

Deliverable 3 Report

Team Loyalty

a. Phase 1 Requirements Status

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

3.1.7.1.	IN-PROGRESS	Site visitors can register and create a customer profile.	✓	Create an account.
3.1.7.1.1.	IN-PROGRESS	First and Last name (required fields)	✓	Enter account information.
3.1.7.1.2.	IN-PROGRESS	Email address (required field, must be unique)	✓	Enter account information.
3.1.7.1.3.	IN-PROGRESS	Secure password (required field)	✓	Enter account information.
3.1.7.2.	IN-PROGRESS	Customers can login with their registered email address and password.	✓	Customer can login with user/pass.
3.1.7.3.	IN-PROGRESS	Customers can log out.	✓	Customer can logout.
3.1.7.4.	IN-PROGRESS	Customer logged-in status is visible on the web site banner.	✓	Customer sees updated page when logged in.
3.1.1.3.	NOT STARTED	Mobile responsive design [Conditional]	✓	Integrate mobile functionality.
3.1.5.	NOT STARTED	Product detail pages	✓	
3.1.6.2.	NOT STARTED	Display items added to cart.	✓	
3.1.6.2.1.	NOT STARTED	Display item quantity.	✓	
3.1.6.2.2.	NOT STARTED	Display item name.	✓	
3.1.6.2.3.	NOT STARTED	Display item (small) image.	✓	
3.1.6.2.4.	NOT STARTED	Display item price (in Dollars).	✓	
3.1.6.3.	NOT STARTED	Display calculated shopping cart subtotal.	✓	
3.1.6.4.	NOT STARTED	Calculate sales taxes (Texas 8.25%)	✓	
3.1.6.5.	NOT STARTED	Calculate shipping (flat rate)	✓	
3.1.6.6.	NOT STARTED	Calculate purchase total.	✓	
3.1.6.7.	NOT STARTED	Site visitor can change cart item quantity.	✓	
3.1.6.7.1.	NOT STARTED	Update subtotal (in Dollars) when quantity changed.	✓	
3.1.6.7.2.	NOT STARTED	Update sales taxes (Texas 8.25%)	✓	
3.1.6.7.3.	NOT STARTED	Update purchase total.	✓	
3.1.6.8.	NOT STARTED	Site visitor can remove item from cart.	✓	✓
3.1.6.8.1.	NOT STARTED	Update subtotal (in Dollars) when item removed.	✓	
3.1.6.8.2.	NOT STARTED	Update sales taxes (Texas 8.25%)	✓	
3.1.6.8.3.	NOT STARTED	Update purchase total.	✓	
3.1.6.9.	NOT STARTED	Proceed to Checkout	✓	✓
3.1.6.9.1.	NOT STARTED	Site visitor shall be prompted to login if necessary.	✓	✓
3.1.6.9.2.	NOT STARTED	Logged-in customer details are pre-populated:	✓	
3.1.6.9.2.1.	NOT STARTED	Populate name.	✓	

3.1.6.9.2.2.	NOT STARTED	Populate shipping address.	✓
3.1.6.9.2.3.	NOT STARTED	Populate email.	✓
3.1.6.9.2.4.	NOT STARTED	Populate phone.	✓
3.1.6.9.3.	NOT STARTED	Logged-in customer may update shipping details for order.	✓ ✓
3.1.6.9.3.1.	NOT STARTED	Update name	✓
3.1.6.9.3.2.	NOT STARTED	Update shipping address	✓
3.1.6.9.3.3.	NOT STARTED	Update email contact	✓
3.1.6.9.3.4.	NOT STARTED	Update phone	✓
3.1.6.9.4.	NOT STARTED	Logged-in customer can enter payment details.	✓
3.1.6.9.4.1.	NOT STARTED	Enter card type.	✓
3.1.6.9.4.1.1.	NOT STARTED	Multiple payment tender types supported.	✓
3.1.6.9.4.2.	NOT STARTED	Enter card number.	✓
3.1.6.9.4.3.	NOT STARTED	Enter card expiration.	✓
3.1.6.9.4.4.	NOT STARTED	Enter name on card.	✓
3.1.6.9.4.5.	NOT STARTED	Enter payment zip code.	✓
3.1.6.9.5.	NOT STARTED	Logged-in customer may submit order.	✓
3.1.6.9.5.1.	NOT STARTED	Validate shipping details complete.	✓
3.1.6.9.5.2.	NOT STARTED	Validate payment details complete.	✓
3.1.6.9.5.3.	NOT STARTED	Submit transaction details for processing.	✓
3.1.6.9.5.4.	NOT STARTED	Authorize transaction payment.	✓
3.1.6.9.5.4.1.	NOT STARTED	Return message on failure (e.g., Payment declined)	✓
3.1.6.9.6.	NOT STARTED	System will record authorized transaction to customer account.	✓
3.1.6.9.6.1.	NOT STARTED	Transaction Header data is recorded.	✓
3.1.6.9.6.1.1.	NOT STARTED	Transaction Date	✓
3.1.6.9.6.1.2.	NOT STARTED	Transaction Location (Web store)	✓
3.1.6.9.6.1.3.	NOT STARTED	Transaction Number (System generated)	✓
3.1.6.9.6.1.4.	NOT STARTED	Transaction Dollar Net Total	✓
3.1.6.9.6.1.5.	NOT STARTED	Transaction Dollar Gross Total	✓
3.1.6.9.6.2.	NOT STARTED	Transaction Line Detail data is recorded.	✓

3.1.6.9.6.2.				
1.	NOT STARTED	Line number	✓	
3.1.6.9.6.2.				
2.	NOT STARTED	SKU	✓	
3.1.6.9.6.2.				
3.	NOT STARTED	Quantity	✓	
3.1.6.9.6.2.				
4.	NOT STARTED	Unit price	✓	
3.1.6.9.6.2.				
5.	NOT STARTED	Net amount	✓	
3.1.6.9.6.2.				
6.	NOT STARTED	Tax amount	✓	
3.1.6.9.6.2.				
7.	NOT STARTED	Gross amount	✓	
3.1.6.9.6.2.				
8.	NOT STARTED	Discount amount applied.	✓	
3.1.6.9.6.3.	NOT STARTED	Transaction Tender data is recorded (single tender).	✓	
3.1.6.9.6.3.				
1.	NOT STARTED	Payment type		
3.1.6.9.6.3.				
2.	NOT STARTED	Tender code		
3.1.6.9.6.3.				
3.	NOT STARTED	Payment details		
3.1.6.9.6.3.				
4.	NOT STARTED	Tender amount		
3.1.6.9.6.4.	NOT STARTED	Charge payment method.		
3.1.7.1.4.	NOT STARTED	Phone number (optional field)	✓	Enter account information.
		Physical address - street, city, state, zip (optional, must be complete)	✓	Enter account information.
3.1.7.1.5.	NOT STARTED	Logged-in customer can update their profile data.	✓	✓
3.1.7.5.	NOT STARTED	Update first and last name.	✓	
3.1.7.5.1.	NOT STARTED	Update email address (must be unique).	✓	
3.1.7.5.2.	NOT STARTED	Change secure password.	✓	
3.1.7.5.3.	NOT STARTED	Update phone number.	✓	
3.1.7.5.4.	NOT STARTED	Update physical address.	✓	
3.1.7.5.5.	NOT STARTED	Update birthday.	✓	
3.1.7.5.6.	NOT STARTED	Customer can view purchase history.	✓	
3.1.7.6.	NOT STARTED	Summary list of transactions including data, transaction number, total dollar amount, status.	✓	
3.1.7.6.1.	NOT STARTED		✓	

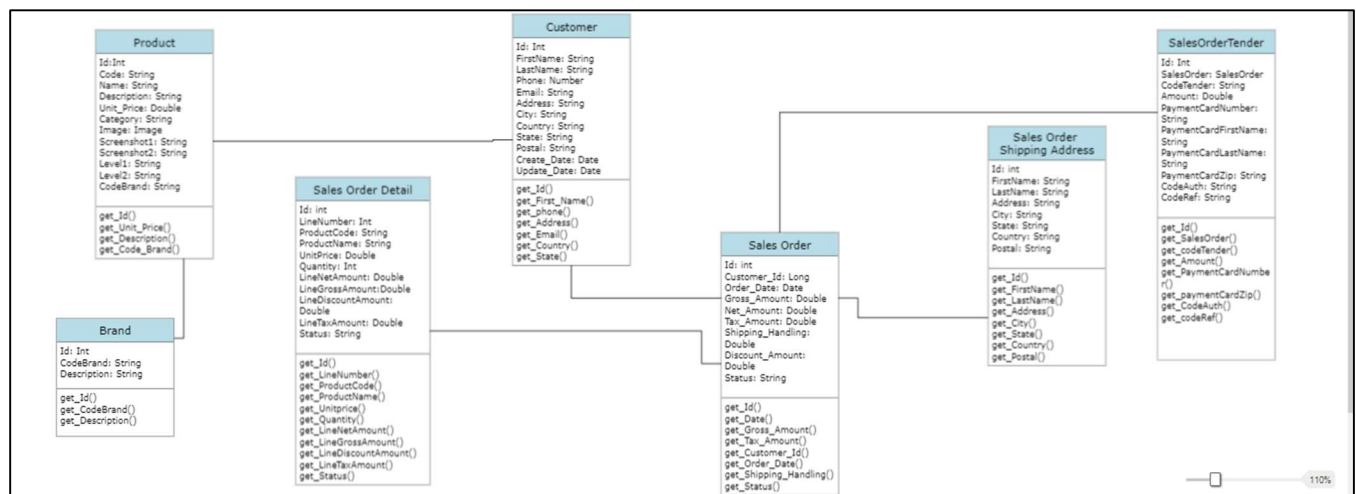
Overall, the API is 95% complete for phase 1, with minor refactoring or additional detail added in future phase. UI is 10% complete for phase 1 requirements.

To mitigate the gap most Phase 1 UI requirements will migrate to Phase 2. This should be feasible because Phase 2 is light on UI change requirements and will mostly involve back-end enhancements to support basic Loyalty capabilities.

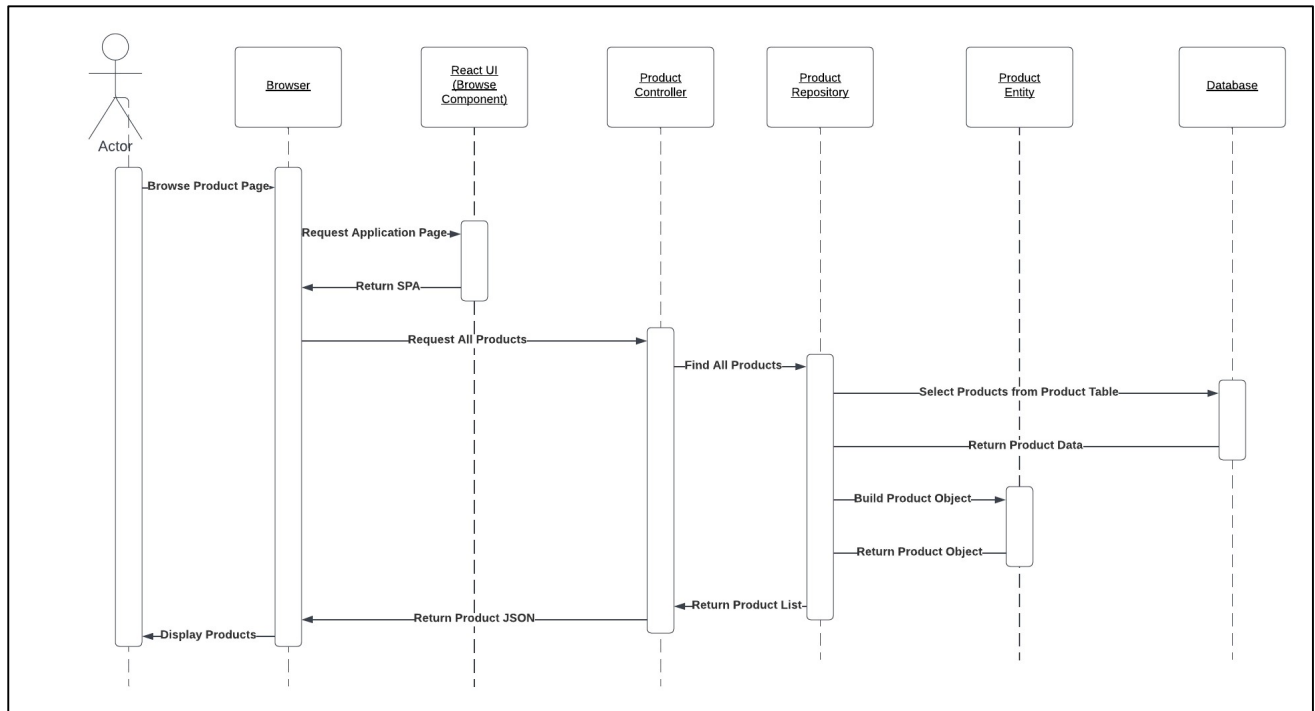
All current deliverable code was merged into the “phase1” branch on GitHub and has also been merged from there into the “main” branch.

b. UML design for phase 1. You must include the following diagrams:

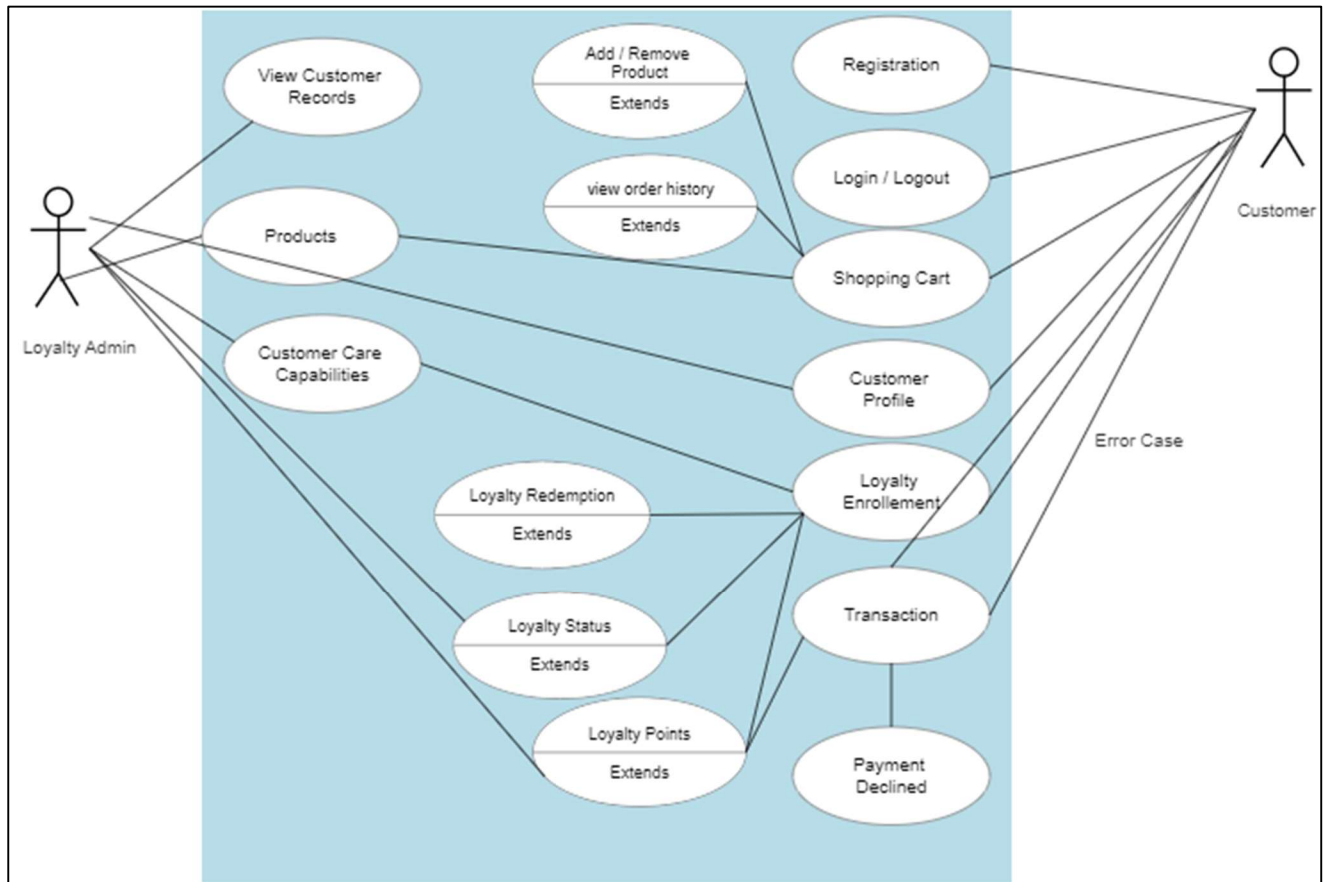
- Class diagram – Express status of classes, and interaction between classes.



- 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.



c. Test Cases (unit tests) for phase 1.

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 are able to retrieve real data from a sample in-memory database instance integrated into the API application.

API Test Cases

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.

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

- d. 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 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 e.

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 now, we are only supporting local launch demo configuration in support of ongoing front-end development, as will be described below in section e.

- e. **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:

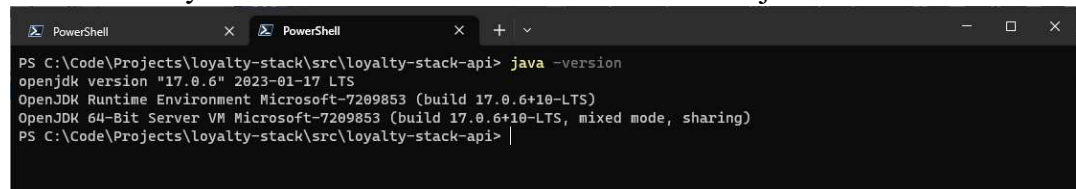
- Be sure to have a compatible JDK (JDK 17) installed. We are using and recommend the Microsoft build of Open JDK 17, found here:

<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**

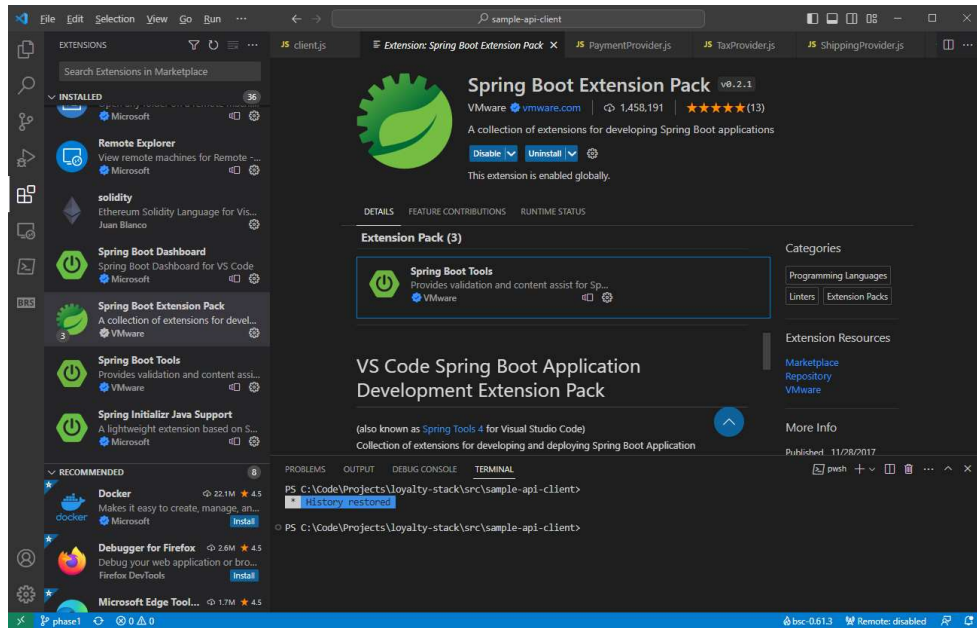
This file is an installer that you can run after downloading which should fully install and configure the JDK on a Windows system. There are similar builds available from the download link for other operating systems.

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

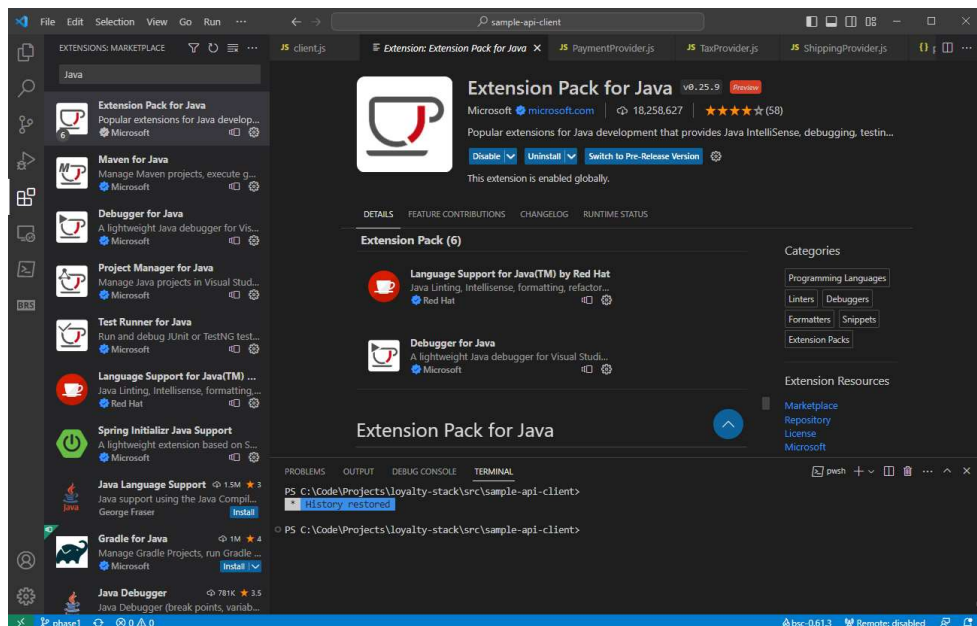


```
PS C:\Code\Projects\loyalty-stack\src\loyalty-stack-api> java -version
openjdk version "17.0.6" 2023-01-17 LTS
OpenJDK Runtime Environment Microsoft-7209853 (build 17.0.6+10-LTS)
OpenJDK 64-Bit Server VM Microsoft-7209853 (build 17.0.6+10-LTS, mixed mode, sharing)
PS C:\Code\Projects\loyalty-stack\src\loyalty-stack-api>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

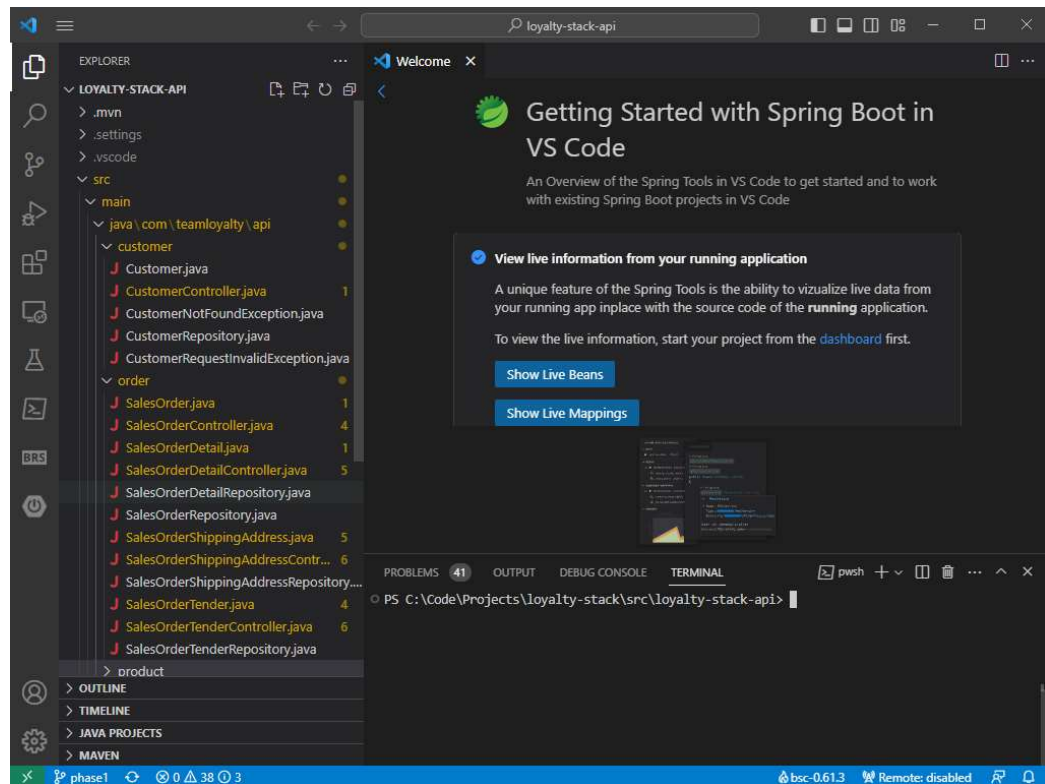
- The Spring Boot API is a Maven-based Spring Boot project. It is compatible with multiple IDEs and was primarily developed with a current version of Eclipse. However, for simplicity we recommend Microsoft VSCode for compiling and launching. Please install VSCode for your operating system from: <https://code.visualstudio.com/>
- 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.



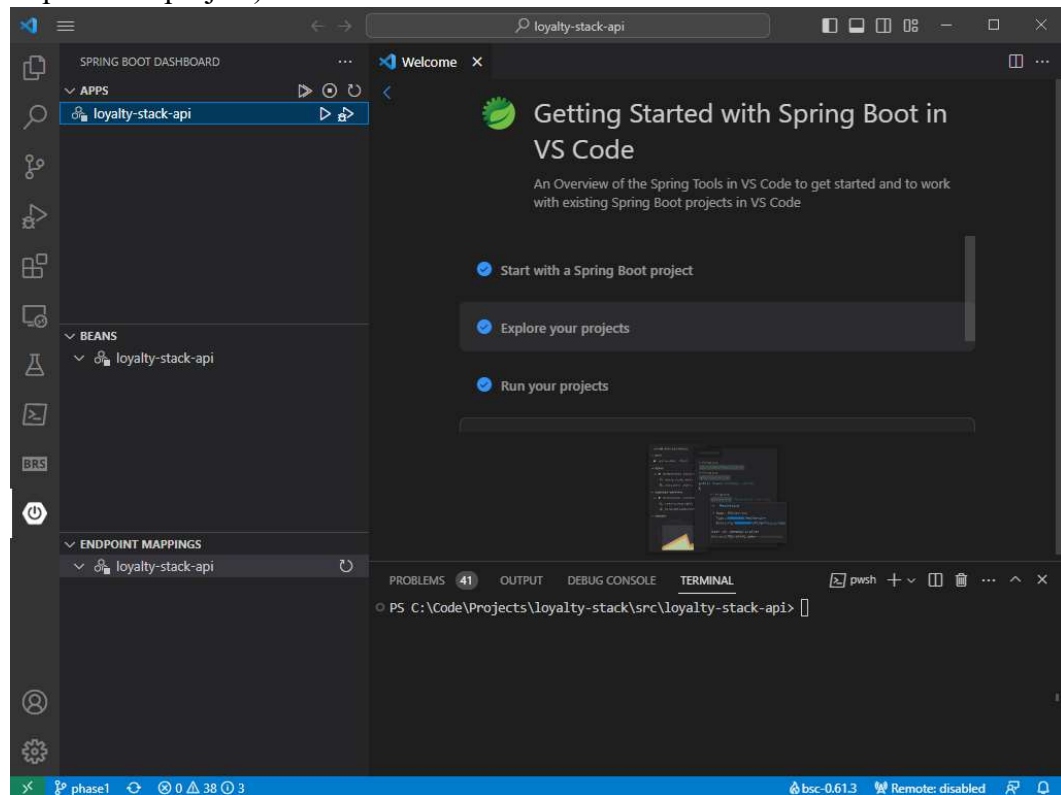
- We also recommend the Extension Pack for Java



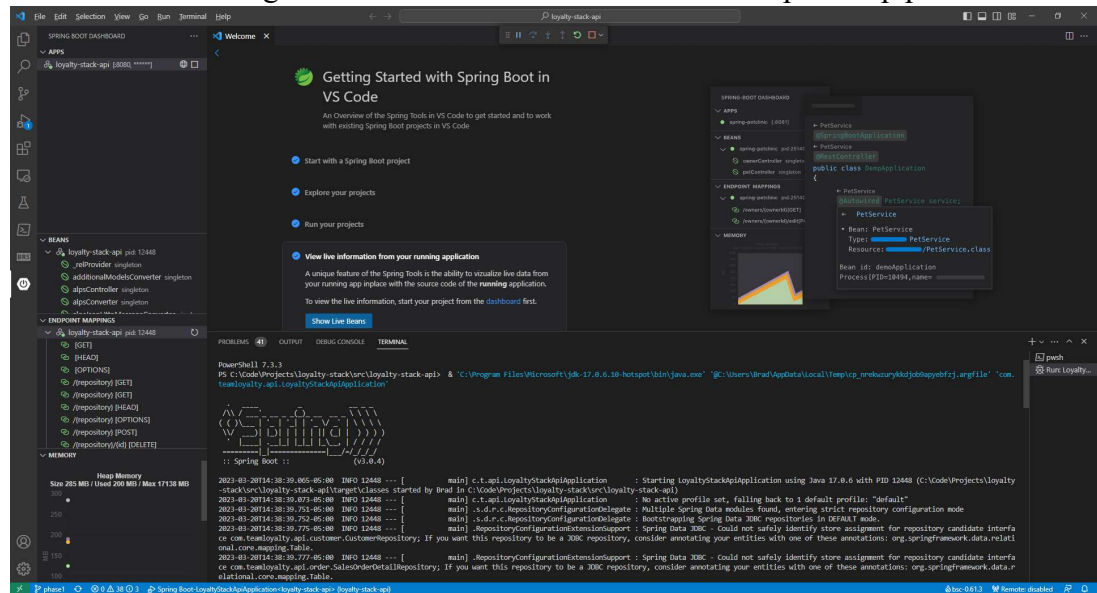
- Do a “git clone” of the full loyalty-stack repository to a local directory.
- Under the top-level repository folder, the Spring Boot API project can be found under src/loyalty-stack-api. This is a fully contained Spring Boot project. Please open just this folder directly in VSCode. You can right-click 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:



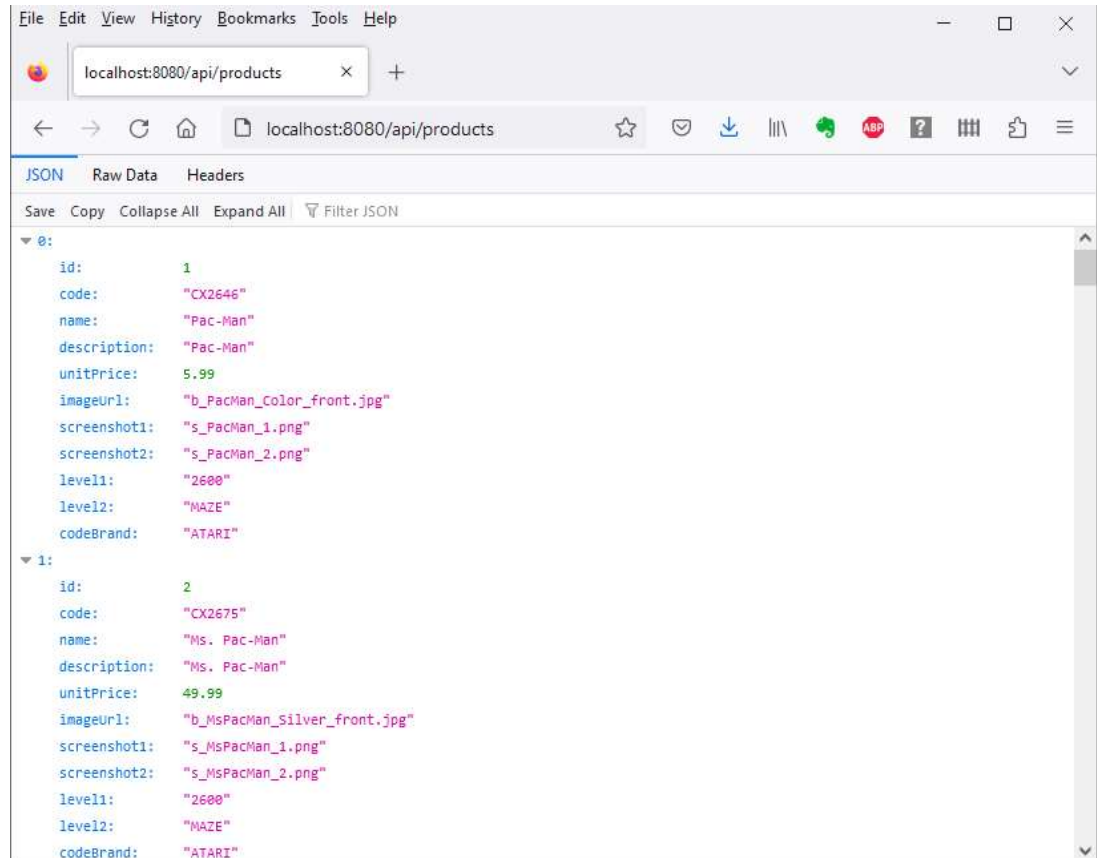
- On the left hand side the last navigation icon toward the middle of the screen accesses the Spring Dashboard. Click it to get this view (may take a few minutes to parse the project):



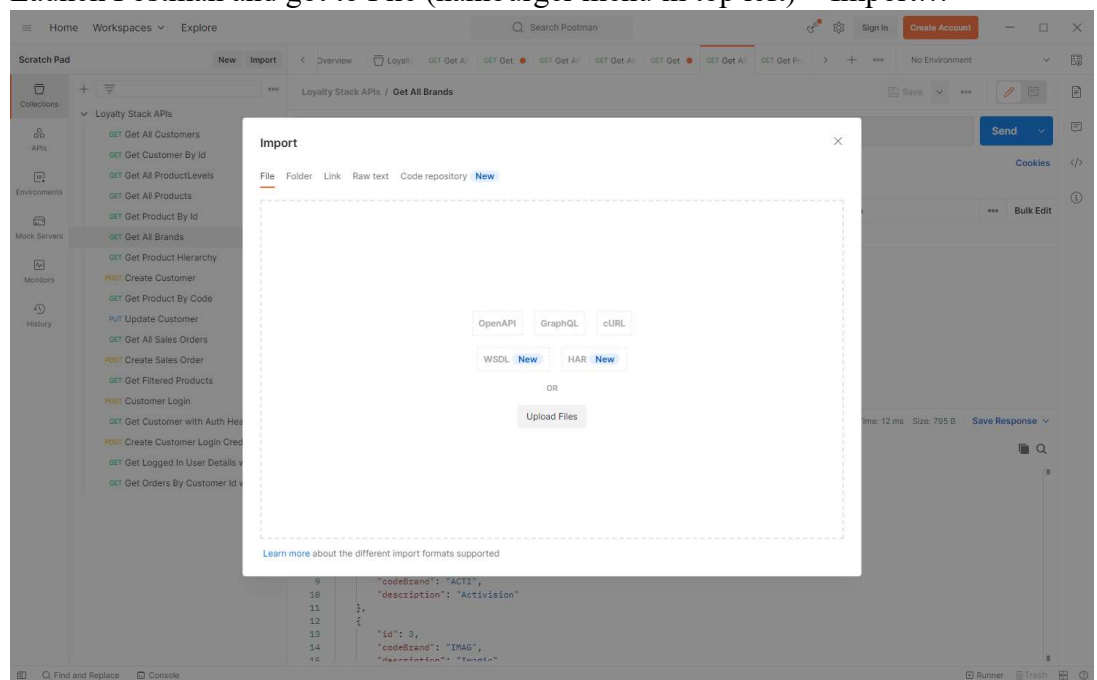
- In the top portion of the Spring Dashboard you can highlight the APP `loyalty-stack-api` and click the “Play” button to launch/run the API. You will see a banner and diagnostic text scroll through the terminal in the lower right corner. The API has an integrated Tomcat server and should come up on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



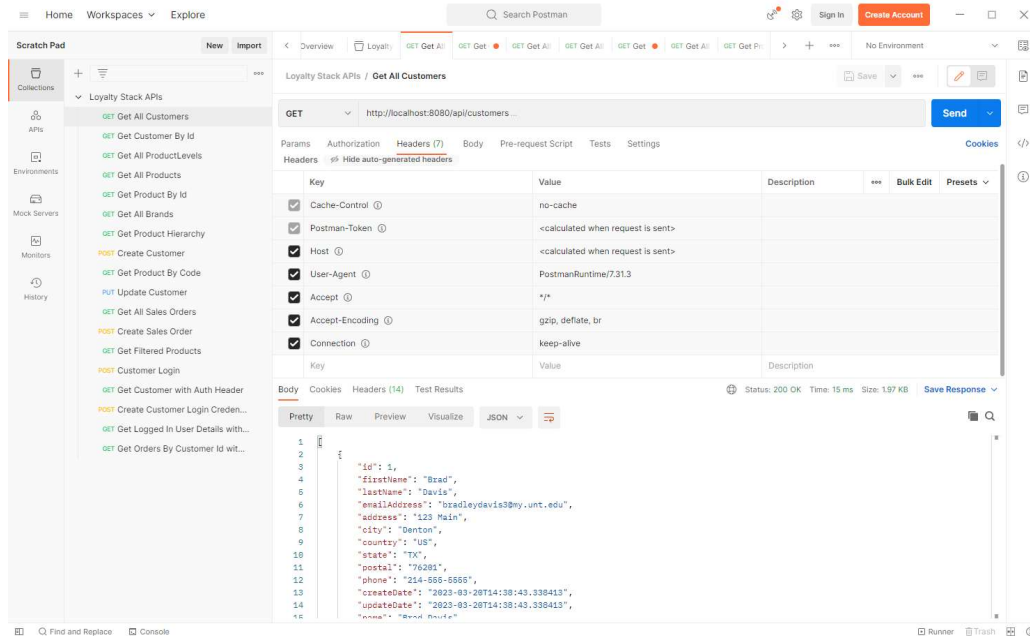
- Now you can do a simple validation that the API is running and responding. Be sure to allow access to any local firewall warnings and alerts. Place the URL <http://localhost:8080/api/products> into a web browser and you should get a response JSON message like so:



- 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!

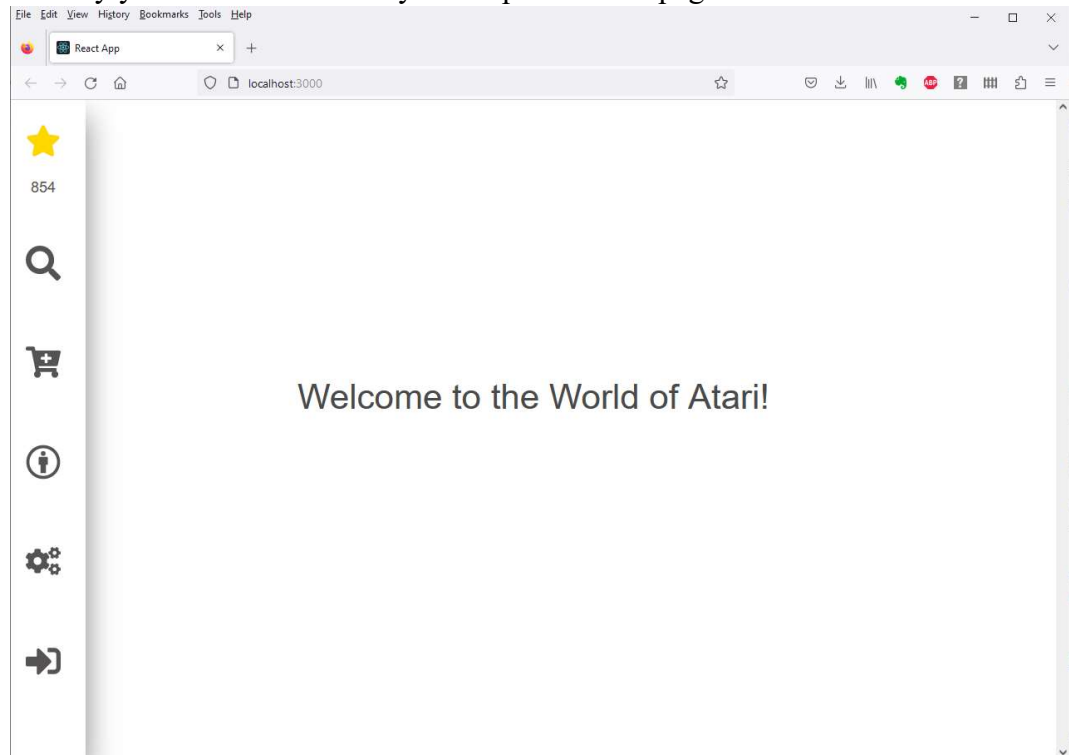
You can now view loyalty in the browser.

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

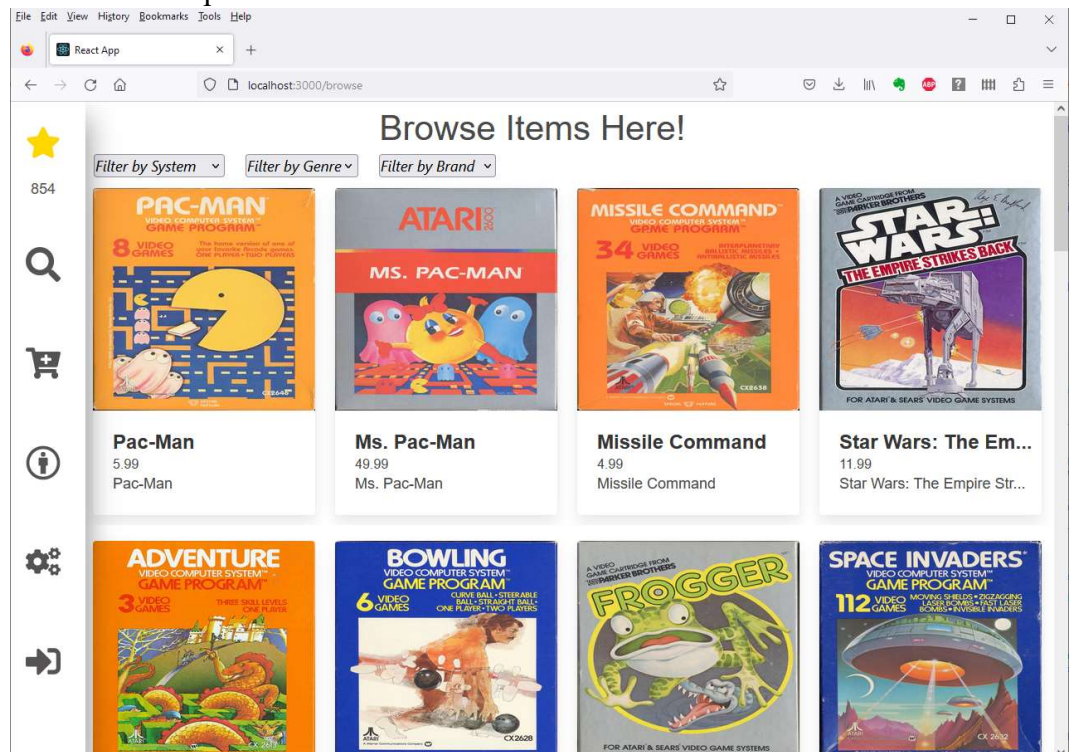
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
|
```

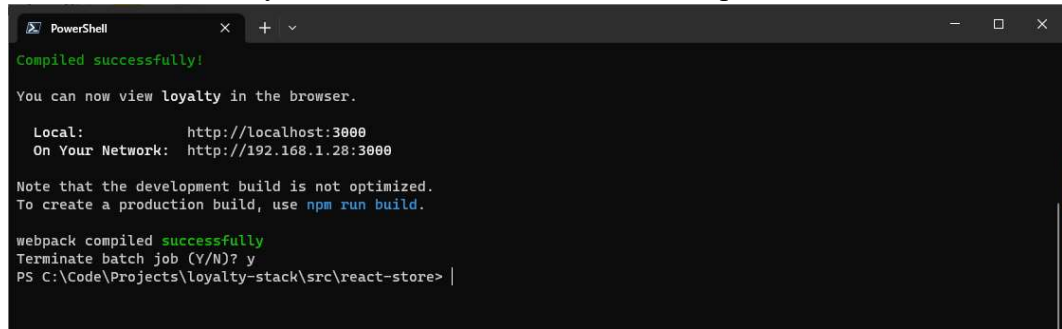

- Initially you'll see the currently incomplete Home page:



- By clicking on the Browse (Search) icon in the left nav bar you can currently view and filter product tiles retrieved from the API



- 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> |
```

f. A section that briefly describes feedback received during the peer review session and actions taken based on the feedback.

During team meetings we reviewed the layout of the API features and discussed an approach to organize a simplified e-commerce website around a few basic views or pages: Products lists and individual Product pages, Customer profile creation/management, Customer Login, Shopping Cart and Checkout. Future phases will add Loyalty aspects to this list.

We also discussed the approach to navigating the sight, the layout of navigation and the necessary elements. This is still a fluid area of design subject to change going forward.

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

The API supporting phase 1 was substantially completed and fully functional, although it lacks hardening for missing or erroneous data. The foundational UI application was started but not advanced very far. To make more progress the team needs to become more familiar with and understand the API model a little better. Some team members still need to become more familiar with the implementation technologies and philosophies (React, Spring Boot, REST), although that is perhaps best done by coding and doing.

Progress on the UI should accelerate in the next week or two, and phase 2 is light on new UI requirements anyway, introducing primarily back end and API loyalty features.

Team Member Contributions

Team Member	Contributions
Brad Davis	API design and implementation, Postman unit tests, sample data
Sandy Martinez-Echegoyen	React product page, filters, stylesheets
Tyler Parks	React app navigation bar, component pages
Vishpendra Chahar	UML diagrams, meeting minutes
Rama Reddy Venkata	React coding, testing