THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

8. Project Intro

CSCI 2541 Database Systems & Team Projects

Gabe

Slides adapted from Prof. Bhagi Narahari, Wood,; and Silberschatz, Korth, and Sudarshan

The Project

A significant part of this course is a large database systems **team** project.

In the project you will design & implement a database system — Full stack development:

- Front End (HTML/CSS & optional Javascript)
- Application server – in Python with Flask
- DBMS backend – MySQL

The project will involve working in teams of 3.

- Form to declare teams is active on schedule, due tonight!

University Management System

You will build a web application to manage many aspects of a university

1. Application processing system (APS)

2. Course registration system (REGS)

3. Advising system (ADS)

Application Processing System (APS)

Think: grad student application to a department

a. Applicant (registration, etc…) applies
   - Can check status at any point
b. Recommendation letters are submitted (emulating email)
c. Multiple reviewers evaluate
d. A single chair/director decides
   - admit
   - admit with aid
   - reject
e. Update admission status of applicant

Course Registration System (REGS)

- Students create accounts
- Can add/drop courses
- Class history for students
  - Grades
  - Classes, time, place
- Faculty assign grades
- Retrieve transcript
- Check for schedule conflicts
- Add course offerings in valid slots
  - location conflicts
- PhD students have limited registrations

Advising System (ADS)

- DEGREEMAP-ish
- Student registration
- Assumes students have a class record (manual)
- Students have a class plan
  - checked for completeness
- Faculty advisor views and approves plan
- Graduation audits
  - check if degree requirements are met
  - advisor must still approve
- Alumni tracking

Academic Honesty

- You **cannot** share or view other student's code outside of your group
  - in the class or not
- Online documentation
  - You can read/use online documentation
  - Use general advice online not specific to your assignment
  - Emulate "real world"
  - You *cannot* use automated assistants to generate/suggest code – no LLMs!
- This policy replaces conflicting AH policies for the class

Two Phases: Impl + Integration

Project broken into 2 phases:

- Phase 1: teams build an application assigned to the team
  - Includes reports containing the design
  - Demo of the Application
- Phase 2: Work in new teams to integrate different applications and produce the final project assigned to you
  - Teams chosen by me based on teamwork in Phase 1
  - This requires integration and NOT redesign

Team Project

You **have** to work in teams

- Each team member required to ‘produce’ equitable share ‘product’
- Teamwork will be assessed…
  - Not all team members may get the same grade on the project!
  - You must bring teamwork issues to attention of the instructor

Mentors and Grading

Mentors

- “Mentor meetings” in the schedule
- One of the class periods/week
- Track progress, teamwork

Grades

- Mentor meetings
- Demo

Github + Github Projects

Team Project….Warnings!

Requirements (for each Phase) will be provided, and your project must meet minimum requirements

- This only gets you a 80% - need to innovate to earn more!

You have to submit a working project

- No partial credit if your project does not work – you get a zero!

What is a working project ?

Must meet all the specifications

Must have correct and complete workflow

- Workflow specified in the assignment

“Reasonable” user interface

- Easy to use
- You have to come up with “easy to use” based on user interfaces you have seen/used.

Extra features – after you have a working system

Working in Teams

Each team works on their assigned project

Team members have to take lead on some aspect of the project

- Workload distribution is important
- Grades for team members can vary based on the project evaluation of each aspect of the “product” you produce

Every team member has to contribute to all aspects of ‘full stack’ development

- You can’t work on just HTML/CSS !!

We will be using some of the weekly class sessions (lecture or lab) for teamwork – but this is not enough to work as a team!

Warning: Team work

Team members must:

- Be respectful
- Divide work up evenly between all members
- Work on both front and backend

Teamwork Rules

Extensive set of rules you must follow

- Communication – early and often
- Respect – responsibility and follow-through
- Planning – everyone on the same page
- Allocating work – clear division of labor/plan
- No procrastination – makes everything else impossible
- Early problem notification – let me help you
- Flexibility – different working styles/schedules
- Professionalism – co-worker not college kid, SEH not dorm
- No type-casts – everyone does interesting work

Teamwork Traps

Extensive set of rules you must follow, and **common traps**

- Radio silence
- “do the documentation” “...project management”
- “I’ll just do it all”
- Altering other’s code without their approval
- Their code, your commit
- Falling behind, but not helping a back-up plan
  - “I’ll have it done tomorrow…”
- Gossip
- Asserting organization – need consensus on, e.g. when to meet

Team Assignment

You will form a team of **3 students** or you can be randomly assigned, discuss NOW!

Are we done? Do we have the teams?

– form on class webpage in schedule to declare them!

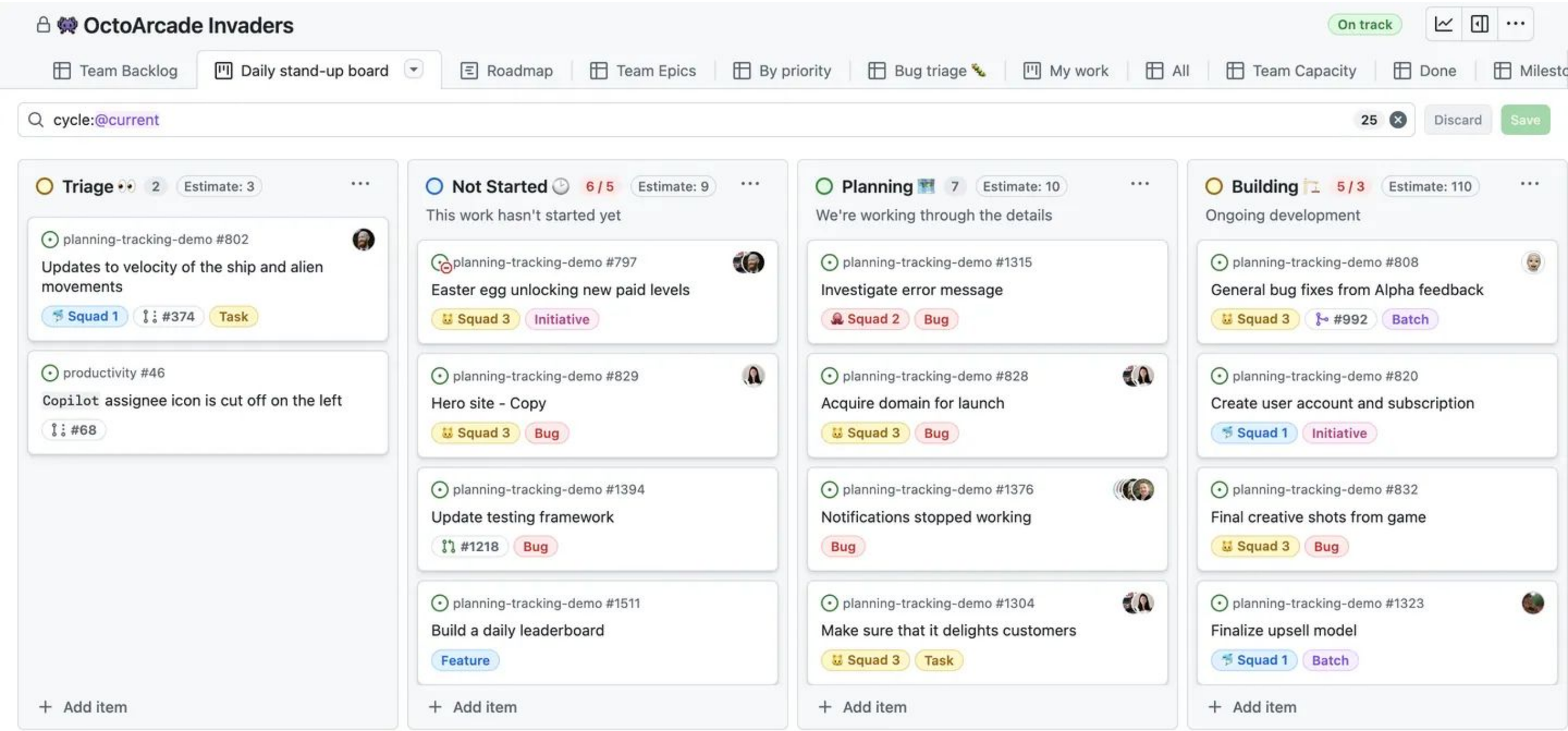
Amazon RDS

Must use this for your projects.

- Will setup the week after spring break
- Will need to migrate from sqlite -> RDS

Using Github Projects + Issues

- Activate Projects (link in project webpage)
- Create issues, and manage them as cards



Using Github Projects + Issues

- Issues are your plan + TODO list
- They track bugs, features, etc…
- They assign the work to specific team members
- They can/should refer to commits relevant to them
- Projects enables you to visually manage them

- You'll work with your mentors by showing them your Github Project, and walking them through progress

Using git Branchs

- You'll use the feature branch git pattern
  - New branch for each feature, bugfix, etc…
  - Team members will merge into main with discussion and coordination
  - Don't forget to merge from master in your branch frequently to avoid big conflicts
  - Prefer `git pull –rebase` over merging (lab today)

git Commits: Commit Messages

`git commit -a` (not `git commit -am "..."`)

```
type/name: single-line, brief description

Longer description in a paragraph of the
commit.

- brief summary of a change
- another change
- etc...
```

git Commits: Commit Messages

Why

- Your commit history should tell a story of your features
- Others should be able to navigate changes only form your commit history
- `git log` should be USEFUL
- Which commit causes a bug? `git bisect`

- db commits? `git log -n 100 | grep "db:"`
- db bugfixes? `git log -n 100 | grep "bugfix/" | grep "db:"`

git Commits: Size

Your commits should not be larger than 100 lines.

- Example: `git diff | wc -l` should be < 100
- Commits should be small and intentional
  - If not well tested, “Note: not well tested” in commit message
  - Never push an untested set of commits
- Many commits means you need to plan the progression of your code
  - What are the steps for what you’re trying to accomplish?

What's Next

Will post ADS/REGS/APPS pdfs on pdf for immediate consumption

- Will post classroom link for teams to clone repo.
- Fill out form on webpage (in the schedule)
- Will share a spreadsheet assigning each team to a module
- You need to be off to the races!!! Start ASAP!