E-Commerce Project

Full Project Explanation Document

Goal

The goal of this assignment is to create and deploy an E-Commerce website for a fictional client using the technologies you have studied this term. The type of web store you create is up to you.

How Your Project Will be Marked

This document lists the possible features that can be included in the e-commerce store you are building for your WEBD-2007 final project.

Each feature that you implement is worth a percentage of your project mark. Features are categorized into three levels of difficulty, worth 2%, 4%, 6% or 8% per feature.

Failing to implement a feature marked with a star ★ will result in a 3% deduction.

For example, a student would be awarded a final project mark of 80% if they:

● Complete twenty-one 2% features	(42%)
● Complete eight 4% features	(32%)
● Complete two 6% features.	(12%)
● Fail to implement two starred features.	(-6%)

Keep track of your project progress using this google spreadsheet. It's recommended that you make a copy of this spreadsheet to your google account. Downloading the spreadsheet as an Excel spreadsheet is untested and not recommended.

The project itself is worth half your grade in this course. A minimum of 50% on this project is required to pass the course.

You should strive to complete at least one feature every day so that you are not swamped with work by the end of the term.

When Projects Will be Marked

Projects will be marked during class time starting the week of November 16th to 20th. The final version of your project must be submitted to Learn by **Sunday, December 6th** at 11:50pm. The final in-class marking session will occur during your final class the week of December 7th to December 10th.

Your project mark will be based *only* on marks you receive during in-class marking.

During an in-class marking session you will demonstrate your project's features to your instructor. For each feature demonstrated your instructor will determine if that feature will be marked as completed or not. It is your responsibility to come prepared to a marking session with a list of the features you wished to have marked. This list should include the feature number and the feature text from the list below. In order for a feature to be considered complete, you must have spent sufficient time and effort on its implementation. When in doubt, check with your instructor.

IMPORTANT: For features listed with a green number you must read over the extra clarifications listed for that feature at the end of this document before you implement it and again before you ask to have it marked.

With this marking process you are accumulating marks throughout the process, or in video game terms you are levelling up your mark. After any of the in-class marking sessions you will know your current project mark.

Epic Failure

Mistakes are how we learn. So let's make some.

In the spirit of encouraging experimentation, I will be awarding marks for well-documented failure on project requirements. If you struggled for 2 or more hours without success while implementing a feature, configuring a gem, or deploying your application, and you have the git commits to prove it, I will award you half of the marks associated with that feature. If you later complete the feature you can end up with marks both for the failure and for the completion.

I will award marks for epic failure for up to two features on your project.

Epic Failure Documentation Requirements: A commit history that shows a 2+ hour code battle, with enough commits to show that you tried various different solutions and strategies. The commits should be regularly spread out across the 2+ hours, and the commit messages should describe all the assumed problems and solutions you explored during your battle. You will also be asked to show a list of all guides, tutorials, blog posts and stack overflow questions that you followed during your code battle, along with which commits link to which resource used and the outcome of each. In other words, your epic failure needs to involve research and must be well documented.

The List of Possible Features

*Requirements marked in **green** are subject to extra clarifications found at the end of this document. Be sure to read these clarifications **before** you begin to implement a feature.*

Above each group of features the type of user the feature applies to is listed. It is important that you pay attention to these details. A feature implemented for the incorrect type of user will not be considered complete.

You can keep track of your marks by way of this marking spreadsheet.

1. Product Administration

As an administrator I should be able to: (Worth 2% Each)

1.1 Gain access to an admin dashboard by providing a username and password. ✯
1.2 Add, edit or delete product listings by way of an admin dashboard. ✯
1.3 Add or update images associated with new or existing products.
(Or images for another suitable model.)
1.4 Edit the content of the website's contact and about page.
1.5 Create and maintain a list of product categories and assign categories to new or existing products.

As the website programmer you should be able to: (Worth 2% or 4% Each)

1.6 Seed your product database with products and associated categories. *2%*
1.7 Scrape your seed data for products and categories from a 3rd party website. *%4*
1.8 Extract your seed data for products and categories from an existing dataset, which isn't Faker. *%2*

Seeding your database from a data source or a web scrape is optional. When seeding from a data source you will need to create a minimum of 100 products and 4 categories. If you choose to invent your own products and categories, you only need 10 products.

2. Product Display

As a customer I should be able to: (Worth 2% Each)

2.1 Navigate through the available products by way of a front page. ★
2.2 Navigate through the available products by category.
2.3 View the details of any of the available products on their own product page. ★
2.4 *Two or more of the following.* Filter the products to see only the products that are:

- on sale.
- new.
- recently updated.

2.5 Products listing are paginated. (There's a gem for that.)

As a customer I should be able to: (Worth 4%)

2.6 Search through the available products using a keyword search *by category*. ★

3. Product Orders

As a customer I should be able to: (Worth 4%)

3.1.1 Add various products to a shopping cart saved in session. ★
3.1.2 Edit quantity of items in shopping cart and remove items from cart.

As a customer I should be able to: (Worth 4% or 8%)

3.1.3 Complete a checkout process after filling their shopping cart. ★ *8%*
3.1.4 Sign up for an account with a username and password. *8%*
3.1.5 Save their address details (including province) during or after sign-up. *4%*

As a customers who can submit orders I should be able to: (Worth 4% Each)

3.2.1 List all past orders along with the order details. (Can also be implemented for all customers from the admin dashboard.)

As an administrator with customers who can submit orders I should be able to: (Worth 2% Each)

3.2.2 Manually and programmatically change the status of outstanding orders (example: pending, paid or shipped) as required.
3.2.3 Change the tax rates associated with the various Canadian provinces and territories.

As the website programmer you should be able to: (Worth 6% Each)

3.3.1 Integrate a 3rd party payment processor like Stripe or Paypal such that you can actually receive credit card payments as part of the checkout process.

As the website programmer you should be able to: (Worth 2% Each)

3.3.2 Design the database schema and the order system code such that changes to product prices or changes to tax rates will not affect past orders.

4. Layout and Application Design

As the website designer you should be able to: (Worth 2% or in two cases 4%)

4.1.1 Create valid markup and CSS for all pages on the website.
4.1.2 Design a consistent look and feel for all pages on the website. *4%*
4.1.3 Implement location-based breadcrumbs up to three levels of depth.
4.1.4 Make use of Rails "View Partials" to DRY up your views. ✭
4.1.5 Use the SASS (SCSS) pre-processor for all styling rules instead of CSS, including the use of nesting, variables, inheritance and operators.
4.1.6 Use the SLIM pre-processor for all views instead of HTML/ERB.
4.1.7 Build your markup & styling with a CSS framework like Bootstrap, Semantic UI, Materialize or Bulma. *4%*
4.1.8 Your CSS is screen-size responsive such that your store works equally well on desktop, tablet and mobile devices.

As the website programmer you should be able to: (Worth 2% Each)

4.2.1 Use Rails validations in *all* of your models (except join models) to ensure that data submitted to the site by administrator and customers is present and correctly formatted. ✭
4.2.2 Implement a database schema that involves both one-to-many and many-to-many relationships.
4.2.3 Make use of a custom flash hash message after a redirect and use the session hash in some way.
4.2.4 Implement file uploads in such a way to support the automatic scaling of images into multiple sizes (example: thumbnail images, and large image) for use within your views.

As the website programmer you should be able to: (Worth 4% and 6% Each)

4.2.5 Implement a many-to-many relationship with a join table/model along with a way to work with this association in the admin dashboard without directly manipulating the join table. (See this optional lecture video.) *4%*

4.2.6 Enhance the user experience of your website by implementing one aspect of your user-facing views with a component powered by a Javascript framework like Vue.js or React. Run your idea by your instructor first before starting to see if it qualifies. *6%*

As the website programmer your should be able to: (Worth 8% Each)

4.2.7 Build your Rails application as a JSON API and use a Javascript framework like Vue.js or React for the front-end views. WARNING: You will need to do a lot of self-learning to implement this feature. The amount of work required for this feature is not necessarily well reflected in the 8 marks that it is worth. *(You can't get marks for both 4.2.6 and 4.2.7.)*

5. Source Control, Deployment, Testing, and Dependency Management

As the website programmer you should be able to: (Worth 4% Each)

5.1 Use git and github to keep your source under control with the ability to push from master to origin. 32 commit and three branch minimum. ★
5.2 Project implements image uploading using Active Storage. ★
5.3 Active Storage uploads are stored to Google Cloud or AWS S3.
5.4 You have installed the Rubocop gem and when run it lists *no* offenses for the code that you personally wrote. (Be sure to check the setup doc.)

As the website programmer you should be able to: (6% Each)

5.5 Using an end-to-end testing solution like Cypress you've written tests for the happy and unhappy paths of your login or checkout.
5.6 You can deploy your store to a cloud system like Heroku or AWS, or a VPS like Digital Ocean. (This should be a non-containerized form of deployment.)

As the website programmer you should be able to: (4% Each)

5.7 You've containerized your app using Docker and can launch this container on your laptop.
5.8 You've containerized your app using Docker and you've configured a remote server (or a cloud service) to host your containerized app.

6. Interesting and Inventive Additions

As the website programmer you have integrated an interested 3rd party service, tool, or library. *(Worth up to 8%)*

6.1 Your store incorporates one of the following services, tools, libraries in a novel way:

- Discord by way of a Discord Bot (Using DiscordRB)
- Twilio for Voice or Text Messages (Using Twilio-Ruby)
- Social Media Bot for FB, Insta, Twitter, etc.
- A CI/CD workflow using Github Actions (or with Gitlab)
- An email sending API like MailGun.
- Some other cool and inventive use of an API or service. Ask for instructor approval first, please.

You will receive 8% if your feature is unique amongst all students in your sections, 6% if you are the first to develop a popular idea, and 4% if your feature has been implemented by one or more other students in your section before you. Yes, it's a bit of a gamble. It's also a gamble because I'm not sure of how difficult these integrations will be. Tread carefully!

7. Project Management

As the project manager you should be able to: (Worth up to 4%)

7.1 Complete and submit the project proposal and review this document in person with your instructor.

As the project manager you should be able to: (Not worth extra marks)

7.2 Meet or exceed the milestone goals listed in this document. (Deduction for missed milestones detailed along with milestone dates in the clarifications at the end of this document.)

Marking Clarifications

*Requirements marked above in **green** are subject to the following clarifications.*

1.2 Your database must include at least 10 products with real names and actual descriptions. No key mashing or lorem ipsum allowed.

1.4 The contact and about pages must be editable from a web-form from within the admin part of your site.

2.2 Navigating by category usually means having a menu that lets you navigate to a particular category and see all the products that belong to that category. Your solution to this requirement should be different from your solution to feature 2.6.2 (search by category).

2.4 Implementing two or more of these filtering options will count towards **one** completed requirement.

2.6 Users should be able to search for products by keyword. Found products will contain the keyword somewhere in the product title or description. When searching the user will be able to select a category to search within using a drop-down HTML select. The search will still be based on user supplied keywords, but will be restricted to a specific category. There should still be a way to search through all products regardless of category. Your solution to this requirement should be different from your solution to feature 2.2 (navigate by category).

3.1.2 Edit quantity of items in shopping cart and remove items from cart:

- User can change the quantity associated with an item using a textfield and/or some up/down buttons.
- User can remove items from the shopping cart.
- Remove functionality must be independent of quantity editing.

3.1.3 A checkout process will involve the following:

- User provides their address details including province (if they aren't already saved).
- System will display an invoice for the product(s) the user wishes to purchase which includes taxes. (Tax rates will depend on the user's province.)
- The user's order and address details are saved to an orders and users table. The entry in the orders table is associated with the user's entry in the users table.

Actual payment processing is covered by feature 3.3.1.

3.1.4 User accounts should be implemented in a secure manner, with passwords being saved to the database in a hashed and salted format. Once a user has created an account there should be a way for the user to login using their password. Users should remain logged in until they log out and/or close their browser. It is recommended that you use the devise gem to implement the login functionality, rather than rolling your own custom implementation.

3.1.5 During the signup process or once logged in a user must be able to add their address including a province. The user's province must be saved as an association to a provinces table.

3.2.1 As a customer: There must be some way to review all of your past orders on the site. Past orders should be shown along with a list of the products ordered, taxes, and the grand

total.
 - or -
3.2.1 As an admin: Your admin backend must include some way to list all the customers who have associated orders along with a list of products ordered, the taxes, and the order grand total. *This admin process should not include manually looking up orders or customers or products by navigating to other pages.*

3.2.2 You must have already implemented a checkout process with credit card payment processing for these marks to be applicable. Your order status must be switched from unpaid to paid once the 3rd party credit card processor confirms the payment. Flagging an order as shipped can be done manually using your admin dashboard.

3.2.3 You must have already implemented a checkout process for these marks to be applicable. Not only must you be able to set the tax rates for the provinces, but your database must include all provinces and territories with the correct tax rates set for GST, PST and HST. These tax rates must then be used during the checkout process and the customer must see the taxes applied to their order when checking out.

3.3.1 Payment integration can be added using a 3rd party API and/or Ruby gem. You should be sure that the payment processor you are using supports some form of "sandbox mode" so that you can test out the functionality without having to transfer actual funds. Once the 3rd party has confirmed that the payment has gone through your code should mark the order as paid in some way. There must also be some way to associate orders/customers in your system with orders/customers in the 3rd party system. This is best done by saving the 3rd party customer id and the 3rd party payment id with your order. Stripe is the recommended choice of 3rd party API.

3.3.2 Your schema should be designed such that if prices / taxes changed and you were audited by the government you could still determine historical order details including the prices of products at the time of purchase, the taxes at the time of purchase, and the grand total at the time of purchase.

4.1.1 These marks are only available during your final marking session. The HTML from all your pages validates according to the W3C validator. Likewise for the CSS. The easiest way to test HTML validation is to install a validation browser extension:

- Validity for Chrome
- HTML Validator for Firefox

4.1.2 These marks are only available during your final marking session. There must be a consistent look and feel to all pages of your store. Your design need not be complex, but the look must be professional. If you can't imagine yourself shopping at this store based on your design, then you have not met this requirement. If your instructor would not shop at this store based on your design, then you have also not met this requirement.

4.1.3 Breadcrumbs do not need to have a memory or state using session. Not all sites will work well with breadcrumb navigation. A good use-case would be if you have a one-to-many category to products relationship. You can use a breadcrumb gem for this feature, but it'll likely be easier to implement this feature without a gem.

4.1.7 At a minimum your website layout should be built around your CSS framework's grid system, should include a framework-based menu navigation component, and at a minimum one other framework-based component that is used in various places throughout the design.

4.1.8 These marks are only available during your final marking session. You must have obtained required 4.1.2 (consistent look and feel) and 3.1.3 (order checkout) for these marks to be available. You must be able to demonstrate using a mobile device or your browser dev tools emulator that:

- Various aspects of your website reconfigure themselves to accommodate devices with various screen sizes.
- All aspects of your store including the menu and shopping cart must be usable and must look professional/polished for both desktop and mobile devices.
- Your full checkout process is usable and looks professional/polished for both desktop and mobile devices.

5.1 Commits should be spread out over the entire project timeframe. All commits messages should be descriptive of the actual committed changes. You instructor may ask to review your commit messages and the commit deltas. You must show that at least two features were developed in their own branches and merged into master.

5.4 Rubocop is a gem that tests Ruby projects to ensure that they conform to the Ruby and Rails community style guides. You must follow these instructions to test your project.

5.5 Your tests must cover the sequence of user interactions required to a) sign-up and login or b) fill up a shopping cart, modify cart, check out. As such the feature under test must be completed before you can receive marks for this feature. Tests must cover both the happy path and any potential unhappy paths. For example, for sign-up/login the happy path means the user is able to input all sign-up and login details without triggering any form validation errors. Example unhappy path tests would ensure correct user feedback is provided when sign-up fails due to username being taken or when login fails due to incorrect credentials.

5.6 All aspects of your project (aside from image uploading on Heroku) must function properly once deployed. As you add features to your project your deployed version must also be updated. You can deploy to a Linux based VPS (that you pay for) like Digital Ocean or use the free Rails hosting plan available Heroku or a similar service. Before deploying to Heroku it's recommended that you get your app working with a locally installed version of Postgres. A

VPS-based install is therefore recommended. Digital Ocean has a “one-click-install” option for Rails apps, but it involves much more than one click. You can also follow my Digital Ocean Rails Deploy Tutorial but it’s two years old, so beware.

The Github Student Pack comes with a $50 Digital Ocean credit. You can also use this link to get a $10 Digital Ocean credit. (Full disclosure: If you use the second link and end up spending $25 on future hosting, the not-for-profit Open Democracy Manitoba will receive a $25 referral credit.)

5.7 All aspects of your project must function properly once containerized.

5.8 When the container is deployed to a server or cloud service your project must be accessible via the internet by domain or direct ip. If you are hosting on a server, a reboot to the server should result in the containerized app automatically running without manual intervention. Deployment should not involve you manually starting your containerized app in any way.

7.2 The project milestones are:

- Project mark of **15+** during or before the week of **Nov 16th to Nov 20th.**
- Project mark of **35+** during or before the week of **Nov 23rd to Nov 27th.**
- Project mark of **55+** during or before the week of **Nov 30th to Dec 4th.**
- Project mark of **75+** during or before the week of **Dec 7th to Dec 10th**.

Each milestone that you miss will make it harder to level up your mark:

- **One Missed Milestone:** Marks received above 90 are worth half their normal value.
- **Two Missed Milestones:** Marks received above 80 are worth half their normal value.
- **Three Missed Milestones:** Marks received above 70 are worth half their normal value.
- **Four Missed Milestones:** Marks received above 60 are worth half their normal value.