ICE 1: Project Bootstrap, Kanban & Team Sync

- **Objective:** Set up the team's repository, bootstrap it with standard process files, create a project Kanban board, establish the feature branching workflow, add a minimal Flask app, and perform a full-team `git pull` to confirm all members are operational.
- **Time Limit:** 40 minutes
- **Context:** For `Angband`, you cloned a "brownfield" repo. Today, you build the "greenfield" repo from scratch. Your first task is to bootstrap the repository with the standard "Ministry" process files (`LICENSE`, `CONTRIBUTIONS.md`, etc.), set up your team's task board, and conduct a "team sync" to prove everyone can successfully push and pull code.

Role Kit Selection (Strategy 1: Parallel Processing ⚡)

For this ICE, we will use the **Project Setup Kit**. Assign these three roles immediately. *Remember the course policy: You cannot hold the same role for more than two consecutive weeks.*

- **Repo Admin:** (GitHub & Repo): Creates the repo, adds starter files to `main`, sets up the Project Board, and manages collaborators.
- **Process Lead:** (Branching & Process): Manages the Git feature branching, leads the `CONTRIBUTIONS.md` edit, and manages the final PR.
- **Dev Crew:** (Code & Local Env): Sets up the `venv`, installs dependencies, writes `app.py`, and confirms the app and tests run locally.

Task Description: "The Ministry is Founded"

This task is sequential. Call out when you are done with your part so the next person can begin.

Part 1: Repo & Project Board Bootstrap

- **(Repo Admin)**
  1. Create a new, **public** GitHub repository. **You must use the format:** `FA25-SE1-TeamXX-moj` (e.g., `FA25-SE1-Team01-moj`).
  2. Go to `Settings` > `Collaborators` and add all team members, your TA, and the instructor (`@seiffert`).
  3. In your *local terminal* (not the web UI), `git clone` the new, empty repo.
  4. Download the `moj-starter-files.zip` (provided on Canvas) and unzip them into your cloned repo. This should add all starter files (`LICENSE`, `CONTRIBUTIONS.md`, `requirements.txt`, `tests/test_placeholder.py`, etc.).
  5. Commit these files directly to the `main` branch and push:

```
git add .
git commit -m "chore: initial project bootstrap with process
files"
git push
```

6. **Project Board Setup:** In the GitHub repo, go to the `Projects` tab. Create a new "Project." Select the "Team backlog" or "Kanban" template.
7. **Add Starter Tasks:** Add 5 "tasks" (as draft issues) to your new board: `Set up CI/CD pipeline`, `Define database models`, `Create user registration endpoint`, `Implement authentication`, `Test joke submission endpoint`.
8. **ANNOUNCE:** "The repo is live and bootstrapped. Everyone `git clone`."

Part 2: Feature Branching

- **(Process Lead)**
  1. After *everyone* has cloned the repo, you will create the first feature branch.
  2. In your terminal, run: `git checkout -b ice1-setup` (Note: `ice1-setup`, not `bootstrap`!)
  3. Push this new branch to the remote repo: `git push -u origin ice1-setup`
  4. **ANNOUNCE:** "The `ice1-setup` branch is live. Everyone run `git fetch` and then `git checkout ice1-setup`."

Part 3: "Hello, Ministry!" & Local Verification

- **(Dev Crew)**
  1. Make sure you are on the `ice1-setup` branch.
  2. Create a virtual environment: `python3 -m venv venv`
  3. Activate it: `source venv/bin/activate` (Mac/Linux) or `.\venv\Scripts\activate` (Windows).
  4. Install the dependencies from the file the `Repo Admin` added: `pip install -r requirements.txt`
  5. Create a file named `app.py` and add the "Hello, Ministry" code:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_ministry():
    return "Hello, Ministry!"

if __name__ == '__main__':
    app.run(debug=True)
```

  6. **Verify everything locally!** Run these two commands:
     - `flask run` (and check `http://127.0.0.1:5000` in your browser).
     - `pytest` (and confirm it finds and passes `1 test` from `test_placeholder.py`).
  7. Commit your work to the `ice1-setup` branch and push:

```
git add app.py
git commit -m "feat: add initial 'Hello Ministry' flask app"
```

```
git push
```

8. **ANNOUNCE:** "The Flask app is pushed. Everyone `git pull`."

`CONTRIBUTIONS.md` Log Entry & Team Sync

One team member share their screen. You are now editing the **existing** `CONTRIBUTIONS.md` file.

1. **Process Lead:** Make sure everyone on the team has pulled the `app.py` file from the `Dev Crew`.
2. **Process Lead:** Open `CONTRIBUTIONS.md` and **fill in the template** for ICE 1 (the header and summary are already there). You only need to add:
   - The `Date`
   - The `@github-user` for all `Team Members Present`
   - The `@github-user` for each `Role`
3. **Process Lead:** Commit and push this change:

```
git add CONTRIBUTIONS.md
git commit -m "docs: log ICE 1 team sync"
git push
```

4. **CRITICAL STEP: All other team members** must now run `git pull` one last time.

Definition of Done (DoD) 🏁

Definition of "Class-Done" (End of Class)

Your team's work is "Done" for class when you can check all of the following:

- ☐ **Team Sync:** ✅ **All team members** have successfully `git pull`-ed the final `CONTRIBUTIONS.md` update to their local `ice1-setup` branch.
- ☐ **Local Functionality:** All members have confirmed they can `flask run` the app and `pytest` runs successfully.

Definition of "Done-Done" (Due: Midnight Tonight)

You can complete these steps outside of class.

- ☐ **Repository:** Repo is named correctly (`FA25-SE1-TeamXX-moj`).
- ☐ **Collaborators:** All team members, TA, and Instructor are added.
- ☐ **`main` Branch:** `main` is clean (contains *only* the starter files).
- ☐ **Project Board:** The GitHub Project (Kanban) is created and has 5 starter tasks.
- ☐ **`CONTRIBUTIONS.md`:** The file is updated on the `ice1-setup` branch with date, members, and roles.
- ☐ **Submission:** A Pull Request is open (`ice1-setup` -> `main`) with your TA assigned as a "Reviewer".

- [ ] **Canvas:** The PR URL is submitted to Canvas.

Grading Rubric (10 points total)

Points	Category	Criteria
5 pts	**Repository Setup**	[ ] Repo is named correctly (`FA25-SE1-TeamXX-moj`). [ ] All members + TA + Instructor added as collaborators. [ ] `main` branch is clean (only starter files). [ ] `ice1-setup` branch contains `app.py`. [ ] PR is correctly opened from `ice1-setup` -> `main`.
2 pts	**Project Board**	[ ] GitHub Project (Kanban) is created. [ ] At least 5 starter tasks are added to the board.
2 pts	`CONTRIBUTIONS.md` **File**	[ ] The *updated* file (with date, members, roles) is committed to the `ice1-setup` branch. [ ] The ICE 1 entry is complete and correct.
1 pt	**PR Submission**	[ ] TA is correctly assigned as a "Reviewer" on the PR.

TA Follow-up Guide

- **In-Class Goal (Workshop):** Your #1 priority is **blocker removal**. The team's goal is *not* to submit the PR in class, but to get every member to a state where they can `git pull`, `git push`, and `flask run`. The "Team Sync" (`push`/`pull` of `CONTRIBUTIONS.md`) is the test for this.
- **Final DoD (Due 11:59 PM):** The team submits a *perfect* PR for you to review.
- **Common Pitfalls & Coaching (Revised):**
  - **The #1 Blocker: SSH Key Hell.**
    - **Symptom:** `git clone` or `git push` fails, or asks for a password.
    - **The Fix:** This is your job. Walk them through `ssh-keygen`, adding the `.pub` key to GitHub, and ensuring their remote URL is the `ssh` version (not `https`). The new `push`/`pull` requirement is *designed* to force this error to happen in class.
  - **The #2 Blocker: Python/`venv` Path Hell.**
    - **Symptom:** `python -m venv venv` fails, or `source venv/bin/activate` fails (especially Windows vs. Mac), or `pip install` fails with "permission denied."
    - **The Fix:** Coach them on how `venv` works. Show them how to check (`which python3`) and be explicit (e.g., `python3.10 -m venv venv`).
  - **The #3 Blocker: Merge Conflict on `CONTRIBUTIONS.md`.**
    - **Symptom:** `git pull` on the final step fails due to a merge conflict.
    - **The Fix:** This happens if they didn't follow the "one person screen-share" rule. This is a perfect "coachable moment." Walk them through a `git pull`, fixing the merge conflict (accepting both changes), `git add`, `git commit`, and `push`ing the resolved file.
- **Coaching Questions (for next team meeting):**

    - "Your `Repo Admin` added several files you didn't write, like `LICENSE` and `DOCUMENTATION_POLICY.md`. Why does a real-world software team bootstrap a project with these 'process' files *before* writing any code?"
    - "The main goal today was the 'Team Sync' (the final `git pull`). Why did we make that the *in-class* goal, instead of finishing the PR?" (Looking for "teamwork," "ensuring everyone's environment works," "finding setup bugs early").
    - "I see you set up your Project Board. How do you plan to use that board to coordinate your work for the first project cycle?"

- **Feed-Forward Prompts:**

    - "You've proven `pytest` runs *locally*. In our next class (ICE 2), we will build the GitHub Action to run it *automatically in the cloud* every time you push. Before then, find the 'Actions' tab on your GitHub repo. What do you see there?"

Pedagogical Analysis

- **Core Goal (SE Experience):** This simulates "Day 1" on a real team. A senior dev (`Repo Admin`) sets up the repo with company-standard files (`LICENSE`, `CONTRIBUTIONS.md`), a standard naming convention, and the initial task backlog (the Kanban board). The dev team (`Dev Crew`) then checks out the repo, adds the first feature (`app.py`), and verifies the *existing* test/build infrastructure works (`pip install`, `pytest`).
- **Developer Workflow Competency:** This is hyper-focused on the *collaborative* Git workflow: `clone`, `checkout -b`, `push`, and (most critically) `pull`. The final "Team Sync" step is a scaffolded exercise in team synchronization.
- **Ethical Challenge:** This artifact does not include an ethical challenge. The task is purely technical setup.
- **Historical Context:** This artifact does not include a historical hook.
- **Creative Engagement:** The exercise uses the "Ministry of Jokes" narrative ("The Ministry is Founded") to frame the technical setup.
- **Cognitive Load Management:** This version has excellent load management.
    1. **De-risked Time:** By moving the PR to midnight and focusing the in-class goal on the "Team Sync," the "finish" pressure is gone.
    2. **Scaffolded Files:** The cognitive load of *creating* `requirements.txt`, `.gitignore`, `tests/`, and `CONTRIBUTIONS.md` is removed. The team only has to *use* them, which is a much more focused task.
    3. **Clear Goal:** The objective is a single, clear team action: "everyone `git pull`." This is an achievable, concrete, and high-value goal for the first day.
    4. **Parallel Tasks:** The `Repo Admin`'s new "Project Board" task can be done in parallel while the `Process Lead` and `Dev Crew` are working on their items, making good use of time.