Standard Course Project: Mini-Amazon

See the overall Course Project Overview (available from the Schedule section of the course website) for general information and important dates. This document is specific to the standard project option.

Overview

In this project, you will build a miniature version of Amazon. On this website, sellers can create product listings and an inventory of products for sale. Users can browse and purchase products. Transactions are conducted within the website using virtual currency. Users can review products and sellers who fulfill their orders.

This project is designed to help you divide work among a team of 5. Each of you will be responsible for implementing a subset of the functionalities described further below:

- Users Guru: responsible for Account / Purchases
- Products Guru: responsible for Products
- Carts Guru: responsible for Cart / Order
- Sellers Guru: responsible for Inventory / Order Fulfillment
- Social Guru: responsible for Feedback / Messaging

That said, you cannot work in isolation: you will need to collaborate closely on the overall database and website designs, and on integrating and testing your code together.

Functionalities

Account / Purchases

Basic requirements:

- A new user can register for a new account; an existing user can log in using email and password.
- Each user account has a system-assigned id. Other account information includes email, full name of user, address, and password. Users can update all information except the id. Ensure that email is unique among all users.
- Each account is associated with a balance. It starts out as $0, but can be topped up by the user. The user can also withdraw up to the full balance. In practice these actions would require some real payment mechanism, but it is not required for this project.
- Users can browse their history of purchases, sorted in reverse chronological order by default. For each purchase in this list, show a summary (e.g., total amount, number of items, and fulfillment status) and link to the detailed order page (see Cart / Order).
- Provide a public view for a user. It will show the account number and name as well as any other summary information you deem necessary. If the user also acts as a seller,

show also email, address, and include a section with all reviews for this seller (see Feedback / Messaging).

Possible additional features:

- Search/filter purchase history by item, by seller, by date etc.
- Visualize history of balances, spending amounts and purchases by category, etc.

Products

Basic requirements:

- There is a list of predefined product categories, and each product belongs to one category. At the minimum, a product should have a short name, a longer description, an image, and a price.
- Users can browse and search/filter all products. For each product in the result list, show a summary (e.g., image, name, average review rating, etc.) and link to the detailed product page. At the minimum, support browsing by category, searching by keywords in name/description, and sorting by price.
- A detailed product page will show all details for the product, together with a list of sellers and their current quantities in stock. For each seller, provide an interface for adding a specific quantity of the product from this seller to the user's cart (see Cart / Order). The page should also include a section showing all reviews for this product (see Feedback / Messaging).
- Users can create new products for sale. The user who created a product will be able to edit the product information.

Possible additional features:

- There are many more enhancements you can make to filtering/sorting a list of products: e.g., sorting by average review rating or total sales, filtering by rating, price, and/or availability of highly rated sellers, etc.
- A flat list of top-level product categories is constraining. Consider a hierarchy of labels, or in general a set of "tags" for labeling products. What is the process for creating and maintaining these labels?
- For the same product, different users in our system may create their own versions of this product in our system. Is there a process for "standardizing" products across sellers, and should we allow different sellers to charge different prices?

Cart / Order

Basic requirements:

- Each user has a cart. The detailed product page (see Products) will provide a way of adding to this cart. Each line item in the cart refers to one product from one seller with a specific quantity. A detailed cart page should list all line items (quantities and unit prices) and the total price. It should also provide ways to change quantities, remove line items, and submit the entire cart as an order.

- When submitting the order, make sure to check available inventories and balances. Beware that inventories and prices can be constantly changing. For simplicity of this project, we will update inventories and balances at the time of order submission. The buyer's balance will be decremented, and the sellers' balances will be incremented and inventories decremented. Once the order is placed, the cart becomes empty.
- The cart contents are persistent: i.e., they should remain after users leave the site and log back in.
- A detailed order page (from the buyer's perspective) should contain all the information that would have been found on the cart page, but with prices "final." In addition, for each line item, it should show if and when that line item has been fulfilled by the seller (see Inventory / Order Fulfillment).
- The entire order should be marked as fulfilled if all line items are fulfilled. Once submitted, an order cannot be changed by the buyer.

Possible additional features:

- Divide the cart into "in cart" and "saved for later" so that a user can check out certain items while keeping the others saved to be purchased at a later time. Items can then be added back into the cart.
- Implement promotional coupon codes for certain items, allowing a user to enter some code to get a discount on either a single item, a group of items, or their entire cart.

Inventory / Order Fulfillment

Basic requirements:

- A user who wishes to act as a seller will have an inventory page that lists all products for sale by this user. There should be a way to add a product to the inventory. For each product in the user's inventory, the user can view and change the available quantity for sale by this user, or simply remove it altogether from the inventory.
- A seller can browse/search the history of orders fulfilled or to be fulfilled, sorted by in reverse chronological order by default. For each order in this list, show a summary (buyer information including address, date order placed, total amount/number of items, and overall fulfillment status), but do not show information concerning other sellers (recall that an order may involve multiple sellers), and provide a mechanism for marking a line item as fulfilled. (Recall from Cart / Order that order submission automatically decrements the available quantity in the seller's inventory; so fulfillment should not further update the inventory.)

Possible additional features:

- Add visualization/analytics to the inventory and/or order fulfillment pages to show popularity and trends of one's products.
- Add analytics about buyers who have worked with this seller, e.g., ratings, number of messages, etc.

Feedback / Messaging

Basic requirements:

- A user can submit a single rating/review for a product. The submission link will be incorporated in the detailed product page (see Products). The user cannot submit multiple ratings/reviews for the same product, but can edit/remove any existing ratings/reviews by this user.
- A user can submit a single rating/review for a seller, provided that the user has ordered something from the seller. Incorporate the submission link in appropriate places in the website, e.g., the detailed order page (see Cart / Order) and the public view of a seller (see Account / Purchases). Again, the user cannot submit multiple ratings/reviews for the same seller, but an existing rating/review can be edited or removed.
- Each user should be able to list all ratings/reviews authored by this user, sorted in reverse chronological order by default. From this interface the user should be able then select ratings/reviews to update. Incorporate the link to this interface in user account view (see Account / Purchases).
- Produce summary ratings for products and sellers; pages or sections showing lists of reviews for products and sellers. At the very least, the summary needs include the average and number of ratings; the reviews lists should be sorted by rating or date.

Possible additional features:

- For an order placed by a user, and for any seller fulfilling this order, the user can start a message thread. This message thread is private to the user and the given seller. Messages should be listed in chronological order, and old messages cannot be edited or deleted. Links to these message threads will be shown in buyers' and sellers' views of orders.
- Upvote functionality for rating/reviews to allow certain reviews to be marked as more or less helpful. By default the top 3 most helpful reviews would be shown first, and then the most recent following these.
- Include the ability to submit a limited number of images in the reviews, and make these easily available to users on the website.

Putting Them Together

Basic requirements:

- Make sure all functionalities described above under different parts are put together into coherent designs, for both frontend and backend. The website needs to support smooth user task flows across clicks and seamless transitions between different parts.

Possible additional features:

- Develop some methods for recommending products based on past purchase history (both a user's personal history and the history across all users) as well as reviews. The recommendations can be shown in a "You may also like…" section when a user is looking at products, the current cart contents, or even the purchase history.
- In reality, orders don't get fulfilled instantaneously. Design a process by which sellers can indicate when items have been shipped and buyers can verify when items have been

received, as well as protocol for resolving issues (i.e., when orders are not fulfilled by a reasonable deadline, when they are returned, or when there are disputes). You might want to rethink when balances are actually updated in your system.

Getting Started & Getting Help

A simple skeleton project implemented using Python, Flask, and PostgreSQL can be found here: https://gitlab.oit.duke.edu/compsci316/mini-amazon-skeleton/. Please read the README.md to set up the project. Next, have everybody go through TUTORIAL.md to get to know the code base and the development process better. Unless your entire team is already well versed with source control and full-stack development, we highly recommend that you **go over the tutorial together first**, to make sure that everybody has a good, common starting point, before you start making any serious changes to the code.

The course website also provides a separate Flask tutorial under its Help section, with an example website built on the familiar beer drinkers' database (although it was coded in a slightly different style using an object-relational mapping framework).

Duke OIT/CoLab offers a wealth of relevant Roots courses/tutorials/workshops that you might find useful to this project.

What to Submit

Milestone 1

- `README.txt`: list your team members, state that you choose the standard project option, and pick a short, catchy name for your team.

Milestone 2

At this point, your team is expected to have set up your own project repo; everyone should have gotten the skeleton code to run in their own development environment and successfully completed the tutorial; and finally, you have collectively come up with an overall design for your database and website.

- `README.txt`: list your team members and their roles (see Overview), briefly summarize what each of you have done since the last milestone, and include a link to your GitLab/GitHub repo.
- `REPORT.pdf`: showing:

  - Your database design. A diagram would be helpful but is not required. List and describe all tables, constraints, and any assumptions you are additionally making or deviating from our standard assumptions.
  - A preliminary page-by-page design of your website, as well as how users interact with different parts. We are not expecting any fancy website mock-ups, just the content descriptions and logical flow/connections among pages.

Milestone 3

At this point, each of you should have completed the implementation of at least one backend API endpoint involving executing SQL, as well as the corresponding frontend elements for displaying results and interacting with backend API endpoints. The endpoint you implement will depend on which guru you are working as:

- Users Guru: Given a user id, find all purchases of that user.
- Products Guru: Given an integer *k*, find the top *k* most expensive products.
- Carts Guru: Given a user id, find the items in the cart for that user.
- Sellers Guru: Given id of a merchant/seller, find the products that are in their inventory.
- Social Guru: Given a user id, find the 5 most recent feedback they posted.

You also need to prepare a **short demo video** (< 3 minutes) of your entire project team demonstrating the working functionality of all APIs and frontend elements above, end to end. You can have one member controlling the app the entire time, but the person who coded each endpoint should be providing the voiceover. For each member, a demo may look something like the following (as an example):

- Type something into some textbox, press a button;
- Show and interact with the results;
- Type something else into the textbox, and rinse and repeat…

Submit the following:

- `README.txt`: as before, plus:
  - A link to your short video.
  - Where to find in your code the implementation of the required items above.
- `CODE.zip`: all your code.

Note on zipping for submission: You may crash Gradescope if you submit a big zip file with too many files. Therefore, when zipping directly from your working directory, try excluding unnecessary files, such as those in the hidden `.venv` directory. Or, submit the zip file that you download from your Git repository, which should exclude these files by default.

Milestone 4

At this point, you should be well under way towards completing your project. In particular, you should have successfully integrated some different pieces that you and your teammates have

worked on separately previously. Finally, you should also create a much larger database that can be used for more comprehensive testing and final demo. The database should be large enough to show how you cope with data at scale, e.g., pagination for lists with lots of contents. Instead of creating the contents by hand, you can write a data generator or scraper.

You also need to prepare a **second video** (< 5 minutes) demonstrating a working (though perhaps not complete) prototype involving both frontend and backend components.

Submit the following:

- `README.txt`: as before, plus:
  - A link to your short video.
  - Where to find in your code the implementation of required items above.
  - Each of you may optionally indicate whether you prefer to be individually scored on the final product (by default, you will **not** be individually scored). Your preference will be respected, except when the majority of the team prefers individual scoring, every member of the team will be individually scored.
    - Note that this is your last chance to indicate your preference. This deadline is intended to give everybody a chance to adapt. After this point, complaints of the form "my teammates did nothing" will not be considered.
- `REPORT.pdf`: as before, plus:
  - A refined database design, highlighting what has changed since the last submitted design.
  - An updated design for your website. Highlight what has changed since the last milestone. Again, focus on contents/flow instead of presentation.
  - A list of advanced features that you have implemented or plan to implement and the proposed bonus points for each of these.
- `CODE.zip`: all your code.

Final Report and Demo Video

For the final submission, prepare a video about ~10-15 minutes in length that demonstrates all functionalities of your website. We recommend the following flow for your video:

- Browsing as a guest (unauthenticated user)
  - Should succeed: browse product details; search products; rank and filter
  - Should fail: all sell/buy/comment actions; checkout
  - Flexible: add product to cart
- Register / login / logout / user profile
  - Should succeed: create a new user with a new email; log in with the newly created user; log out the current authenticated user; update user profile legally (e.g. change password)

    - Should fail: create a new user with the same email; log in with incorrect password; update user profile illegally (e.g. change email to an existing email address)
    - Also show: user input validation (e.g., cannot register with an ill-formatted email); prevention of SQL injection (e.g., quotes and other special characters are handled safely)
- Authenticated buyer
    - Should succeed: add products to carts / update carts; review products; checkout; view transaction history
    - Should fail: add unavailable product to cart; check out with insufficient balance (if any); accessing sensitive areas after signing out
- Authenticated seller
    - Should succeed: manage inventory (e.g. add/delete products, change quantity); view order history
    - Should fail: add same product multiple times
- Anything else you implemented or thought of during the demo

In addition, if you would like to be considered for the Audience and Staff Choice awards, please prepare an additional short video showcasing the highlights of your project (<5 minutes). This video can be more promotional in nature and does not need to show every detail of the project like your long demo video. We will post an Ed thread for sharing and voting on these short promotional videos.

Submit the following:

- `README.txt:` as before, plus:
    - A link to your complete demo video.
- `REPORT.pdf`: showing:
    - A refined database design, highlighting what has changed since the last submitted design.
    - The final list of features you have implemented or attempted to implement. For each feature, indicate its status: fully functional, buggy, partially implemented, or not implemented at all.
- `CODE.zip`: all your code.

Grading Criteria

For the four milestones, points will be assigned mostly based on completion or effort made towards completion. Your design may not be perfect and your implementation may still have issues — that's fine; we will give feedback but not take points off, unless there is clearly a lack of effort. The final product, however, will be graded according to its quality.

- Milestone 1: 3 points
- Milestone 2: 10 points (5 for database design and 5 for website design)
- Milestone 3: 15 points (5 for video plus 2 per endpoint)

- Milestone 4: 10 points (5 for video and 5 for the large testing database)
- Final submission:
    - Report: 12 points (7 for video and 5 for the rest of the report)
    - Final product: 50 points

See Functionalities for features expected on your final product. Here we provide additional guidelines on how to break down the 50 points.

Team Score Option

A project with all 5 parts working together meeting the basic requirements will get you a baseline score of 43/50. Going above and beyond will earn points for your whole team, up to the maximum score of 50/50. Starting from the baseline:

- (Basics) Buggy or missing features in the basic requirements for each team member's part: -0.5 to -8 points, depending on how serious the issues are. You lose 8 points for each part that is totally missing.
- (Integration) For each team member's part that has buggy interaction with other parts or is not integrated into the final website: -0.5 to -1 point, depending on how serious the issues are. You can lose up to 5 points for lack of integration.
- (Safety Penalty) Some code is susceptible to SQL injection attacks: -2 points overall.
- (Style Penalty) Not following good coding practices (e.g., unnecessarily hard-coding values that should have been obtained from the database; excessive duplication of code without refactoring): -0.5 to -2 points overall, depending on how serious/prevalent the issue is.
- (Performance Penalty) Unreasonably slow response time that could have been easily fixed (e.g., with appropriate database indexing, or by incrementally receiving and paginating results in the frontend): -0.5 to -1 point overall.
- (Interaction Design Bonus) Well-designed, smooth user interaction: +0.5 to +1 point.
- (Aesthetics Bonus) Well-designed and impressive-looking frontend: +0.5 to +2 points.
- (Documentation Bonus) Commenting your code and documenting your design/setup (beyond what's required in `REPORT.pdf`): +0.5 to +1 point.
- (Data Bonus) Well-designed large testing database with real and/or realistic data: +0.5 to +2 points.
- (Feature Bonus) Each additional feature: +0.5 to +2 points, depending on how impressive the added functionality is and how much effort it requires. Features that require extensive coordination among multiple teammates earn more.

You will not lose points by going with a plain-looking website coded in plain HTML. After all, this is not a course about web or frontend development!

Individual Score Option

Successfully meeting the basic requirements for your own part will give you a baseline score of 43/50. Going above and beyond the basic requirements, by yourself, can get you a maximum

score of 47/50. In other words, getting a total project score in the high A/A+ range does require your whole team's success! Starting from the baseline:

- (Basics) Same as the guideline above for the team score option, but limit to your part and multiply the penalty by 5.
- (Integration) Not applicable.
- (Feature Bonus) In addition to the guideline above for the team score option, you can also earn bonuses for implementing basic features for other parts not assigned to you.
- Other penalties and bonus: Same as the guideline above for the team score option.