Project Painter: Developer Documentation

The developer documentation should include at least the following information:

- **How to obtain the source code [Jacob Porter].** If your system uses multiple repositories or submodules, provide clear instructions for how to obtain all relevant sources.
    - Source Code can found at the projects GitHub page in the PainterExtension subdirectory: https://github.com/Canvas-Painter/Group17-Project/tree/main/PainterExtension
        - Independent features are split up by folder, with the Chrome Extension related files such as the *manifest* outside of folders
    - Developer Documentation can be found here (this page): https://docs.google.com/document/d/13YbMxTzsQXYc2uVJYm61ArJ58H_LvJ6tH59HM6h7rzQ/edit?usp=sharing
    - User Documentation can be found here: https://docs.google.com/document/d/1xUaHkDol8P2Ght-Y0p4gwRxqx8HyKfweMmLJVI2JGq8/edit?usp=sharing
- **The layout of your directory structure[Samuel Jamieson].** What do the various directories (folders) contain, and where to find source files, tests, documentation, data files, etc.

  ProjectPainter/ (Root Directory)

  |

  |-> src/ (Source Code - Core Functionality)

  | |-> Features/ (Canvas LMS Enhancements)

  | | |-> gamified_calendar (Folder that contains all gamification files for project

  | | | |-> gamification_points.js (File to add points for completing assignments)

  | | | |-> gamified_calendar.js (File for adding gamification features)

  | | | |-> gamification_rewards.js (File for adding rewards and badges for achieving certain milestones)

  | | |-> Sidebar (Folder containing sidebar information

| | | |-> sidemenu.js (File for displaying sidebar information on canvas main screen)

| | | |-> ta_schedule.js (adds feature to display TA office hours on sidebar

| | | |-> pdf_syllabus_parser.py (File for parsing PDF information)

| | |-> faster_canvas.js (Script for speeding up canvas load times)

| |-> background/ (Handles background tasks for API request and data storage)

| | |-> background.js (Manages persistent extension operations

| | |-> storage.js (Handles local storage if needed)

| |-> popup/ (UI Elements and plus website and extension popups)

| | |-> popup.html (Main UI for extension popup window

| | |-> popup.js (Handles interactions within popup.html)

| | |-> popup.css (Handles extension popup styling)

| | |->rewards_display.js (Handles the rewards user has achieved within gamified calendar)

| | |-> Website/ (Houses code for settings and description website)

| | | |-> index.html (Website html)

| | | |-> index.css (Website CSS)

| | | |-> index.js (Handles interactions on website)

| |-> content_scripts/ (Scripts injected into Canvas LMS Pages)

| | |-> content.js (Coordinates interactions between extension and canvas LMS)

| | |-> dom_manipulation.js (Modifies and enhances Canvas UI elements)

|-> test/ (Automated tests for core features)

| |-> test_gamification.js (tests Gamification mechanics (Pass or Not Pass Tests))

| |-> test_pdf_parser.js (Feeds a series of syllabi into the script monitoring the outputs manually)

| |-> test_api_calls.js (Verifies Canvas LMS API)

| |-> test_ui.js (A series of pass or no pass tests)

| |-> Test_faster_canvas.js (Tests the faster canvas feature)

|-> docs/ Project Documentation

| |->README.md (overview of project)

| |-> Reports/ (Folder of weekly reports and contributions

| |-> API_DOCS.md (Instructions on how to make a successful API call in canvas)

| |->ProjectProposal.pdf (Living document)

| |-> User_guide.md (User Guide)

| |->dev_guide.md (Developer Guide)

- **How to build the software [Oscar Ludwig].**
    - Our project, as a web browser extension does not get compiled like most programs, but is instead loaded in an unpacked format onto a compatible chromium browser. To achieve this, download we recommend you download Google Chrome at https://www.google.com/chrome, then input chrome://extensions into the address bar, turn on developer mode, click load unpacked, then select the “PainterExtension” folder (inside root folder for project). This completes the build process for our product.
- **How to test the software [Kai Turner].** Provide clear instructions for how to run the system’s test cases. In some cases, the instructions may need to include information such as how to access data sources or how to interact with external systems. You may reference the user documentation (e.g., prerequisites) to avoid duplication.
    - We have two different test frameworks that are currently at work. The first one is PyTest to run the extension in the browser using Selenium and the second is Jest for testing specifically complex parts of the code.

    - To run the PyTest tests you must go into the extension directory and install the requirements.txt through “pip install -r requirements.txt” then you can just run “pytest”
    - To the Jest tests you also must go into the extension directory and run “npm ci” to install the required libraries and to run the tests you must run “npm test” and that will run the tests
    - Another way to view the test results is through the actions on github
- **How to add new tests [Andrew Vu].** Are there any naming conventions/patterns to follow when naming test files? Is there a particular test harness to use?
    - In order to add new tests and test cases, developers will need to add new test files to the github. The files should be placed in a separate directory which will be called PainterExtension/Test/Python if the tests are written with python. The test files should be named test_(*feature*).py and placed within the test folder within the PainterExtension github repository. For JavaScript-based Jest tests, there will be a separate directory named PainterExtension/Test/Javascript and the name should follow the format test_(*feature*).js.
- **How to build a release of the software [Cameron Dilworth].** Describe any tasks that are not automated. For example, should a developer update a version number (in code and documentation) prior to invoking the build system? Are there any sanity checks a developer should perform after building a release?
    - When releasing a new build/version of the software, it is important that the version number correctly reflects the changes; If the changes were significant and something we’ve been working towards for a while, we should update the main version number, e.g. 1.2.2 -> 2.0.0. Additionally, if it is a smaller change, e.g. bug fixes, we should reflect that: 1.2.2 -> 1.2.3.
    - Whenever a new version is about to be released, it should first be committed and a final copy should be tested before the official release. This will be done with the tests we already have prepared. The automatic tests should all pass, and the user tests should be done by developers or testers. The more features that are implemented, the more tests we need to incorporate and try out, i.e. we will also need more testers to complete these in a timely manner.
    - In the case that not all tests are passed, we will need to postpone the launch and focus entirely on fixing said bug. Depending on the estimated time it will take to fix these bugs, we may need to release an official notice.
    - After a new release is uploaded, it is important that the developers check on any new feedback and use this moving forward with the next upcoming releases.