Project Design: Test-Driven Development

The TDD "Double-Loop" Implementation

1. Objective

This assignment will bridge the gap between your high-level BDD (Behavior-Driven Development) design and the low-level code implementation. Your goal is to implement **two (2) core user stories** from your BDD design using a strict Test-Driven Development (TDD) "Red-Green-Refactor" cycle.

The focus of this assignment is **process over product**. The quality of your test suite and the discipline of your TDD workflow (proven by your Git history) are more important than a fully-featured, styled application.

2. Team Roles

With teams of 6-7, TDD is best practiced in pairs, with specific responsibilities. Please assign the following roles within your team. (For a 6-person team, one of the pairs can be a trio).

- **CI/Test Environment Manager (1 member):**
    - Responsible for initializing the Rails project (`rails new ... -T` to skip Minitest).
    - Installs and configures all testing gems (e.g., `rspec-rails`, `cucumber-rails`, `capybara`, `database_cleaner`).
    - Sets up the Continuous Integration (CI) pipeline (e.g., GitHub Actions) to run *all* tests (`rspec` and `cucumber`) automatically on every push.
- **BDD Feature Lead (1 member):**
    - ***Selects the two NEW user stories*** from the team's application. These should NOT be user stories from the BDD assignment.
    - Translates the Gherkin/user stories into a failing `*.feature` files using Cucumber. This person writes the *first* failing test (the BDD test) that kicks off the development cycle.
- **Backend TDD Pair (2-3 members):**
    - Focuses on the "inner loop" of TDD.

    - When a BDD test fails because a model is missing or lacks logic, this pair writes a failing *model spec* (RSpec) for the required validation, association, or method.
    - Writes the minimal model code to make their RSpec spec pass.
    - Refactors the model code.
- **Controller/Request TDD Pair (2-3 members):**
    - Focuses on the "outer loop" of TDD.
    - When a BDD test fails due to routing, controller actions, or response issues, this pair writes a failing *request spec* (RSpec) or *controller spec*.
    - Writes the minimal `routes.rb` and controller code to make their RSpec spec pass.
    - Refactors the controller code.

3. Assignment Tasks & Workflow

You will complete this workflow **twice**, once for each of the two user stories you selected.

Step 1: Setup (CI/Test Manager) [this step needs to be done only once, not for each user story]

1. Initialize the new Rails application. **Crucially**, disable the default Minitest suite: `rails new your_project_name -T`.
2. Set up your `Gemfile` with `rspec-rails`, `cucumber-rails`, `capybara`, etc. in the `:development, :test group`. Run `bundle install`.
3. Run the RSpec and Cucumber installation generators:
    - `rails generate rspec:install`
    - `rails generate cucumber:install`
4. Set up your CI configuration file (e.g., `.github/workflows/ci.yml`). This file should run `bundle install`, `rails db:setup`, `bundle exec rspec`, and `bundle exec cucumber`.
5. Push this initial setup to your team's GitHub repository.
    - **Goal:** A `main` branch where the CI pipeline runs and passes (with 0 tests).

Step 2: The "Outer Loop" (BDD Lead)

1. Create a new feature branch (e.g., `feature/user-signup`).

2. The **BDD Lead** writes the *first* Cucumber feature file (e.g., `features/user_signup.feature`) based on User Story #1. Write a single "happy path" scenario.
3. Run `bundle exec cucumber`. It must **fail** (this is your first **RED**).
4. Commit this failing test: `git commit -m "Add failing BDD test for user signup"`
5. Push the branch. The CI build should now fail.

Step 3: The "Inner Loop" - Models (Backend Pair)

1. The **Backend Pair** checks out the feature branch. They see the Cucumber test fails (e.g., "no such table: users" or "undefined method 'email'").
2. They identify the need for a `User` model.
3. **TDD Cycle 1 (Validation):**
   - **RED:** Write a *model spec* (`spec/models/user_spec.rb`) that tests a single validation (e.g., `it 'is invalid without an email'`).
   - Run `bundle exec rspec`. This one spec must **fail**.
   - Commit: `git commit -m "Add failing model spec for user email validation"`
   - **GREEN:** Write the *minimal* code in `app/models/user.rb` (e.g., `validates :email, presence: true`) to make the spec pass.
   - Run `bundle exec rspec`. The spec must **pass**.
   - Commit: `git commit -m "Make user email validation spec pass"`
   - **REFACTOR:** Is the code clean? (For a single validation, probably).
4. Repeat this R-G-R cycle for *all* required validations and associations for this user story (e.g., password presence, email uniqueness).

Step 4: The "Inner Loop" - Controllers (Controller Pair)

1. The **Controller Pair** pulls the latest changes. The model specs pass, but the BDD test still fails (e.g., "no route matches GET /signup").
2. **TDD Cycle 2 (Routing/Actions):**
   - **RED:** Write a *request spec* (`spec/requests/users_spec.rb`) that tests the `GET /signup` route (e.g., `it 'renders the new template'`).
   - Run `bundle exec rspec`. This new spec must **fail**.

- Commit: `git commit -m "Add failing request spec for GET /signup"`
- **GREEN:** Write the *minimal* code in `config/routes.rb` and `app/controllers/users_controller.rb` to make the spec pass (e.g., `get '/signup', to: 'users#new'` and an empty `new` method).
- Run `bundle exec rspec`. The spec must **pass**.
- Commit: `git commit -m "Make GET /signup request spec pass"`
- **REFACTOR:** Clean up the controller.

3. Repeat this R-G-R cycle for the `POST /users` (create) action, testing for a successful redirect and that a new `User` is created in the database.

Step 5: Closing the Loop (Whole Team)

1. At this point, all RSpec unit/request tests should pass.
2. Run the BDD test again: `bundle exec cucumber`.
3. It will likely fail on view-related steps (e.g., "cannot find 'Email' field").
4. As a team, implement the minimal `app/views/users/new.html.erb` and any remaining Cucumber step definitions to connect the working models and controllers.
5. Run `bundle exec cucumber` until it **passes** (this is your final **GREEN** for the BDD test).
6. Commit the passing BDD test: `git commit -m "Make BDD test for user signup pass"`
7. Push the branch. The CI build should now be 100% green (all RSpec and Cucumber tests pass).
8. Merge the feature branch into `main`.

Step 6: Repeat

Repeat steps 2-5 for your second user story on a new feature branch.

3. Submission

Submit a link to your team's GitHub repository. Do **not** squash and merge. The individual commits on your feature branches are the primary evidence of your TDD process and will be graded heavily.

4. Grading Rubric

Category	Needs Improvement (0-4 pts)	Satisfactory (5-7 pts)	Excellent (8-10 pts)	Weight
TDD Process (Red-Green-Refactor)	No clear R-G-R cycle. Tests are written *after* code. Commit history is one large "WIP" commit.	Some evidence of R-G-R. Tests are written *around* the same time as code, but not strictly *before*.	**Clear, demonstrable Red-Green-Refactor cycle.** Git history shows distinct "failing test," "passing test," and "refactor" commits.	**40%**
BDD/TDD "Double-Loop"	BDD (Cucumber) tests are ignored or written last.	A BDD test is written, but TDD (RSpec) unit tests are not clearly driven by it.	**A failing BDD (Cucumber) test is the *starting point*.** Failing TDD (RSpec) tests are then used to build the components (models, controllers) needed to make the BDD test pass.	**20%**
Test Quality & Coverage	Tests are trivial or only cover one happy path. Specs are missing for key validations or logic.	Tests cover happy paths. Most validations are tested.	Tests are comprehensive. They cover **happy paths, all model validations, and at least one "sad path"** (e.g.,	**20%**

Category	**Needs Improvement (0-4 pts)**	**Satisfactory (5-7 pts)**	**Excellent (8-10 pts)**	**Weight**
			"invalid email fails signup").	
CI & Environment	CI is not set up, or the final `main` build is failing. Test suites are not configured correctly.	CI is set up and runs, but may be flaky. Only one test suite (e.g., just RSpec) runs.	**CI pipeline is set up correctly.** It automatically runs *all* test suites (RSpec and Cucumber) on every push, and the final `main` branch is 100% green.	**10%**
Code Quality & Functionality	Application code is buggy, does not pass tests, or is poorly written (no refactoring).	Code works and passes tests, but shows little evidence of refactoring (e.g., large controller methods).	Code is clean, DRY (Don't Repeat Yourself), and well-factored. The two implemented features work as defined by the BDD specs.	**10%**
Total				**100%**