Project Evaluation

CSE-112-team8: Cre8tors

How to Evaluate a Repo

- We consider how well the repo embodies the values we champion as a group, not only based on how these are implemented in the project, but also in how much growth opportunity there is in the project for these values.
- For each value, we will grade the project as either being satisfactory, adequate, or unsatisfactory.
    - A project which satisfies a value not only embodies the value, but also shows a great potential of growth in terms of that value.
    - A project which is adequate shows that it is possible for us to implement these values into the project once we begin work, but as it stands lack a substantial amount of effort towards exemplifying that value
    - A project which is unsatisfactory shows very little, if any, possibility of growth for that value.
- For each value, we will also be keeping in mind the constraints that will inevitably show up as we work on the project.
    - **Time:** We have 10 members, CAPES reviews typically average class hours to 8 hours per week. Not including week 1-3, we have 8 weeks left before the deadline, which gives us 10 members * 8 hours * 8 weeks * 0.7 slack factor= 448 hours of work time assuming the CAPE numbers and the slack factor are accurate representations. Leveraging this information, we should take in projects that are in a state where we have enough time to create a production level quality product. Obstacles in repos would include time to learn new technologies as well how long it would take to get acquainted with the repo.
    - **Skills:** How confident we are with the technologies used in the code base. This includes front end, back end, pipeline, project management, design, and testing. Not everyone needs to understand all the technologies at play with the code base, so long as there is someone who is understanding of the technology and can teach it in a sufficient time frame. To get a better grasp of this constraint, we will poll ourselves to know what technologies we are comfortable with. Additionally, we will weigh in how long it would take for us to learn new technologies.

Simplicity & Organization

- A repo should be simplistic and organized because it allows us to leverage more time into developing and improving the project instead of having to spend large amounts of time getting acquainted with the system
- The infrastructure of the project is maintainable and not overly-built:
    - The github repo serves as the single source of truth.
    - Not too many files and dependencies.
    - The structure of the project is self-explanatory.
    - The onboarding documentation, if any, is accessible and easy to understand.
    - Unnecessary and/or overly-complicated tools are not used in the project.
    - Documentation is well worded and diagrams are easy to understand.
    - Issues are detailed and contain meaningful labels
    - Most branches are associated with an issue and there is a meaningful connection between issues and and branches.
    - Having a SCRUM board which organizes the issues.
    - The code is well documented and has a linter to enforce a standard of quality.
    - High quality C4 Diagrams to help with the traversal of the project code.

User Centered Thinking

- Not necessary to see accessibility in the product itself more so that there is a solid foundation of user centered thinking and design in the repo.
- There is evidence of the previous developers designing this product with users in mind (e.g., User stories, personas). General user experience is fundamental to the goals that the previous developers set up.
    - Developed user stories and personas which are accurate to the target audience.
    - Flow charts detailing the use cases of the product from the perspective of the user.
    - ADRs which account for the user experience. This can include noting accessibility features and performance.
    - User focused design diagrams like wireframes and high fidelity mockups.
    - A feature list which details features that emphasize the user experience.
- In short, the repo should tell us who the users of the product would be, and have profiles on the needs of those users.

Transparency

- Documentation of both the code as well as the documentation process. This would capture the pitfalls past developers fell into. This would help us prevent wasting time discovering issues that could have been documented for us.
- Emphasis on looking for how issues and pitfalls were addressed and evidence that these issues were addressed and well documented, otherwise we could potentially be adopting issues we don't even know about.
    - There are ADRs which document every major decision. These ADRs should include the problem, possible solutions with pros and cons, and a decision with the consequences of that decision documented.
    - The issues documented are well detailed and provide not only the goals to be achieved for that issue, but also the reasoning behind why it was set up.
    - A roadmap detailing what features the original team wanted to implement.
    - Known bugs and issues are documented and are not hidden. Issues found in the product are noted in documentation.
    - If there are more branches than just main, are their purposes explained, or are they an anomaly?

How to Compare the Top Two Projects

- We will be taking a deep dive into the final two repos chosen. We will dissect each aspect of the project including the design, documentation, programming, CI/CD, and tests used.
- We will define what we would be doing with the project and we will draft out the potential project timeline for each one, keeping in mind the time constraints.
    - Additionally we will drafting out the potential maintainability of this project even after we reach our goals by the end of the quarter.
- Set up each project and try to push significant changes to both and figure out how simple/easy/understandable it was to do this, if it even was possible.
    - Setting up the project means launching the repo and access the website locally.
- Discuss each value in strong detail. List the strengths and weaknesses of each repo in relation to each value.
- Run a runtime analysis and get a vital from each project. Determine if these issues are brought up by the documentation or are mentioned in the repo. Is it possible for us to come up with a general diagnostic of the performance from reading the documentation, going over design diagrams, and interpreting the code?