PROJECT Design Documentation

> *The following template provides the headings for your Design Documentation. As you edit each section make sure you remove these commentary 'blockquotes'; the lines that start with a > character and appear in the generated PDF in italics but do so only **after** all team members agree that the requirements for that section and current Sprint have been met. **Do not** delete future Sprint expectations.*

Team Information

- Team name: TEAMNAME
- Team members
    - Klejdis
    - Ashrith
    - Sejal
    - Jon
    - Nicholas

Executive Summary

This is a Puzzles E-Store appliation.

Purpose

> The primary goal of this project is to develop an e-commerce website dedicated to selling puzzles. The platform aims to provide a seamless online shopping experience for customers while offering a comprehensive management system for the store owner.

Glossary and Acronyms

Term	Definition
SPA	Single Page
MVVM	Model-View-ViewModel
API	Application Programming Interface
REST	Representational State Transfer
CRUD	Create, Read, Update, Delete
UML	Unified Modeling Language
OOP	Object-Oriented Programming
TDD	Test-Driven Development
DAO	Data Access Object

Requirements

Definition of MVP

> The Minimum Viable Product (MVP) for this ecommerce store project is a basic version of the ecommerce platform that includes essential features to demonstrate the core functionality of the system. The MVP aims to provide a functional and user-friendly platform for both customers and the e-store owner, focusing on the core features that are necessary for the initial launch.

Key Features of the MVP:

> **1. Authentication:**
>
> - ***Minimal Authentication***: The system will implement a basic authentication mechanism that allows users to log in and log out. The authentication process will be simplified, trusting the browser to identify the user. A simple username is required for login, with the assumption that logging in as an admin represents the e-store owner. This approach is not secure and is acknowledged as such, focusing instead on the functionality rather than security for the MVP.

> **2. Customer Functionality:**
>
> - ***Product Listing:*** Customers can view a list of available products.
> - ***Product Search:*** Customers can search for specific products within the store.
> - ***Shopping Cart Management:*** Customers can add or remove items from their shopping cart.
> - ***Checkout Process:*** Customers can proceed to check out their items for purchase, completing the transaction process.

> **3. Inventory Management:**
>
> E-Store Owners can manage the inventory by adding, removing, and editing product data. This functionality is crucial for maintaining the store's stock and ensuring that customers can purchase the items they are interested in.

> **4. Data Persistence:**
>
> The system will save all data to files, ensuring that changes made by one user (such as adding items to a shopping cart) are persisted and visible to other users upon their next login. This approach replaces the use of a database for the MVP, focusing on the use of basic file I/O operations to achieve data persistence.

> **5. 10% Feature Enhancement(s):**
>
> **1.Custom Puzzle Products**
>
> - Customers can submit their own images and order custom puzzles.
> - The custom puzzle will get created and then added to their cart, but not to the actual store inventory.
>
> **2. Gift Options:**
>
> - Customers can choose to gift the puzzles to someone else.

> - That will move them to a different menu where they can write a gift message and the email of the gift recipient.
> - The product will get emailed to the recipient with the message.

MVP Features

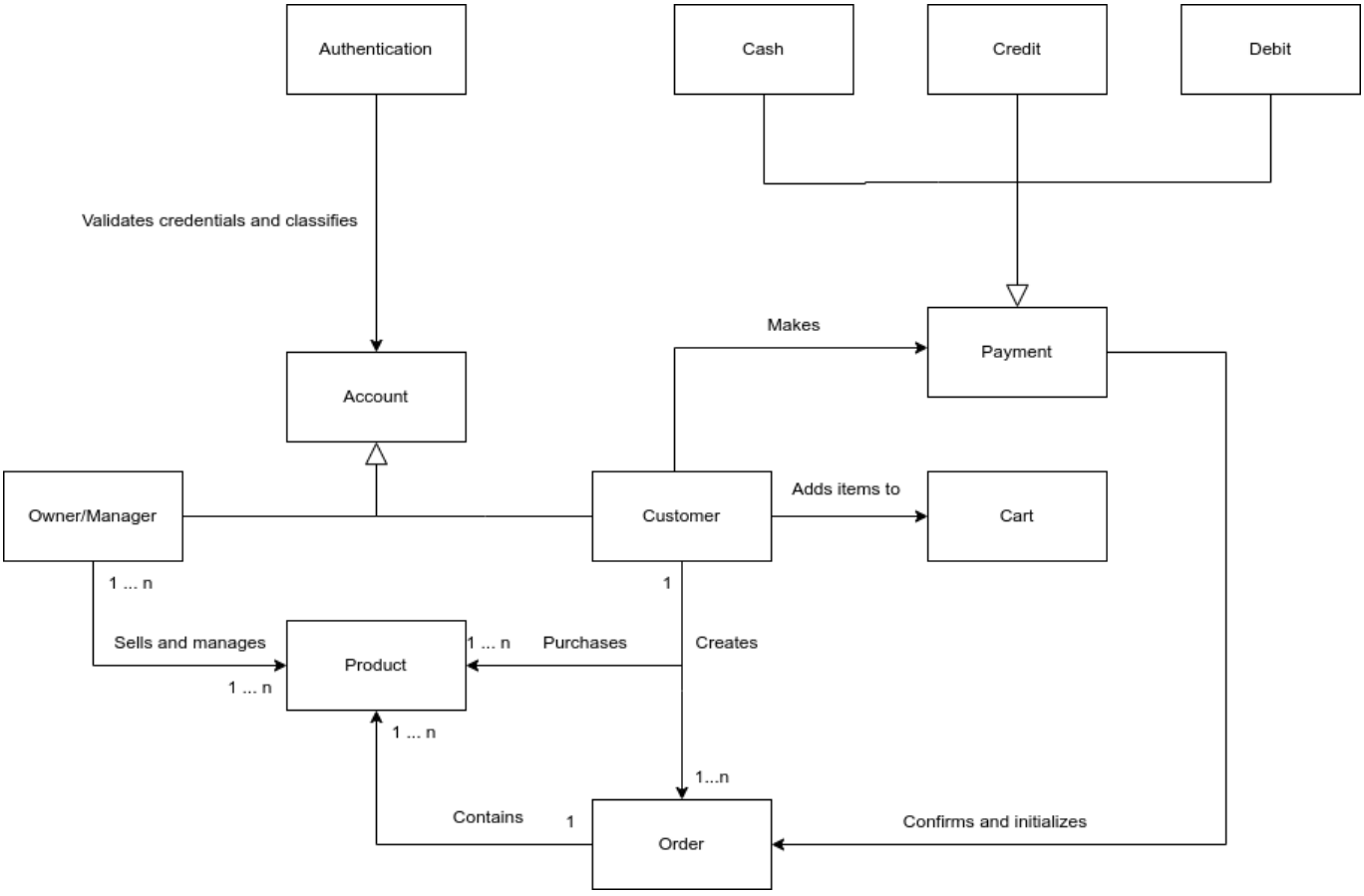
> ***[Sprint 4]*** *Provide a list of top-level Epics and/or Stories of the MVP.*

Enhancements

> ***[Sprint 4]*** *Describe what enhancements you have implemented for the project.*

Application Domain

This section describes the application domain.



> ***[Sprint 2 & 4]*** *Provide a high-level overview of the domain for this application. You can discuss the more important domain entities and their relationship to each other.*

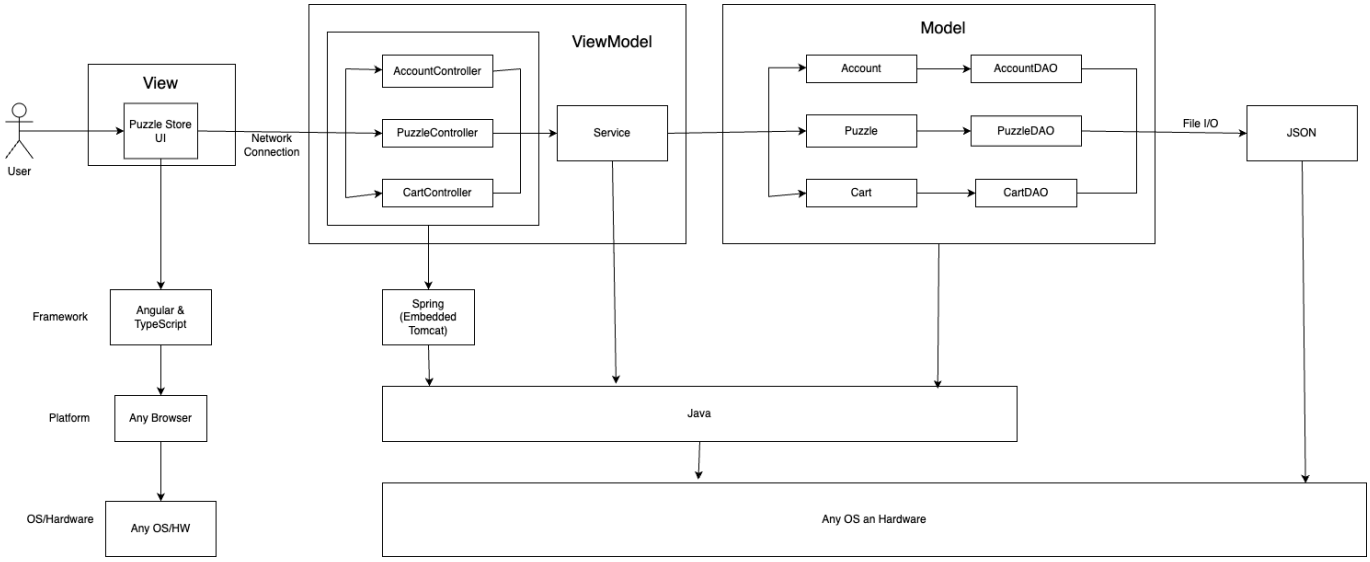
Architecture and Design

This section describes the application architecture.

Summary

The following Tiers/Layers model shows a high-level view of the webapp's architecture. **NOTE**: detailed diagrams are required in later sections of this document.

> ***[Sprint 1]*** *(Augment this diagram with your **own** rendition and representations of sample system classes, placing them into the appropriate M/V/VM (orange rectangle) tier section. Focus on what is currently required to support **Sprint 1 - Demo requirements**. Make sure to describe your design choices in the corresponding **Tier Section** and also in the **OO Design Principles** section below.)*



The web application, is built using the Model–View–ViewModel (MVVM) architecture pattern.

The Model stores the application data objects including any functionality to provide persistence.

The View is the client-side SPA built with Angular utilizing HTML, CSS and TypeScript. The ViewModel provides RESTful APIs to the client (View) as well as any logic required to manipulate the data objects from the Model.

Both the ViewModel and Model are built using Java and Spring Framework. Details of the components within these tiers are supplied below.

Overview of User Interface

This section describes the web interface flow; this is how the user views and interacts with the web application.

> *Provide a summary of the application's user interface. Describe, from the user's perspective, the flow of the pages in the web application.*

View Tier

> ***[Sprint 4]*** *Provide a summary of the View Tier UI of your architecture. Describe the types of components in the tier and describe their responsibilities. This should be a narrative description, i.e. it has a flow or "story line" that the reader can follow.*

> ***[Sprint 4]*** *You must provide at least **2 sequence diagrams** as is relevant to a particular aspects of the design that you are describing. (**For example**, in a shopping experience application you might create a sequence diagram of a customer searching for an item and adding to their cart.) As these can span multiple tiers, be sure to include an relevant HTTP requests from the client-side to the server-side to help illustrate the end-to-end flow.*

> ***[Sprint 4]*** *To adequately show your system, you will need to present the* ***class diagrams*** *where relevant in your design. Some additional tips:*
>
> - *Class diagrams only apply to the* ***ViewModel*** *and* ***Model*** *Tier*
> - *A single class diagram of the entire system will not be effective. You may start with one, but will be need to break it down into smaller sections to account for requirements of each of the Tier static models below.*
> - *Correct labeling of relationships with proper notation for the relationship type, multiplicities, and navigation information will be important.*
> - *Include other details such as attributes and method signatures that you think are needed to support the level of detail in your discussion.*

ViewModel Tier

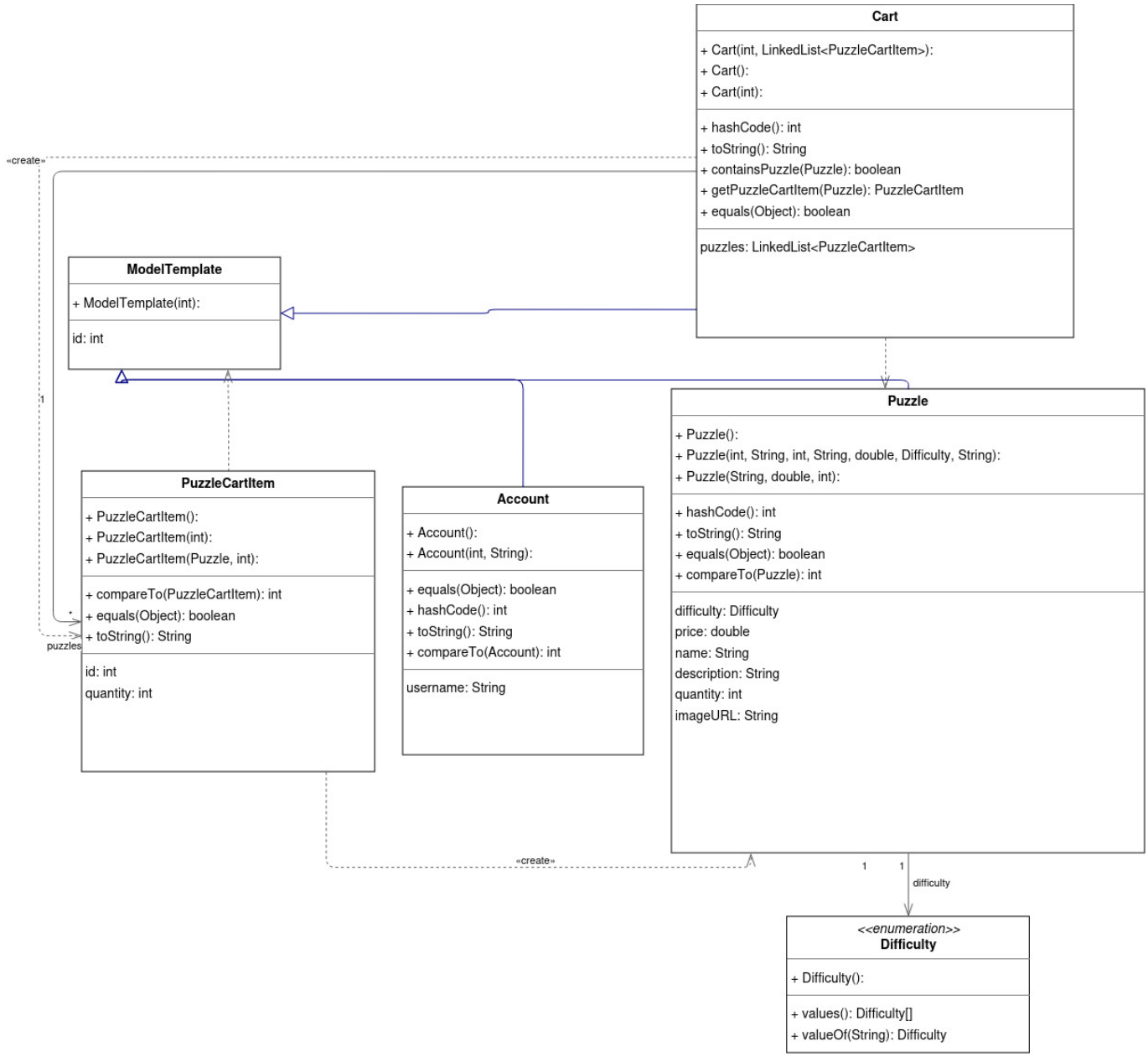
> - AccountController
> - PuzzleController
> - CartController
>
> Each controller provides the url mappings that make the client/server communiation possible. For example, the client makes a request to an /account url, the controller interprets it and delegates the task to the persistence layer.

> ***[Sprint 4]*** *Provide a summary of this tier of your architecture. This section will follow the same instructions that are given for the View Tier above.*

> *At appropriate places as part of this narrative provide* ***one*** *or more updated and* ***properly labeled*** *static models (UML class diagrams) with some details such as critical attributes and methods.*

 Replace with your ViewModel Tier class diagram 1, etc.

Model Tier

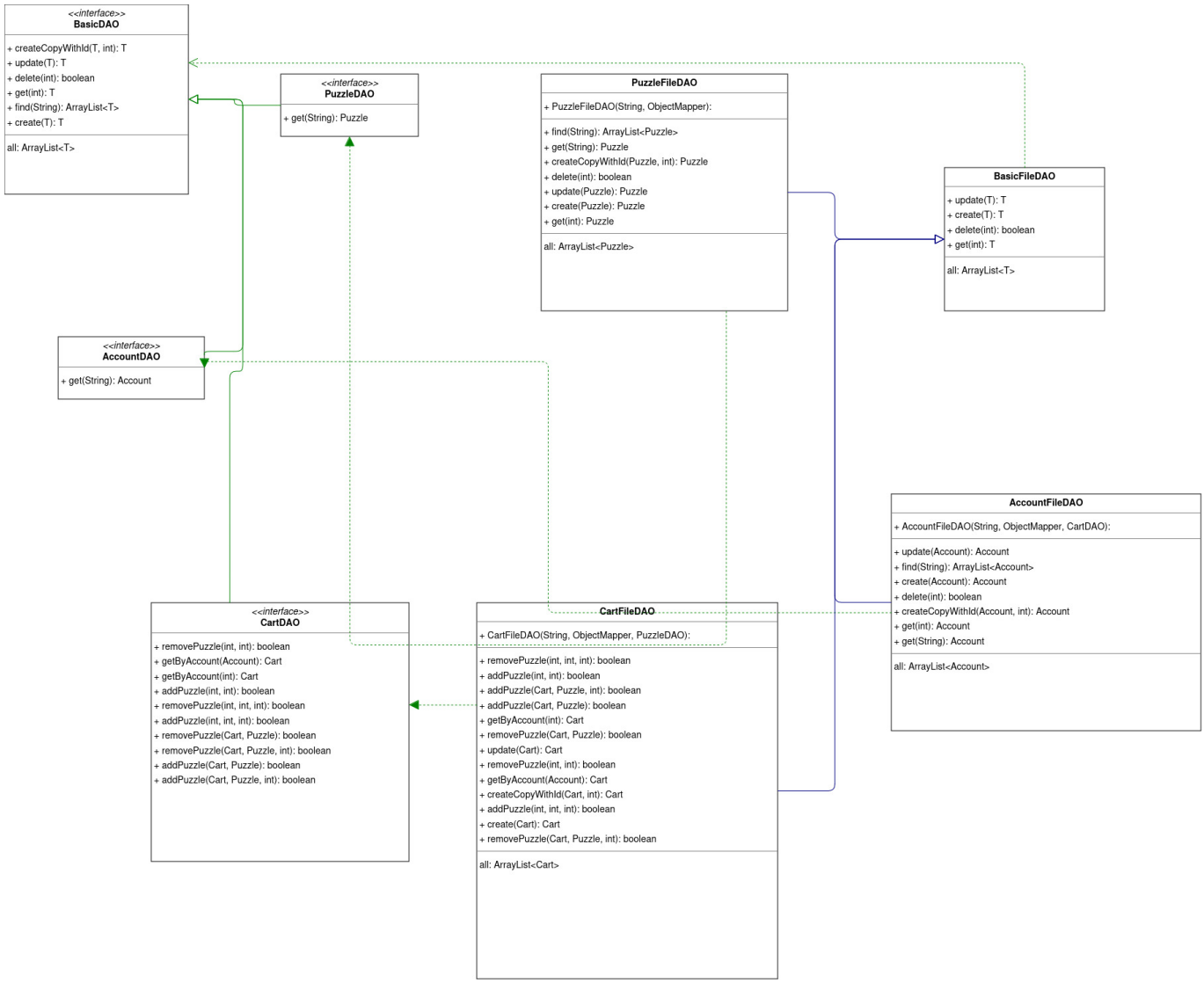


Every class in the model has one common thing, the id. That is why every class that was used to model inherits from it. The classes that inherit from ModelTemplate are:

> - Puzzle
> - Account
> - Cart

`PuzzleCartItem` is used inside the cart, to store the product together with the quantity in the cart.

Persistence



Now we can explain why ModelTemplate is needed. It was needed to create a `BasicDao`, an interface that supports all the basic CRUD operations. Basic Dao has a generic type, but the only condition is that the type must extend `ModelTemplate`. `BasicFileDao` implements it and provides all the basic operations for file based persistence, utilizing the FileStorage class. The following interfaces extend `BasicDao<T extends ModelTemplate>`:

> - `PuzzleDAO<Puzzle>`
> - `AccountDAO<Account>`
> - `CartDao<Cart>`

The following classes extend `BasicFileDao<T extends ModelTemplate` and implement the respective interfaces:

> - `PuzzleFileDAO`
> - `AccountFileDAO`
> - `CartFileDAO`

OO Design Principles

1. Single Responsibility Principle (SRP):

- Classes have well-defined roles and responsibilities.

- Examples:
    - Puzzle class holds puzzle data.
    - Difficulty enum represents difficulty levels.
    - PuzzleDao interface defines data access methods.
    - PuzzleFileDao implements PuzzleDao for file-based persistence.
    - PuzzleController handles communication between client and persistence layer.

2. Dependency Inversion Principle (DIP):

- Constructor injection is used in PuzzleController to favor interfaces over concrete implementations.
- This promotes loose coupling and easier testing.

Areas for Improvement

1. Open/Closed Principle (OCP):

- While enums and the controller structure partially adhere to OCP, there's room for improvement.
- Suggestions:
    - Implement interfaces or abstract classes for core functionalities (like PuzzleDao) to enable future data source or logic changes without modifying existing code.
    - Utilize the Strategy pattern for behaviors likely to change (e.g., puzzle validation) to isolate them from the core logic.

2. Low Coupling:

- Spring Framework's dependency injection helps with low coupling in PuzzleController.
- However, there's potential for further improvement.
- Suggestion:
    - Introduce a service layer between controllers and DAOs to encapsulate business logic and further decouple web handling from data access. interfaces for these services would allow for flexible implementations.

> ***[Sprint 2, 3 & 4]*** *Will eventually address upto* ***4 key OO Principles*** *in your final design. Follow guidance in augmenting those completed in previous Sprints as indicated to you by instructor. Be sure to include any diagrams (or clearly refer to ones elsewhere in your Tier sections above) to support your claims.*

> ***[Sprint 3 & 4]*** *OO Design Principles should span across* ***all tiers.***

Static Code Analysis/Future Design Improvements

> ***[Sprint 4]*** *With the results from the Static Code Analysis exercise,* ***Identify 3-4*** *areas within your code that have been flagged by the Static Code Analysis Tool (SonarQube) and provide your analysis and recommendations.*
> *Include any relevant screenshot(s) with each area.*

> ***[Sprint 4]*** *Discuss* ***future*** *refactoring and other design improvements your team would explore if the team had additional time.*

Testing

We have written 87 JUnit tests(backend only) and all of them pass. Tests span across all tiers.

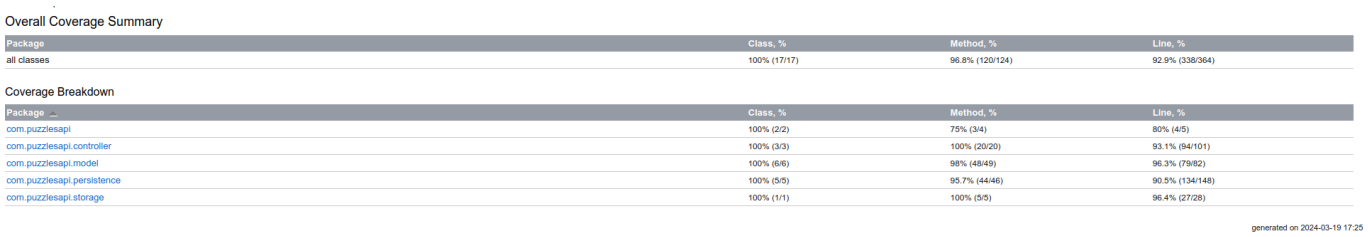
Acceptance Testing

> All 8 user stories passed acceptance testing.

Unit Testing and Code Coverage

> ***[Sprint 4]*** *Discuss your unit testing strategy. Report on the code coverage achieved from unit testing of the code base. Discuss the team's coverage targets, why you selected those values, and how well your code coverage met your targets.*

Coverage Report



Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	100% (17/17)	96.8% (120/124)	92.9% (338/364)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.puzzlesapi	100% (2/2)	75% (3/4)	80% (4/5)
com.puzzlesapi.controller	100% (3/3)	100% (20/20)	93.1% (94/101)
com.puzzlesapi.model	100% (6/6)	98% (48/49)	96.3% (79/82)
com.puzzlesapi.persistence	100% (5/5)	95.7% (44/46)	90.5% (134/148)
com.puzzlesapi.storage	100% (1/1)	100% (5/5)	96.4% (27/28)

generated on 2024-03-19 17:25

Ongoing Rationale

> ***[Sprint 1, 2, 3 & 4]*** *Throughout the project, provide a time stamp* ***(yyyy/mm/dd): Sprint # and description*** *of any* ***mayor*** *team decisions or design milestones/changes and corresponding justification.*