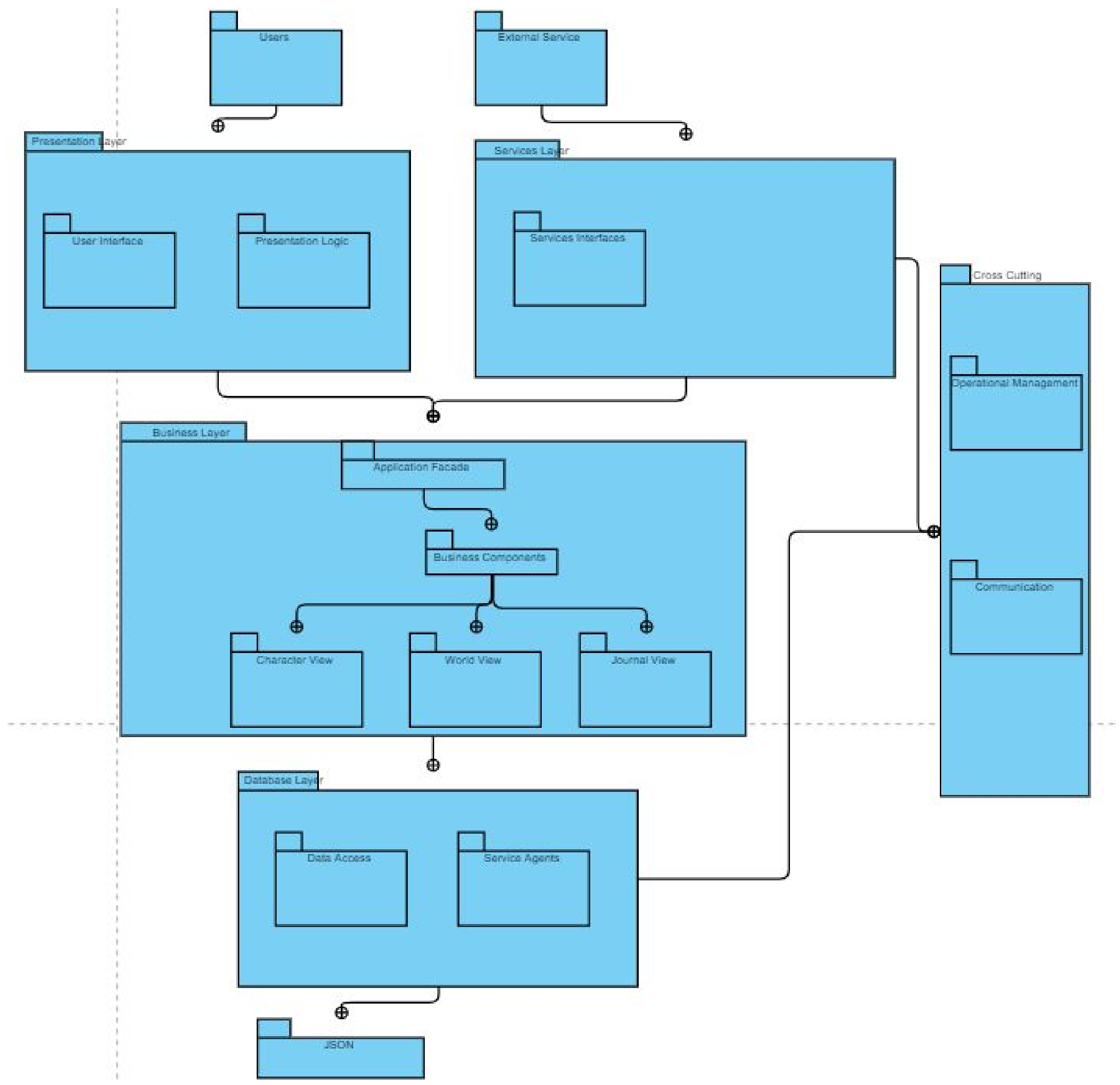
1. Description

The problem of disorganized preparation and running of tabletop roleplaying games (TTRPGs) affects both player and game master; the impact of which is a much less enjoyable game for all parties involved. For game masters who regularly run games, Cosmos Smithy is an easy to use tool that allows game masters to organize and run their game all in one place. Unlike other products that only do one or two things, our product will manage everything from preparation to running the game, and can function as a general note taking app as well. Cosmos Smithy is a web application for Dungeon Masters to organize their games, to create, generate, organize, and run their games all in one place when they need creative assistance. In particular, people who participate in TTRPG's, specifically writers and Game Masters, will have an interest in this system.

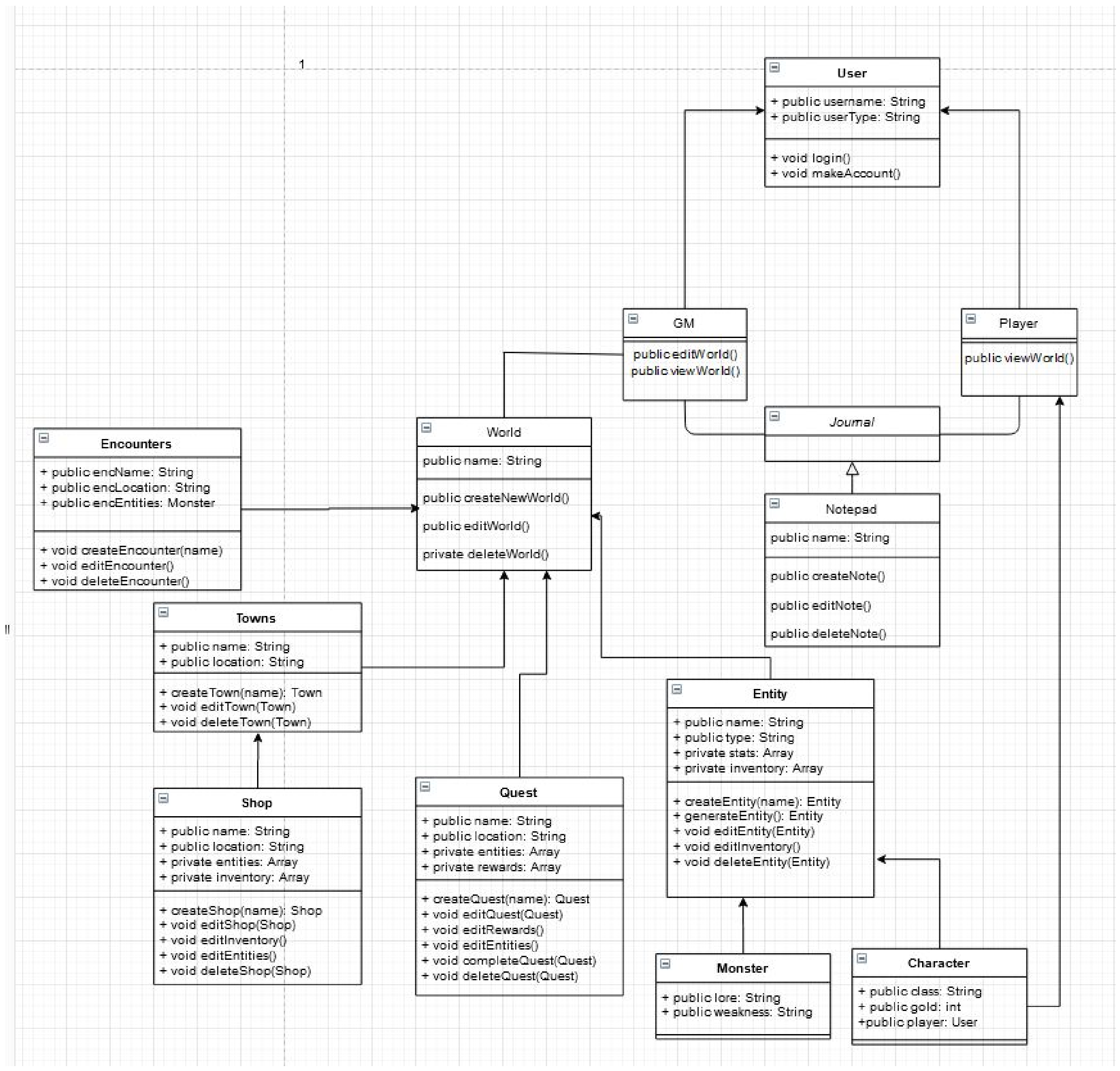
In this system, users can login. Game masters (GMs) can create a new world or access an existing world. Selecting the world by name allows the user to build and access encounters, towns, and quests. These all allow the GM to better keep track of their game and the players progress inside of it. Both Game masters and players may create a journal which contains a notepad. Users may create, edit and delete notes. Players can create or generate player characters. Entities include monsters, characters, and non-player characters, which all serve a different function in the game world.

2. Architecture

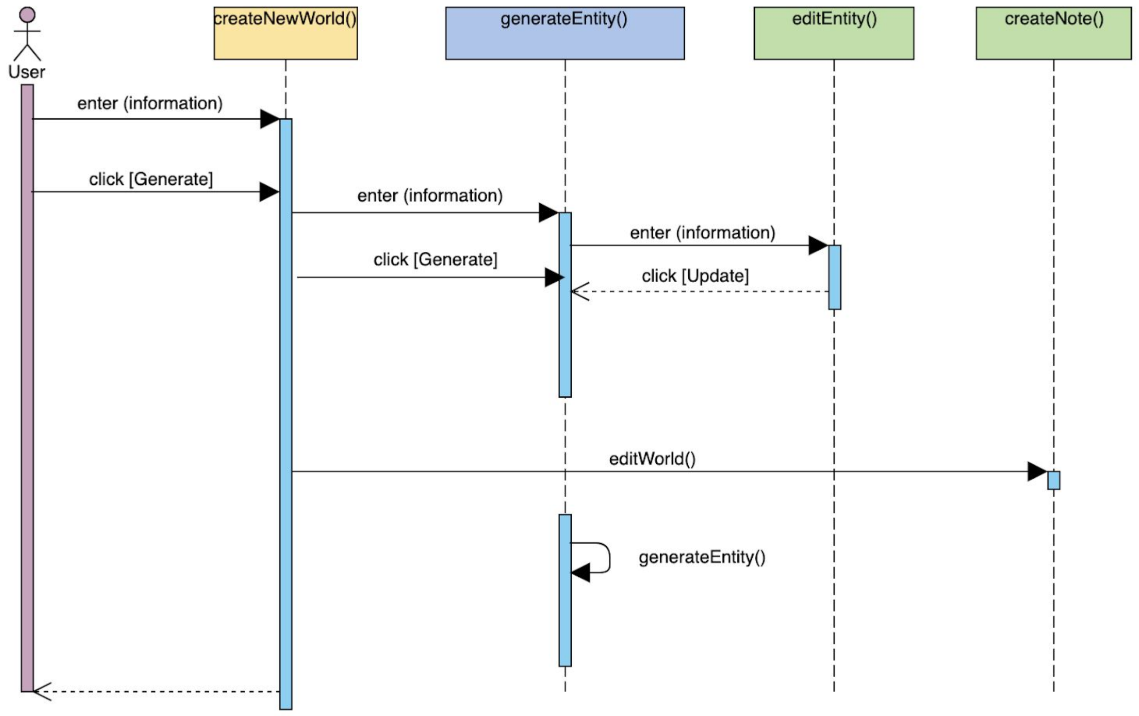


We mainly modeled it after the example given and decided to drop certain models like security and business entities. We elaborated that the business components can be modeled after three distinct views. We also noted that the database model we were accessing was a JSON.
*Apologies if the font is small the application would not allow us to edit the font size nor the box that houses the different layer names.

3. Class diagram

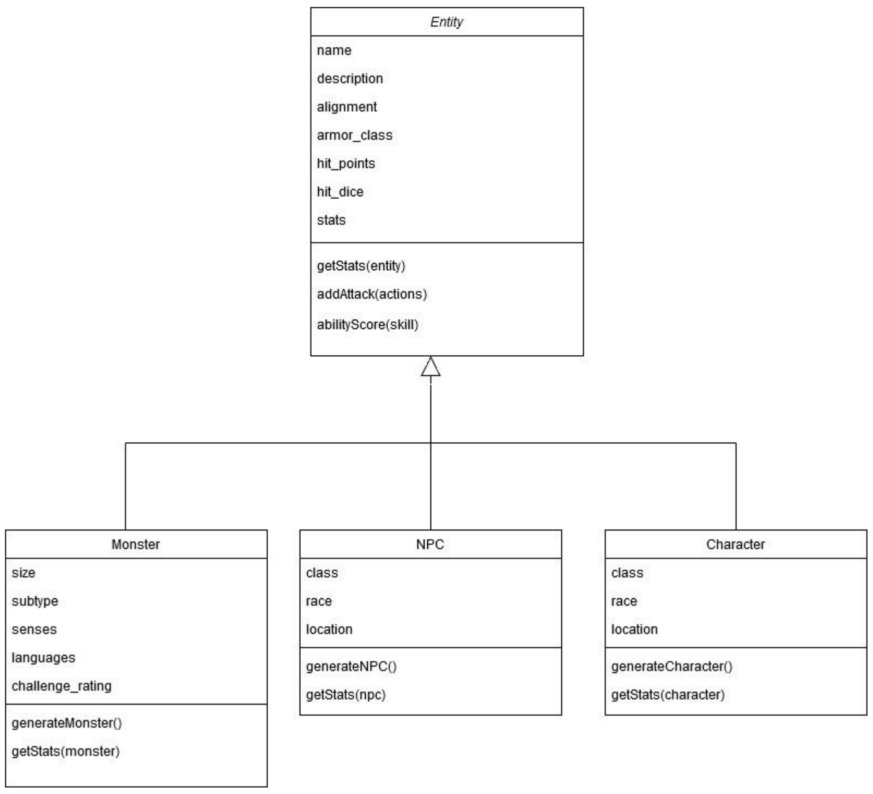

1
User
+ public username: String
+ public userType: String
+ void login()
+ void makeAccount()
GM
public editWorld()
public viewWorld()
Player
public viewWorld()
Journal
Notepad
public name: String
public createNote()
public editNote()
public deleteNote()
Encounters
+ public encName: String
+ public encLocation: String
+ public encEntities: Monster
+ void createEncounter(name)
+ void editEncounter()
+ void deleteEncounter()
World
public name: String
public createNewWorld()
public editWorld()
private deleteWorld()
Towns
+ public name: String
+ public location: String
+ createTown(name): Town
+ void editTown(Town)
+ void deleteTown(Town)
Entity
+ public name: String
+ public type: String
+ private stats: Array
+ private inventory: Array
+ createEntity(name): Entity
+ generateEntity(): Entity
+ void editEntity(Entity)
+ void editInventory()
+ void deleteEntity(Entity)
Shop
+ public name: String
+ public location: String
+ private entities: Array
+ private inventory: Array
+ createShop(name): Shop
+ void editShop(Shop)
+ void editInventory()
+ void editEntities()
+ void deleteShop(Shop)
Quest
+ public name: String
+ public location: String
+ private entities: Array
+ private rewards: Array
+ createQuest(name): Quest
+ void editQuest(Quest)
+ void editRewards()
+ void editEntities()
+ void completeQuest(Quest)
+ void deleteQuest(Quest)
Monster
+ public lore: String
+ public weakness: String
Character
+ public class: String
+ public gold: int
+public player: User

4. Sequence diagram



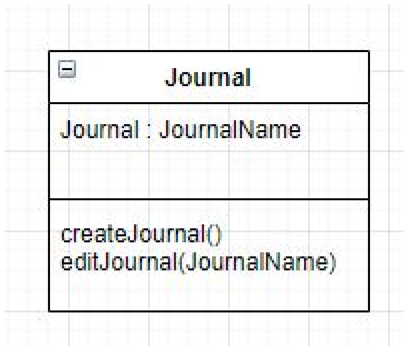
5. Design Patterns

Section 1 : Template Pattern



https://github.com/bmvisoky/CosmosSmithy/blob/master/src/components/pages/journalPage.js

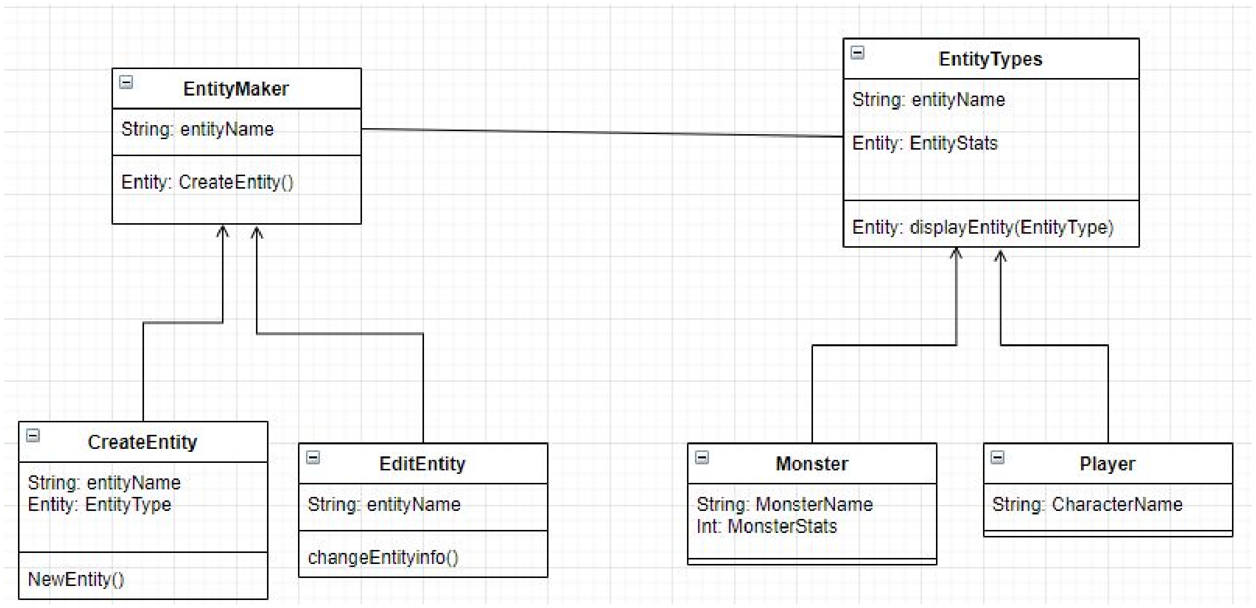
Section 2: Singleton Pattern



NOTE: for purposes of this assignment, the journal class was detached from needing user info to attach the journal to in order to be properly a singleton.

https://github.com/bmvisoky/CosmosSmithy/blob/master/src/components/pages/journalPage.js

Section 3: Bridge Design Pattern



NOTE: For the sake of keeping the Image smaller, entity stats are compressed into a single variable

https://github.com/bmvisoky/CosmosSmithy/blob/master/src/components/pages/characterPage.js

6. Design Principles

How does your design observe the SOLID principles? Provide a short description of the following principles giving concrete examples from your classes.

Single Responsibility Principle: The single responsibility principle is a principle that states an object should only change for one reason.

In our system, the **Entity** class changes only when the DM asks it to, as the creatures of the world exist at the discretion of the DM and no one else.

The Open-Closed Principle (OCP) states that software entities (classes, modules, methods, etc.) should be extendable but not modifiable.

In our system, the **Towns** class is easily extendable, as you can add more towns as desired to the world. Modification, however, is not possible as the general structure of a town is rigid, and should not change for consistency.

The Liskov Substitution Principle (LSP) states that subclasses must be substitutable for their base class.

In our system the **Entity** class and it's subclasses (Monster, NPC, Player Character) is a strong example of LSP, as the subclasses of **Entity** can substitute for it

The Interface Segregation Principle (ISP) states that client methods should not need to use methods that they don't want to use/outright do not use.

In our system, no methods are required for *most* usage. World creation is required, but the creation of towns, encounters, entities, etc. are not required for playing the game (While it would be dull to play a game without most of these). You can call methods for creation and editing of all of these things without needing the others to exist in the world. For instance, you don't need towns or quests in order for an encounter to occur.

The Dependency Inversion Principle says that higher level modules should not depend on lower level modules.

In our system, the **Town** class is good to show this, as you don't need the shop class in order to create a town