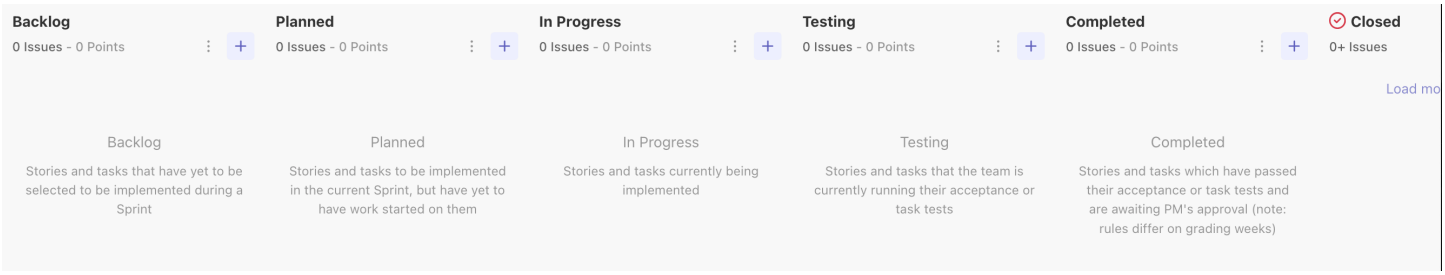
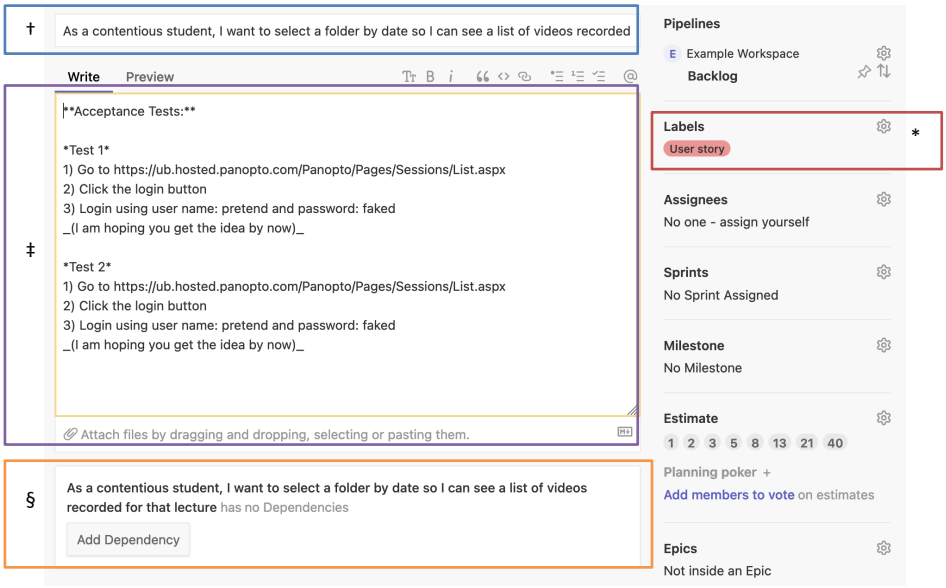
PIPELINES IN ZENHUB

Upon creating their Scrum Board in ZenHub, each team will need to update the pipelines to align with the process used at UB. The name of each pipeline and order that they should occur in ZenHub can be seen in the image below:



USER STORIES IN ZENHUB

Each user story should be created using the "New Issue" button in ZenHub. Below is an image on an example user story and this is followed by an explanation of the required elements.



† (issue title) contains the user story text. It should ***only*** contain the text of the user story. The text must follow user story template: "As a *role*, I want *action taken* so that *observable outcome*". Make certain that the *role* provides insight as to why this feature is important, the *action taken* describes what the role does to trigger the feature, and the *observable outcome* describes how the role knows the action taken completed successfully.

‡ (issue description) contains all of the acceptance tests for the story. Write "Acceptance tests" in bold as the start of this text. All of the acceptance tests needed for this story should follow. Each acceptance test must be labeled and there should be a blank line between each test. Within an acceptance test, each step should be numbered. These tests will be run by an ***untrained user***, so you need to specify all of the details about what to do or what should be seen and avoid any jargon.

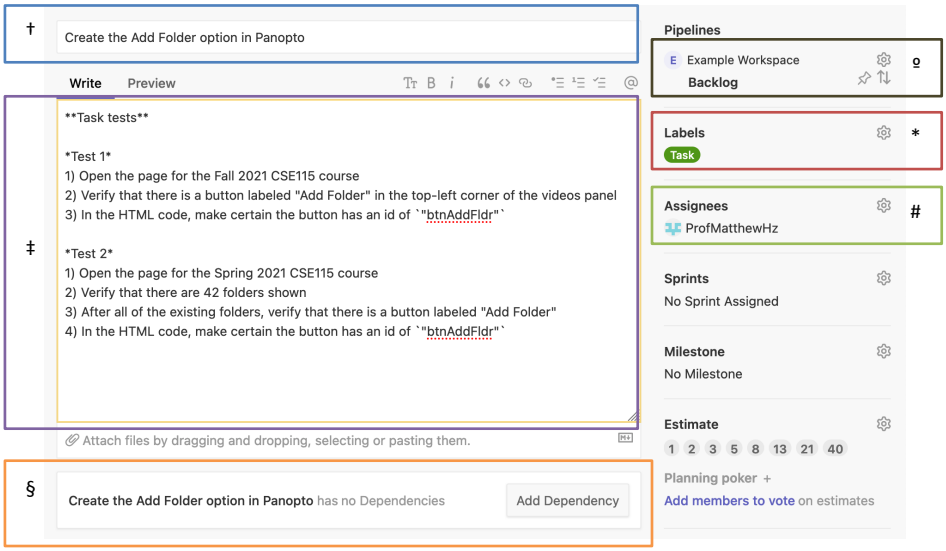
* (labels) should only contain "User Story" when the story is created in the Backlog. The current sprint should be added as a label when the story is added to the "Planned" pipeline. Ideally stories will be completed in the sprint in which it is started. When a story cannot be completed by the end of the sprint, it will have to carry over to the next sprint. In these cases, **do NOT remove the original sprint label**, but instead have the issue include labels from the original sprint label ***and*** the new sprint.

§ (dependencies) should link the user story to the tasks used to implement the story. **User stories always list tasks as dependencies** and never the other way around. Tasks should be added as a dependency as soon as the task is created.

(Not shown in this image) Issue comments can be used to provide updates or communicate information for this user story. This is rarely done, however.

TASKS IN ZENHUB

Each task should be created using the "New Issue" button in ZenHub. Below is an image on an example task and this is followed by an explanation of the required elements.



† (issue title) contains a brief summary of the task. It should ***only*** contain a summary of the task. There is no required template to follow and technical language is acceptable. A developer just joining the group should be able to figure out what the task will do from this summary.

‡ (issue description) contains all of the task tests for the story. Write "Task tests" in bold as the start of this text. All of the task tests needed for this task should follow. Each task test must be labeled and there should be a blank line between each test. If unit tests are being used, then the name(s) of the file. If the task tests use step-by-step instructions, each step should be numbered. These tests will be run by a ***developer*** user, so jargon and technical details are expected. Be certain to specify all of the details about what to do or what should be verified in that test.

º (pipelines) will automatically be updated as the issue is moved from pipeline to pipeline. If the task is for a story in the "Planned" pipeline, the tasks should be added to the "Planned" pipeline when it is created. Otherwise, the task should be added to the Backlog pipeline (which is the default). It is important that the task be moved into each pipeline to show work was done properly. There should not be any commit shown for this task before it is added to "In Development" and there must be at least 1 commit between when it is added to "In Development" and when it is moved to "Completed".

* (labels) should only contain "Task" if the task is created in the Backlog. When a task is added to the "Planned" pipeline, it should also have the current sprint added as a label. A task should be completed in the sprint in which it is started. If a task is added to the Planned pipeline in one sprint, but does not get developed in that time, **DO remove the original sprint label** and replace it with the label from the new sprint.

* (assignees) should contain the student assigned by the PM to complete that task. The student should add themselves as the assignee immediately after the team meeting when the PM assigns them to a task. After that initial assignment, only the PM can add another student as an assignee or change the student to whom the task is assigned.

§ (issue dependencies) will link the task to the user story for which it was created. **Tasks never list user stories as dependencies** so this button should not be used in the task "issue".

(Not shown in this image) Issue comments MUST be used to record the URLs and findings in any research-related tasks. Comments can also be used to provide updates or communicate information in all tasks.

BRANCHES IN GITHUB

Your project's **`main`** branch is used to hold the latest release of your project. **AT NO TIME SHOULD YOU EVER MAKE A COMMIT DIRECTLY TO THE `main` BRANCH.**

At the start of your project, you will need to create a **`dev`** branch off of the **`main`** branch. The **`dev`** branch can be used to track your team's progress in the current Sprint. To do this, it is important that **`dev`** only contains code and documents from the current Sprint's tasks which have been fully completed ***and tested***. Before the grading meeting at the end of each Sprint, your team needs to "release" the next version of your project, by merging the changes in **`dev`** back into the **`main`** branch.

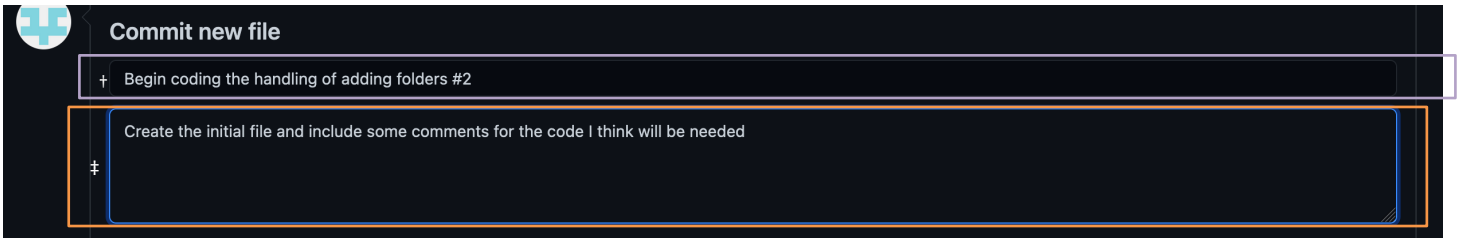
When starting work on a task, you will need to create a branch for that task from the **`dev`** branch. Be certain to give the branch a meaningful name that allows your manager, your manager's manager, and other outsiders to know what the changes in this branch were accomplishing. You can include the task number in the branch's name if it helps in clarifying the work being done, but this is **not** required. You should be making regular, consistent commits to this branch as you work on the task. These regular commits help your manager see your progress, insures nothing gets lost, and cannot harm anyone else's progress since they will not be working on this branch. After adding your task to the Completed pipeline (or Close pipeline in a grading week), you should merge the task's branch back into the **`dev`** branch.

The history of branches and commits can be seen in GitHub by selecting the "Insights" tab, and then clicking on "Network" from the menu bar on the left. An example of a good use of branches can be seen below:

COMMITS IN GITHUB

It is expected that students will make regular commits as they are working on a file. In addition to ensuring progress gets saved, regular commits allow others to recreate their process and understand the decisions they made while coding. Frequently pushing changes to the server (and pulling down any updates) also keeps merge conflicts to a minimum and simplifies the cleanup process when it occurs.

Below is an image showing a sample commit using the GitHub web interface. Using this interface is **NOT** required, but it can be used and provides a good visual to explain the standards used in this software engineering class.



† (subject) This should be under 50 characters and give a brief overview of why the commit matters. **End the subject with a space the hashtag and the number of the task** for which this commit was made. Since you are working on 1 task at a time and are making frequent commits, **only tag 1 task**. If this commit finishes the code, move the task to the Testing pipeline and start your testing. Remember that the audience for your commit messages are your boss and the people who will need to update and support this code, so focus on the information they will need to know.

‡ (description) This is optional, but can be used if you need more than the summary to document this commit. If the commit breaks existing code or has any important side-effects, document those effects here ***and explain why these changes are valuable***. Remember that the audience for your commit messages are your boss and the people who will need to update and support this code, so focus on the information they will need to know.