Agile

User Story 1:
As a vanilla git power-user that has never seen GiggleGit before, I want to…

- Title: Onboarding Git Power-User
- Acceptance: onboardGitPwrUser
- Priority: 2
  - This is a lower tier of priority because the onboarding process for an experienced user is less rigorous and less common compared to users with little to no experience with Git.
- Story Points: 5
  - Creating an onboarding process specific to someone who has a lot of experience in Git can be a large undertaking, so a higher number of story points is appropriate
- Description: As a vanilla git power-user that has never seen GiggleGit before, I want to quickly understand what makes GiggleGit different and what features it offers so I start using it immediately.

User Story 2:
As a team lead onboarding an experienced GiggleGit user, I want to…
- Title: Team Lead Onboarding Experienced GiggleGit User
- Acceptance: onboardExpGigGitUser
- Priority: 1
  - This is a high priority since we are onboarding a team member.
- Story Points: 3
  - This has a medium amount of work because the user is experienced, but there is a many team-specific things this user needs to learn.
- Description: As a team lead onboarding an experienced GiggleGit user, I want to ensure they are quickly integrated into the team so they can start to contribute as soon as possible.

User Story 3:
- Title: Team Lead Viewing Team Commit Activity
- Priority: 1
  - Important for this to be done as soon as possible to support continual monitoring activities for the team lead.
- Story Points: 2
  - Is is a relatively simple feature to implement since it is just pulling existing data.
- Acceptance: viewTeamCommits
- Description: As a team lead I want to see an overview of my team's commit activity in GiggleGit so I can easily track contributions and monitor progress without having to open everyone's repositories and manually searching.
- Task: Create a component on the team dashboard that displays the commit data.

- Ticket 1: Create Component on Dashboard
    - Utilize front-end technologies to create a user friendly table and graph view of the data.
- Ticket 2: Add Data to Component on Dashboard
    - Integrate commit data into the component through the back-end.

As a user I want to be able to authenticate on a new machine.

- This is not a user story because there is no benefit. User stories must have a benefit.

Formal Requirements

- Goal: Create a user-friendly interface for SnickerSynce that allows users to easily merge changes with a snicker.
- Non-goal: Allow users to utilize snickers in other parts of GiggleGit outside of merges.
- **Non-functional requirements:**

  1. Access Control
      - The system shall utilize role-based access control to ensure only certain people can maintain the different snickering concepts.
          - **Functional:**
          - The system shall allow PMs to log in with elevated privileges to edit snickering concepts.
          - The system shall track all actions taken by PMs regarding creation, editing, or deletion of snickering concepts.
  2. User Study Assignment
      - The system shall ensure participants are randomly assigned to control groups and variants to guarantee unbiased results of the user study.
          - Functional
          - The system shall implement a random algorithm to assign participants to control and variant groups.
          - The system shall provide researchers with an interface showing the assignments and groups.

In this exercise, you are provided with a

- formal requirements document,
- a formal specification document describing
    - the variables tracked by the system
    - the functions implemented by the system
- empty class definitions separated into distinct "modules" employing a Manager / Mediator pattern

You must do the following:

1. Convert the variables and functions to properties and behaviors in the appropriate classes
2. Make this folder an importable module
3. Diagram the modules and their dependencies

Details in the following slides.

This system is designed for managing wildlife populations, their habitats, and migration patterns. It allows users to register, update, and remove animals, habitats, and migration events, as well as retrieve and manage detailed information about them.

The **Animal Management** module enables users to track specific animals and modify their details.

The **Habitat Management** module handles the creation of habitats, their details (such as size, environment, and location), and the assignment of animals to them.

The **Migration Management** module allows users to create and manage migrations along predefined paths between habitats, including scheduling, viewing, and canceling migration events.

Users can also manage **Migration Paths**, specifying starting and ending habitats, species involved, and migration duration. The system provides powerful tools for organizing and monitoring the movement and habitat assignment of animals in conservation efforts.

Modules:
folder: animal_management

- animal_manager.py
- animal.py

folder: habitat_management

- habitat_manager.py
- habitat.py

folder: migration tracking

- migration_manager.py
- migration_path.py
- migration.py

```
from typing import Any, List, Optional


age: Optional[int] = None
animal_id: int
animals: dict[int, Animal] = {}
animals: List[int] = []
current_date: str
current_location: str
destination: Habitat
duration: Optional[int] = None
environment_type: str
geographic_area: str
habitat_id: int
habitats: dict[int, Habitat] = {}
health_status: Optional[str] = None
migration_id: int
migration_path: MigrationPath
migrations: dict[int, Migration] = {}
path_id: int
paths: dict[int, MigrationPath] = {}
size: int
species: str
species: str
start_date: str
start_location: Habitat
status: str = "Scheduled"


def assign_animals_to_habitat(animals: List[Animal]) -> None:
    pass
```

```python
def assign_animals_to_habitat(habitat_id: int, animals: List[Animal]) ->
None:
    pass

def cancel_migration(migration_id: int) -> None:
    pass

def create_habitat(habitat_id: int, geographic_area: str, size: int,
environment_type: str) -> Habitat:
    pass

def create_migration_path(species: str, start_location: Habitat,
destination: Habitat, duration: Optional[int] = None) -> None:
    pass

def get_animal_by_id(animal_id: int) -> Optional[Animal]:
    pass

def get_animal_details(animal_id) -> dict[str, Any]:
    pass

def get_animals_in_habitat(habitat_id: int) -> List[Animal]:
    pass

def get_habitat_by_id(habitat_id: int) -> Habitat:
    pass

def get_habitat_details(habitat_id: int) -> dict:
    pass

def get_habitats_by_geographic_area(geographic_area: str) ->
List[Habitat]:
    pass

def get_habitats_by_size(size: int) -> List[Habitat]:
    pass

def get_habitats_by_type(environment_type: str) -> List[Habitat]:
    pass
```

```
def get_migration_by_id(migration_id: int) -> Migration:
    pass

def get_migration_details(migration_id: int) -> dict[str, Any]:
    pass

def get_migration_path_by_id(path_id: int) -> MigrationPath:
    pass

def get_migration_paths() -> list[MigrationPath]:
    pass

def get_migration_paths_by_destination(destination: Habitat) ->
list[MigrationPath]:
    pass

def get_migration_paths_by_species(species: str) -> list[MigrationPath]:
    pass

def get_migration_paths_by_start_location(start_location: Habitat) ->
list[MigrationPath]:
    pass

def get_migrations() -> list[Migration]:
    pass

def get_migrations_by_current_location(current_location: str) ->
list[Migration]:
    pass

def get_migrations_by_migration_path(migration_path_id: int) ->
list[Migration]:
    pass

def get_migrations_by_start_date(start_date: str) -> list[Migration]:
    pass

def get_migrations_by_status(status: str) -> list[Migration]:
    pass
```

```python
def get_migration_path_details(path_id) -> dict:
    pass

def register_animal(animal: Animal) -> None:
    pass

def remove_animal(animal_id: int) -> None:
    pass

def remove_habitat(habitat_id: int) -> None:
    pass

def remove_migration_path(path_id: int) -> None:
    pass

def schedule_migration(migration_path: MigrationPath) -> None:
    pass

def update_animal_details(animal_id: int, **kwargs: Any) -> None:
    pass

def update_habitat_details(habitat_id: int, **kwargs: dict[str, Any]) ->
None:
    pass

def update_migration_details(migration_id: int, **kwargs: Any) -> None:
    pass

def update_migration_path_details(path_id: int, **kwargs) -> None:
    pass
```