
3.2.2.7.1.1.3.	FINISHED	Rules shall be active for specific date ranges.	Checkout		✓
3.2.2.7.1.1.4.	FINISHED	New rules can be entered into the system after it is live.	Checkout		✓
3.2.2.7.1.2.	FINISHED	Authorized transactions for loyalty customers are	Checkout		✓

		evaluated in real-time by the points engine and point outcomes are issued to customer account immediately.			
3.2.2.7.2.	DESCOPED	Points may be issued upon customer profile creation/registration in Loyalty program [Optional]	Sign-Up		

c. Phase 3 Requirement Status

Section Number	Status	Requirement Description	Page	UI	API
3.1.1.	FINISHED	Loyalty Redemption	Checkout	✓	✓
3.1.1.1.	FINISHED	Shopping cart enhancement – Loyalty customers' available points balance is displayed during checkout.	Checkout	✓	✓
3.1.1.2.	FINISHED	Points redeemable for a single reward type: Purchase Discount.	Checkout	✓	✓
3.1.1.3.		Loyalty customer must have a minimum threshold points balance to redeem [TBD]			
3.1.1.4.	FINISHED	Points convert to dollars discount at a system-defined rate [100 point per dollar]	Checkout	✓	✓
3.1.1.5.	FINISHED	Shopping cart enhancement – available points can be applied to basket at checkout as \$X dollars off.	Checkout	✓	✓
3.1.1.5.1.	FINISHED	Discount is prorated and applied to net amount of individual basket items.	Checkout	✓	✓
3.1.1.6.		Customer's earned points are redeemed and converted to discount rewards in FIFO			

		sequence (oldest points first).			
3.1.1.7.	FINISHED	All redemptions are final, no points returned to balance, no residual cash value.	Checkout	✓	✓
3.1.1.8.	FINISHED	The conversion of points amount to reward of dollar amount is recorded in the system.	Checkout	✓	✓
3.1.1.9.		Transaction processing enhancement – Points redemption reward code/number and dollar amount recorded with transaction.			
3.1.2.	FINISHED	Loyalty Status			
3.1.2.1.	DESCOPE	E-commerce web site enhancement – Logged in Loyalty customers will see Loyalty status indicator and loyalty card number on web site banner.			
3.1.2.2.	FINISHED	E-commerce web site enhancement - Loyalty status page/dashboard.	Loyalty Status	✓	✓
3.1.2.2.1.		Display Loyalty status.			
3.1.2.2.2.	FINISHED	Display Loyalty card/account number.	Loyalty Status	✓	✓
3.1.2.2.3.	FINISHED	Display points balance.	Loyalty Status	✓	✓
3.1.2.2.4.	FINISHED	Display recent points history.	Loyalty Status	✓	✓
3.1.2.2.4.1.	FINISHED	Display points earned records, including points amount, date and points bonus rule code/description.	Loyalty Status	✓	✓
3.1.2.2.4.2.	FINISHED	Display point redemption records, including points amount, date, transaction number.	Loyalty Status	✓	✓

3.1.2.3.	FINISHED	E-commerce web site enhancement – Loyalty transaction history	Profile	✓	✓
3.1.2.3.1.	FINISHED	Loyalty customer transactions will display points earned for each transaction.	Checkout	✓	✓

Phase 3 Requirements Review

Phase 3 involved catching up on all of the critical outstanding items from phases 1 and 2, including product display, search and filtering enhancements, individual product pages, shopping cart and checkout functionality, basic profile management functionality and transaction and points history display functionality.

In addition, there were planned phase 3 enhancements, including the ability to redeem points on a purchase transaction and see point balances and history in the UI. Less critical features or features for which we didn't have time to address were de-scoped and remain unimplemented. In general, there are also some missing validations and possible undiscovered defects due to data conditions and corner cases. However, the main use cases are operational, and the software is alpha quality and a functioning proof-of-concept product.

All current deliverable code was merged into the “phase3” branch on GitHub and has also been merged from there into the “main” branch.

The final production deployed application UI is hosted at the following URL:

<https://reactstore.z13.web.core.windows.net/>

The hosted API backend is located at the following address:

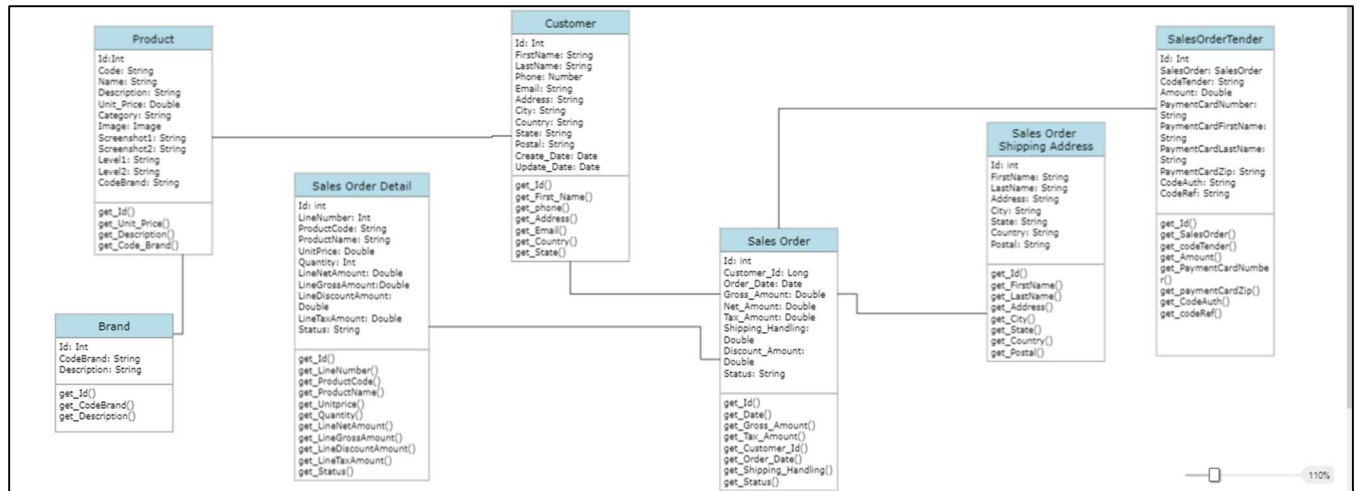
<https://loyalty-stack-api-loyalty-stack-api.azuremicroservices.io>

The API is self-documenting, and the documentation can be accessed at

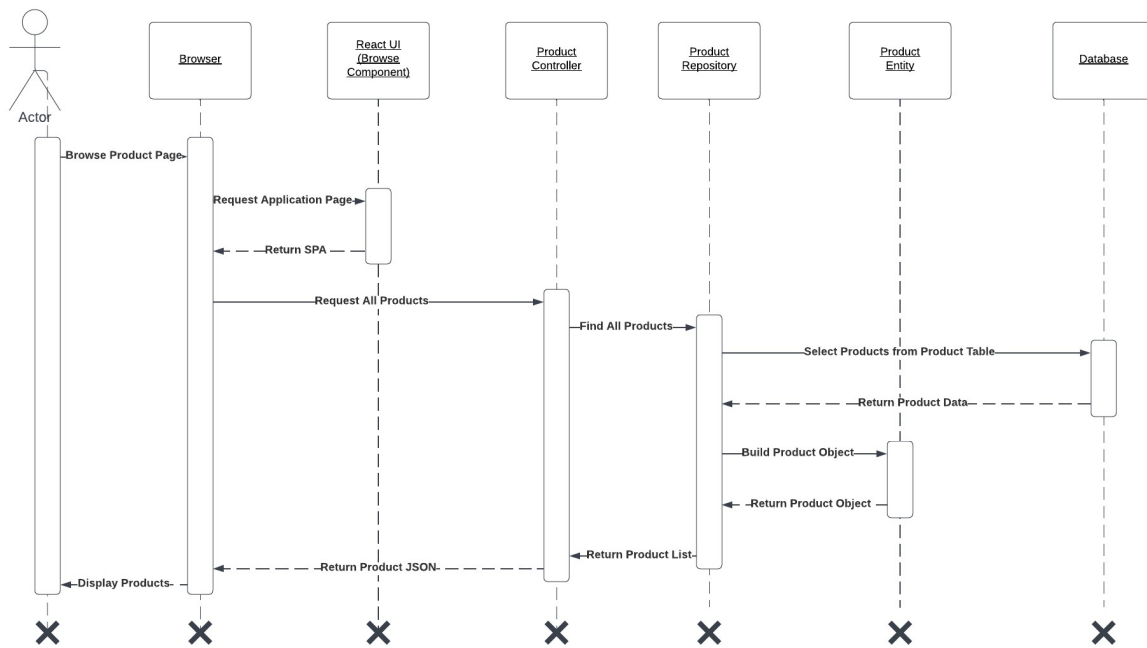
<https://loyalty-stack-api-loyalty-stack-api.azuremicroservices.io/api/swagger-ui/index.html>

d. UML design. You must include the following diagrams:

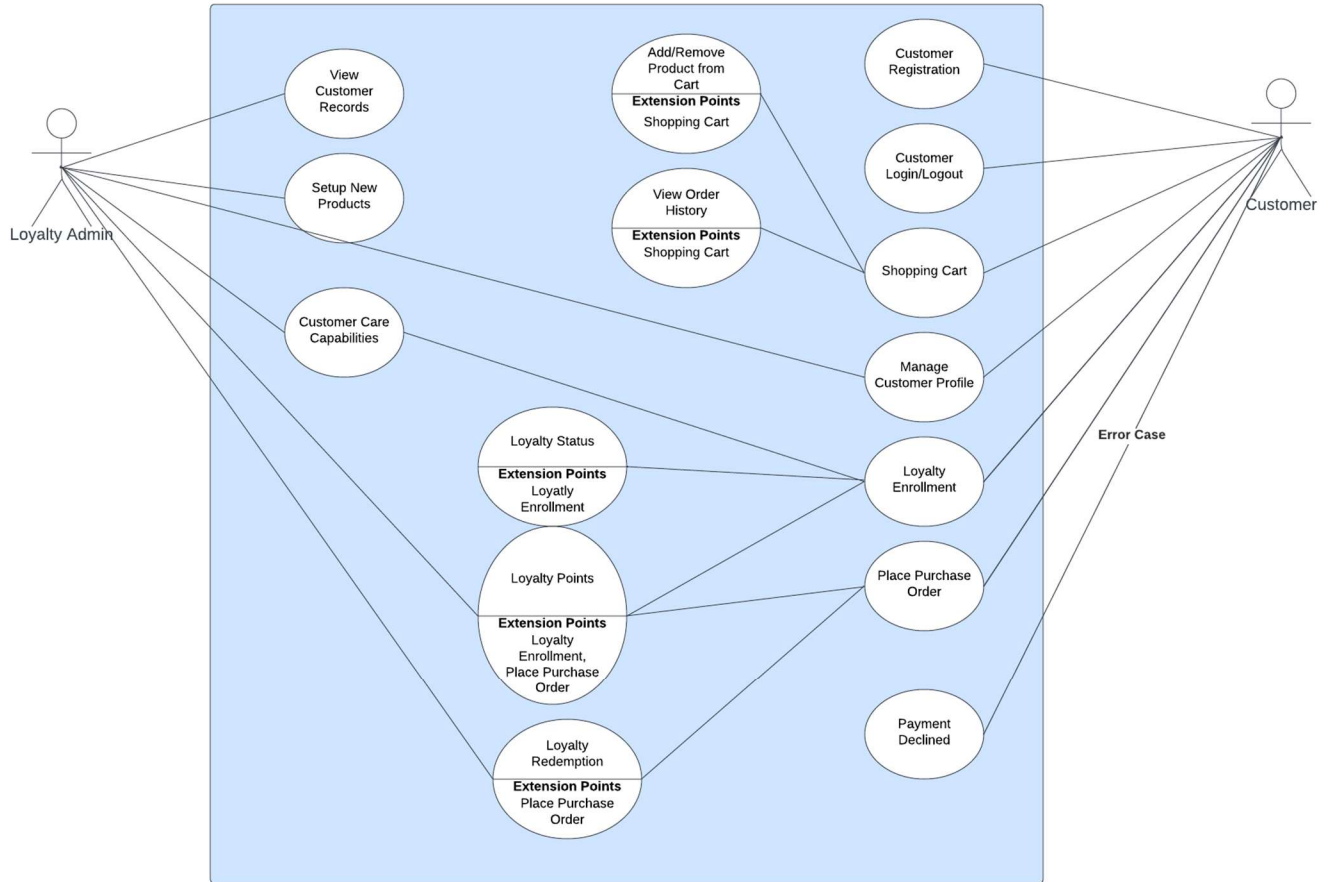
- Class diagram – Express status of classes, and interaction between classes.



- Sequence diagram – Interaction between object of classes, object interaction of class diagram.



- Use case diagram – at least one normal case and one error case should be included.



e. Test Cases (unit tests) for phase 2.

List a set of test cases used for testing the working program including descriptions of tests (e.g., what functionality they test, and inputs/outputs for them).

For the API portion of the project a suite of basic request/response tests were developed in the Postman tool. Instructions on how to import and use the Postman collection are in section e. The Spring Boot framework supports integrated unit testing frameworks, and we will probably implement some of these as framework unit tests in the next phases. These current tests can retrieve real data from a sample in-memory database instance integrated into the API application.

API Unit Test Cases (added for Phase 3)

Test	HTTP Request	HTTP Response
Get All Customers	GET request to /api/customers	An array of JSON Customer entities
Get Customer by Id	GET request to /api/customers/{customerid}	A single JSON Customer entity
Get All ProductLevels	GET request to /api/productlevels	An array of JSON ProductLevel entitites
Get All Products	GET request to /api/products	An array of JSON Product entities
Get Product by Id	GET request to /api/products/{productid}	A single JSON Product entity
Get All Brands	GET request to /api/brands	An array of JSON Brand entities
Get Product Hierarchy	GET request to /api/producthierarchy	An array of JSON Product Hierarchy level descriptions
Create Customer	POST request with JSON Customer body to /api/customer	A single JSON Customer entity with newly created customerId
Get Product by Code	GET request with “code” querystring parameter to /api/product	A single JSON Product entity with matching

		product code, if found.
Update Customer	PUT request with JSON Customer body, including customerId, to /api/customer/{customerid}	The updated single JSON Customer entity.
Get All Sales Orders	GET request to /api/customers/{customerid}/orders	An array of JSON SalesOrder entities.
Create Sales Order	POST request with JSON SalesOrder body to /api/orders	A newly created JSON SalesOrder entity with orderId. <i>Updated this use case to now earn Loyalty points, reflected in the response.</i>
Get Filtered Products	GET request with “System”, “Genre” or “Brand optional query string parameters to /api/products	An array of JSON Product entities matching the filter criteria parameters.
Customer Login	POST request with JSON Authorization body (username/password) to /api/auth/signin	A JSON Web Token with username and token values.
Get Customer with Auth Header	GET request with Authentication header to /api/customers/{customerid}	A single JSON Customer entity (access will be secured to matching token in future phase).
Create Customer Login Credentials	POST request with JSON User body to /api/auth/register	Returns HTTP OK on success.
Get Logged in User Details	GET request with Authentication header to /api/me	Returns JSON User entity for current user, including username and customerid
Get Orders By Customer Id with Auth Header	GET request with Authentication header to /api/customers/{id}/orders	For an authenticated customer, retrieve their sales orders

Product Name Search	GET request to /api/productsearch with “name” query string parameter.	Simple case-insensitive string-matching search on product names.
Create Customer Enroll in Loyalty	POST to /api/customers with a Customer create request and new “joinLoyalty” flag	Create a Loyalty program enrollment record with a newly-issued loyalty account/card number at the time of Customer profile creation.
Get Customer Points Balance	GET request with Authentication header to /api/customers/{id}/pointsbalance	Return the number of points earned by a customer.
Get Available Offers	GET request to /api/offers	Returns active bonus rules with descriptions of when points will be issued.
Get Loyalty Customer By Id	GET request to /api/customers/{customerid}	Same as Get Customer By Id, but this use case is now a “Loyalty” customer and has an Enrollment record attribute in the response.
Redeem Points by Customer Id	PUT request to /api/customers/{id}/redeempoints/{amount}	Request to debit specified amount of points from loyalty member points balance and return a dollar value discount certificate.
Get Customer Points History	GET request to /api/customers/{id}/pointshistory	Request to retrieve an array all all points activity on loyalty member’s

		account, including bonus point credits and point redemption debits.
--	--	---

UI Functional Tests

Test	Description	Outcome
View Welcome	Navigate to the application Welcome screen.	Currently available offers, configured as Loyalty bonuses, and based on date range compared to the current date will be displayed on the Welcome screen.
Browse Products	The browse product page displays products as informed by the product APIs and user can interact with product listings.	Products from various categories are displayed with images and product details. Filters can be applied individually or in combination, or specific products can be search by name. Clicking a product displays an individual product information page.
Add Product to Cart	On Product information page, clicking Add to Cart	A line item will be created in the Shopping Cart for the product displayed.
Change Quantity and Add Product to Cart	On Product information page, change the Quantity control element and then click Add to Cart.	A line item will be created in the Shopping Cart for the product displayed with the expected quantity. If the item is already in the cart the quantity will be increased as expected.
Sign-in Redirect	Click on Loyalty Service or Profile navigation menu items.	User should be redirected to sign in first.
User sign up	Complete the sign-up form, addressing all validation messages and submit.	Customer account with provided login credentials is created. Customer can now login with selected username and password.

Profile Page	After sign in, user can visit the profile page and manage profile data.	Any changes to profile data are persisted after clicking Save Changes. Any Customer order history is displayed.
Loyalty Service Page	After sign in, user can visit the Loyalty Service/Status page and view Loyalty account information.	If the Customer registered as a Loyalty member, then Customer Loyalty account number, points balance, and point activity history will be displayed.
No Checkout	Visiting the Shopping Cart when not items added.	Message will display “Your car is empty” and nothing else can be done on the page. User may navigate back to browser products or elsewhere.
Simple Checkout	Click Checkout in the Shopping Cart and proceed.	User can see Shipping Address defaulted to Customer account info, but can modify. User can submit payment card information and click Continue.
Submit Order	User is shown final cart totals including shipping and taxes and clicks Complete Checkout button.	Order Confirmation page is displayed with a new Order ID and information about any points earned. Point balance is updated and displayed in navigation menu (star) and on Loyalty Status page.
Redeem Points	Loyalty Customer with point balance greater than zero can enter points to redeem on Submit Your Order page and click Apply to debit points and generate a discount.	Redeemed points are deducted from points balance. Equivalent (100pts/dollar) discount is applied in prorated fashion to Shopping Cart line items. Net and gross cart totals are recalculated and updated.

- f. **A user manual that tells us how to install/use your program. This is meant for the end-user of the software. You may include screen shots, where appropriate.**

There are two components to install and deploy in Phase 1: a React storefront UI and a Java/Spring Boot API layer. Technically there is a database, but in this phase we are still using a zero-configuration ephemeral in-memory database (H2) integrated into the API stack for testing and demo purposes.

Installing the React storefront UI:

The React UI application is a JavaScript and React framework-based SPA (single-page application). Assuming the packages have been loaded and it has been configured for a live environment (it has not at this point), the UI can be hosted by virtually any Internet web server application (Apache, IIS, NGINX, etc) by copying the source files and packages to a web server and configuring a new web site appropriately. For this phase the application is still a prototype/POC at best with incomplete features, and we are only relying on an integrated Node.js server bundled with the project. Instructions for launching the UI with the demo configuration are in section g.

For hosted installation, Azure Storage was used for the React UI frontend. The React UI was compiled to a production build using the command “`npm run build`” from the source directory of the UI application (/src/react-store in the repository). The contents of the resulting build directory (/src/react-store/build) were then deployed on Azure Storage using the general instruction found here:

<https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-static-website>

Installing the Spring Boot API:

The Spring Boot API layer is 95% feature-complete for phase 1, covering all major use cases. However, it is currently using a non-production in-memory database (H2 database) integrated into the API stack. This database solution offers zero configuration, automatic DDL, and quick testing support. It is currently being pre-loaded with test customers, products and orders.

Deploying Spring Boot supports several options, none of which we are utilizing yet. However, the target model will likely be deploying the Spring Boot API application as an executable jar configured as a service on a Linux-based host. This is covered in detail in the standard documentation here:

<https://docs.spring.io/spring-boot/docs/current/reference/html/deployment.html#deployment.installing>

For hosted installation of the Java Spring Boot API, we used Azure Spring Apps, following the general guide found in the standard documentation here:

<https://spring.io/guides/gs/spring-boot-for-azure/>

Local launch demo configuration in support of front-end development and testing is described below in section g.

- g. Clear instructions on how to compile/run both your program and your test cases (the program must compile/run).**

Compiling and Running Spring Boot API:

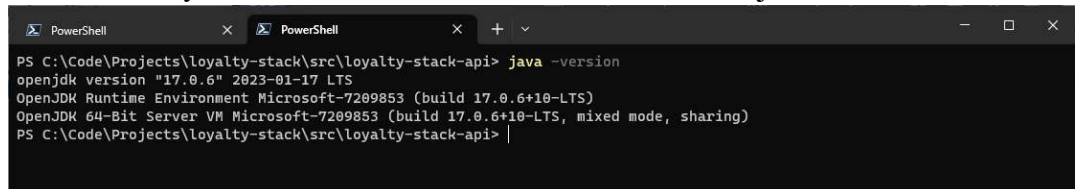
- Be sure to have a compatible JDK (JDK 17) installed. We are using and recommend the Microsoft build of Open JDK 17, found here:

<https://learn.microsoft.com/en-us/java/openjdk/download>

The appropriate file to download for Windows is **microsoft-jdk-17.0.6-windows-x64.msi**

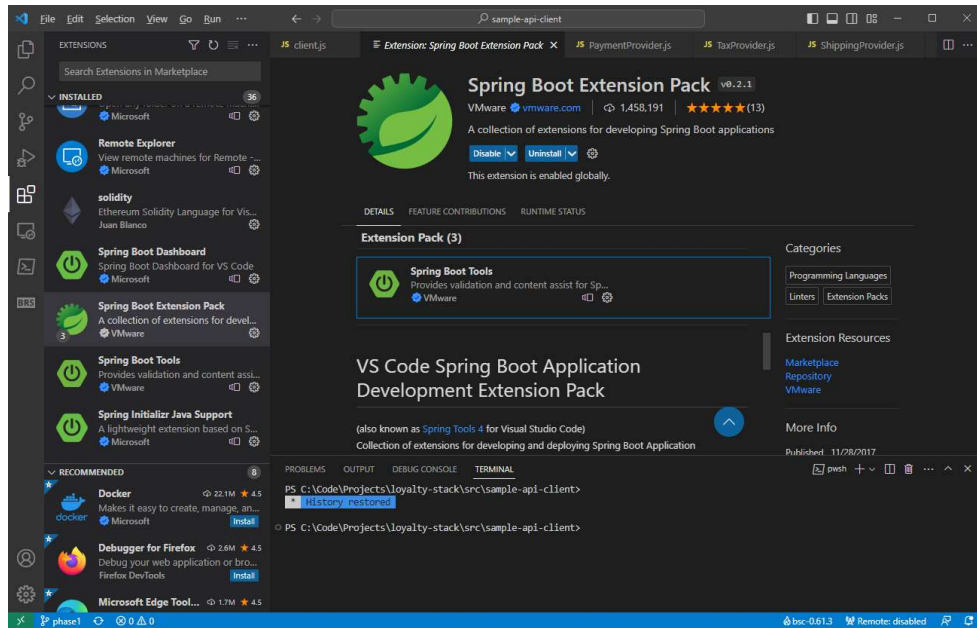
This file is an installer that you can run after downloading which should fully install and configure the JDK on a Windows system. There are similar builds available from the download link for other operating systems.

You can verify successful JDK install with the command “java -version”:

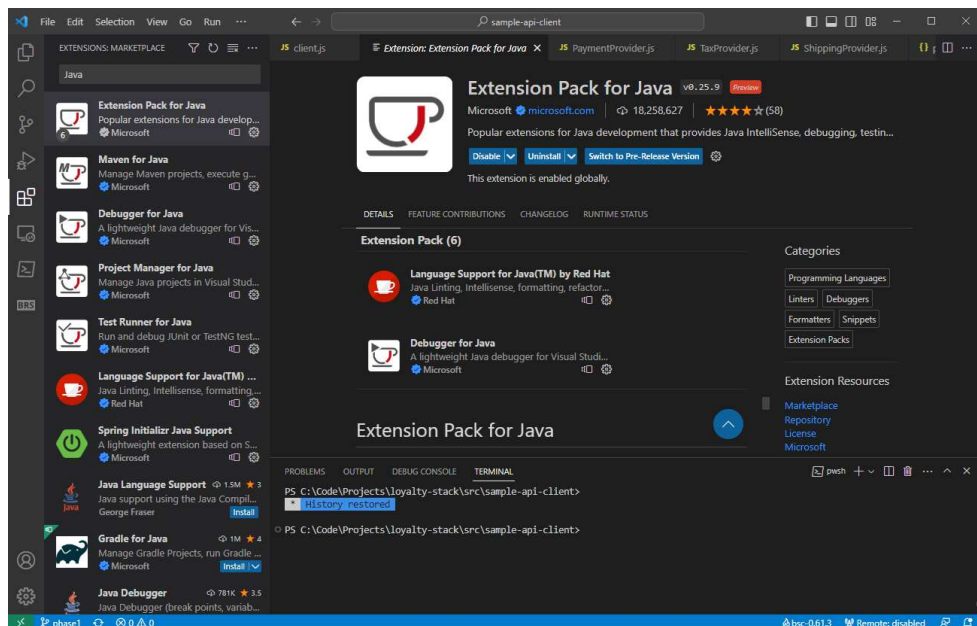


```
PS C:\Code\Projects\loyalty-stack\src\loyalty-stack-api> java -version
openjdk version "17.0.6" 2023-01-17 LTS
OpenJDK Runtime Environment Microsoft-7209853 (build 17.0.6+10-LTS)
OpenJDK 64-Bit Server VM Microsoft-7209853 (build 17.0.6+10-LTS, mixed mode, sharing)
PS C:\Code\Projects\loyalty-stack\src\loyalty-stack-api> |
```

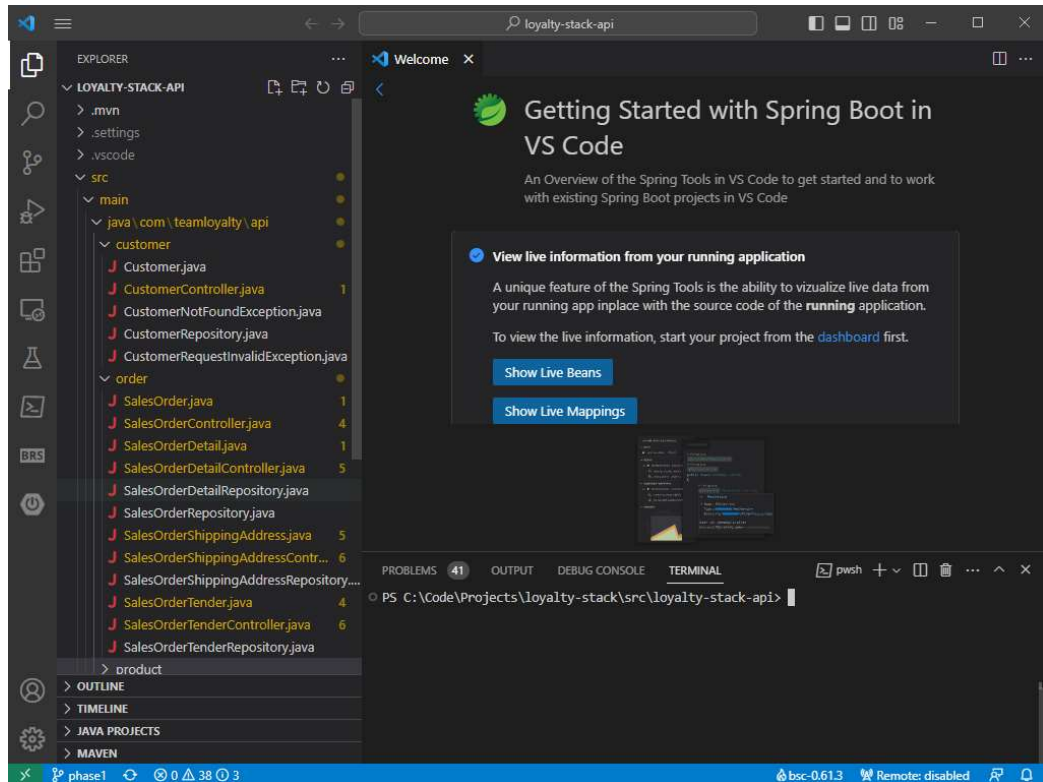
- The Spring Boot API is a Maven-based Spring Boot project. It is compatible with multiple IDEs and was primarily developed with a current version of Eclipse. However, for simplicity we recommend Microsoft VSCode for compiling and launching. Please install VSCode for your operating system from: <https://code.visualstudio.com/>
- In VSCode navigate to the Extensions view on the left, or click View menu > Extensions. In the extensions Search box search for and install the Spring Boot Extension Pack extension, which will install all relevant Spring Boot extensions.



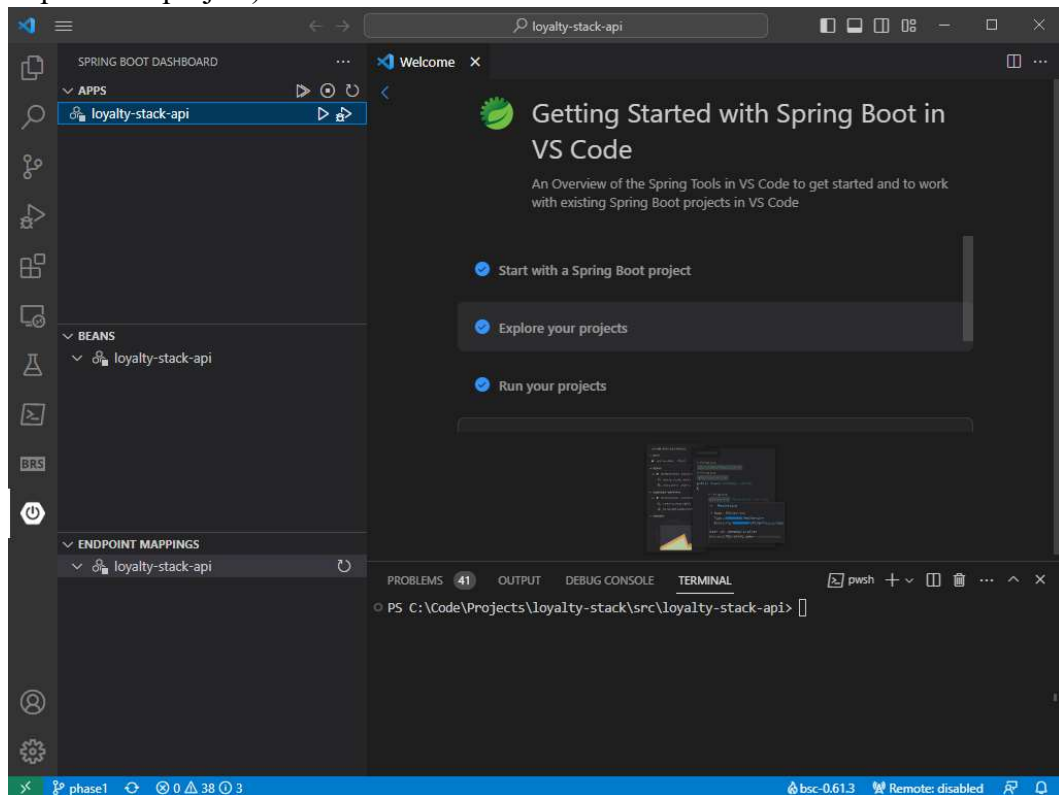
- We also recommend the Extension Pack for Java



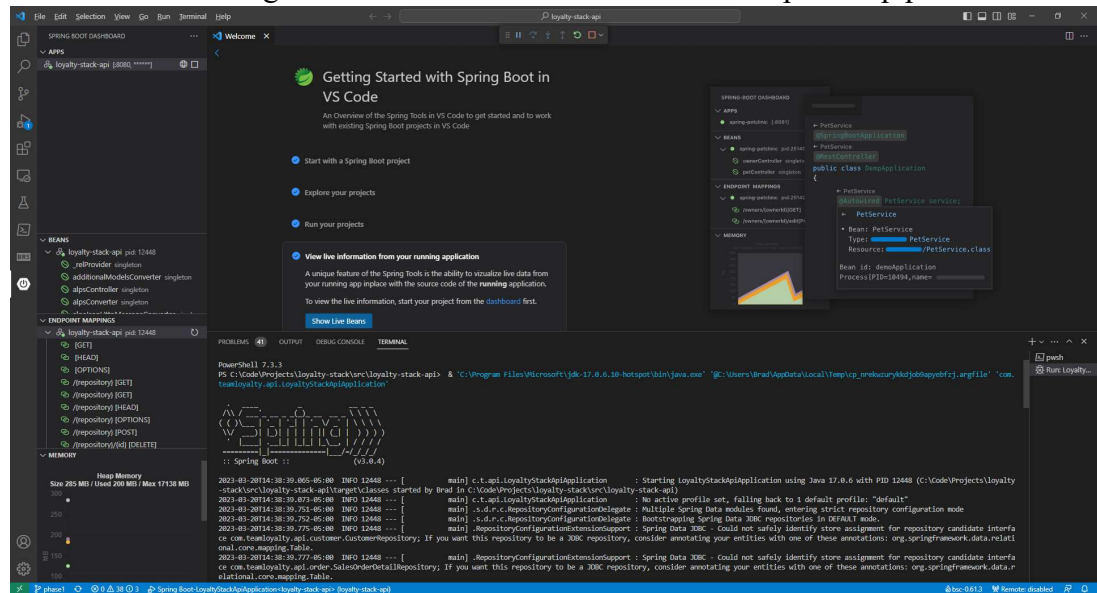
- Do a “git clone” of the full loyalty-stack repository to a local directory.
- Under the top-level repository folder, the Spring Boot API project can be found under src/loyalty-stack-api. This is a fully contained Spring Boot project. Please open just this folder directly in VSCode. You can right-click on the loyalty-stack-api folder under /src and select “Open With Code” or you can go to File>Open Folder in an already-running instance of VSCode. The opened project should look something like this:



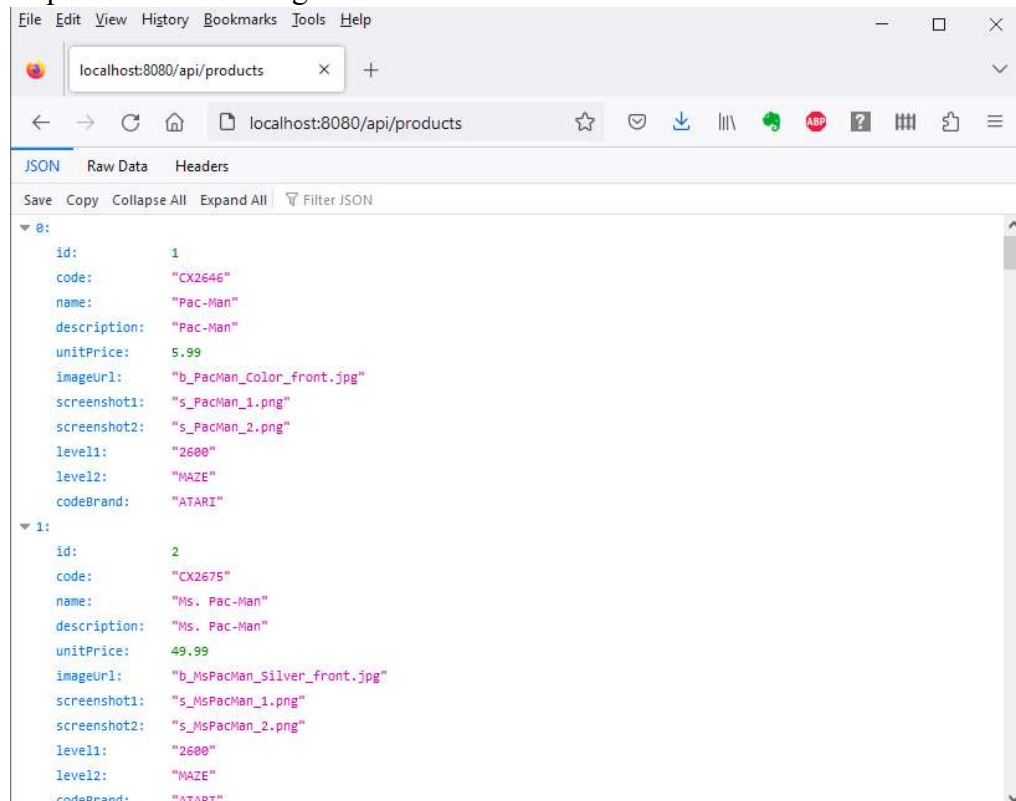
- On the left hand side the last navigation icon toward the middle of the screen accesses the Spring Dashboard. Click it to get this view (may take a few minutes to parse the project):



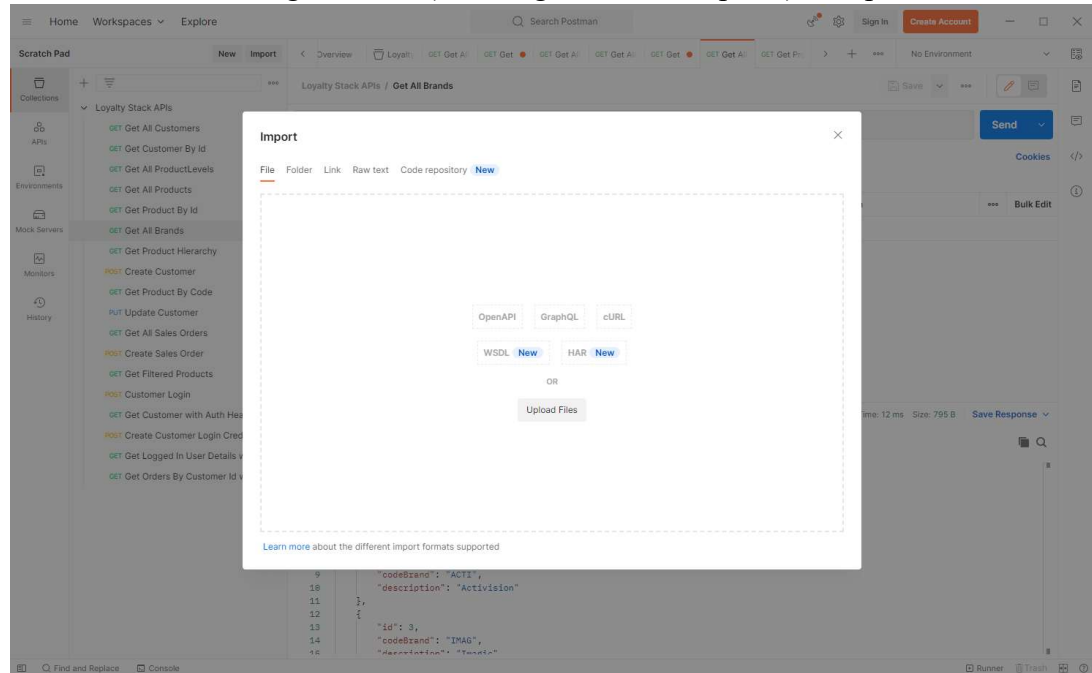
- In the top portion of the Spring Dashboard you can highlight the APP loyalty-stack-api and click the “Play” button to launch/run the API. You will see a banner and diagnostic text scroll through the terminal in the lower right corner. The API has an integrated Tomcat server and should come up on http port 8080.



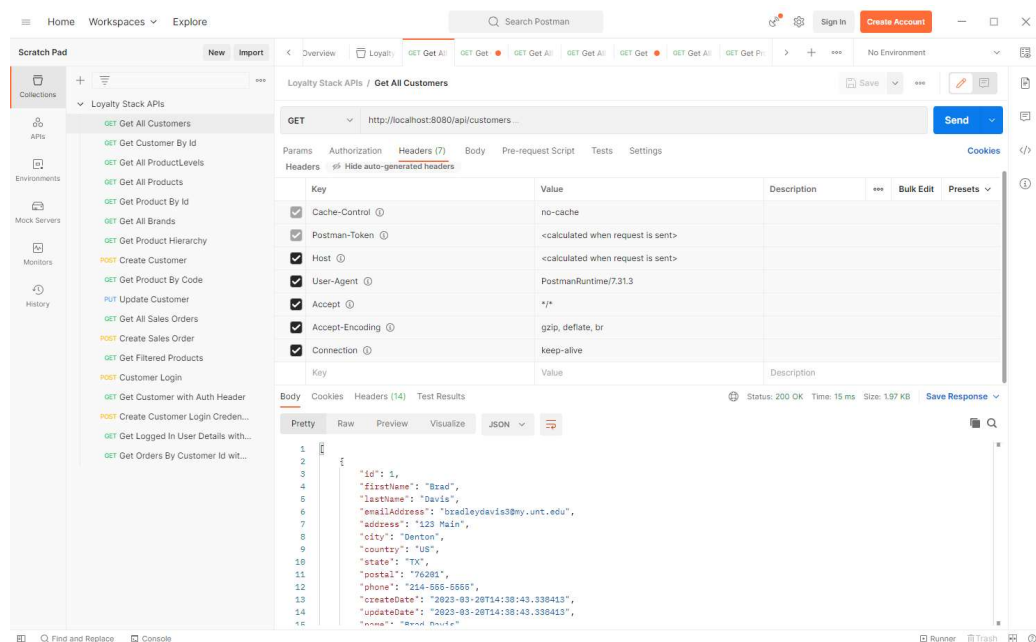
- Now you can do a simple validation that the API is running and responding. Be sure to allow access to any local firewall warnings and alerts. Place the URL <http://localhost:8080/api/products> into a web browser and you should get a response JSON message like so:



- There is also a complete Postman collection available with the project for testing the available APIs. Download and install Postman for your platform from <https://www.postman.com/>
- Launch Postman and got to File (hamburger menu in top left) > Import...



- Browse to the file named “Loyalty Stack APIs.postman_collection.json” under the tools/postman directory in the top level of the loyalty-stack project repository.
- Once loaded you can interact with the APIs using the provided samples. Select an API test from the list under Loyalty Stack APIs (expand if necessary) and click the blue “Send” button to see the API in action:



- The Body tab in the lower right section will show any API response JSON bodies.

- The API can be shut down using the Stop button next to the loyalty-stack-api APP in the Spring Dashboard in VSCode.

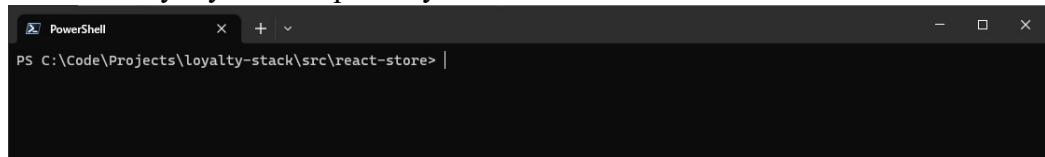
Compiling and Running React storefront UI:

- The React storefront UI is written primarily in JavaScript and does not technically require compilation. But there are still some setup steps in order to run it successfully.
- You will need a current version of Node.js. We are using and supporting Node.js version 18.15.0 LTS. Browse to <https://nodejs.org/en> to download and install it for your operating system.
- You can validate successful Node.js install by issuing the command “node --version” from a terminal or prompt:



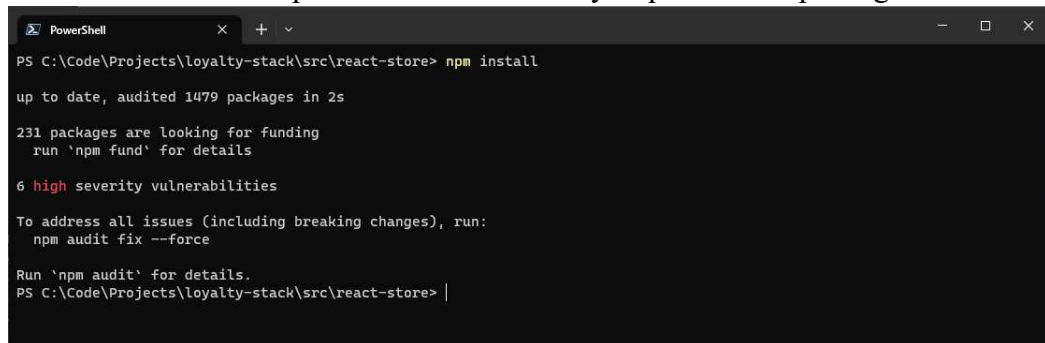
```
PowerShell
PS C:\Users\Brad> node --version
v18.15.0
PS C:\Users\Brad> |
```

- Open a terminal or command prompt and browse to the /src/react-store directory under the loyalty-stack repository:



```
PowerShell
PS C:\Code\Projects\loyalty-stack\src\react-store> |
```

- Issue the command “npm install” to install any required node packages:



```
PowerShell
PS C:\Code\Projects\loyalty-stack\src\react-store> npm install

up to date, audited 1479 packages in 2s

231 packages are looking for funding
  run 'npm fund' for details

6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run 'npm audit' for details.
PS C:\Code\Projects\loyalty-stack\src\react-store> |
```

- Now the application should be ready to launch. Issue the command “npm start” to launch the integrated Node.js web server. The app should automatically launch in a web browser, or you can navigate to it on port 3000 at <http://localhost:3000>



```
Windows PowerShell
Compiled successfully!

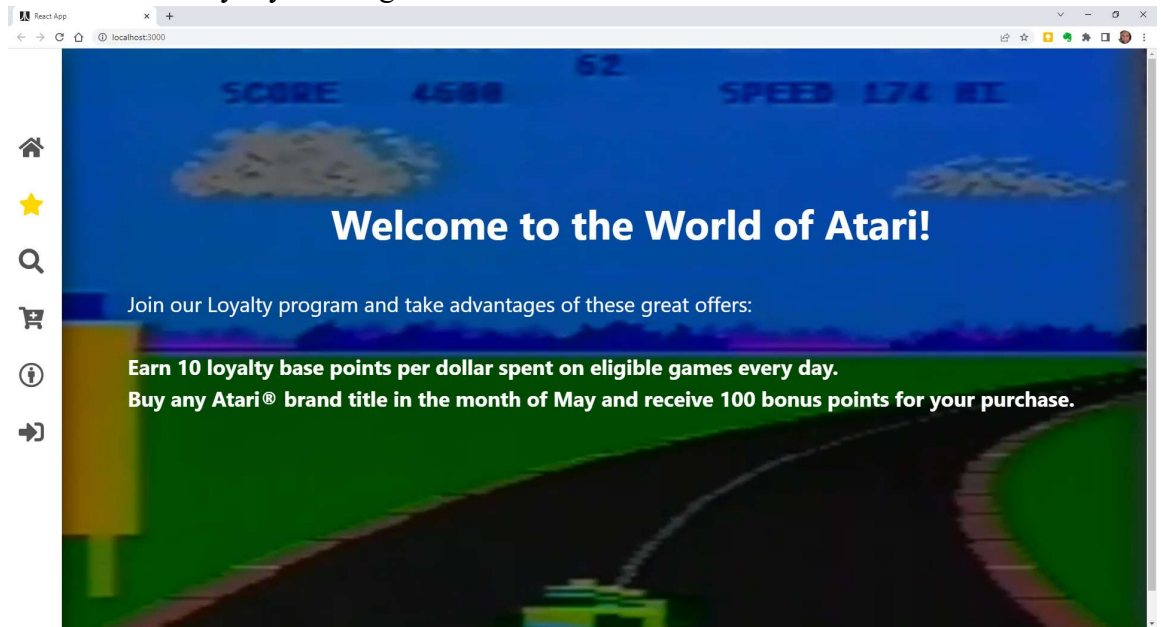
You can now view loyalty in the browser.

Local:      http://localhost:3000
On Your Network: http://192.168.1.28:3000

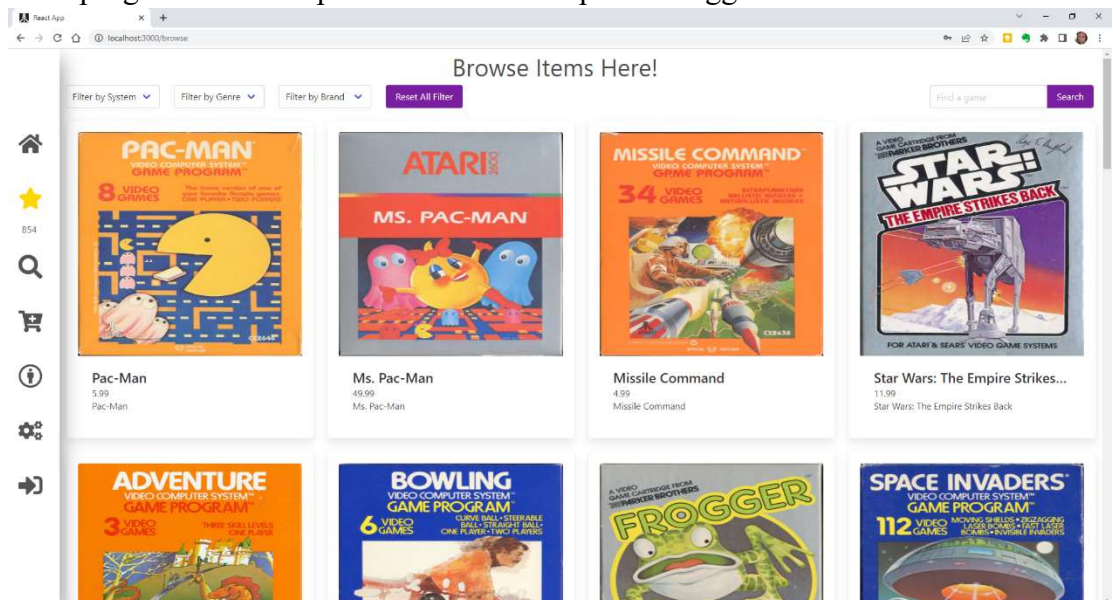
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
|
```

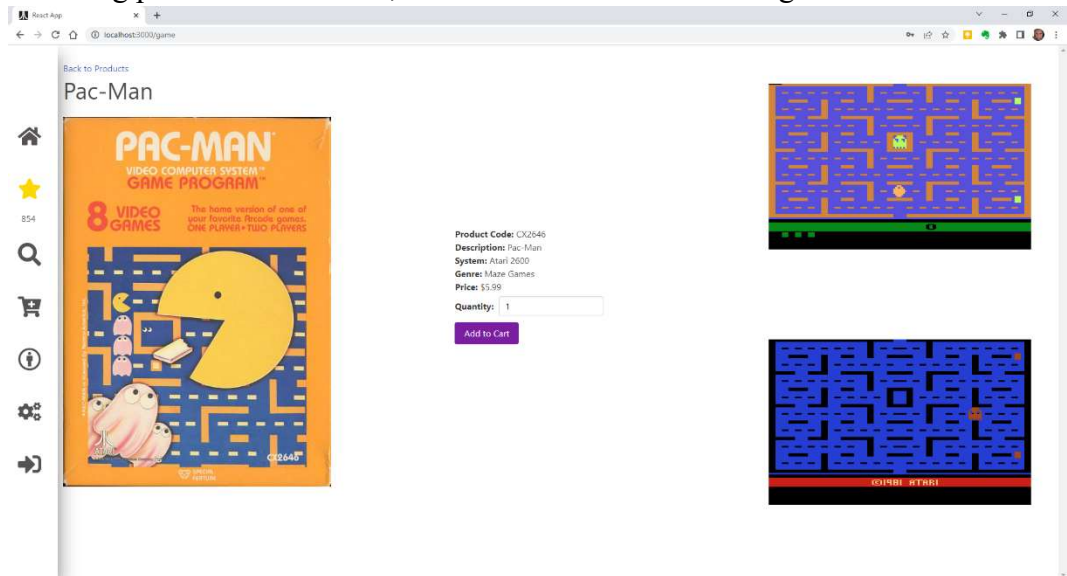

- Initially you will see the updated site Welcome page with background animation and available Loyalty offerings:



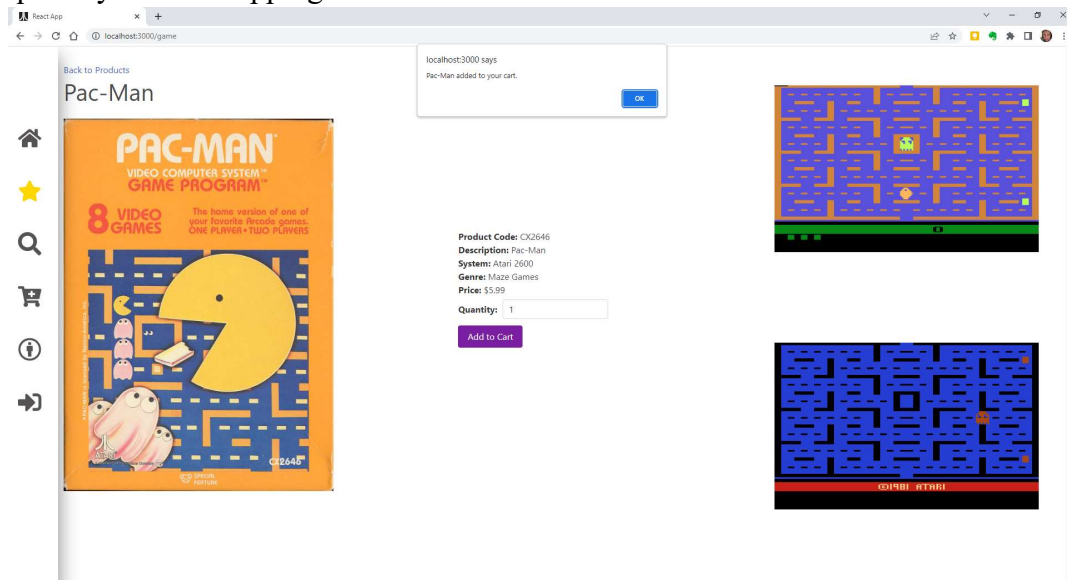
- By clicking on the Browse (Search) icon in the left nav bar you can currently view, and filter product tiles retrieved from the API. There is also a Search box in the top right corner that provides search completion suggestions.



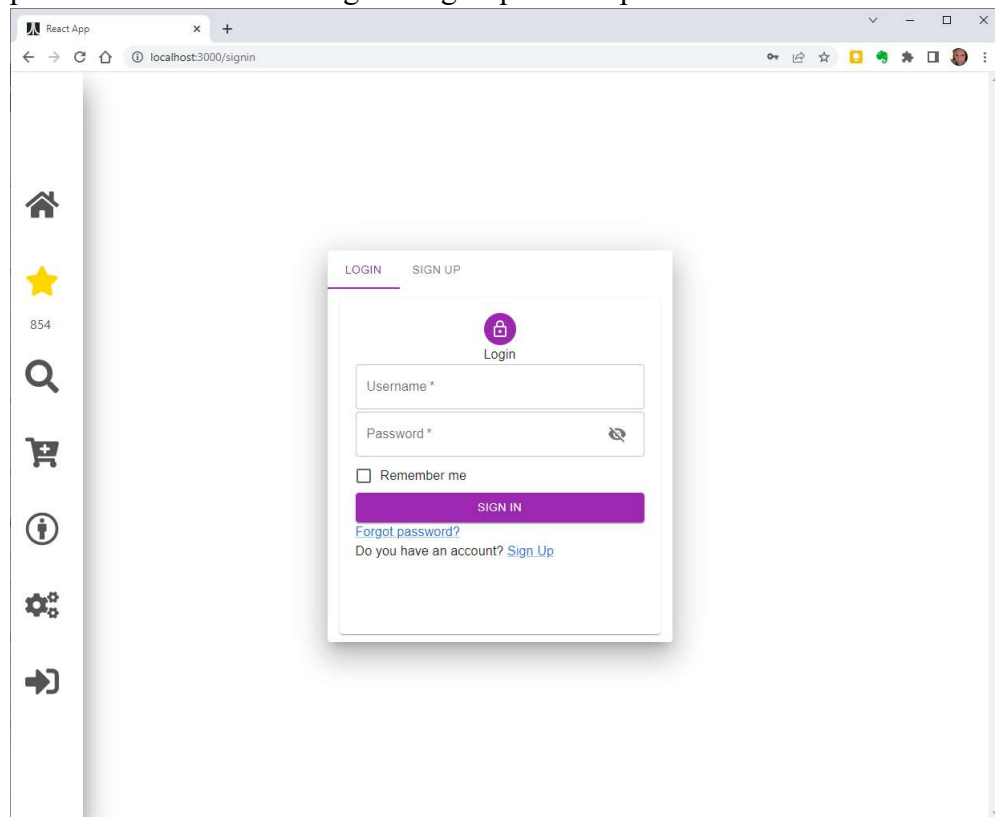
- The individual product tiles are clickable and lead to individual product pages, including product screenshots, which can be clicked to enlarge.



- Clicking on the “Add to Cart” button will add the current game product and selected quantity to the Shopping Cart.



- By clicking the Sign-In icon in the navigation bar (bottom left item) a customer is presented with a combo sign-in/sign-up UI component:



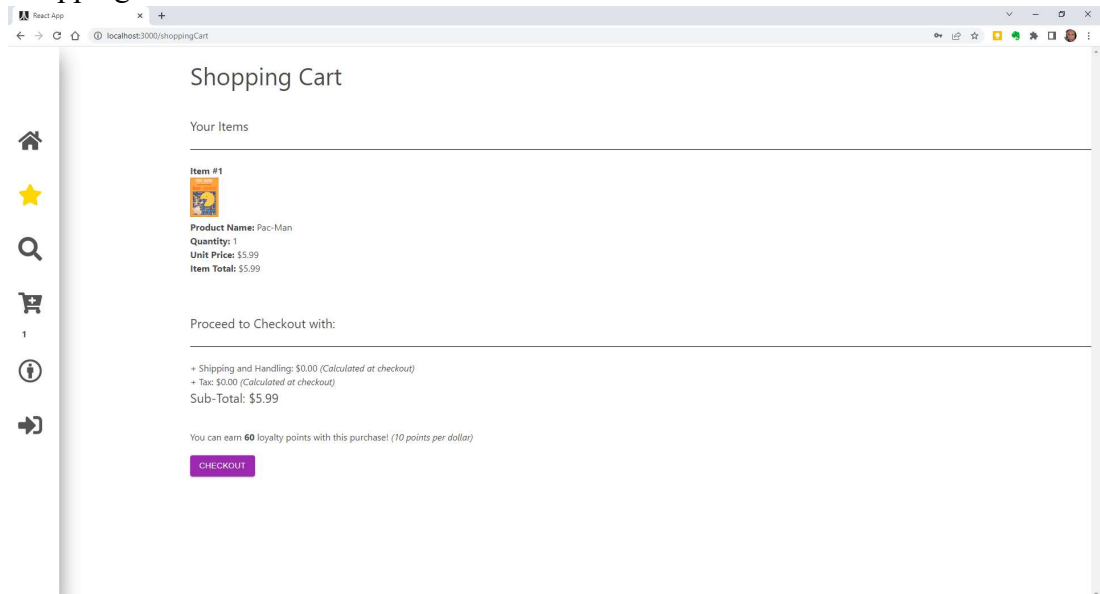
- Click SIGN UP and fill in the form to create an account. All required fields must be completed. Check the box to accept T&C's. Optionally leave the checkbox to join the site's Loyalty program at registration.

The screenshot shows a web browser window with the URL `localhost:3000/signin`. The page has a sidebar on the left with icons for Home, Favorites (854 items), Search, Cart, User Profile, Settings, and Logout. The main content area displays a modal form titled "SIGN UP" with a purple plus icon. The form includes fields for First Name, Last Name, Email, Username, Password, Confirm Password, Address, City, State, Zip Code, and Phone Number. At the bottom, there are two checkboxes: "I accept the terms and conditions." (unchecked) and "Enter loyalty program and earn points!" (checked).

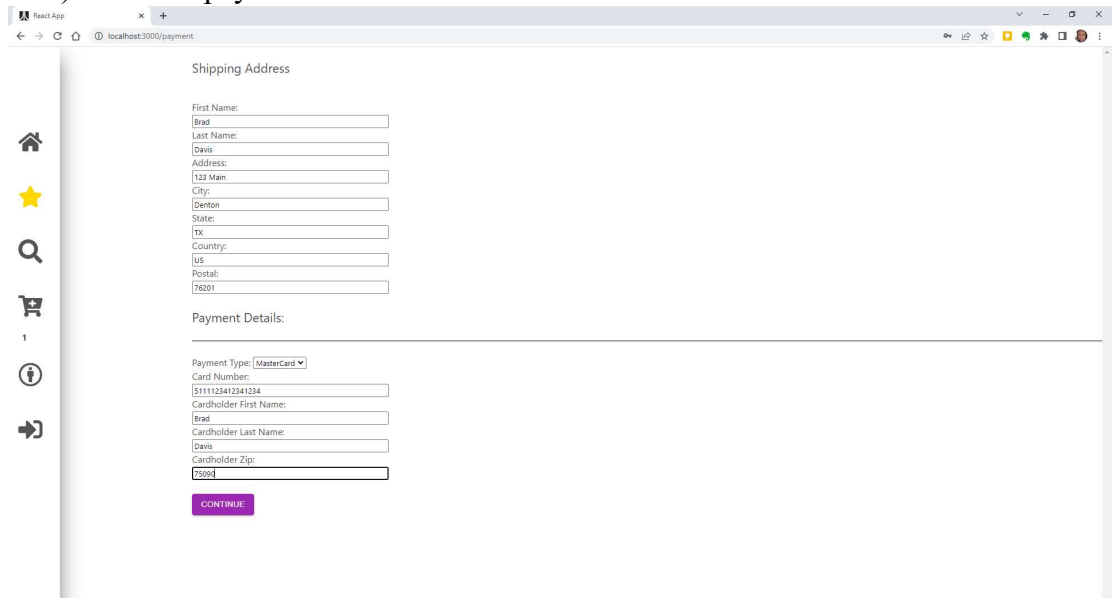
- After registering for the web site, sign in to be taken to the Customer Profile page, where the user can manage and update their Customer personal information and see Order History:

The screenshot shows the "Profile" page in a web browser. The sidebar is identical to the previous screenshot. The main content area displays the user's profile information in a form: First Name (Brad), Last Name (Davis), Email (bradleydavis3@my.unl.edu), Phone (274-555-5555), Address (123 Main), City (Denton), State (TX), Country (US), and Postal (76201). Below the form is a purple "SAVE CHANGES" button. Under the "Order History:" section, the following information is displayed: Order ID: 1, Date: 2023-05-01T19:25:10.046603, Total: \$11.50.

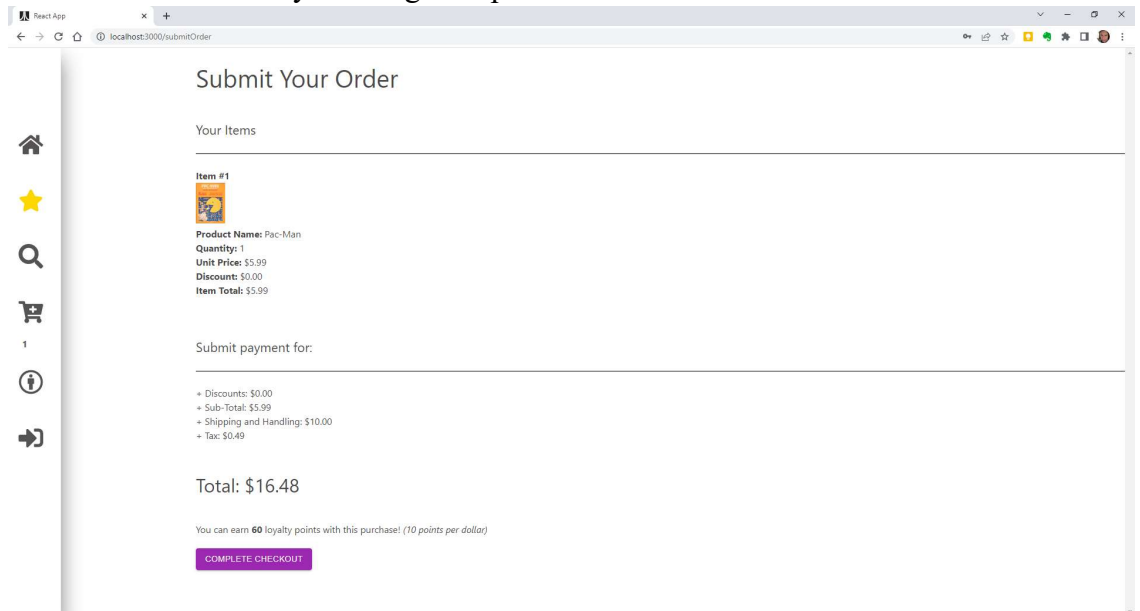
- Click the Shopping Cart in the navigation menu to see items currently in the Shopping Cart. You can click the Checkout button to continue.



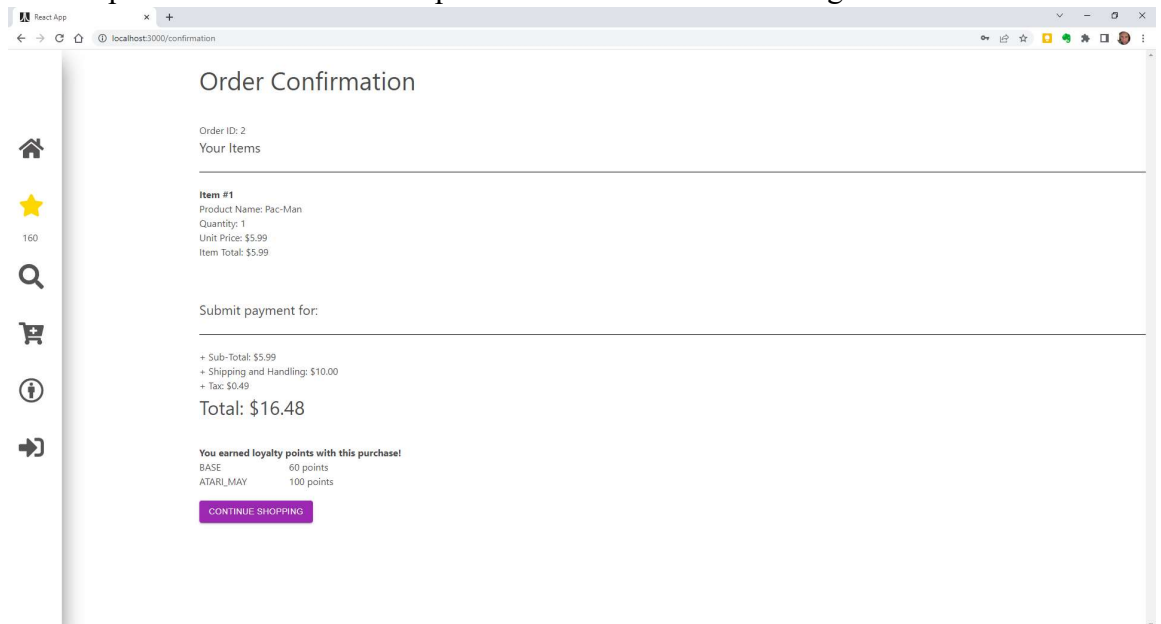
- Next the customer can update their Shipping Address (defaults to Customer Profile data) and enter payment card information for the order. Click the Continue button.



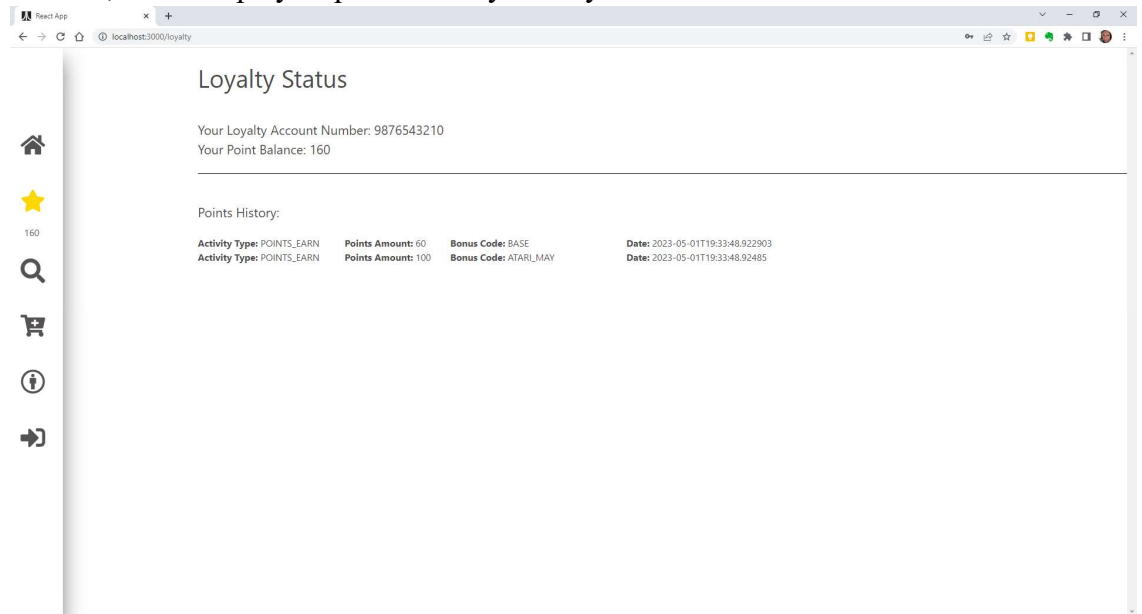
- A final total, including taxes, shipping and any discounts, is displayed and the user can submit the order by clicking Complete Checkout button.



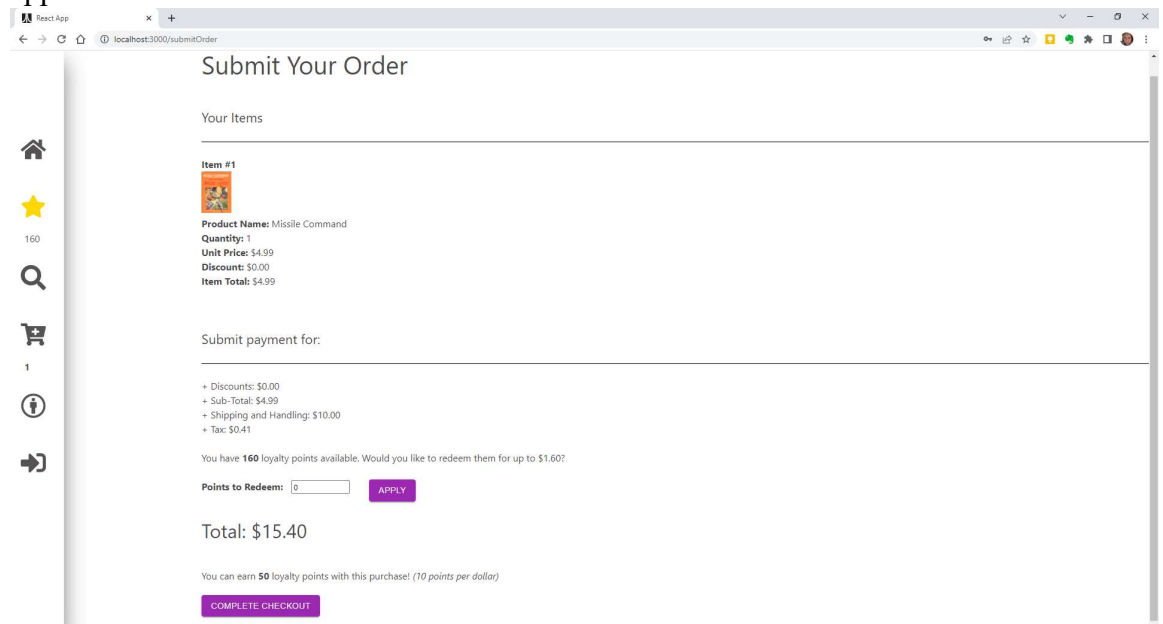
- And order confirmation page is displayed, and if the user joined the Loyalty program while registering, any points earned for this transaction are also displayed, and the point balance should be updated near the star in the navigation menu.



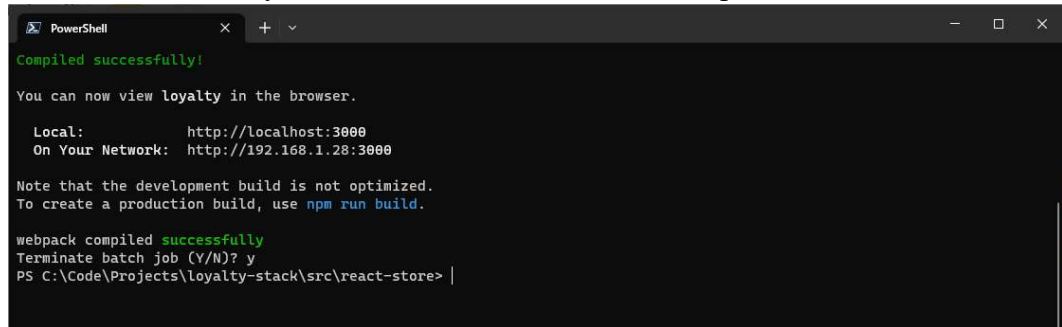
- The customer can now click on the Loyalty Status (star) in the navigation menu to see their Loyalty account information, including enrolled account number, point balance, and a display of point activity history.



- If a Loyalty Customer has a non-zero point balance they will have the opportunity at checkout to redeem some or all of their points balance for a discount on their purchase during checkout on the Submit Order screen. Simply enter the desired amount of points and click Apply to see the points redeemed for a discount and applied to the order.



- Generally the web UI can be shut down by closing your browser and issuing the Ctrl-C or Cmd-C key combination in the terminal to stop the web server.



```
PowerShell
Compiled successfully!

You can now view loyalty in the browser.

Local:      http://localhost:3000
On Your Network: http://192.168.1.28:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
Terminate batch job (Y/N)? y
PS C:\Code\Projects\loyalty-stack\src\react-store> |
```

- h. A section that describes which features you have successfully implemented and any limitations (i.e., any known problems or issues with your program) such as features that you were unable to complete on time including feedback received during peer review sessions. It should include discussions of your plans for the next phase of your project. These are the next features and/or improvements you plan to make to your program (assume that you will continue working on this project next semester).

Team Loyalty accomplished a lot this semester and delivered a comprehensive proof-of-concept product for an ecommerce-based Loyalty solution. We built a RESTful API suite providing more than 25 APIs to support ecommerce and Loyalty capabilities and a web-based ecommerce customer experience for shopping and earning and redeeming Loyalty points on purchases. Core functionality included customer registration with optional Loyalty enrollment, customer sign-in, customer profile management, product browsing, searching, filtering, and listing capabilities driven by APIs, a comprehensive shopping cart experience with ability to add items, submit shipping and payment details, calculate shipping, taxes and totals, earning points based on multiple rules entered into a system points engine, and redeeming points for discounts. Customers can also view their point balances and some basic purchase transaction and points activity histories.

Much remains to be done, and some basic usability features are still lacking. Although not difficult, no sign-out capability was implemented as it was not a critical priority. There is no way to remove items from the shopping cart currently. Sessions are not currently persisted when the site is manually refreshed or when the user navigates the browser away and returns later. Logins are not persistent either. More tests need to be implemented for the API layer, preferably in a standard unit testing framework. Some of the functional elements of the UI need styling, layout and flow improvements and there needs to be more consistent application of stylesheets. The UI is not currently mobile-friendly. Many features lack basic validations and are reliant on users providing correct input to function properly. At times the production deployment of the application behaves slightly differently from a locally executed version, displaying

obscure errors in some corner cases (e.g., financial/numeric figures sometimes displaying as NaN – Not-a-Number).

All of this and more will be addressed in future phases. Additionally, there are many Loyalty capabilities and enhancements that were not even included in the initial proposed feature requirements, but which could be ideated further and implemented in future phases, such as badges, social login, gamification, surveys and quizzes, leader boards, new types of point redemptions (free items, etc), charitable donations and more.

i. A brief reflection on what has been accomplished, what went well and could be improved.

The end result has a reasonably high-quality API and application design that provides a solid foundation for future enhancements. Although some code could be refactored to more complete implementations, very little of it represents shortcut or stopgap coding. The overall coding quality is high but is lacking in some basic validations and error handling. Even so, most existing bugs currently do not manifest as critical errors or application failures but as unexpected output. The application behaves gracefully most of the time. The current code base is probably an alpha-level product and a decent proof-of-concept for demonstration purposes. The application also exhibits a clean layered architecture, including a presentation layer, an API layer and a data store, with clear separation of responsibilities. The majority of visible functionality is actually implemented and not just mocked up in front-end code.

What could have gone better would have been making more progress earlier on and ramping up on technology familiarity more aggressively. This could have perhaps been facilitated by having each team member produce sample projects based on tutorials and exercises to learn the languages and libraries by key deliverable dates. We did gather substantial resources including documentation, tutorials, and videos, but it was left to each team member to use them as needed. A more rigorous approach by our group to team meetings and individual tasks and deliverables probably would have been helpful as well and more analogous to industry practice.

Finally, we are reasonably pleased and proud of the result of the work we have done. We managed to implement and deliver a multi-tiered enterprise-style application with more than just basic ecommerce features, using technologies and tools and are widely employed in industry today. Hopefully this exercise has been instructive and informative for everyone on the team, and we can all point to this as a successful body of work in our academic careers.

Team Member Contributions

Team Member	Contributions
Brad Davis	API design and implementation, Points calculation engine and rules, React coding, Postman unit tests, sample data
Sandy Martinez-Echegoyen	React product browse and item pages, filters, stylesheets, search, customer sign up and sign in.
Tyler Parks	React app navigation bar, component pages, landing page with animation, profile - shopping cart page template.
Vishpendra Chahar	UML diagrams, meeting minutes, react coding, testing
Rama Reddy Venkata	React coding, testing