SlugTrade Team Working Agreements

Meeting Times
Scrum: MWF 9:10 - 9:25

Project Repository
Git repo: https://github.com/Sidney-/slugtrade
Documentation: https://drive.google.com/drive/folders/0AAefs5hNkKCCUk9PVA

Communication Channels
Communicate over Slack: friendbnb.slack.com

Development Environment
TBD

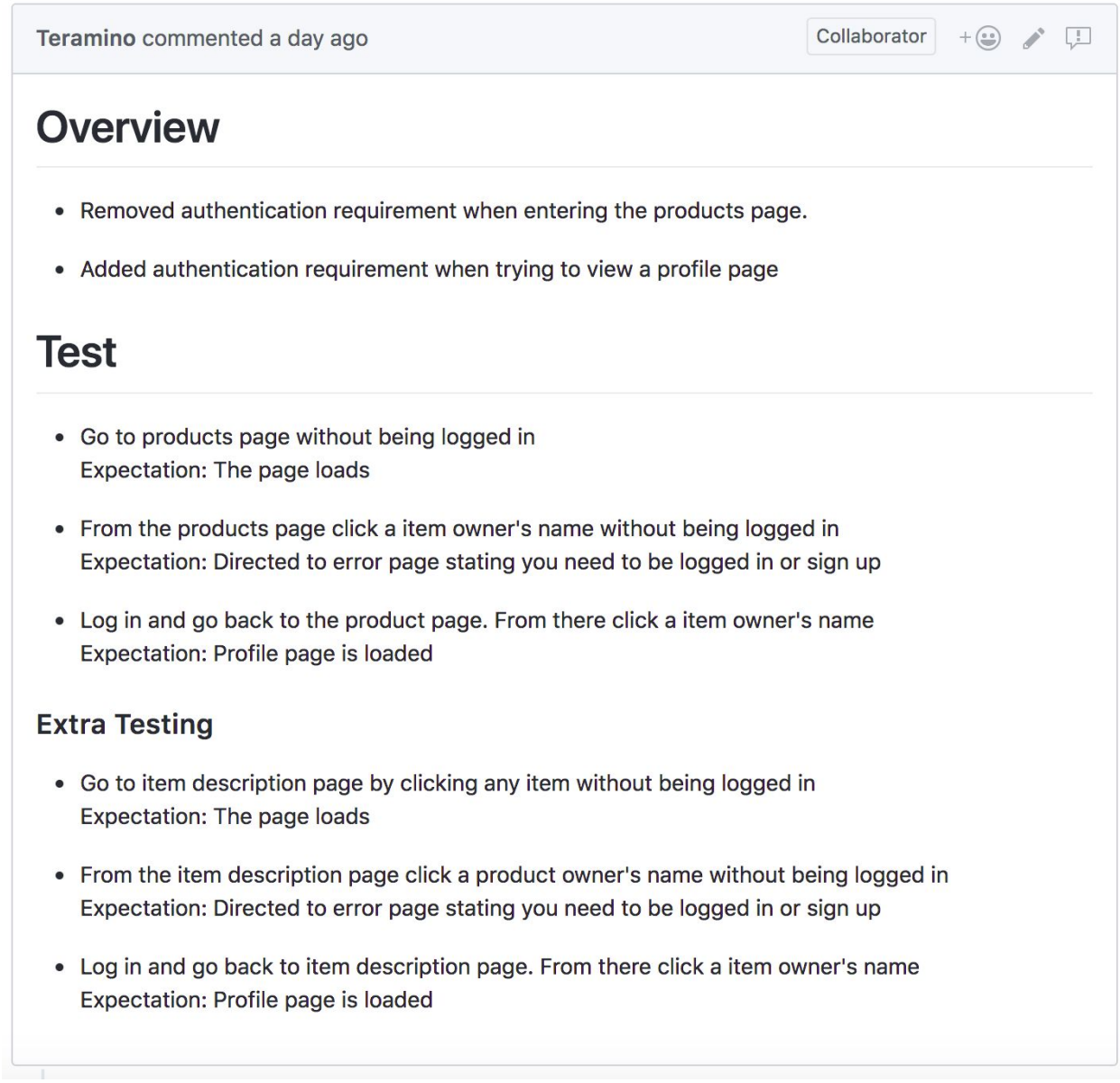
Work Patterns
Definition of Done: A feature is done when it has been merged to develop via a pull request.
Integration: Never push directly to master or to develop. Create a pull request from the feature branch into develop and notify the group on Slack#github.
Pull Requests:

- Pull requests are for merging features into develop (thus develop will always be the base branch).
- One feature per pull request.
- Making changes to develop (refactoring, code styling, commenting out code, removing commented-out code, etc.), is considered a feature.
- Features will not be merged until all comments on the pull request have been resolved.
    - Comments are resolved when the pull requester and the commenter agree that each item has been fixed, will be fixed after merge, or won't be fixed.
    - Regressions (a feature works as expected in dev but not in the feature branch) and bugs that cause the app to crash must be fixed before the pull request is merged.
- Someone other than the feature creator (or collaborators) must review the pull request.
- If a pull request is a bug fix for a reported issue, include the issue number in the commit.
- Ensure that the pull request has no conflicts with the base branch at the time the pull request is created.

Describing a Pull Request:

- Create a descriptive title. If the pull request is a bugfix, include "fix" somewhere in the title.
- Give a list of the functionality changes as a user would perceive them.
- Describe how to test the new functionality: for each item to be tested, describe:
    - how to navigate to the item,
    - what a user would do to trigger the new functionality, and

    - what the user would expect to see.
- Include any notes that are relevant to the reviewer of the pull request (Examples of relevant information: unrelated bugs that affect the feature, items that might need extra scrutiny, known implementation compromises). This reduces back-and-forth discussion where the reviewer brings up issues that were already resolved by the pull requester.
- Example of a beautiful pull request (Authentication fix #108):



Teramino commented a day ago — Collaborator

Overview

- Removed authentication requirement when entering the products page.
- Added authentication requirement when trying to view a profile page

Test

- Go to products page without being logged in
  Expectation: The page loads
- From the products page click a item owner's name without being logged in
  Expectation: Directed to error page stating you need to be logged in or sign up
- Log in and go back to the product page. From there click a item owner's name
  Expectation: Profile page is loaded

Extra Testing

- Go to item description page by clicking any item without being logged in
  Expectation: The page loads
- From the item description page click a product owner's name without being logged in
  Expectation: Directed to error page stating you need to be logged in or sign up
- Log in and go back to item description page. From there click a item owner's name
  Expectation: Profile page is loaded

- Example of a well-formed test plan for a feature (Received offers feed #119):

> # Test Place Offers
>
> ## Setup
>
> Log in as any user
> find an item that is "cash only" and make an offer on it
> find an item that is "trade only" and make an offer on it
> (Note: test with multiple items for the "trade only" offer)
> find an item that is "free" and make an offer on it
>
> ## Execute
>
> Go to offers page and view "Offer Placed"
>
> ## Assert
>
> Verify that these three items are there

Handling a Pull Request:

1. Ensure that there are no local commits in develop or the branch you are testing.
2. Change to the primary working directory: cd slugtrade/slug_trade
3. Check out the branch: git checkout -b origin/{{pull-request-branch}}
4. Merge develop into the branch: git merge origin/develop
5. Examine the proposed changes to develop: git diff origin/develop HEAD
6. Run the first-time startup script: ./first_start.sh
    a. Alternatively, run each startup script separately:
        i. Ensure that your python virtual environment is correct: ./venv_setup.sh
        ii. If needed, recreate the database for testing: ./renew_db.sh
        iii. Start the server: ./start.sh
7. Do the test steps laid out in the pull request description.
8. Ensure that all new functionality laid out in the pull request description is present.
9. Test for regressions: ./run_tests.sh (Note: automated testing is not currently sufficient to check for regressions; manual testing is also required.)

Note: Whenever feasible, indicate the issue severity for each item.

- Severity 0: unambiguously a bug, such that the feature can't be merged
    - Examples: the app crashes or gives 500 error, a key existing feature breaks (regression), the machine catches fire
- Severity 1: issue might be severe enough to prevent the feature from being merged; bring the issue to the whole team to discuss if there is debate between the pull requester and the reviewer
    - Examples: possible regression, "astonishing" behavior, missing functionality

    - Severity 2: issue might be severe enough to file an issue after the feature is merged; bring the issue to the whole team to discuss if there is debate between the pull requester and the reviewer
        - Examples: potentially ambiguous UX, correct but inconsistent behavior between features, typos in user-facing content

Issues:

- Report issues on GitHub. If you're not sure if the behavior you see is a problem, discuss it with the group on whichever Slack channel is appropriate (frontend, backend, etc.).
- Describe the steps to reproduce the issue on the GitHub issue. Verify the steps yourself (in a clean environment, if at all possible).
- Update the issue with a root cause when you find it.
- When working on an issue, assign yourself to the issue on GitHub so others know the bug is being worked on.
- When committing a fix, include the issue number in the pull request commit message.

Starting Up the App for Development: See the Installation Guide

Running Tests: Note that testing is currently only supported on Linux or MacOS; also note that Chrome is required for testing. Run the bash script run_tests.sh to run all tests. The script will report which tests pass and which fail to STDOUT. Note that catastrophic failures (improperly configured environment, incomplete test, etc.) may result in a crash of the test framework.

Collaboration Guidelines: TBD

Coding Standards

Frontend - HTML

- All html must be indented +1 tabs for all child element making ">" shape(s) on all templates pages
- HTML classes are used elements that may have repeated css styles on them, IDs are used to target specific elements with directed css code
- HTML class naming convention should first state the name of the template that is being worked on, type of container name, and specific use (products-list-wrapper, products-list-item-detail-title)
- HTML must be structured in a way that each section of an area of a page has a wrapper and body div to segment code into smaller working chunks doing to the individual div items. This should scale down to individual elements. The developer may have a template wrapper and template body, a preview wrapper and preview body within it, profile image wrapper (on left) and profile details wrapper (right side), then have the image in the profile image wrapper and a list of profile details in the profile details wrapper. This approach contains all of the elements into smaller divs allowing for modularity and clarity. It is similar to Russian Nesting Dolls.
- All class names use dashes for spaces and not underscores. "-" not "_"
- File names use underscores "_"

Front-end - CSS

- All alignment styling is completed using Flex-Box
- All grid styling is completed using CSS-Grid
- No or minimal inline CSS
- Each template should have a long comment above its section clearly stating where the css for that template belongs (/********** Home page **********/)
- CSS must generally flow with div structure. This doesn't have to be 100% but we should aim for about 80% accuracy. This means the parent-most div would be at the highest point of the css page within a template section and the child-most div css would be at the lowest point of a template section
- CSS code should stay near its parents and nearby items. Meaning an element's css at the top of a template should be near the css of parent and child divs of itself. You should not see unrelated CSS next to each other.

Frontend - Javascripts

- All javascript variable names will utilize lower camelcase naming conventions
- All js variable names will be obvious as to what they are
- All js function names should be obvious to what they are
- All js code that is not immediately clear should make use of comments

Backend - Python

- All python function and variable names should use underscores for spaces, not camelcase or dashes
- Follow all typical python conventions (required by python)
- Implement lynter when available
- Be very clear about names of functions, variables, and file names.
- Follow Django/Python file-name conventions where applicable (ex- base.html)

Backend - Django

- We will follow all Django/Python schema conventions
- Our schema will follow relational database standards making use of one-to-one functions, primary keys, and foreign keys
- All model attribute naming should not include the model name
- Model attributes should have underscores when spaces are needed
- Model attribute names should be obvious as to what they are
- Limitations on attributes should be purposeful, meaning not setting limitations on attributes for no reason
- Where possible drop parameters onto the next line for readability. Sometimes parameters for attributes, choices, faults, blanks, etc. can be really long.
- Follow template inheritance
- Render as much logic on the backend as possible and not within a template

Design Patterns

TBD

Product Design Patterns

UI look and feel
Product architecture
Common approach to common problems
Error handling
..