Project 2 general requirements:

Web app with DB for persistent data.

Data Model: At least two many-to-many relationships

User interactions: More than just do a couple buttons.

(View certain things, search, place order with multiple things.)

Authorization: At minimum, at least one with different privileges (like an admin that can do everything).

[CI Pipeline, deployed to app service.]

Got to be MVC, EF, some SQL, plus anything you want to learn yourself.

Need team, in one paragraph what the project is and what the user can do. User stories. Multiplicities in the data model.

Minimum Viable Product (MVP): If you have an overall ambitious project, have a part to focus on to have an acceptable project.

**Idea1 - Sci-Fi Game**

DB Tables –

Users: Login info, adminOrNot.

Characters: Stats. (FK to user). FK to race/class?

Race/class?

Inventory: Quantity, equippedOrNot (Joint table between characters and game items).

Items: Description, stats.

Enemies: Description, stats.

Drops: Quantity (Joint table between enemies and game items).

Save state ?

Could have save state be another column that with the character id is used to uniquely identify where the character is in the game/what is equipped?

In other words, could have duplicate character stats, but different inventory/enemy.

User Interaction -

User can create a character.

User can choose character/save state to load.

User can save/load the current character.

User can equip character with items.

User can fight an enemy.

User can train character (increase stats. Cost gold?).

Admin can view tables.

Admin can add items.

Admin can add/update enemies

**Idea2 – Calendar**

Add/remove holidays and events.

Add/remove tasks.

Mark tasks as done.

Search holidays (get day and description).

Change gui?

Admin can add holidays to db.

Admin can view all events?

**Idea3 – Adventurers' Guild**

User stories -

Any user can create customer account.

Any user can login.

Customers can create requests.

Customers can see their request status.

Receptionist assigns a rank to a pending request.

Receptionist approves/declines a pending request.

Adventurer can accept a request of their rank or below.

Adventurer can update their accepted requests' status (in-progress, completed).

Adventurer can update their stats.

Adventurer can attempt to rank-up.

Guild Master/Leader can view everything in DB.

Guild Master/Leader can set rank's posting fee.

Guild Master/Leader can set an account's permission level.

Guild Master/Leader can assign funding (receptionist and adventurer salary).

Guild Master/Leader can update rank requirements.

Tables –

User: ID, LoginInfoFK, Name (Unique), salary (nullable), stats (nullable), RankIDFK (nullable).

LoginInfo: ID, username, password, permissionLvl.

Request: ID, RankIDFK, description, completionRequirement, cost, adventurerpartyRewards, State (pending, declined, approved, accepted, in-progress, completed).

RequestingParty (Customers\_Requests): ID, Joint table of Customers and Requests they've placed.

AdventurerParty (Adventurers\_Requests): ID, Name, Joint table of Requests and Adventurers who accepted them.

Rank: ID, Name, Fee.

AdventurerRankUpRequirements: ID, CurrentRankIDFK, MinNumberRequests, MinTotalStats, NextRankIDFK.

[PermissionLvls: Lvl1 just customer, Lvl2 just receptionist, Lvl3 just adventurer, Lvl4 customer+receptionist (need check to prevent approving own request), Lvl5 customer+adventurer (need check to prevent accepting own request), Lvl6 Receptionist +adventurer, Lvl7 all 3 (need check to prevent approving/accepting own request), Lvl8 guildmaster. ]

,NET Core with Sonar template

ASP.NET Core for example of publish task

Add publish task: Name Publish, copy buildconfig arg, (Since we're doing separate repos for each, the other defaults are okay?)

Add Azure task:

Nick will show example Monday

/az pipelines

setup granularity

/az pipelines *build-url*

/github subscribe