Music IMDB
Analysis and Design Document
Student: Gyarmathy Tímea
Group: 30433

Revision History

Date	Version	Description	Author
31/03/2018	1.0	First iteration	Gyarmathy Tímea

Table of Contents

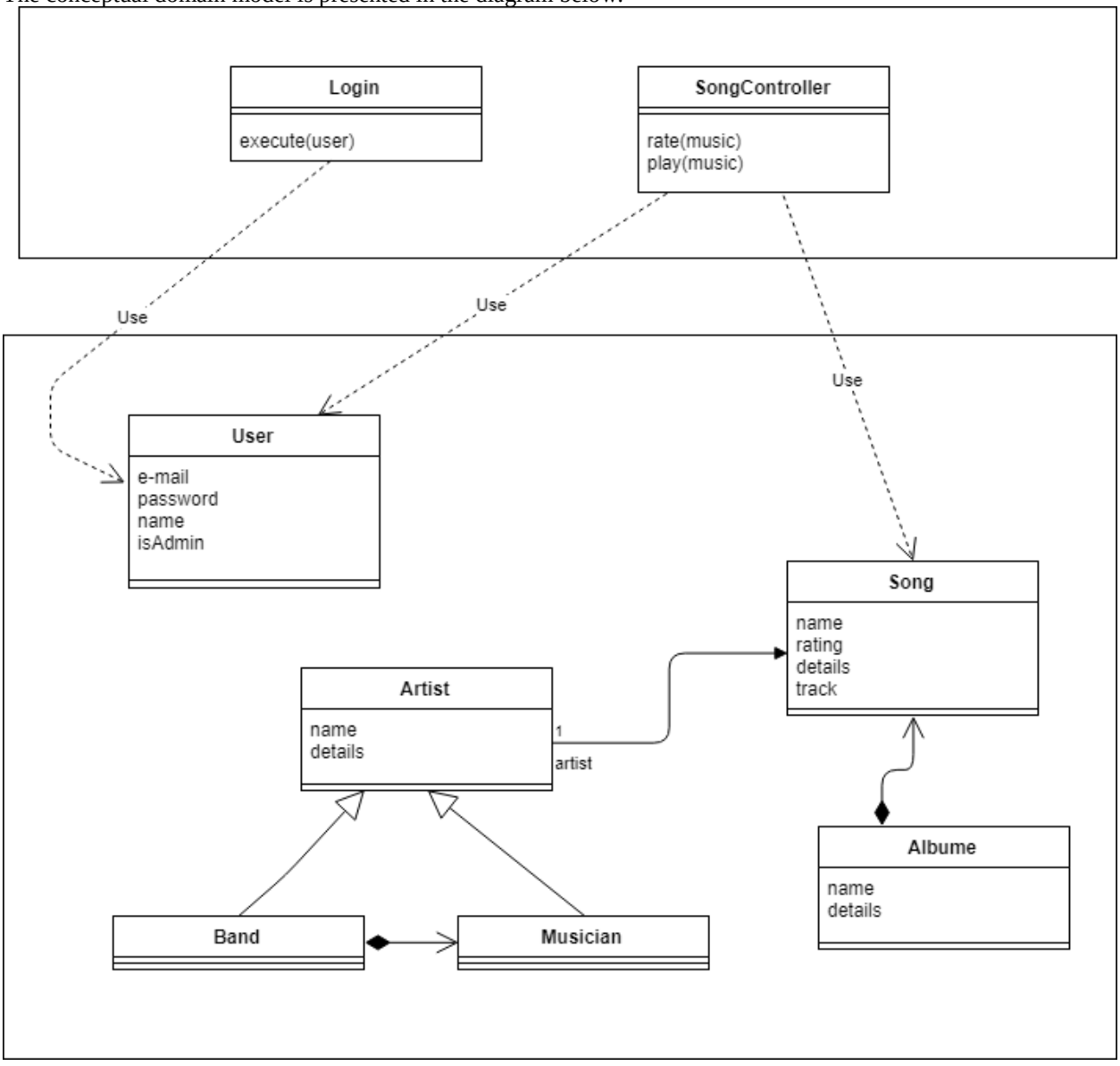
I. Project Specification

A “Music IMDB” should be developed which is a project developed in a 3-tier architecture, based on an officially built database of released songs and registered musical artists. The project should provide an easy to use interface to the user and safe accessibility to the database for the administrators. To provide accessibility, the project should be developed in ASP.NET web framework to build a web application which can be accessed from different platforms.

II. Elaboration – Iteration 1.1

1. Domain Model

The conceptual domain model is presented in the diagram below:



Here we can already observe the layered architecture based on which the app will be developed.

The model will contain elements as Song, Album, Artist, directly representing the data stored in the underlying database. An Artist can be either a single Musician, either a group of musicians, i.e. a Band.
The operations from the User will be processed through a Business Layer, which uses this domain model. This is illustrated by the Login and the SongController classes.

2. Architectural Design

2.1 Conceptual Architecture

The system Music IMDB will be an ASP.NET application, which is by default a Model-View-Controller framework. We will use it with a web page in the front-end, and with the layered architecture, more precisely the three-tier system, to be able to organize the data connection, the actual application logic and the user interface.

Model–view–controller (MVC) is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

Components

- The model is the central component of the pattern. It expresses the application's behavior in terms of the problem domain, independent of the user interface. It directly manages the data, logic and rules of the application.
- A view can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- The third part or section, the controller, accepts input and converts it to commands for the model or view.

The ASP.NET MVC framework offers the following advantages:

- It makes it easier to manage complexity by dividing an application into the model, the view, and the controller.
- It does not use view state or server-based forms. This makes the MVC framework ideal for developers who want full control over the behavior of an application.
- It uses a Front Controller pattern that processes Web application requests through a single controller. This enables you to design an application that supports a rich routing infrastructure. For more information, see Front Controller.
- It provides better support for test-driven development (TDD).
- It works well for Web applications that are supported by large teams of developers and for Web designers who need a high degree of control over the application behavior.

In order to access the underlying database, we will structure this MVC architecture based on another, similar one, which adds another layer to our architecture, the data access which will communicate with the database.
The three-tier architecture is a client–server software architecture pattern in which the user interface (presentation), functional process logic ("business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms.

Web development usage

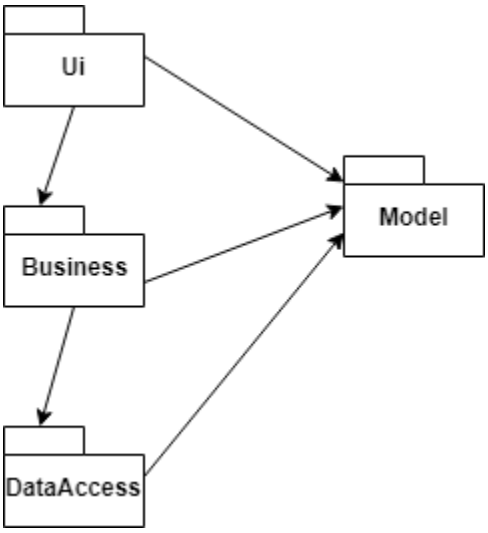
In the web development field, three-tier is often used to refer to websites, commonly electronic commerce websites, which are built using three tiers:

- A front-end web server serving static content, and potentially some cached dynamic content. In web-based application, front end is the content rendered by the browser. The content may be static or generated dynamically.
- A middle dynamic content processing and generation level application server (e.g., Symfony, Spring, **ASP.NET**, Django, Rails).

- A back-end database or data store, comprising both data sets and the database management system software that manages and provides access to the data.

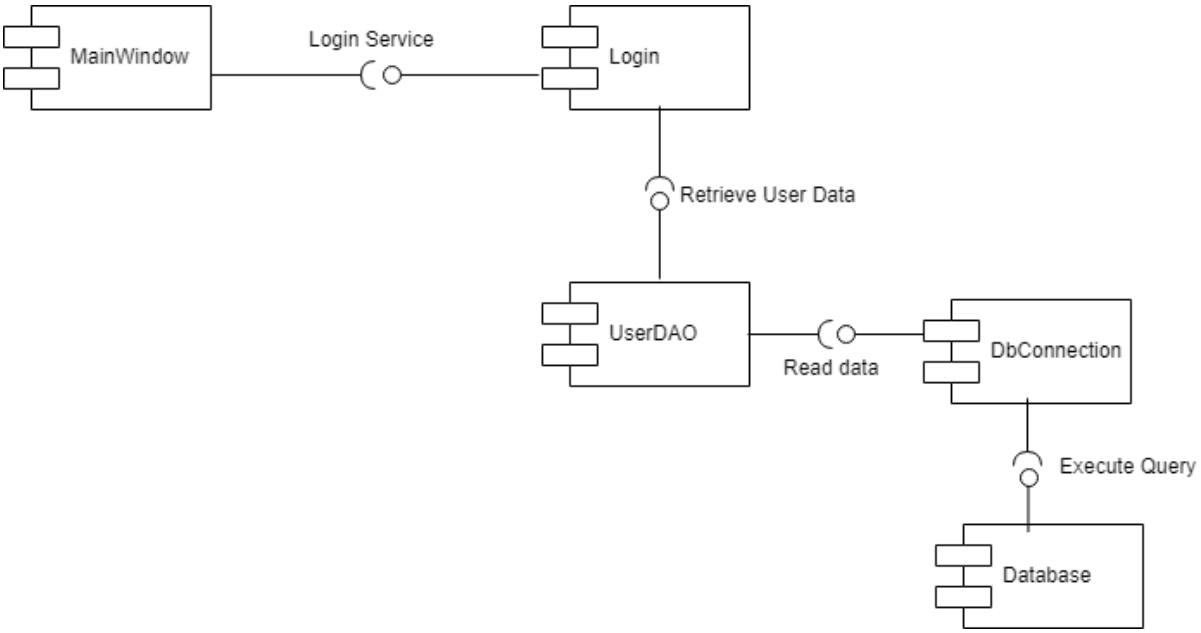
2.2 Package Design

In the case of this system, the packages would correspond to the ones defined in the architecture of the system: model, view, controller, but in our case, view becomes Ui, which contains all user interface related components, and controller becomes Business. The layered architecture adds another package, the DataAccess. The Model is seen from all these.

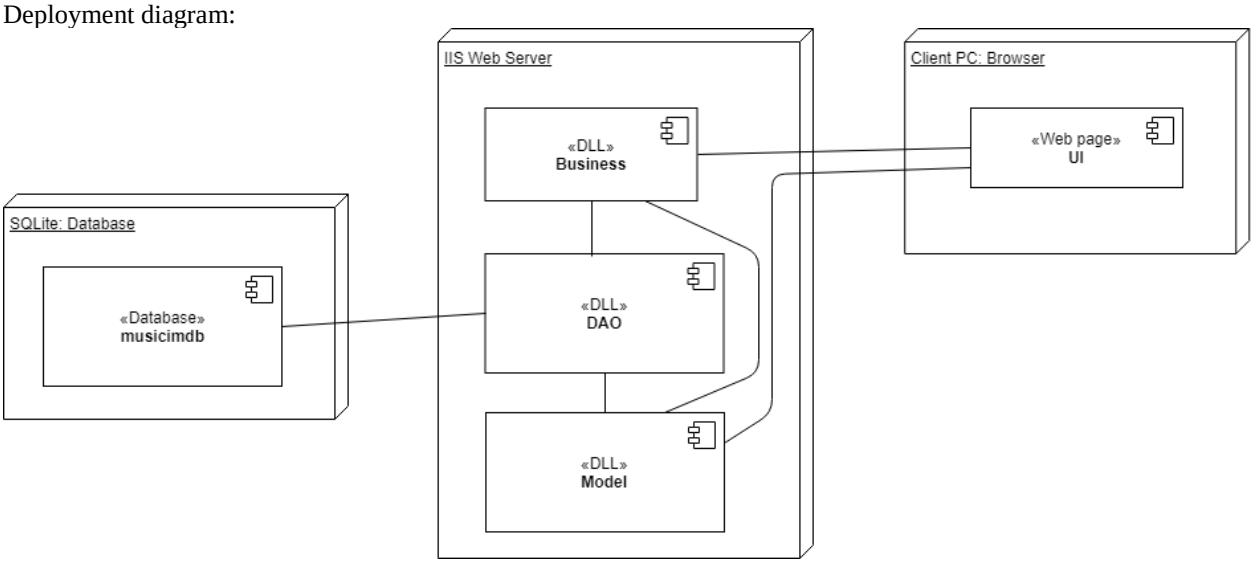


2.3 Component and Deployment Diagrams

As the component diagram would be too elaborate, we can show the main idea of it through the components related to the user login:



Deployment diagram:



The IIS Web Server is an extensible web server created by Microsoft for use with the Windows NT family. IIS supports HTTP, HTTP/2, HTTPS, FTP, FTPS, SMTP and NNTP. The Music IMDB project will have an interface based on Cshtml which is supported by this server.

III. Elaboration – Iteration 1.2

1. Design Model

1.1 Dynamic Behavior

[Create the interaction diagrams (1 sequence, 1 communication diagrams) for 2 relevant scenarios]

1.2 Class Design

[Create the UML class diagram; apply GoF patterns and motivate your choice]

2. Data Model

[Create the data model for the system.]

3. Unit Testing

[Present the used testing methods and the associated test case scenarios.]

IV. Elaboration – Iteration 2

1. Architectural Design Refinement

[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]

2. Design Model Refinement

[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]

V. Construction and Transition

1. System Testing

[Describe how you applied integration testing and present the associated test case scenarios.]

2. Future improvements

[Present future improvements for the system]

VI. Bibliography