2018

Software Design Document

VERSION 2.0

X.Z.

Contents

Table of Figures

1 Introduction

It is a players' data management system, which is a web application based on PHP. The features include displaying players' information and write a player's information to the database. The system is composed of three primary components: the View component that provides display methods, the Controller component that responds to the requests from the pages, and the Model component which manages databases of players information.

2 Run the Application

Run index.php. There are three types of databases. Switch between them by modifying the first line of Config.json in the Model folder. There are three options: array, json, and file.

3 Design Overview

The system is in a layered architecture fitted with MVC (Model–View–Controller) pattern. There are three layers. The top layer is the View, which provides display methods. The middle layer is the Controller, which responds to the information requests from the View. It can be extended with other services, which provides other information for the View. The bottom layer is the Model, which reads and writes data through databases. In the layered architecture, a lower layer offers services to its upper layer so that the change in the upper layers does not affect the lower layers. A Façade pattern is used in the Model layer. The interface of the Model layer lists the functions that the layer provides, which reduces dependencies of the upper layers on the inner workings of the Model layer. Therefore, the change in the bottom layer will cause only minimal changes in the upper layers. Figure 1 below is the class diagram of the system.

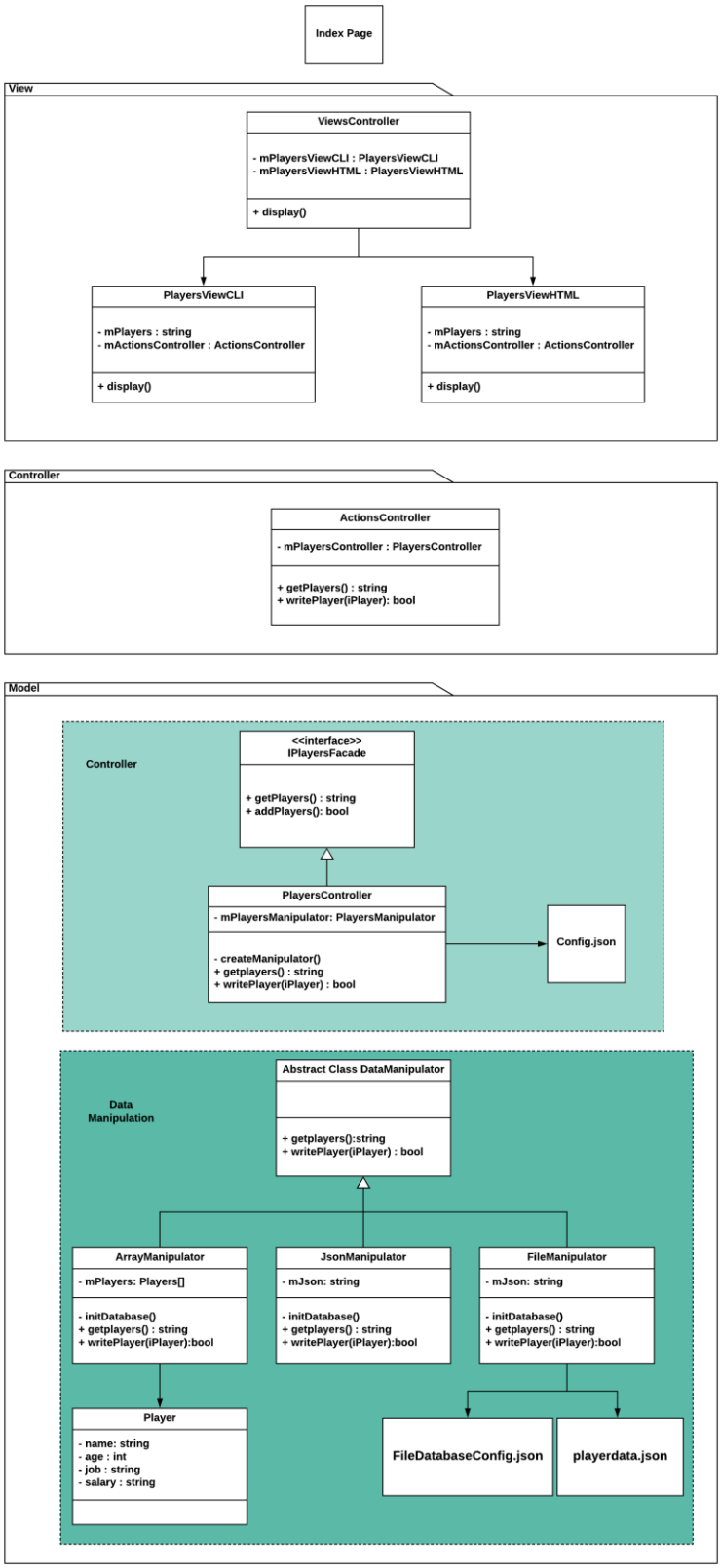

Figure 1. System Overview

4 The Design of Layers

4.1 View

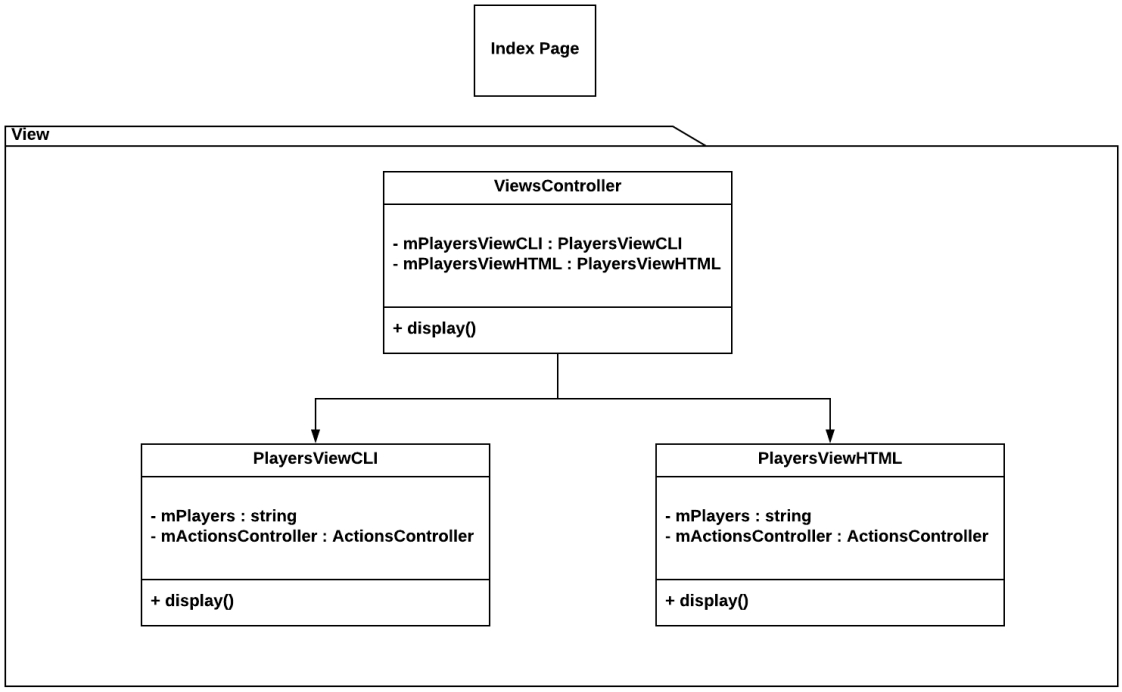

Figure 2. Class Diagram of View

Index.php is loaded first. It creates a *ViewsController*, which decides which display method should be used. In index.php, the display() function of the *ViewsController* is called. The function creates either a view in strings or a view in HTML based on the type of interface between web server and PHP. If the interface is CLI, the *ViewController* creates a *PlayersViewCLI,* which outputs strings. In the rest of the cases, the *ViewController* creates a *PlayersViewHTML*, which displays players information in HTML. After creating a view, the *ViewContorller* calls the display() function of the corresponding view.

PlayersViewHTML and *PlayersViewCLI* both keep the players' data as a member variable. Once they get the players' data, they do not need to call the getPlayers() function again. The member variable is a string in JSON format. If we would divide the system to a client (contains View and Controller) and a server (contains Model) in the future, the data is transmitted in the form of string. In consideration of this, the players' data is passed in string format through the layers.

PlayersViewHTML and *PlayersViewCLI* both create an *ActionsController*, which responds to the requests from the views.

4.2 Controller

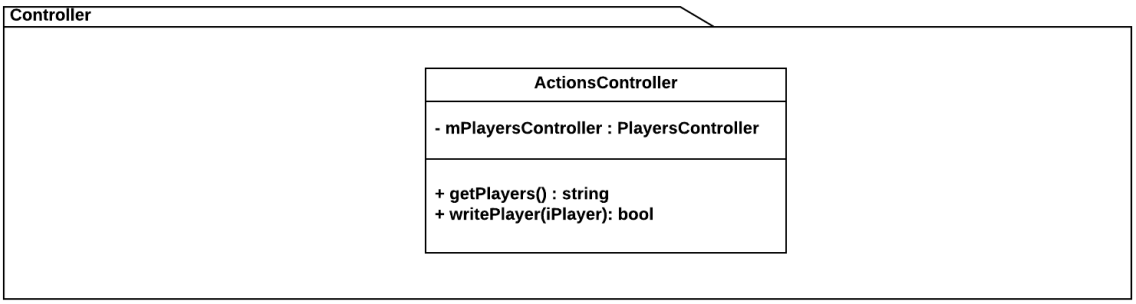

Figure 3. Class Diagram of Controller

ActionsController provides getPlayers() function, which return a string of Players' data read by Model. Besides, its writePlayer() function calls functions in Model to write a player's information to a database. *ActionsController* creates a *PlayersController*, which implements the Façade interface of Model.

4.3 Model

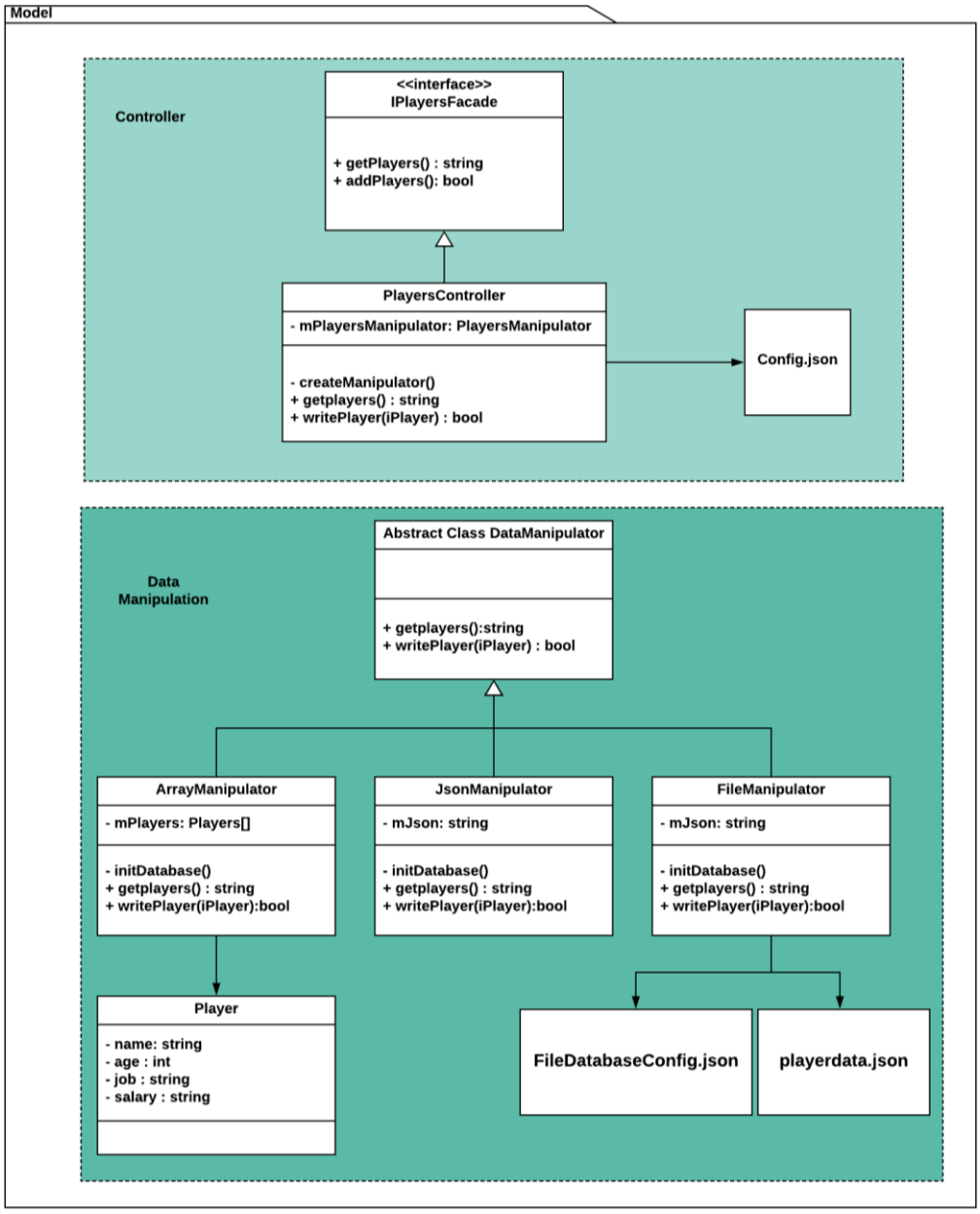

Figure 4. Class Diagram of Model

There are two layers in Model layer: the Façade interface and its implementation, and the data manipulation layer.

The data manipulation layer has three types of databases: an array of player objects, a JSON format string, and a JSON file. Because they need to provide some functions requested by the upper layers, they have an abstract parent class (*DataManipulator*) that defines the functions.

An *ArrayManipulator* inherits *DataManipulator, which* keeps an array of *Player* objects. A *Player* contains name, age, job, and salary information.

A *JsonManipulator* that inherits *DataManipulator* maintains a string of players' information in JSON format.

Another subclass of *DataManipulator* – *FileManipulator,* manages a JSON file as a database. It reads a configuration file (FileDatabaseConfig.json) to get the file path of the JSON file (playerdata.json). The file paths are not hardcoded, so it's easier to modify the file name and path of the JSON file.

The Façade interface (*IPlayersFacade*) of Model defines two methods: getPlayers() and writePlayer(). It's implemented by *PlayersController*. When the aforementioned methods are called, *PlayersController* creates a *DataManipulator* if is not created yet, and keeps it as a member variable. It chooses a type of *DataManipulator* (*ArrayManipulator, JsonManipulator or FileManipulator*) based on the config.json file.

5 How to Extend the System

5.1 Client-Server

Add data transmission components between the Controller layer and the Model layer. The Controller layer sends the data requests to the Model layer. The Model layer gets the requests, interprets them and calls the corresponding functions to read and write information.

5.2 More Databases

To add a new database, add a new class that inherits *FileManipulator,* and implements all methods in *FileManipulator.* Modify the Config.json in the Model folder if this database is in use.

5.3 More Display Methods

Add a new display component in the View folder. Modify the display() function in *ViewsController* so the new component is called.