TEAM LOYALTY PROJECT PROPOSAL

CSCE 5430.004 (SPRING 2023)

PROJECT INFORMATION

Project Title: eCommerce Loyalty Stack

Group Name: Team Loyalty

Group Members:

Student Name	GitHub Username
Bradley Davis (Leader)	*lightsixer*
Tyler Parks	*Tyler Parks*
Sandy Martinez-Echegoyen	*sandymarech*
Vishpendra Chahar	*Vishpendra Chahar*
Rama Reddy Venkata	*ramatvrr14*

PROJECT DESCRIPTION

As hinted at in the name of the group, Team Loyalty's project will be an implementation of a working commercial enterprise loyalty stack, integrated into a mock eCommerce application site. The main UI and customer experience will be a conventional Internet shopping website, selling a range of products yet to be determined. However, the capabilities of the product to be delivered by the team are a set of loyalty features designed to incentivize and reward customer behaviors.

CORE BEHAVIORS

The initial design phase will be focused on identifying a baseline set of loyalty features for a minimum viable product (MVP). Customers will of course need the ability to register their personal information and create a loyalty/shopping account through the eCommerce website. This will include the ability to capture customer PII (personally identifiable information) data, as well as establish a set of secure personal login credentials for authenticated access to the customer's profile. Additionally, the customer will need to be able to manage their profile after creating it, such as by updating or adding personal information. The loyalty system will need to establish a primary identifier for the loyal customer, such as a unique loyalty number to track enrollment status of a loyalty account (active, closed, other options) and date of registration.

Customers can earn loyalty points for their eCommerce purchases. Typically, this is accomplished through a rule-based engine, and implementing one will be a critical component of the project. In most programs, customers can earn a fixed number of base points per eligible dollar spent. Eligible dollars, or eligible revenue, is typically a subset of net revenue. Net revenue is dollars spent on services and merchandise, excluding taxes, fees, shipping and handling, and other recovered costs. It may also exclude specific categories of products, such as gift cards, charitable donations, and certain highly regulated or restricted products (alcohol, tobacco, lottery, etc.). For an

MVP, we can probably avoid most of this complexity, but at minimum, we will need to address net revenue versus gross revenue, including taxes and shipping, in a standard eCommerce transaction.

Additionally, the engine should allow the eCommerce site to issue bonus points for specific products or product categories during specific windows of time for promotional and customer incentive purposes. These types of bonuses could be threshold-based (spend $50, receive 1000 extra points), point multipliers (receive 2X points on product XYZ), or additional points per dollar. Depending on implementation complexity of the points engine we may limit the variety of these options for the MVP to one or two use cases.

Obviously, points are no good to a customer if they can't use them. A critical feature of the MVP will be the customer's ability to redeem their points for something of value, such as a reward. Most often loyalty programs in the retail sector allow a customer to convert their points at a fixed rate to a reward discount certificate or coupon. This can be on demand or converted automatically as the customer reaches a certain threshold of points. For example, a program could give a customer a $5 reward for every 1000 points earned. The reward can then be redeemed as a discount in their basket on their next purchase. Other programs may allow customers to redeem from a catalog of physical and digital redemption items, gift points to friends, family, or charities, or exchange them with other loyalty programs. For the MVP we will target basic on-demand conversion of points to dollar discounts.

Finally, there are customer-facing experiential elements to the program. For an MVP the eCommerce site should feature a special banner or loyalty status page/dashboard for members. Standard information shall include their personal data (name, email, etc.), membership card number, status (active, elite status, or a higher tier based on spend threshold), and their current available points balance. The point balance is critical and non-trivial to implement. It involves bookkeeping for newly issued points earned and points deducted for redemptions, and accuracy of points balances, and liability is non-negotiable for businesses. Additionally, the customer may be able to see a history of their own purchase transactions, points earned by date and source, and points redeemed.

STRETCH GOALS

Many of these may prove unlikely or unrealistic for a single semester project, but depending on the team's development speed and capacity, there are numerous optional capabilities to create a much more enhanced loyalty product. This list is by no means exhaustive but should prove that there will be no shortage of work for the team for the duration of the semester.

LOYALTY TIERS

Many retail loyalty programs have different tiers of membership to provide premium benefits and higher points earning rates to customers who spend beyond a certain threshold per year. Two or three tiers with a program is common.

GAMIFICATION AND ENGAGEMENT

Loyalty programs often like to create ways to engage their customers in their brand in ways that are not transactional. This can be a vehicle to either foster spend and share of wallet, or just to gather more customer data. Programs can find ways to incentivize or award points for customers who take surveys, quizzes and polls, interact with the brand on social media, check in via geolocation and mobile apps, or complete challenges to earn badges and recognition. Certain activities could have leaderboards, avatars and public profiles. Giving customers a small bonus for signing up with the program is also common.

LOYALTY RELATIONSHIP FEATURES

Often a points program will allow customers to link multiple accounts together within their control or with members of their household or close friends for the purposes of pooling their points together. A linked points pool can then be controlled either exclusively by the primary account holder or setup so that any member of the pool can redeem. Another option, typically via customer service, would allow a member with multiple accounts to merge and consolidate them together, particularly when each account has a balance of points or rewards and some activity history. Another capability featured in some programs it he ability to gift or transfer some earned points from one member account to another. And some programs will of course allow members to donate points to charity.

MOBILE EXPERIENCE

Many programs feature either a white label or fully custom mobile app for their program, which allows members on the go or in store to conveniently check their account status, points balance, available rewards and offers, and even to make certain types of redemptions.

PROGRAM ADMINISTRATION AND SUPPORT

A program of any size requires both management of the program (altering rules, bonus points configurations, creating new offers) as well as a customer care function (typically a call center or support inbox). To facilitate this there is typically a set of non-customer-facing application, usually web-based, that feature program configuration UIs as well as customer care capabilities to look up and review member customer accounts, view their point balances and transactional histories and to even appease unhappy customers with points and rewards.

BLOCKCHAIN INTEGRATION

In theory the bookkeeping for issuing and redeeming loyalty points could be implemented on blockchain technology as a type of token for the brand, allowing customers to earn a form of cryptocurrency in their digital wallet by engaging with the program. Other options could be the redemption of points for some other form of cryptocurrency or for brand NFTs.

REPORTING

A typical retail loyalty program with need to provide some form of standardized reporting to various types of stakeholders. Critical concerns include membership numbers and new enrollments, points issued, points redeemed, and total points liability, among other things.

TECHNOLOGY & IMPLEMENTATION

The team will implement the Loyalty Stack project as an N-tier architecture (N-tier architecture style n.d.). Such architecture, also referred to as multi-tier architecture, features separate logical layers and sometimes separate physical tiers for different components of the solution. In the most conventional approach with a closed layer architecture, each logical layer can only communicate with the next layer immediately down in the stack.

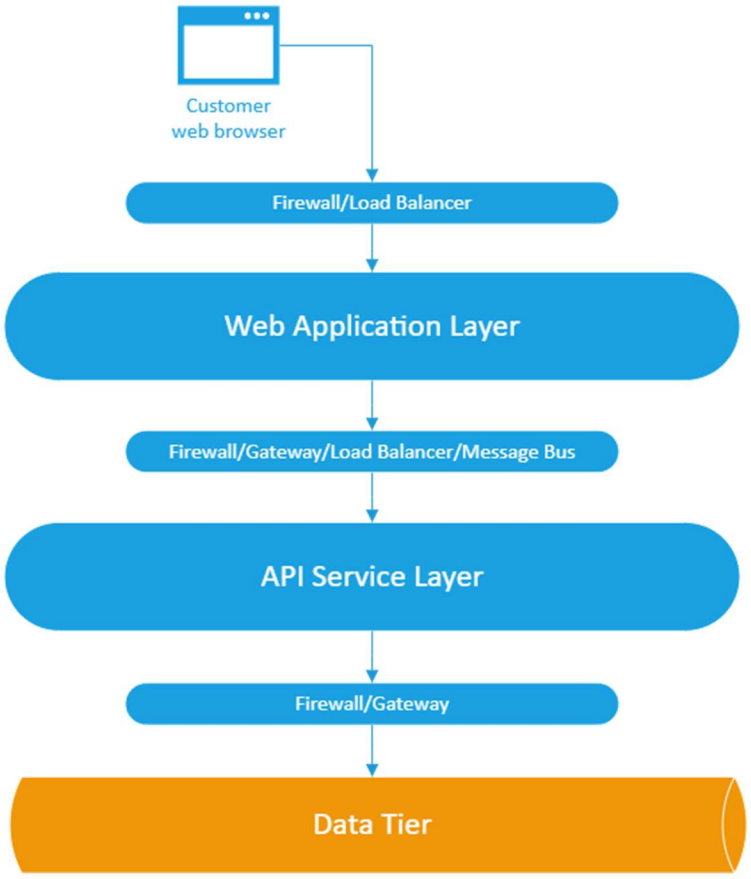


Figure 1. N-tier Application Architecture

The lowest layer would be the data layer or tier, responsible for storing and providing authenticated access to the most asset of the solution, the application's data. This layer is usually highly protected and not directly accessible from the application's user-facing elements. Above the data layer would be an application layer or "middle" tier. This usually takes the form of a logically and physically separate set of API services that mediate and control access to the data layer and implement substantial business rules and logic. At the highest level is the presentation layer, which can take the form of one or more user-facing applications at the same level, such web sites, mobile applications or even desktop software. Sandwiched in between these major logical layers and tiers can be several infrastructure-oriented component layers, such as service buses, message queues, caches, load balancers, gateways, firewalls, etc. For our project we will try to keep such complexity to a minimum, but we will adopt a logically separated web application layer, an API service layer, and a data layer.

WEB APPLICATION LAYER

For the front-end we will be building a mock eCommerce website. This web application will be programmed primarily in JavaScript (ES2022) [Add References], and necessarily rendered as HTML5 and CSS3. However, the application will be built using the React framework running on Node.js.

JavaScript is the language of the Web and React is an extremely popular and well-established framework for component-based development for web experiences. Some of our team members are already experienced in React, while others should hopefully find it an approachable and valuable learning opportunity. JavaScript is a

dynamic, loosely typed scripting language that runs at high speed in modern web browsers and features a C-like syntax.

Furthermore, we will be evaluated for a React-based eCommerce framework, preferably free and open-source, to accelerate the implementation and standup of the base eCommerce experience for the project, on top of which we will layer the custom loyalty capabilities. Ideally, the eCommerce framework will provide mock implementations of products and inventory management as well as payment processing, which we can hopefully re-use and modify as needed, and allow us to build custom connectors to our loyalty API stack. Key elements of the customer-facing loyalty experience will be implemented as custom React components.

SERVICE LAYER

To provide loyalty capabilities to the eCommerce website we will implement a set of RESTful APIs (application programming interfaces) in the service layer. An attempt will be made to adopt the REST style as fully as possible, including leveraging HTTP verbs, JSON request/response payloads, and HATEOAS [Ref]. The service layer will provide APIs for all the core capabilities and business logic of the solution, including creating and managing customer profiles, evaluating purchase transactions for the awarding of points, providing point and activity history, converting points to rewards, redeeming rewards and any other features that rely on business rules and CRUD operations.

The chosen service layer implementation language is Java, using OpenJDK 17, the current LTS release version. Additionally, to facilitate rapid prototyping and build of RESTful services we will use the Spring Boot framework, including Spring Web for J2EE-style MVC approach to building RESTful services, Spring Data for data layer persistence, and elated libraries for HATEOAS, Entities, Logging, etc. In addition to providing a stateless RESTful API, the service layer will also need to feature some basic security and access control to authenticate and authorize application use of the APIs via some industry standard such as JWT or OAuth 2.0, whichever can be implemented more simply for our purposes here.

Java has a massive presence in industry software development, and the Spring framework is dominant for business applications, microservices, cloud computing, and numerous other use cases. The Java language syntax is in the C family of languages and should be comfortable for students experienced in C or C++. Java is a strongly typed, structured compiled language with a runtime virtual machine, security, automatic garbage collection and memory safety and a massive class library. It supports coding at a higher level than C or even C++, has many of the most advanced and modern language features, runs extremely fast on modern implementations, and is ready-made for modern web-centric enterprise applications Not only is the Java and Spring Boot combination a valuable toolset to rapidly build high-quality RESTful services, but experience with both is highly valuable in the industry.

DATA LAYER

The Data Layer will be implemented as a SQL-based RDBMS, specifically on the current version of PostgreSQL. Although there are modern alternatives like NoSQL and distributed databases, for the style of application we are building SQL presents a mature technology that is approachable, easy to use, and a valuable skillset. PostgreSQL is free and open source, has a large support community, features standard ACID capabilities, is performant and is easily containerized. We can design an ERD for our logical data model and then easily implement it in PostgreSQL with commonly available SQL tools and bind it to entities in our service layer.

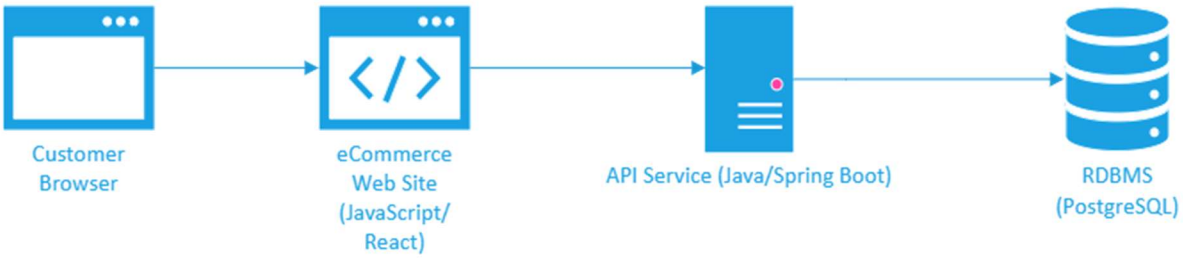


Figure 2. Logical Project Application Architecture

PLATFORM & DEPLOYMENT

The technologies chosen are all platform-agnostic. The different layers of the solution will be containerized and could most easily be deployed on a virtualized Linux instance of almost any flavor. An Ubuntu-based OS would be the most likely choice. Containers for both the service layer and the data layer, and the Web layer at deployment, will be packaged and built with industry-standard Docker technology [ref].

In addition to easing the management and deployment of dependencies in a cloud-hosted environment, containers will also make it easy for developers to get the full stack up and running on their machines without substantial software installation and configuration. For example, front-end developers could simply download and launch the latest versions of the service and data layers on their local machines and quickly start to focus on coding just the customer-facing Web experience. These containers could be similarly deployed with ease to some shared development instance. The configuration and building of containers will become part of our building process.

The source code control system will of course be by Git, and specifically a repository on GitHub.

Link to GitHub repository:

https://github.com/lightsixer/loyalty-stack

Project planning, tracking and management will occur on a Kanban board established on Trello.

Link to Trello/Kanban:

https://trello.com/b/EJKI3Vdw/kanban

For some project management and collaborative creation of project documents and artifacts we will use Microsoft Teams, although many artifacts will make their way to the Git repository in relatively final form.

Link to Teams:

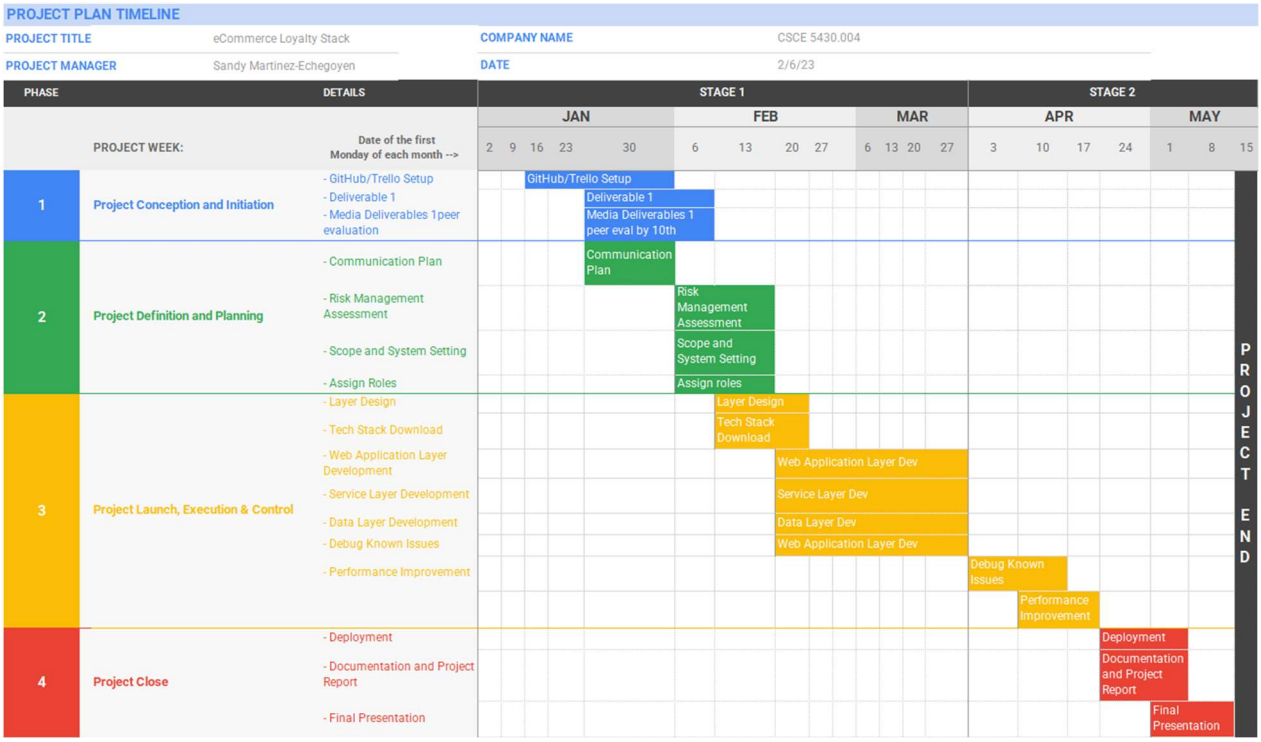
https://teams.microsoft.com/l/team/19%3auBy7rK2W1Mgugk_iPeqdTP4FriveOoqFQDypSYT_ERc1%40thread.tacv2/conversations?groupId=cc3764b7-0de4-424f-b3ee-94039ec6d614&tenantId=70de1992-07c6-480f-a318-a1afcba03983

Although recommendations will be made, developers on the team may use their own choice of tools depending on their comfort or familiarity. There may be a preference of one tool or another for Web vs service layer development. Microsoft's free VSCode editor features numerous language plugins, is lightweight and very popular in industry now – it should be useable for almost any aspect of the project. Eclipse is another strong choice,

particularly for developing Java code for the service layer. Microsoft Visual Studio or JetBrains IntelliJ IDEA are also viable options. For SQL access and database administration there are powerful open-source tools such as pgAdmin or more generic JDBC-based tools such as DBeaver. Most of these tools are available across platforms, and developers should have no issue working and coding on Linux, macOS or Windows.

PLANNING

Please find our proposed Project Plan Timeline below. For a more detailed view, click on this link.



RISK MANAGEMENT

TOP 3 RISKS

1. Over-extended Scope

   The scope of this project is large and ambitious, as it should be. Our group plans to execute and deliver a functional web storefront, layered with loyalty features; however, we also plan for scope constraints in that not all projected features and capabilities will be completed.

2. Learning Curve and Technical Challenges

   Depending on the technologies we choose to build our project, our group takes the risks of performing on a shrunken timeline to make room for learning said new technologies.

3. Exceeding Budget 'Hours'

   As a group of five members, the raw number of hours we will be able to put into our project will be large, but possibility not large enough to complete the project in its entirety.

RISK MONITORING

To monitor and mitigate risks to our milestones and overall project, each team member will need to keep track of a list of pending, completed, and in-progress action items. This list will likely be created on a collaborative document so that each team member has access to each other member's workflow. This will promote group member accountability and provide transparent checks-and-balances between members.

An example of this could look like the following:

Team Member	**Action Item**	**Date Due By**	**C/I/IP**	**Verified By**
Member 2	Build Website	3/12/23	IP	Member 1
Member 2	Finish Website	4/6/23	IP	Member 3
…	…	…	…	…

RISK REEVALUATION

At each major milestone, our group will be able to reassess and redefine current risks, objectives, and plans. At these milestone intervals, we'll be able to track how much and what work has been completed; as well as any action items that slipped through the cracks. Being able to track each team member's contributions will enable each of us to reinforce each other's work habits and ensure overall project completion.

RISK MITIGATION AND CONTINGENCY PLANS

To mitigate and effectively combat risk fruition, our group will need to take steps to secure our project in case of an unexpected obstacle. This could include a loss of a team member, lack of contributions from one or more members, timeline challenges, scope issues, and many more.

Our plan to address these issues is one of cooperation and timely assessment. If any issues should arise during project development, each team member should identify the problem situation and cooperate with other members to determine the best path forward. This might include splitting and distributing additional action items in order to finish the project on time.

TEAM MEMBER ROLES

Role	Team Member(s)
Project Manager	Sandy Martinez-Echegoyen
Requirements Lead	Tyler Parks
Design Lead	Brad Davis
Implementation Lead (Front End)	Sandy Martinez-Echegoyen
Implementation Lead (Back End)	Brad Davis
Configuration Management Lead	Rama Reddy Venkata
Testing Lead	Vishpendra Chahar
Documentation Lead	Vishpendra Chahar
Demo and Presentation Lead	Tyler Parks
System Administrator	Rama Reddy Venkata
DBA	Brad Davis

MEMBER CONTRIBUTIONS

Team Member	Contributions
Brad Davis	Project Description, Technology & Implementation, Diagrams, Presentation Video
Sandy Martinez-Echegoyen	Project Planning, Planning slides, Presentation Video
Tyler Parks	Risk Management, PowerPoint Deck, Presentation Video
Vishpendra Chahar	Pert Chart, Presentation Video
Rama Reddy Venkata	References, Edit and Review, Presentation Video

REFERENCES

n.d. *N-tier architecture style.* Accessed February 1, 2023.

https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier.