Term Project: Financial Web Application

0 Points Possible

Add Comment

Details

In previous FinTech 512 offerings, student teams built a financial trading system similar to E*Trade or RobinHood (aka, “Big Bucks”). This semester, teams will create a financially related web application of their choice. Teams will utilize Scrum, an agile framework focusing on iterative progress, transparency, inspection, adaptation, and collaboration.

The project is divided into the following structure:

Team Selection	Students form teams of five members of their choosing.
Project Proposal	Teams present their lean canvas and initial screen designs during a five-minute in-class presentation.
Sprint Phase 1	Project planning, initial design
Sprint Phase 2-6	Teams continue to incrementally build the system’s functionality following Scrum. This culminates in the final sprint (6).
Video Creation	Teams create three videos covering various aspects of the project.
Final Presentation	During the final period, teams present their project for 12 minutes, including a live system demonstration, followed by 5 minutes of questions from fellow students, Tas, and instructors.

Project Logistics

- You will work in teams of five members for the project. You will choose your team members, but the instructor(s) reserve the right to make changes.
- Projects will follow the Scrum Framework: https://scrumguides.org/ (https://scrumguides.org/):
  - *Sprints* are generally one week long, ending at 11:00 PM on Friday evenings.
  - A teaching assistant (TA) will act as the *product owner*. However, the TA will not have a say in prioritizing work. You should speak freely with your TA throughout the semester. The TA will also act as a mentor, providing advice and guidance on proceeding. TAs are responsible for validating requirements with the associated test cases. TAs will maintain a spreadsheet throughout the project, monitoring your progress and grading.

- Your team should have a *sprint meeting* with your TA towards the end of each sprint. This meeting covers the following aspects:
  - A *sprint review* covering the work completed during the Sprint. The TA should validate the implemented requirements and associated tests during this session.
  - A *retrospective* in which the team plans to increase quality and effectiveness. What went well in the past sprint? What has not gone well? What changes should be made? We value honesty – the grading is based upon the completion of the retrospective, regardless of any negative comments. Pointing out deficiencies should be seen as a positive as those provide a basis for improvement.
  - *Sprint planning* - Discuss and decide what will be worked on in this sprint based on the remaining project requirements. Create one or more 'Planning' issues for each requirement, and assign each issue to a team member. A planning issue must be created before work begins to get full credit for implementing project requirements.
- *Product Backlog*: The project requirements form the initial product backlog. The backlog may shrink or grow as requirements are completed and new information is discovered.
- Specific to this class, each project team must complete a sprint status report at the end of each spring with the following elements:
  - A list of the GitLab issues opened or closed during the week.
  - UML class diagram (The first report will be the most difficult; later reports will need to update the diagram.)
  - A paragraph summarizing your sprint review
  - A paragraph summarizing your sprint retrospective
  - A paragraph summarizing your sprint planning
  - Any additional comments and/or concerns
- Create a single GitLab repository, 'fintech512-project_*team_name*'. Add each team member as a 'Maintainer'. Add your instructor and all of the course TAs as 'Reporter'. In addition to any deliverables (code, test cases, documents, diagrams, etc.), your repository must include the following files:
  - README.md: Provides an overview of your project and specific steps to install and run the software. These steps must be of sufficient detail so that an experienced developer can follow those steps without having to research specifics in the code.
  - CodeReviewChecklist.md: Your team's code review checklist.
  - CONTRIBUTING.md: Process description that your team follows. A definition of "done" that includes:
    - Everything in the **PSP script (https://canvas.duke.edu/courses/51277/files/2422662?wrap=1)** **(https://canvas.duke.edu/courses/51277/files/2422662/download?download_frd=1)** for each piece of work you define. Add anything that you find helpful or necessary.
    - Each "increment issue" (see below) must include
      - Pull request (PR) where the code implementing the increment is located
      - Each code file must contain appropriate comments: **functions (https://fintechpython.pages.oit.duke.edu/jupyternotebooks/1-Core%20Python/07-Functions.html#docstrings-and-providing-function-documentation)**, **modules**

(https://fintechpython.pages.oit.duke.edu/jupyternotebooks/1-Core%20Python/20-Modules.html#module-docstrings) , classes (https://fintechpython.pages.oit.duke.edu/jupyternotebooks/1-Core%20Python/25-ClassesAndObjects.html#docstrings%20)
        - Code review results, based on your code review checklist
        - Test cases for the increment.
        - Test results for the increment based on the defined test cases.

- Translate each project requirement into one or more **increments** (https://scrumguides.org/scrum-guide.html#increment) . Create an issue for each increment in your GitLab repository. Each increment must include -
    - A description of the specific project requirement(s) that the increment addresses.
    - A **user story** (https://www.mountaingoatsoftware.com/agile/user-stories) describing the goal of the increment.
    - Test cases: examples of input/output showing that the increment has been completed. Ideally, these should be automated test cases.
    - Notes on any user documentation required.
    - List of changes to be made.
    - Estimated hours to make and test the changes.
    - Anything else the team believes is necessary to assure that the increment is complete and usable.
    - Once the TA has approved the implemented increment, they will provide a comment on the issue.
- Obtain a **VCM** (http://vcm.duke.edu/) and designate it as your production environment. Email the machine's name and IP address to your assigned TA and your course instructor. If you find it helpful, you can also use a VCM as a staging environment to test developer changes before sending them to the production machine.
- For writing code and tests, each team member should use one of these three options:
    - fork their team's repo, and work on their changes in their own forks
    - create a branch within the team's repository, and work on their changes in their own branches
    - both fork the repository and create a branch.

  Regardless of the option, they must perform merge requests to bring their efforts together in the main repo.
- If you have team problems, you should attempt to resolve them yourselves. If that fails, involve your TA and/or professor.

First Sprint

For the first sprint, your team must accomplish several activities:

1. Schedule the six sprint meetings with your assigned TA.
2. Create a lean canvas for your project.

3. Develop two or more paper prototype screens to highlight the significant functionality of your application.
4. Present your project proposal (BMC and paper prototypes) during a five-minute session in class.
5. Establish your Gitlab repository.
6. Based upon the overall project requirements, establish the specific requirements for your project.
7. Based upon those requirements and items specified in this document, create a “product backlog”. This backlog can be tracked in several ways: a GitLab issue, a shared document, etc. Choose something that will work well for your team.
8. Perform the initial design for your application. Items to consider:
    - Architecture
    - Class design
    - Database design
    - APIs
    - Testing approach
    - Error handling
    - Implementing necessary non-functional requirements (security/performance)
9. Outline your initial approach to incrementally building your application.
10. Hold your first weekly sprint meeting.
11. Complete and submit your first sprint status report.

Sprints 2 – 5

The remaining sprints will function more similarly to an industry-based sprint:

1. During the prior sprint’s review meeting, determine what to implement in this sprint.
2. Create the necessary issues for each increment to be implemented.
3. Divide up the coding, testing, and review work among the team members
4. Perform the necessary activities to implement the planned increments
5. Update project artifacts such as the class and database designs.
6. Meet with your teaching assistant to perform the sprint review meeting.
7. Complete and submit your sprint status report.

Videos

You must create three five-minute videos and post these videos to the MS Teams Project Channel:

1. Website pitch
    1. Unique value propositions from the lean canvas
    2. Why your application versus others?
    3. Revenue streams
2. Website demonstration highlighting the significant functionality of your website. This should include both a regular user’s view of the site and an administrator’s view. Include anything unique or that you're proud of.
3. Presenting the website's design, including UML, patterns used, and UI/UX principles applied, and presenting lessons learned. What went well? What could have gone better?

Final Presentation

Each team will have 17 minutes, covering the following:

1. Site demonstration: Using a live version of your application running on the VCM, demonstrate the essential features of your application.
2. Design discussion: Explain how you designed and built the site.
3. Process discussion: Discuss the successes and challenges experienced in working together to build the site. Emphasize any 'lessons learned' about project management and software engineering.
4. Five minutes for questions and answers from other students, TAs, and instructors.

Grading

As outlined in the syllabus, the programming project is divided into the following components (normalized to represent 100% of the project rather than 40% of the class):

- 50% Tested, implemented functionality
- 30% Sprint status reports (5% each)
- 20% Presentations
  - 5% Project proposal
  - 5% Videos
  - 10% Final presentation

List of factors affecting the team's grade:

- Consistency of effort. Defining and completing increments of work every sprint to implement the listed requirements will provide maximum credit for the project grade.
- Functionality
- Design
- Documentation (including initial UML review + final UML diagrams)
- Testing
- Other Code Quality Factors (e.g., naming, formatting, smells)
- Process (Issue tracking, CI/CD, Code Reviews, etc)
- Team participation - we will look at team member commit frequency, size, complexity, and adherence to good practices (e.g. following instructions, tests) to assess relative effort among team members.
- The individual score for the group project will be 80% based on the overall team score and 20% based on their consistency of effort.