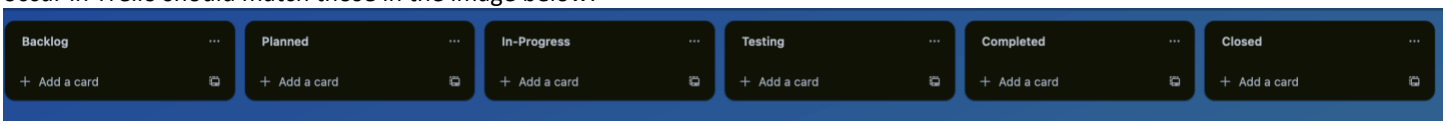
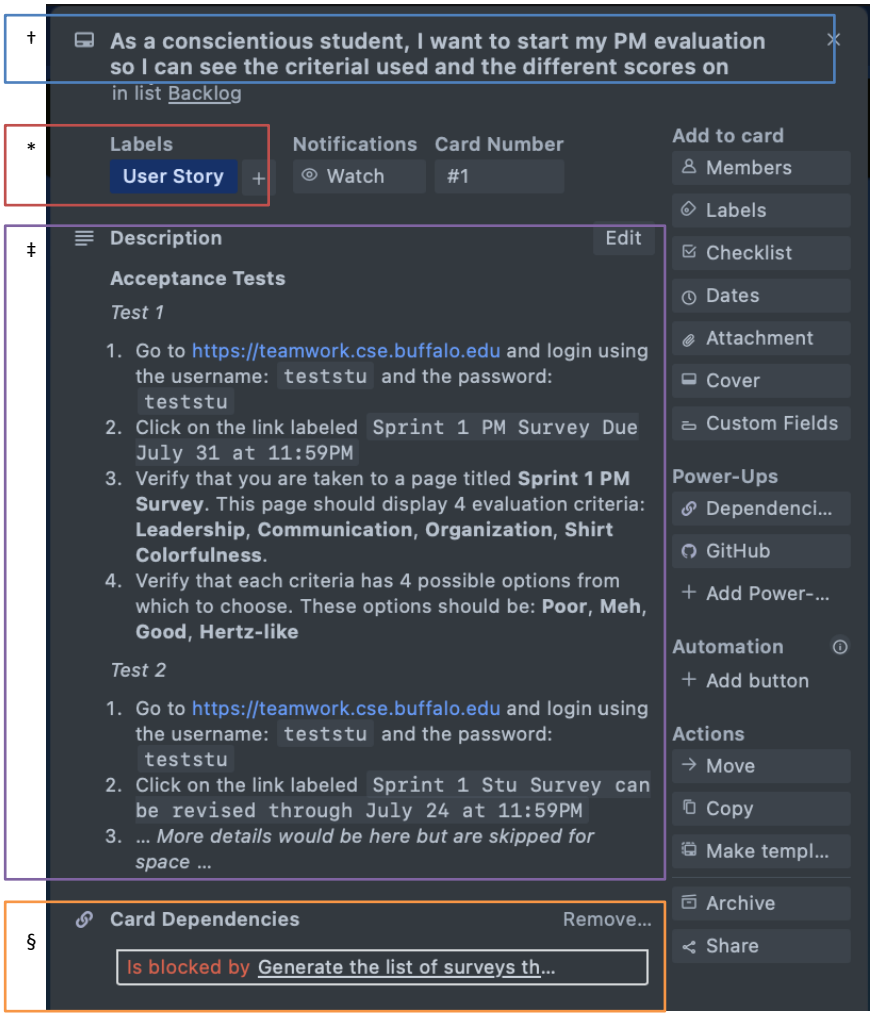
LISTS IN TRELLO

When you get access to your team's Scrum Board in Trello, it will already have been setup with everything you will need for this course. This should include containing lists that align with the process used at UB. The name of each list and order that they should occur in Trello should match those in the image below:



USER STORIES IN TRELLO

You will need to create a card in Trello for each user story. Each user story card documents exactly 1 potential feature in your program. Below is an example of a user story and then an explanation of each of the required elements.



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

‡ (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 contain an italicized label and be followed by a numbered list of steps to take. Add a blank line between each of the tests. These tests will be run by an ***untrained user***, so make certain you specify all the inputs that need to be entered and exact details of what they should see.

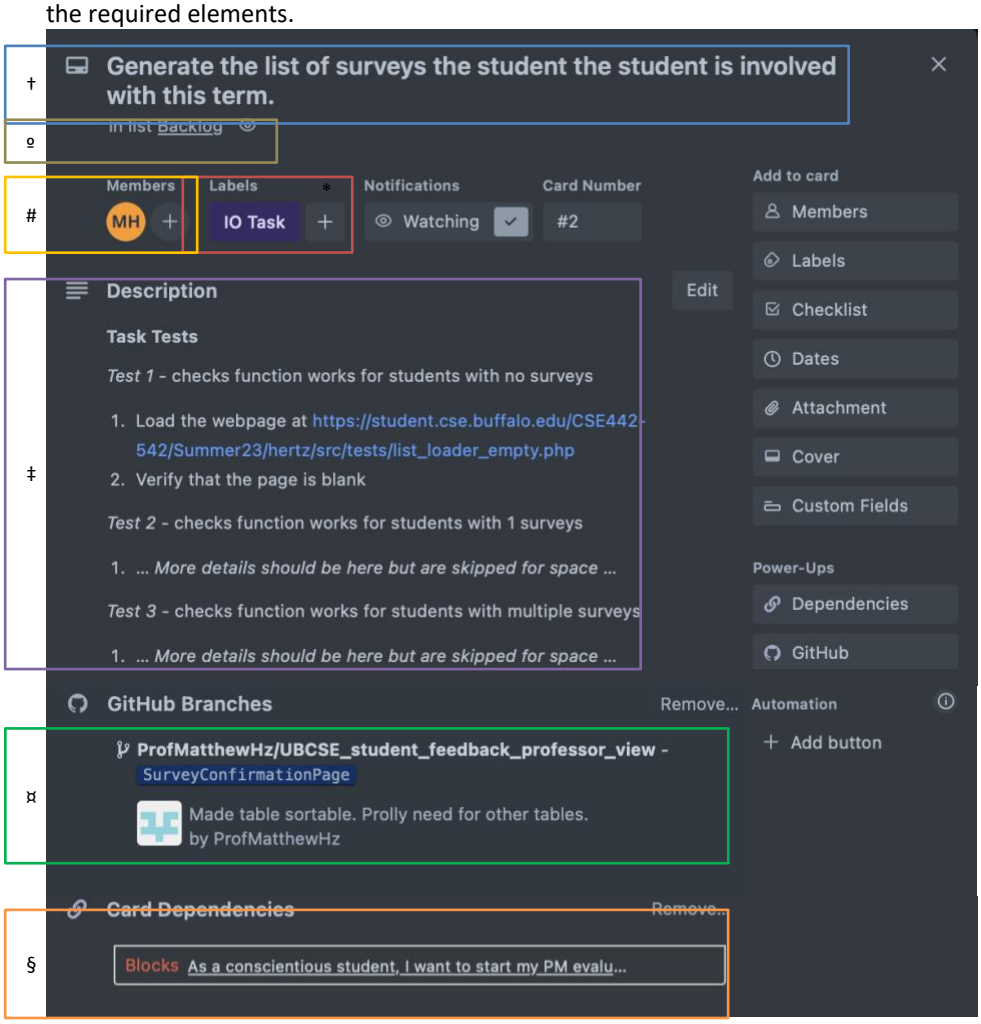
* (labels) "User Story" should be added to the card when it is created in the "Backlog list. The current sprint should be added as a label when the story is added to the "Planned" list. Ideally stories will be completed in the sprint in which it is started. When a story cannot be completed by the end 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 card include labels from the original sprint label ***and*** the new sprint.

§ (dependencies) Each user story **is blocked by** all of the tasks that are used to implement the story. When included in a sprint, user stories must have at least 1 (and almost always more than 1) task as a dependency.

(Not shown in this image) Comments can be added in the activity portion of the card to provide updates on this user story.

TASKS IN TRELLO

Each task should also be created as a card in Trello. Below is an image on an example task and this is followed by an explanation of the required elements.



† (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.

‡ (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 list 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 (lists) will automatically be updated as the card is moved from list to list. If the task is for a story in the "Planned" list, the story CANNOT be in the “Backlog” list. When a story is in the “Backlog” list, its tasks are usually also in the “Backlog” list. It is important that the task be moved into each list as work is being done to document the process was followed and testing was performed. Commits should only occur when the task is in the "In Development" list and there must be at least 1 commit between when it is added to "In Development" and when it is moved to "Testing".

UB Software Development Standards

* (labels) should identifying this card as containing a task. The specific labels used for tasks will vary by course. When a task is added to the "Planned" list, 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 include in the Planned list in one sprint and gets carried over to a second sprint, **DO NOT remove the original sprint label**. When a task is carried over into a second sprint, it add the new sprint label while also keeping the old one.

(members) 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.

¤ (branch) A new branch off of **`dev`** needs to be created with the first commit for this task. The card should then be updated to add a link to this branch to document this relationship.

§ (dependencies) link the task to the user story for which it was created. **Tasks are never blocked by user stories**.

(Not shown in this image) Card 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.

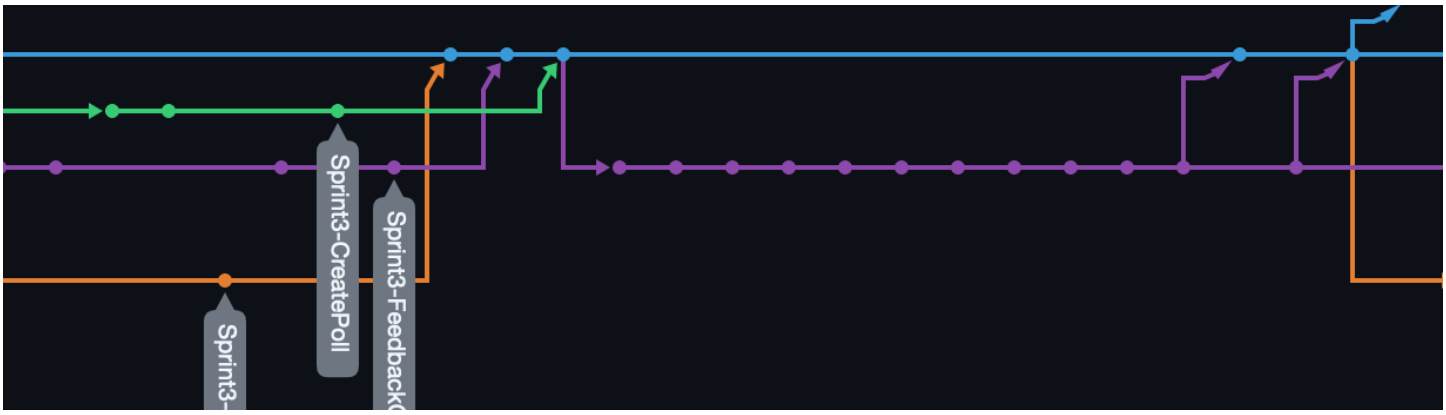
BRANCHES IN GITHUB

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***. By the deadline set by your PM for that Sprint, your team needs to have all of the code merged into the **`dev`** branch. Your PM will then "release" the next version of your project by merging the changes in **`dev`** 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 should also add this branch to the task's card using the GitHub link. 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" list, you should merge the task's branch back into the **`dev`** branch.

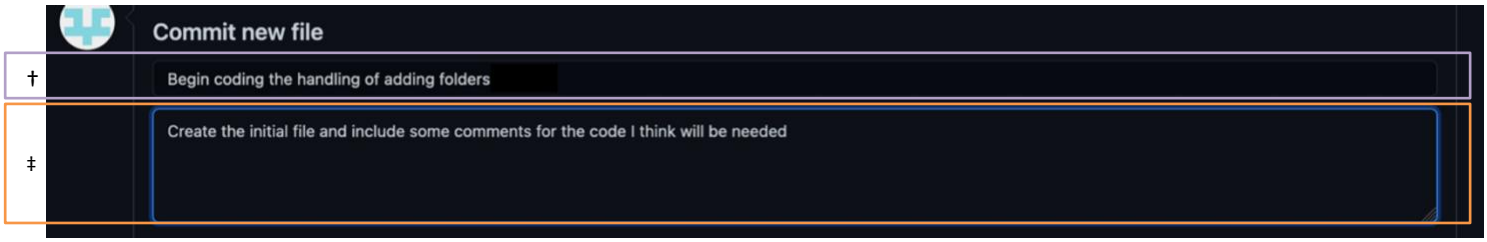
The history of branches and commits can be seen in GitHub by selecting the "Insights" tab, and then clicking on "Network" from the menu bar on the left. An example of a good use of branches can be seen below:



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. The **1** task whose branch this commit will be should be in the “In Development” list. If this commit finishes the code, move the task to the “Testing” list and start your testing.

‡ (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.