Object Oriented Analysis and Design  
(INT3110 21)

*Course Project*

*Netflix.com Website*

**Instructor:** Assoc. Prof. Dr. Trương Ninh Thuận

**Project team:**

|  |  |  |
| --- | --- | --- |
| Trần Quang Vinh | K62-CACLC3 | Student No. 17021357 |
| Phạm Thái Sơn | Student No. 17021330 |

Table of Contents

[1 Requirements 4](#_Toc23883454)

[1.1 Problem statement 4](#_Toc23883455)

[1.1.1 Addressing the problem 4](#_Toc23883456)

[1.1.2 Solution 4](#_Toc23883457)

[1.2 Glossary 5](#_Toc23883458)

[1.3 Supplementary specifications 6](#_Toc23883459)

[1.4 Use-case models 8](#_Toc23883460)

[1.4.1 Create account 9](#_Toc23883461)

[1.4.2 Log in 10](#_Toc23883462)

[1.4.3 Upload movie 11](#_Toc23883463)

[1.4.4 Delete movie 12](#_Toc23883464)

[1.4.5 Edit movie’s information 14](#_Toc23883465)

[1.4.6 Watch movies 15](#_Toc23883466)

[1.4.7 Update account’s information 16](#_Toc23883467)

[1.4.8 Buy subscription 16](#_Toc23883468)

[1.4.9 Cancel subscription 17](#_Toc23883469)

[1.4.10 Add movie to watch list 18](#_Toc23883470)

[1.4.11 Remove movie from watch list 19](#_Toc23883471)

[2 Use case analysis 21](#_Toc23883472)

[2.1 Architectural analysis 21](#_Toc23883473)

[2.1.1 High-level organization of the model 21](#_Toc23883474)

[2.1.2 Key abstractions 21](#_Toc23883475)

[2.2 Use-case realizations 22](#_Toc23883476)

[2.2.1 Use-case realizations: Sequence diagrams 22](#_Toc23883477)

[2.2.2 Use-case realizations: Views of participating classes 32](#_Toc23883478)

[2.2.3 Describe analysis mechanism 37](#_Toc23883479)

[3 Use-case design 39](#_Toc23883480)

[3.1 Architectural refinement 39](#_Toc23883481)

[3.1.1 Identify design elements 39](#_Toc23883482)

[3.1.2 Identify design mechanisms 43](#_Toc23883483)

[3.2 Describe the run-time architecture 43](#_Toc23883484)

[3.3 Describe distribution 44](#_Toc23883485)

[3.4 Use-case design 45](#_Toc23883486)

[3.4.1 Design sequence diagrams 45](#_Toc23883487)

[3.4.2 Design views of participating classes 54](#_Toc23883488)

[3.5 Subsystem design 54](#_Toc23883489)

[3.6 Class design 54](#_Toc23883490)

[3.7 Database design 60](#_Toc23883491)

# Requirements

## Problem statement

### Addressing the problem

As quality of life standard increases, therefore, the need for entertainment increases. At the same time, people are more occupied than ever before. People wants a less time-consuming way to watch movies without ever standing in a queue or having to go to a store to purchase them. This evokes the need for a system for a service to help accommodate such needs. Users can watch a movie they want without the need of going out to a physical store or waiting at a queue in a movie theater. Movies distributors can distribute their movies on the system so that users can watch them. This has come to a demand for a system to solve this problem.

### Solution

Netflix.com is built as an online entertainment platform so that users can watch the movie directly on the website. Netflix is a streaming service that offers a wide variety of award-winning movies on thousands of internet-connected devices. You can watch as much as you want, whenever you want without a single commercial – all for one low monthly price. There's always something new to discover and new movies are added every week!

**Description**

The system will be developed as a web application. End users will interact with the system over the Internet via a wide range of devices (smartphones, PCs, smart TV…).

People can register for an account then log into the system and buy a monthly subscription to start watching movies.

Users can search for movies they are interested in and watch it. They can also view the information of a movie such as synopsis, trailers, etc. Moreover, they can add a movie to their watch list to watch later.

Content providers can upload movies, edit their information directly on the system.

**End users**

The system’s end users are as follow:

* Visitor

Visitors are people that don’t have an account on the system. They can register an account to become users.

* Users

Users are visitors registered to become registered users. They can buy subscription to watch movies. Users can view information about the movies and add them to their watch list.

* Content providers

Content providers are responsible for managing their movies on the system. They can upload movies, edit their information. They can also delete movies from a system if the contract expires.

## Glossary

**Introduction**

This document is used to define terminology specific to the problem domain, explaining terms, which maybe unfamiliar to the reader of the use-case descriptions or other project documents. Often, this document can be used as an informal data dictionary, capturing data definitions so that use-case descriptions and other project documents can focus on what the system must do with the information.

**Definitions**

The glossary contains the working definitions for the key concepts in the Netflix.com website.

**Account**

A record about a user/administrator containing information about his/her username, e-mail, password, subscription and payment information. Each account has a unique user ID, which are used to identify the user/administrator and grant them access to specific parts of the system.

**Content provider**

An organization whose job is to manage its movies on the system. They are responsible for uploading movies, adding or editing movies’ information and delete the movies from the system.

**Movie**

A movie made by movie studios and published by a distributor.

**Subscription**

An amount of money that users pay regularly to be able to watch movies on Netflix.com website.

**User**

Any person who has a registered account on the website but is not an administrator. Users can buy or cancel subscriptions, watch movies and view their information.

**Visitor**

Any person who visits the website without having an account. Visitors can create account to become users.

## Supplementary specifications

**Objectives**

The purpose of this document is to define requirements of the Netflix.com system. This Supplementary Specification lists the requirements that are not rapidly captured in the use case of the use-case model. The Supplementary Specification and the use-case model together capture a complete set of requirements on the system.

**Scope**

This Supplementary Specification applies to the Netflix.com system, which is a streaming website in the United States.

This specification defines the non-functional requirements of the system: such as reliability, usability, performance and supportability, as well as functional requirements that are common across a number of use cases. (The functional requirements are defined in the Use Case Specification.)

**References**

None.

**Functionality**

Multiple users must be able to perform their work concurrently.

**Usability**

The software must be easy to use so that a new user can learn how to use the system within 30 minutes.

The user interface has to be friendly and intuitive.

**Reliability**

The system must be available 24 hours a day, 7 days a week. The system must also have less than 5% downtime.

**Performance**

The system shall handle up to 1 petabits per second of simultaneous peak traffic.

The system shall provide access to the database with no more than 5 seconds latency.

The system must be able to complete at least 90% of all transactions within 15 seconds.

**Supportability**

None.

**Security**

The system must prevent users from logging in if they do not provide a correct password.

A movie can only be edited and deleted by the content provider that uploaded it.

Only content providers can upload movies to the server.

**Design Constraints**

The system must provide a responsive web-based interface that is usable on multiple devices, such as computer, smartphone and smart TV.

## Use-case models

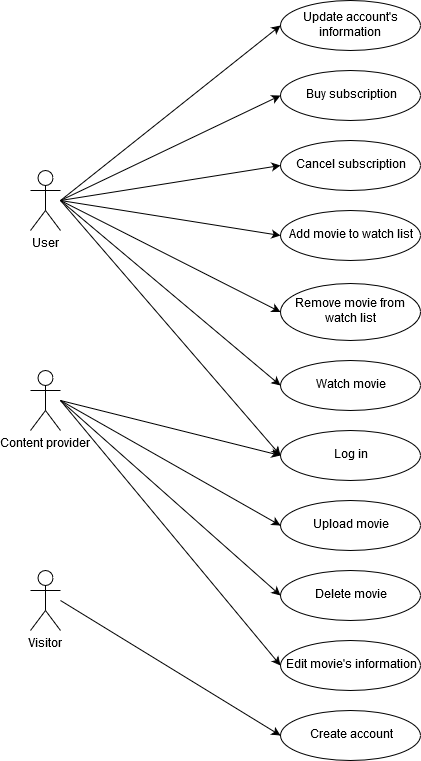


Figure 1‑1. Use-case model for the Netflix.com website system

### Create account

**Brief Description**

This use case describes how a visitor registers for an account on the Netflix.com website.

**Flow of Events**

*Basic Flow*

This use case starts when the visitor requests to create an account on the website.

1. The system displays a form and prompts visitor to enter the following information:

* Email address (required)
* Password (required)
* Email his/her Netflix special offers (optional)

1. Visitor provides the information then press continue.
2. The system check if email address already exists in database and all required fields are filled. It then adds a new account with the specified information to the user database.
3. The system lists available payment method then asks visitor to choose his/her preferred payment methods.
4. Visitor choose a payment method, enter his/her name (required) and billing information (required), then select “Start membership”.
5. The system verifies billing information, then save the specified information to the previously added account.

*Alternative Flows*

**Missing Information**

If any of required fields are not filled in, the system will automatically display an error. The visitor can make changes to continue the registration or cancel it, which end the use case.

**Email address already exists**

If visitor enters an email address already existed in database, the system displays an error message and asks the user to choose another one. The visitor can choose another email address to continue the registration or cancel it, which end the use case.

**Failed to verify billing information**

If the payment processor cannot verify the billing information, the system displays an error message and prompts the user to choose another billing method or billing information. The visitor can choose another billing method or billing information to continue the registration or cancel it, which end the use case.

**Special requirements**

None.

**Pre-Conditions**

None.

**Post-Conditions**

If the operation is successful, a new user is added to the system. Otherwise, the system state remains unchanged.

**Extension Points**

None.

### Log in

**Brief Description**

This use case describes how a user or a content provider logs into the Netflix.com website.

**Flow of Events**

*Basic Flow*

This use case starts when the actor requests to log into the website.

1. The system displays a page that prompts actor for email address and password.
2. Actor enters email address and password and, optionally, tick the “Remember me” checkbox then press “Sign in” button.
3. The system validates the entered information then logs the actor into the system.

*Alternative Flows*

**Missing Information**

If any of required fields are not filled in, the system will automatically display an error. The actor can make changes to continue to log in or cancel operation, which end the use case.

**Email address doesn’t exist in database**

If actor enters an email address doesn’t exist in database, the system displays an error message and asks the actor if he/she want to register a new account. The actor can choose another email address to continue to log in, create a new account or cancel the operation, which end the use case.

**Wrong password**

If the actor entered wrong password, the system displays an error message and asks the actor if he/she want to recover password. The actor can choose to recover password, retry another password or cancel the operation, which end the use case.

**Special requirements**

None.

**Pre-Conditions**

The system is in the login state and has the login screen displayed.

**Post-Conditions**

If the operation is successful, the actor is now logged into the system. Otherwise, the system state remains unchanged.

**Extension Points**

None.

### Upload movie

**Brief Description**

This use case describes how a content provider uploads a movie into the Netflix.com website.

**Flow of Events**

*Basic Flow*

This use case starts when the content provider requests to upload a movie to the system.

1. The system displays a page with a drag-and-drop box to upload the movie and prompt for movie’s information (movie’s name, genre, released year, actor, …).
2. Content provider choose the movie he/she wanted to upload and press “Upload” button and at the same time, fill movie’s information.
3. When the upload process is completed, the system validates the information then asks the provider to confirm the upload.
4. Content provider confirms the upload.
5. The system creates a new record in the database containing information about that movie and its location on the server for later use.

*Alternative Flows*

**Missing Information**

If any of required fields are not filled in, the system will automatically display an error. The content provider can make changes to continue or cancel operation, which end the use case.

**Content provider rejects the uploaded movie**

If the actor rejects the movie that he/she uploaded, the system deletes the movie from the system then goes back to step 1. Actor can choose to continue the upload or cancel the operation, which end the use case.

**Special requirements**

None.

**Pre-Conditions**

The content provider must be logged to the system.

**Post-Conditions**

If the operation is successful, the movie will be available for the users. Otherwise, the system state remains unchanged.

**Extension Points**

None.

### Delete movie

**Brief Description**

This use case describes how a content provider deletes a movie on the Netflix.com website.

**Flow of Events**

*Basic Flow*

This use case starts when the content provider requests to delete a movie on the system.

1. The system displays a list of movies uploaded by the content provider.
2. Content provider choose the movie he/she wished to delete and press “Delete” button.
3. The system displays a message asks if he/she really wants to delete the movie.
4. The content provider chooses “Yes” to confirm the delete.
5. The system deletes the movie’s record from database and movie file from the server then displays a message notify the user that the operation is complete.

*Alternative Flows*

**Content provider denied when asked to confirm the delete**

If the actor rejects the delete, the system goes back to step 2. Actor can choose to continue the process or cancel the operation, which end the use case.

**List of movies uploaded by the content provider is empty (Content provider hasn’t uploaded any movies to the system)**

If the list of movies uploaded by the content provider is empty, the “Delete” button is disabled and the use case is ended.

**Special requirements**

None.

**Pre-Conditions**

The content provider must be logged to the system.

**Post-Conditions**

If the operation is successful, the movie will no longer be available for the users. Otherwise, the system state remains unchanged.

**Extension Points**

None.

### Edit movie’s information

**Brief Description**

This use case describes how a content provider edit information of a movie on the Netflix.com website.

**Flow of Events**

*Basic Flow*

This use case starts when the content provider requests to edit information of a movie on the system.

1. The system displays a list of movies uploaded by the content provider.
2. Content provider choose the movie he/she wished to edit information and press “Edit” button.
3. The system displays all information fields with current information filled each field.
4. Actor edits information as needed then click “Save”.
5. The system displays a message asks the actor to confirm the changes.
6. The content provider chooses “Yes” to confirm the changes.
7. The system edits the movie’s record on database using the new information.

*Alternative Flows*

**Content provider denied when asked to confirm the changes**

If the actor rejects the changes, the system goes back to step 2. Actor can choose to continue the process or cancel the operation, which end the use case.

**List of movies uploaded by the content provider is empty (Content provider hasn’t uploaded any movies to the system)**

If the list of movies uploaded by the content provider is empty, the “Edit” button is disabled and the use case is ended.

**Special requirements**

None.

**Pre-Conditions**

The content provider must be logged to the system.

**Post-Conditions**

If the operation is successful, the movie’s new information will be available for the users. Otherwise, the system state remains unchanged.

**Extension Points**

None.

### Watch movies

**Brief Description**

This use case describes how a user watch a movie on the Netflix.com website.

**Flow of Events**

This use case starts when the user requests to watch a movie on the system.

1. User choose the movie he/she wished to watch.
2. The system starts streaming the movie to the user.

**Special requirements**

None.

**Pre-Conditions**

The user must be logged to the system and the account must have an active subscription.

**Post-Conditions**

If the operation is successful, the movie is streamed to user’s display.

**Extension Points**

None.

### Update account’s information

**Brief Description**

This use case describes how a user updates his/her account’s information on the Netflix.com website.

**Flow of Events**

*Basic Flow*

This use case starts when the user requests to edit account information on the system.

1. User clicks “Account” button.
2. The system displays account’s information.
3. User makes changes as needed. After that, user presses “Save” button to save all changes.
4. The system validates the changes then updates the changes to the database.

*Alternative Flows*

**Verification detects invalid information**

If the system detects any invalid changes, it will display an error message and notify the user about those invalid changes. User can continue to make changes or cancel the operation, which end the use case.

**Special requirements**

None.

**Pre-Conditions**

The user must be logged to the system.

**Post-Conditions**

If the operation is successful, the new account information is displayed to user.

**Extension Points**

None.

### Buy subscription

**Brief Description**

This use case describes how a user buy his/her subscription on the Netflix.com website.

**Flow of Events**

*Basic Flow*

This use case starts when the user requests to buy account subscription on the system.

1. The system displays account’s information and a “Restart membership” button.
2. The system displays a message asks user to choose subscription plans.
3. User chooses a subscription plan then press “Confirm”.
4. The system attempts to charge user based on their billing information and chosen subscription plan.
5. If successful, the system saves user’s new subscription information.

*Alternative Flows*

**The system failed to charge user’s billing method**

If the system cannot charge user’s billing method (wrong credit card information, card doesn’t have sufficient fund, ...), it will display an error message and ask the user to change billing information. User can continue to make changes or cancel the operation, which end the use case.

**Special requirements**

None.

**Pre-Conditions**

* The user must be logged to the system.
* User’s account must not have any active subscription.

**Post-Conditions**

If the operation is successful, the subscription status is changed to “Active” and expiry date is displayed to user.

**Extension Points**

None.

### Cancel subscription

**Brief Description**

This use case describes how a user cancel his/her subscription on the Netflix.com website.

**Flow of Events**

This use case starts when the user requests to cancel subscription plan on the system.

*Basic Flow*

1. The system displays account’s information and a “Cancel membership” button.
2. User press “Cancel membership” button.
3. The system displays a message asks user to confirm the action.
4. User press “Yes” button.
5. The system cancels user’s subscription.

*Alternative Flow*

If the user press “No” when asked to confirm, the use case is ended.

**Special requirements**

None.

**Pre-Conditions**

* The user must be logged to the system.
* User’s account must have an active subscription.

**Post-Conditions**

If the operation is successful, the subscription status is changed to “Inactive”. User will be able to watch movies until the old expiry date.

**Extension Points**

None.

### Add movie to watch list

**Brief Description**

This use case describes how a user add a movie or TV show to his/her watch list on Netflix.com website.

**Flow of Events**

This use case starts when the user requests to add a movie or TV show to his/her watch list for later on the system.

*Basic Flow*

1. User chooses a movie on the system.
2. The system displays movie’s page.
3. User presses “+ My list” button.
4. The system adds the movie to user’s watch list.

*Alternative Flow*

If the chosen is already added to user’s “My list”, the use case is ended.

**Special requirements**

None.

**Pre-Conditions**

The user must be logged to the system.

**Post-Conditions**

If the operation is successful, the plus (“+”) sign will change to the checkmark (“✓”) sign, indicating that the movie has been added to user’s watch list.

**Extension Points**

None.

### Remove movie from watch list

**Brief Description**

This use case describes how a user remove a movie or TV show from his/her watch list on Netflix.com website.

**Flow of Events**

This use case starts when the user requests to remove a movie or TV show from his/her watch list on the system.

Basic Flow

1. User chooses a movie on the system.
2. The system displays movie’s page.
3. User presses “✓ My list” button.
4. The system removes the movie from user’s watch list.

*Alternative Flow*

If the movie is not already added to user’s “My list”, the use case is ended.

**Special requirements**

None.

**Pre-Conditions**

* The user must be logged to the system.

**Post-Conditions**

If the operation is successful, the checkmark (“✓”) sign will change to the plus (“+”) sign, indicating that the movie has been removed from user’s watch list.

**Extension Points**

None.

# Use case analysis

## Architectural analysis

### High-level organization of the model

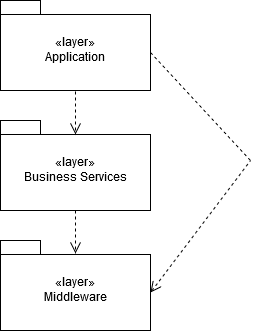


Figure 2‑1 Layering approach

The above figure describes the high-level organization of the software system. The system consists of three layers:

* The *Application* layer contains the design elements that are specific to each use case of the system.
* The *Business Services* layer encapsulates some key abstractions and services common to all use cases. It is accessible from the *Application* layer.
* The *Middleware* layer offers services to enable data communication and management on distributed systems.

### Key abstractions



Figure ‑ Key abstractions used in the application

* *Account*: A record about a user/content provider. Each account has a unique email address, a password, which is used to identify the user/content provider and grant them access to secure parts of the system. Also, user’s account may contain billing information.
* *Movie*: A video content that is distributed by the content provider and is available for all users in the system.

## Use-case realizations

### Use-case realizations: Sequence diagrams

Figure ‑ Sequence diagram for the Remove movie from watch list use case

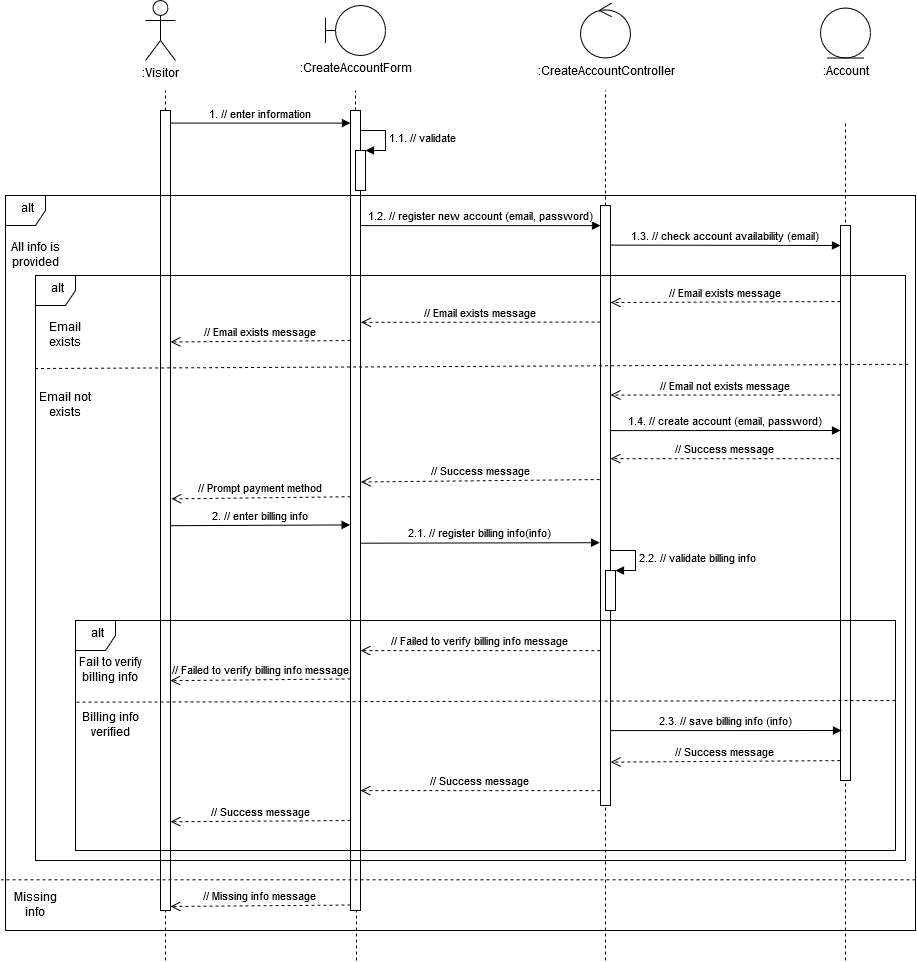


Figure 2‑4 Sequence diagram for the Create Account use case

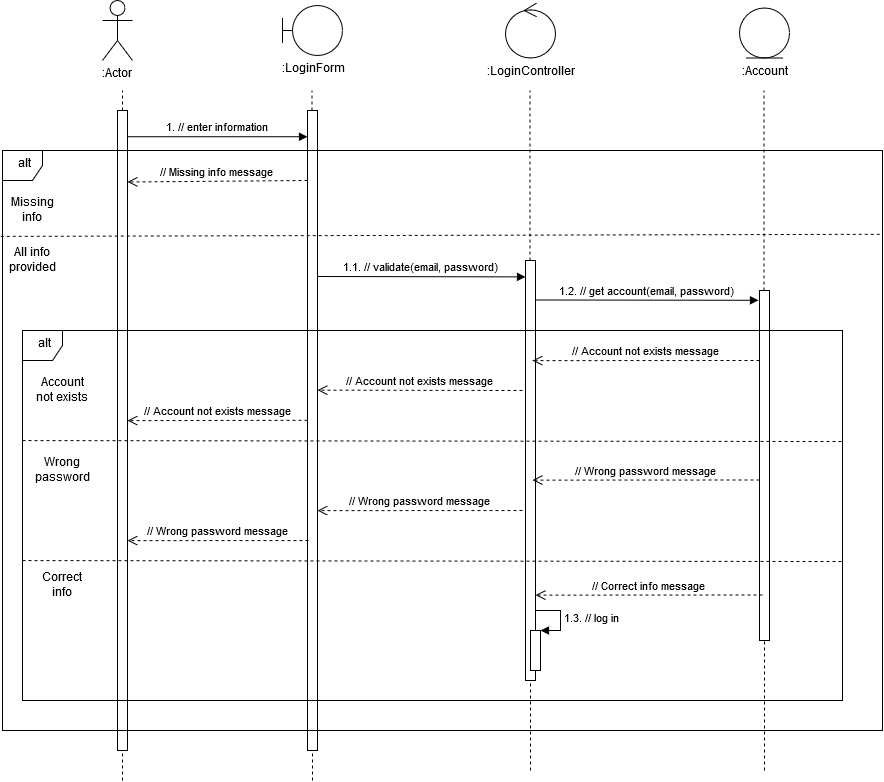


Figure ‑ Sequence diagram for the Log In use case

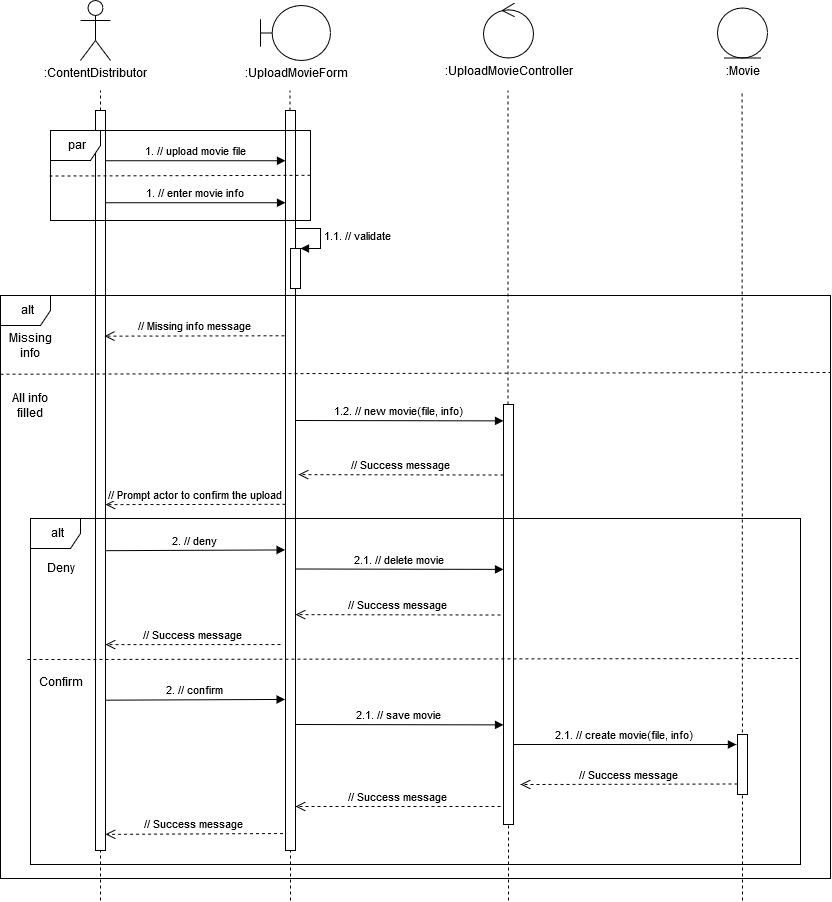


Figure ‑ Sequence diagram for the Upload Movie use case

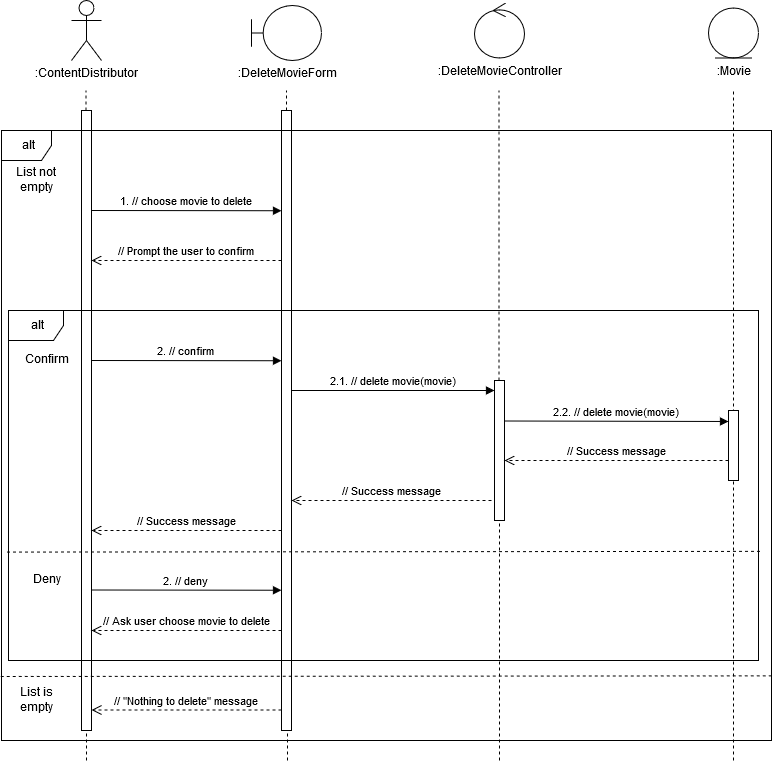


Figure ‑ Sequence diagram for the Delete movie use case

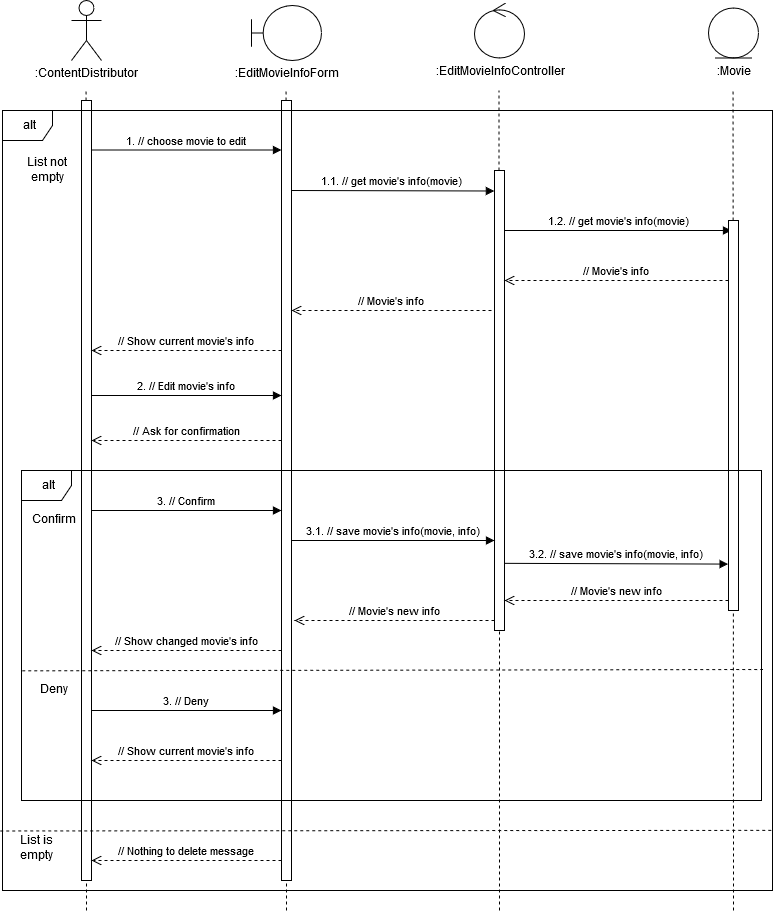


Figure ‑ Sequence diagram for the Edit movie's information use case

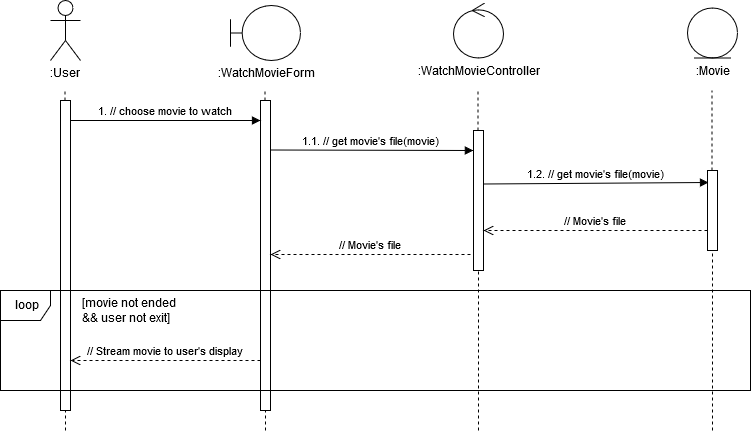


Figure ‑ Sequence diagram for the Watch movies use case

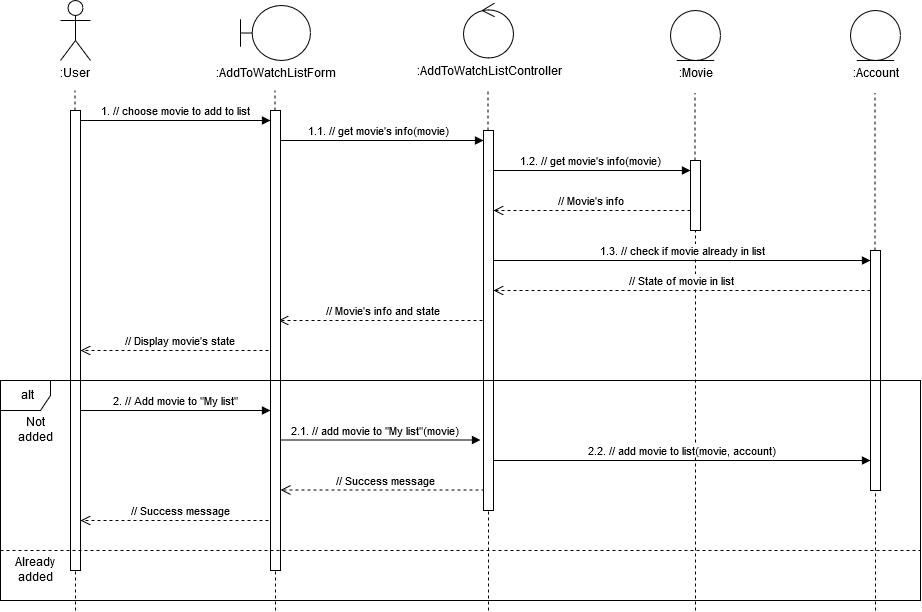


Figure ‑ Sequence diagram for the Add movie to watch list use case

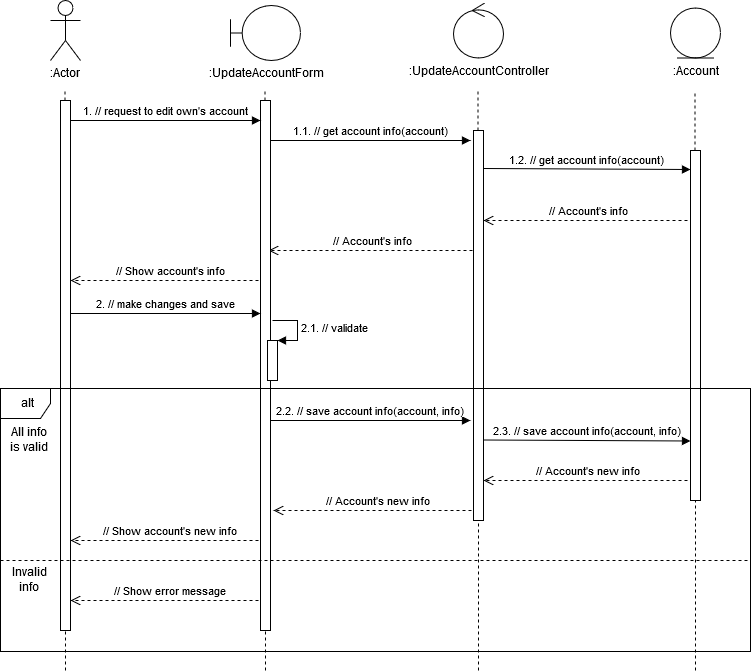


Figure ‑ Sequence diagram for the Update account's information use case

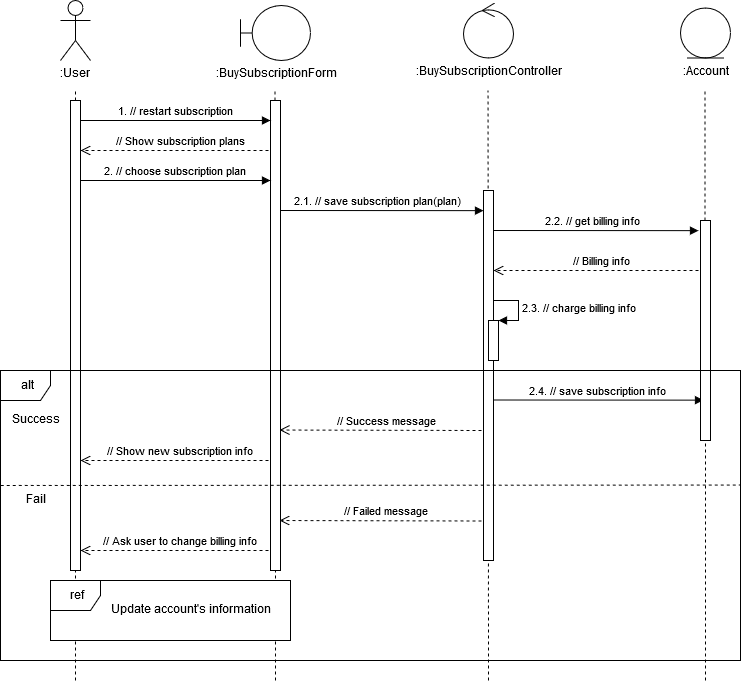


Figure ‑ Sequence diagram for the Buy subscription use case

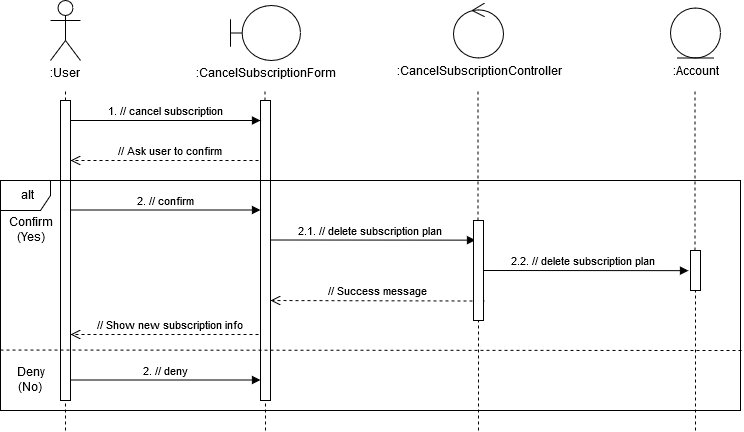


Figure ‑ Sequence diagram for the Cancel subscription use case

### Use-case realizations: Views of participating classes

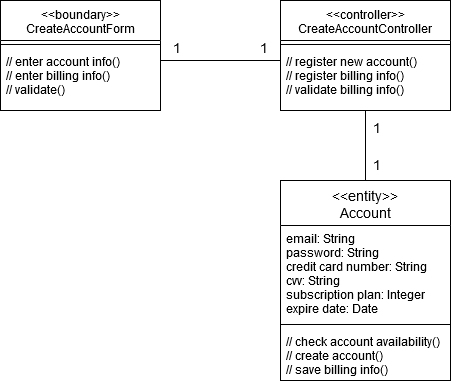


Figure ‑ VOPC for the Create Account use case

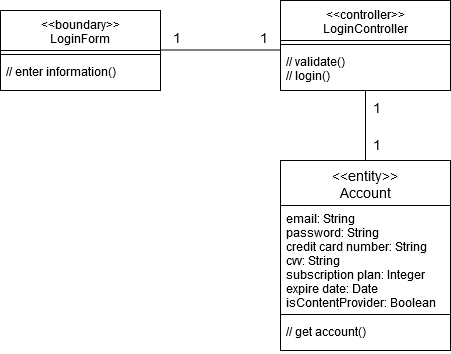


Figure ‑ VOPC for the Log In use case

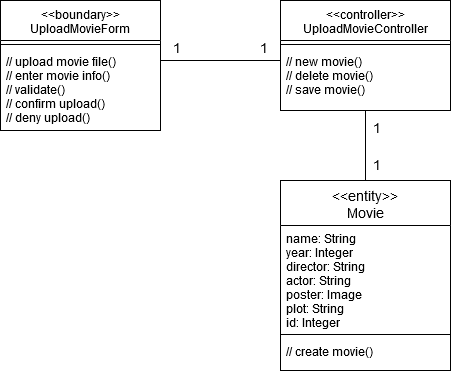


Figure ‑ VOPC for the Upload movie use case

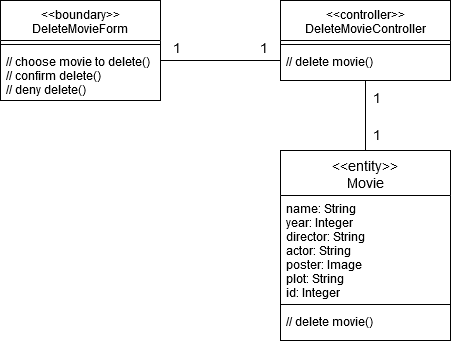


Figure ‑ VOPC for the Delete movie use case

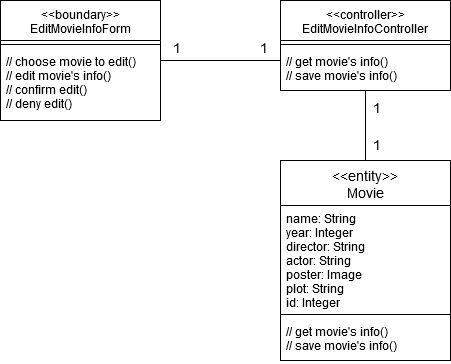


Figure ‑ VOPC for the Edit movie’s information use case

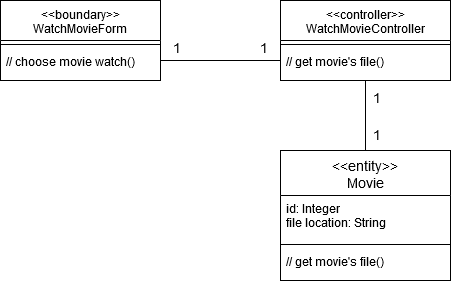


Figure ‑ VOPC for the Watch movies use case

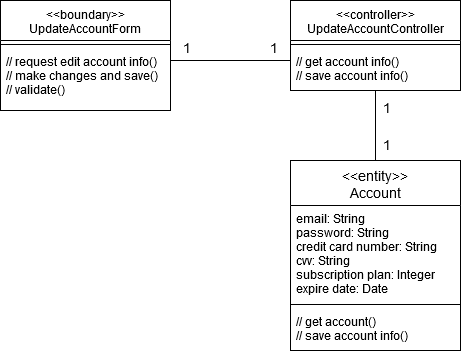


Figure ‑ VOPC for the Update account’s information use case

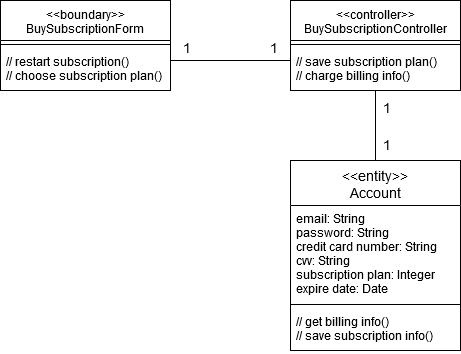


Figure ‑ VOPC for the Buy subscription use case

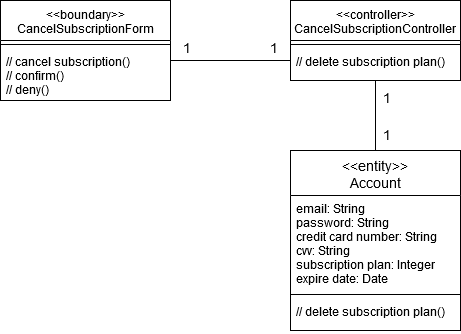


Figure ‑ VOPC for the Cancel subscription use case

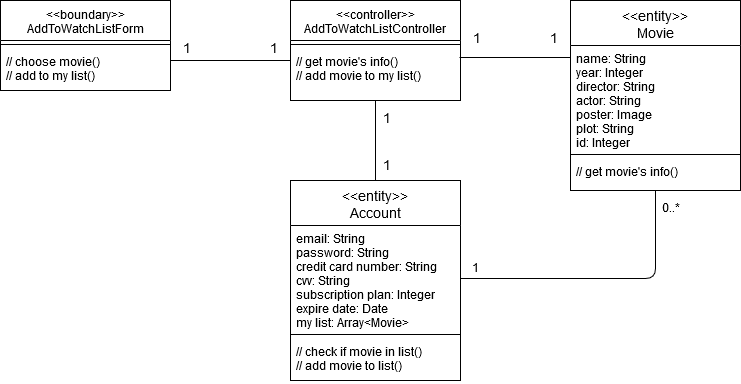


Figure ‑ VOPC for the Add movie to watch list use case

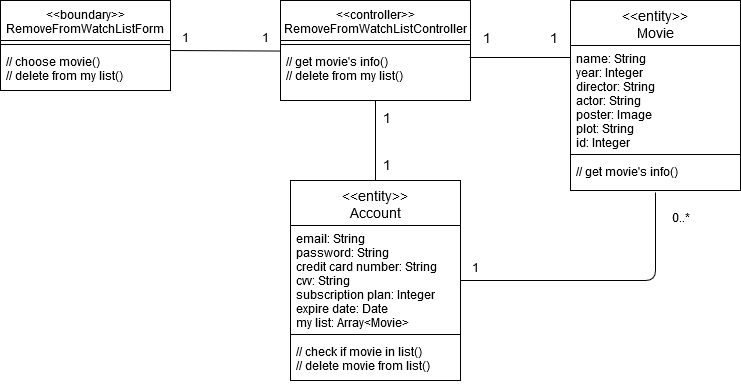


Figure ‑ VOPC for the Remove movie from watch list use case

### Describe analysis mechanism

|  |  |
| --- | --- |
| **Analysis class** | **Analysis mechanism** |
| Account | Persistency, Security |
| Movie |
| CreateAccountController | Distribution |
| LoginController |
| UploadMovieController |
| DeleteMovieController |
| EditMovieInfoController |
| WatchMovieController |
| UpdateAccountController |
| BuySubscriptionController |
| CancelSubscriptionController |
| AddToWatchListController |
| RemoveFromWatchListController |

Table ‑ Analysis-Class-To-Analysis-Mechanism map

**Analysis mechanism characteristic**

**Security**

* Data granularity: attribute level
* User granularity: three roles – unregistered users, registered users and content distributors
* Security rules:
  + Only registered users may log into the system.
  + Only logged in users may view and edit their own account information, buy or cancel their subscription, add or remove movies to/from their watch list.
  + Only logged in users may watch a movie
  + Only logged in content distributors may upload movies.
  + A movie could only be edited or deleted by its owner.

**Persistency**

|  |  |  |
| --- | --- | --- |
| Class | Account | Movie |
| Granularity | 1 MB per product | 1 to 30 GB per product |
| Volume | Up to 2,000,000,000 | Up to 500,000 |
| Access frequency | Create: 10,000 per day  Update: 5,000 per day | Create: 50 per day  Delete: 5 per day  Update: 25 per day  Read: 25,000 per day |

# Use-case design

## Architectural refinement

### Identify design elements

#### Identify classes

|  |  |
| --- | --- |
| **Analysis class** | **Design element** |
| Account | Account, Database subsystem |
| Movie | Movie, Database subsystem |
| CreateAccountController | Map directly to design classes |
| LoginController |
| UploadMovieController |
| DeleteMovieController |
| EditMovieInfoController |
| WatchMovieController |
| UpdateAccountController |
| BuySubscriptionController |
| CancelSubscriptionController |
| AddToWatchListController |
| RemoveFromWatchListController |
| CreateAccountForm |
| LoginForm |
| UploadMovieForm |
| DeleteMovieForm |
| EditMovieInfoForm |
| WatchMovieForm |
| UpdateAccountForm |
| BuySubscriptionForm |
| CancelSubscriptionForm |
| AddToWatchListForm |
| RemoveFromWatchListForm |

Table ‑ Analysis-Class-To-Design-Element map

#### Identify subsystems and interfaces

The *Database* subsystem provides support for relational databases written in the SQL language. The subsystem is designed as follow:

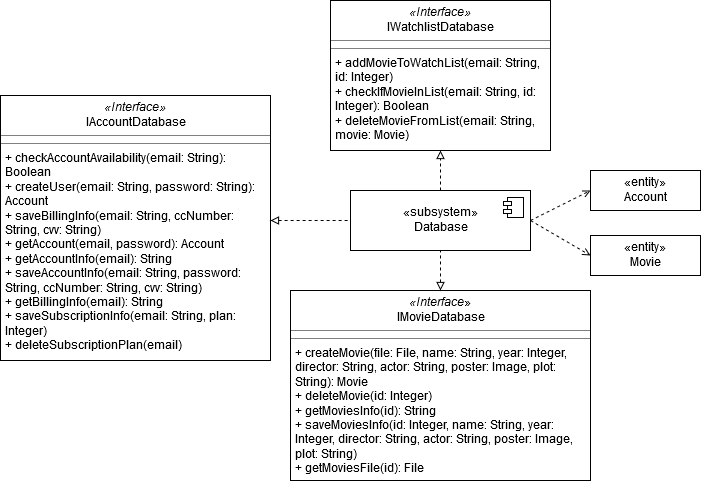


Figure ‑ The Database subsystem and its interfaces

#### Identify packages

Each layer in the analysis corresponds to a high-level package in the system.

**The “*Application”* package**

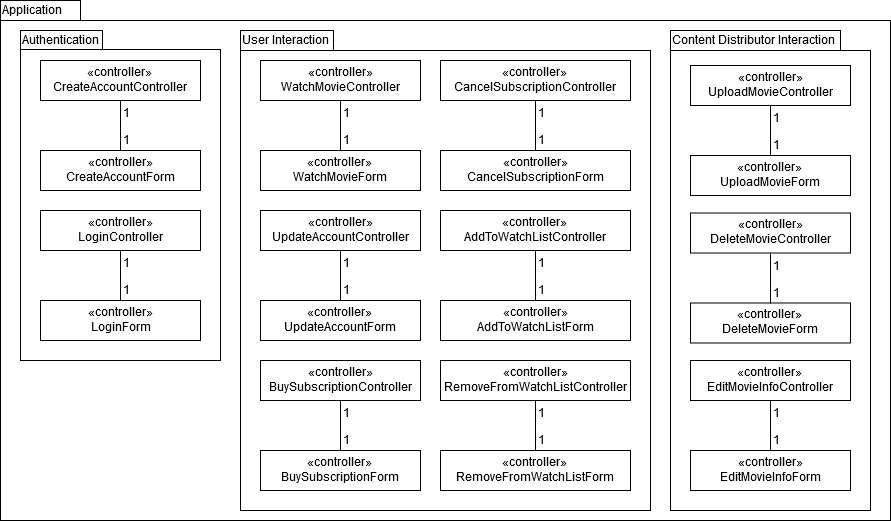
****

Figure ‑ The Application package and its sub-packages

The *Application* package contains the boundary and control classes, which are present in the client application. It is further divided into three sub-packages, each responsible for a different part of the application:

* The *Authentication* sub-class handles account creation and login.
* The *User Interaction* sub-class contains classes involving actions which require the user to be logged in: watching movies, updating account, buying/cancelling subscription and add/remove movies in watch list.
* The *Content Distributor Interaction* sub-class contains utilities that help content distributors maintain their movies. Only accounts marked as “Content Distributor” have access to these tools.

**The *Business Services* package**

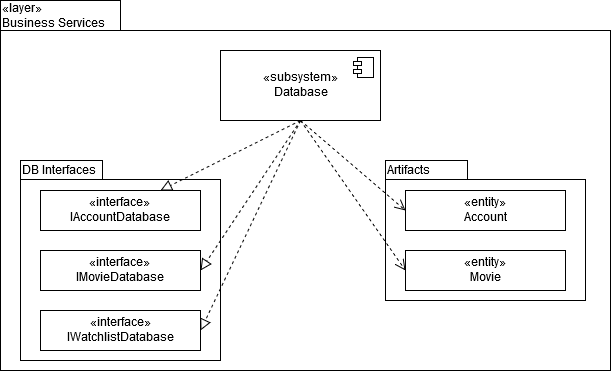


Figure ‑ The Business Services package

The *Business Services* package contains the *Database* subsystem and its interfaces, as well as the entity classes. These elements are common to all use cases.

**The *Middleware* package**

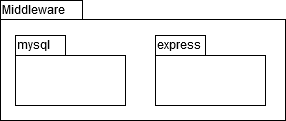
****

Figure ‑ The Middleware package

The *Middleware* package includes NodeJS’s MySQL package, which provides access to MySQL databases and the Express framework, which provides video streaming services.

**Packages and their dependencies**

As already stated, the *Application* package depends on the *Business Services* package and the *Middleware* package.

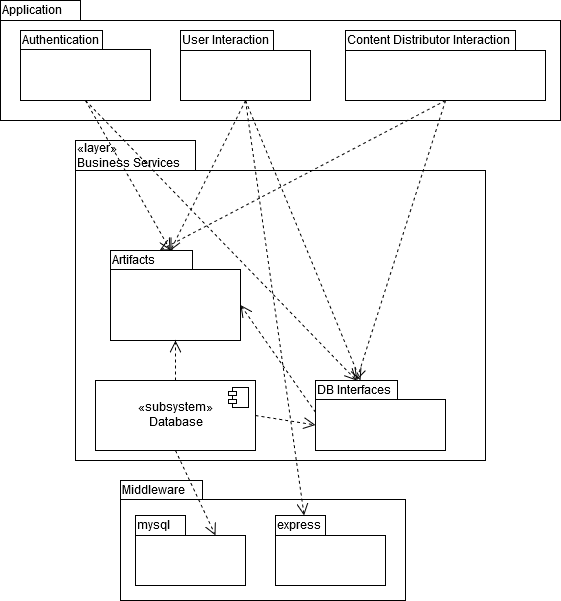


Figure ‑ Package dependencies diagram

### Identify design mechanisms

|  |  |  |
| --- | --- | --- |
| **Analysis mechanism** | **Design mechanism** | **Implementation mechanism** |
| Persistency | RDBMS | mysql connector |
| Security | Authentication | Cookies-based |
| Distribution | Client-Server | NodeJS, Express framework |

Table ‑ Design and implementation mechanisms

## Describe the run-time architecture

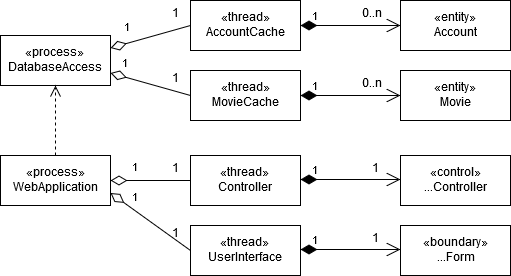


Figure ‑ The system’s process model

## Describe distribution

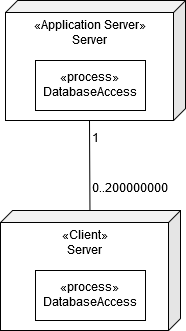


Figure ‑ The deployment view of the system

## Use-case design

### Design sequence diagrams

After incorporating the *Database* subsystem, the model’s sequence diagrams are updated as follows. Some method parameters are omitted for conciseness and readability – they are shown in full in the *Class Design* section.

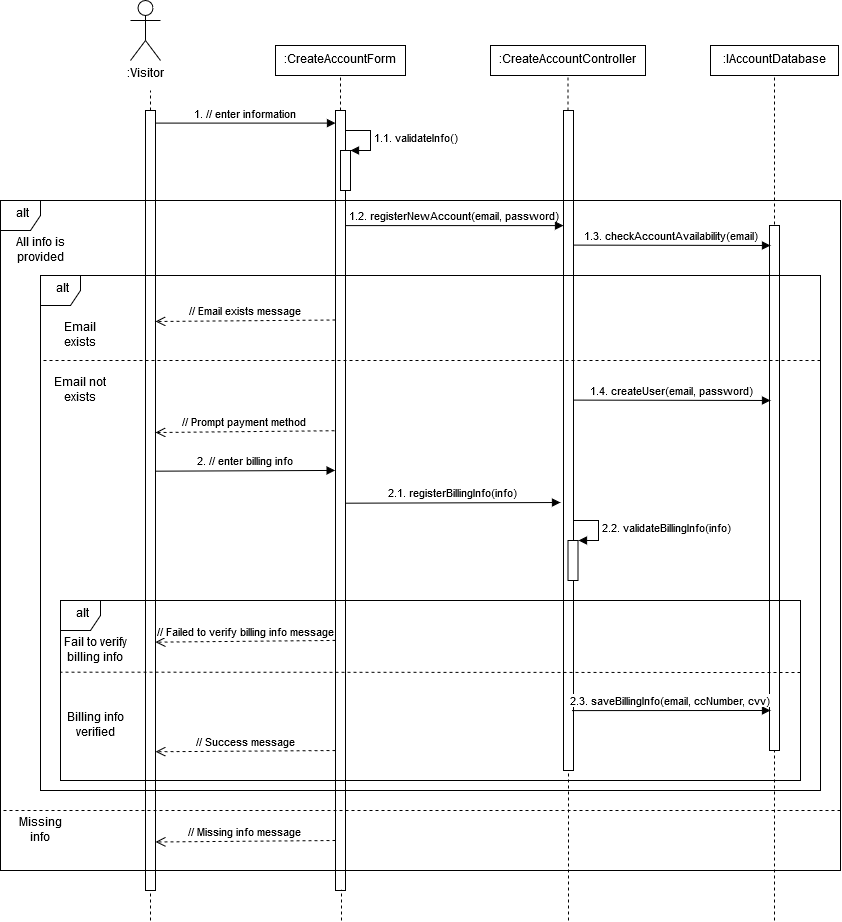


Figure ‑ Design sequence diagram for the Create Account use case

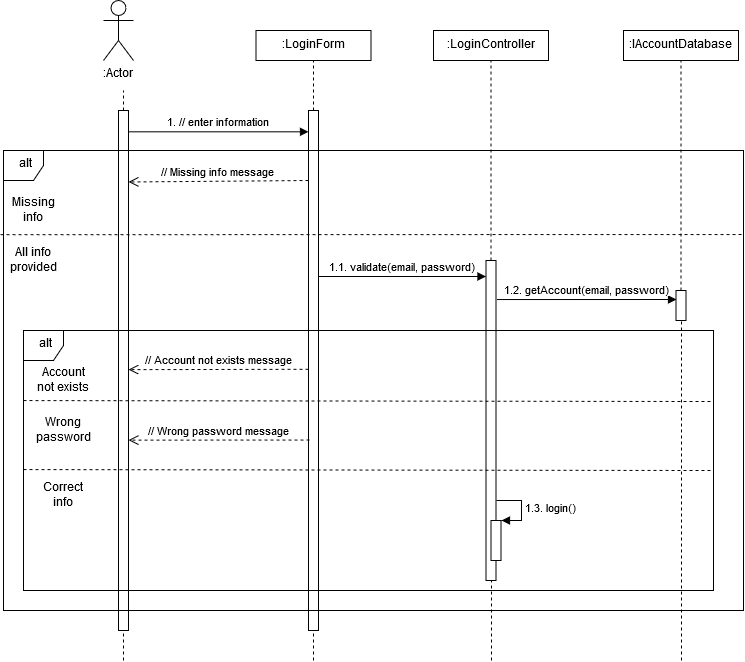


Figure ‑ Design sequence diagram for the Log In use case

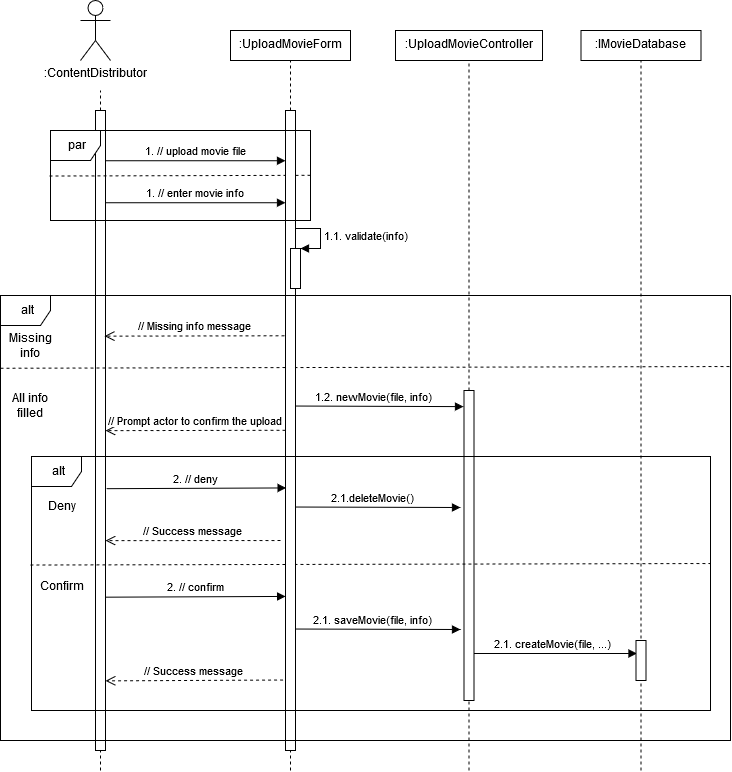


Figure ‑ Design sequence diagram for the Upload movie use case

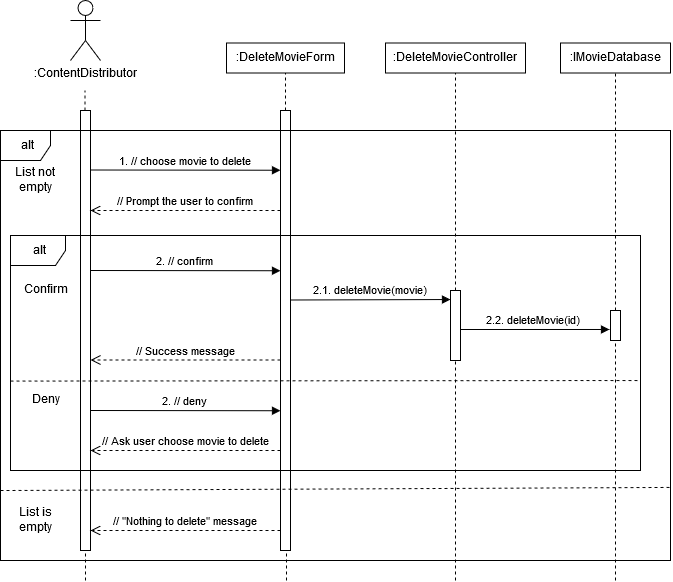


Figure ‑ Design sequence diagram for the Delete movie use case

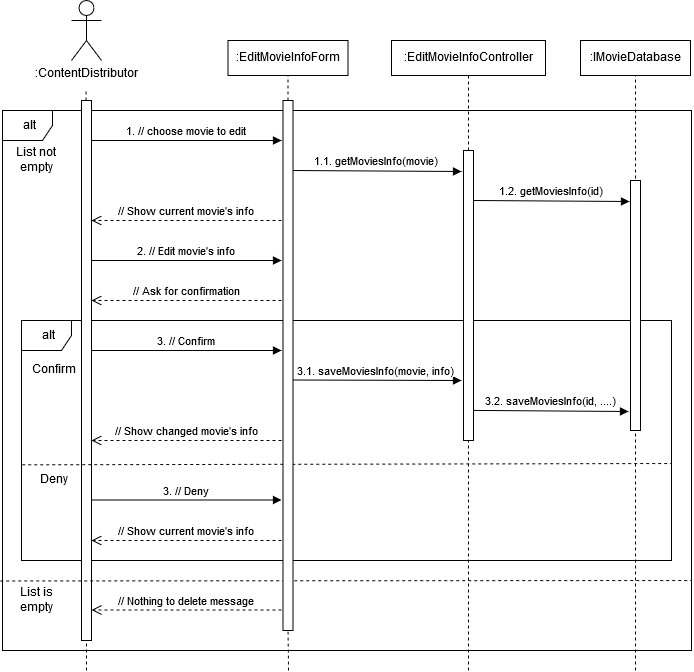


Figure ‑ Design sequence diagram for the Edit movie’s information use case

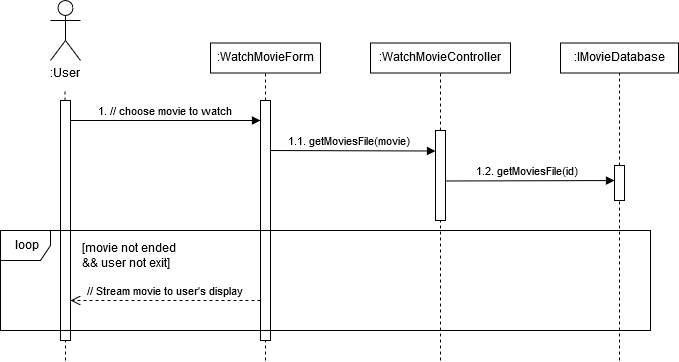
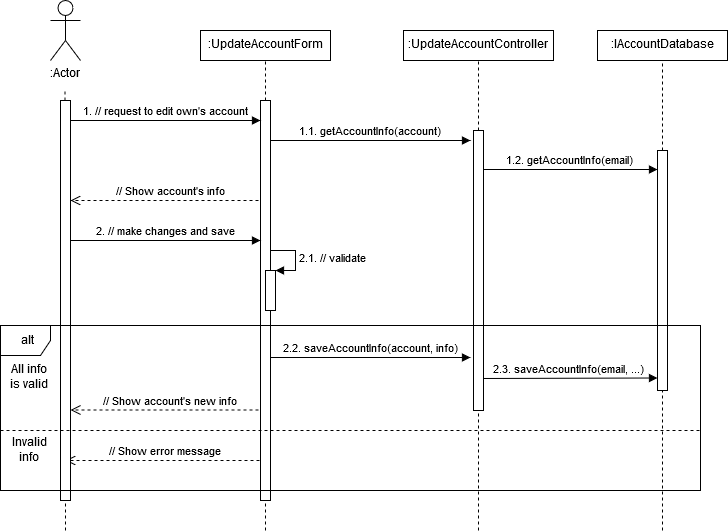


Figure ‑ Design sequence diagram for the Watch movies use case



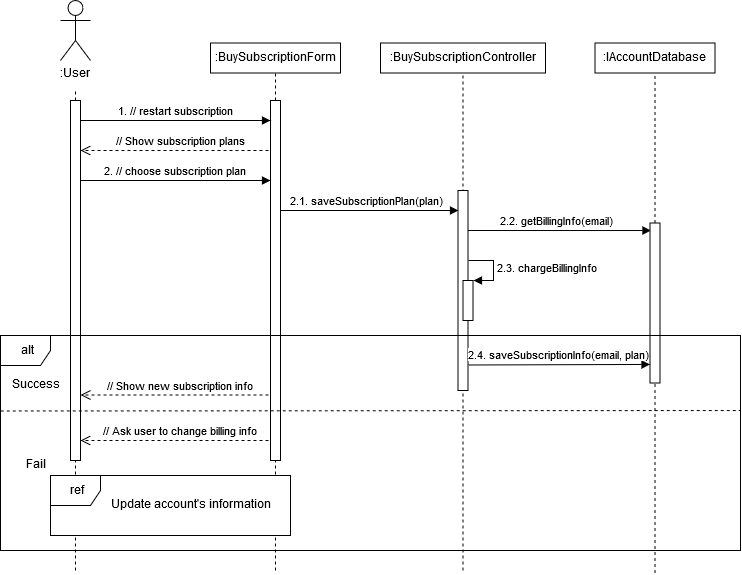
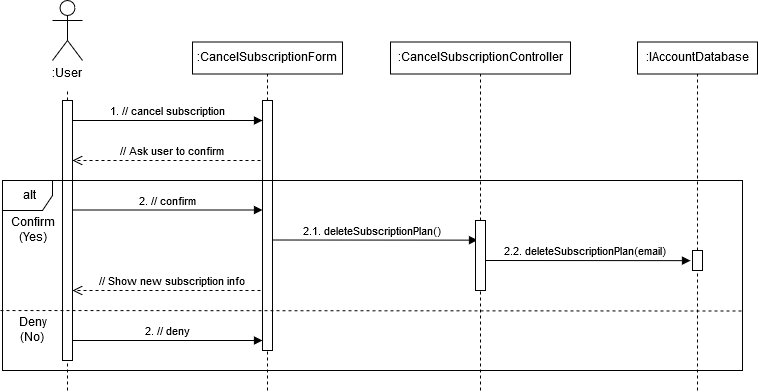
Figure ‑ Design sequence diagram for the Update account’s information use case

Figure ‑ Design sequence diagram for the Buy subscription use case



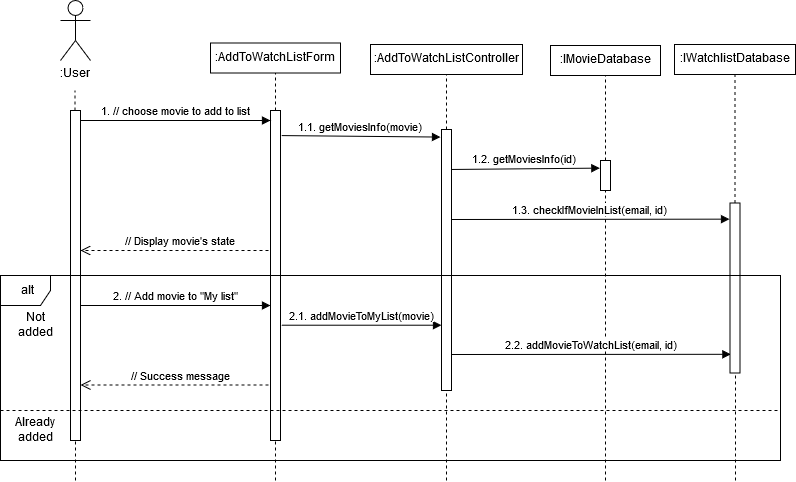
Figure ‑ Design sequence diagram for the Cancel Subscription use case

Figure ‑ Design sequence diagram for the Add movie to watch list use case

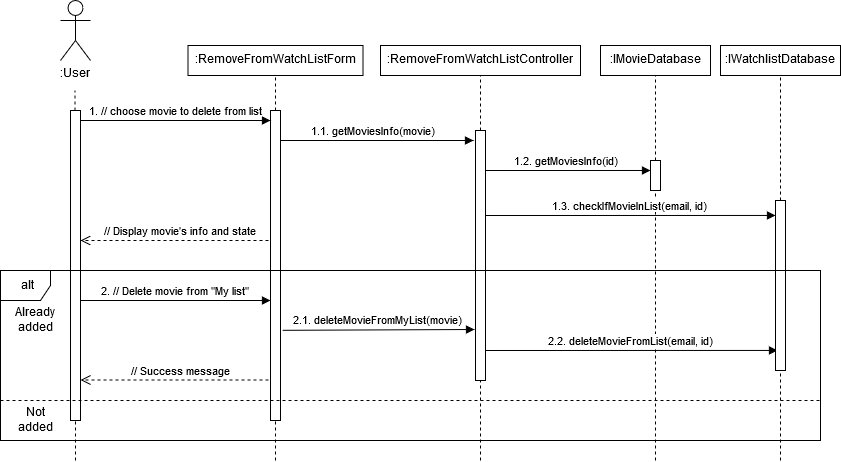


Figure ‑ Design sequence diagram for the Remove movie from watch list use case

### Design views of participating classes

The updated views of participating classes for each use case are described in the *Class Design* section.

## Subsystem design

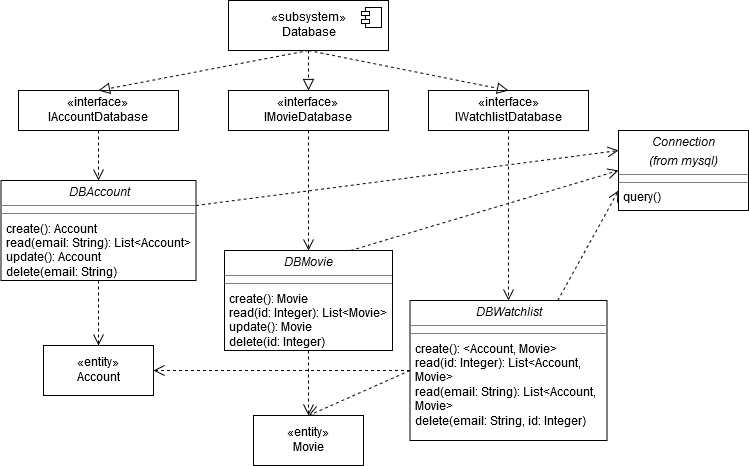


Figure ‑ Database subsystem elements diagram

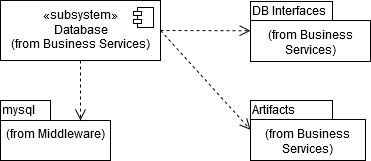


Figure ‑ Subsystem dependencies class diagram

## Class design

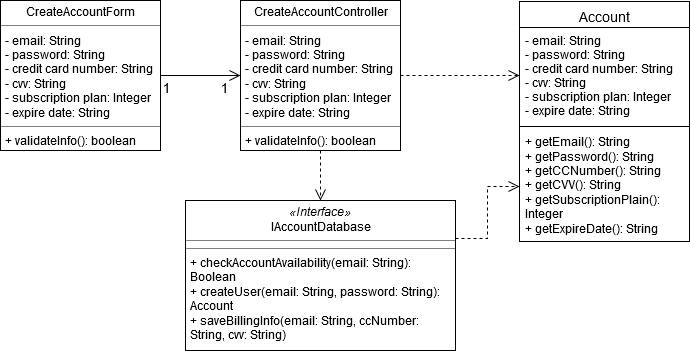


Figure ‑ Design VOPC for the Create Account use case

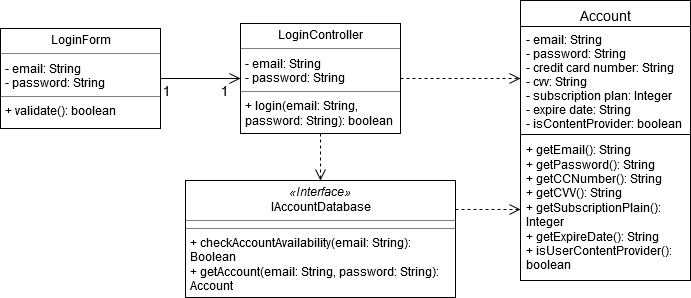


Figure ‑ Design VOPC for the Log in use case

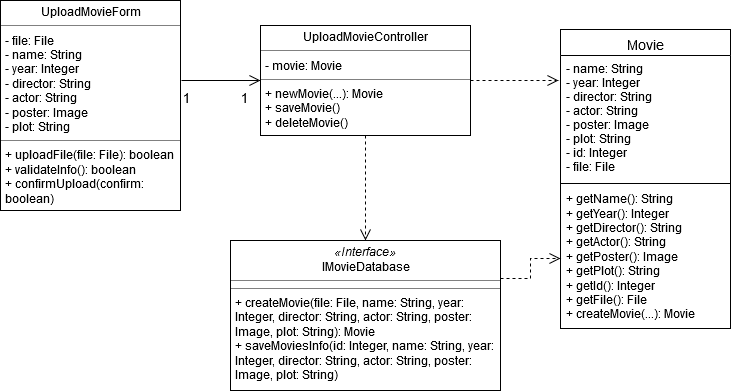


Figure ‑ Design VOPC for the Upload movie use case

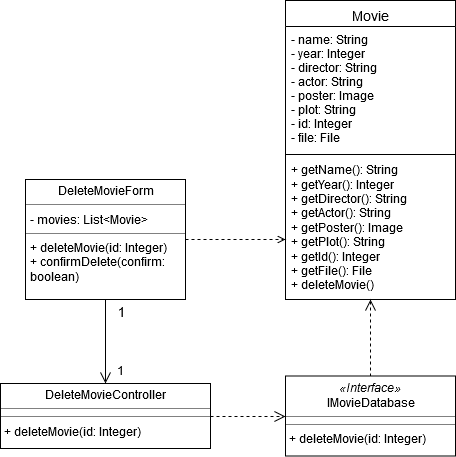


Figure ‑ Design VOPC for the Delete movie use case

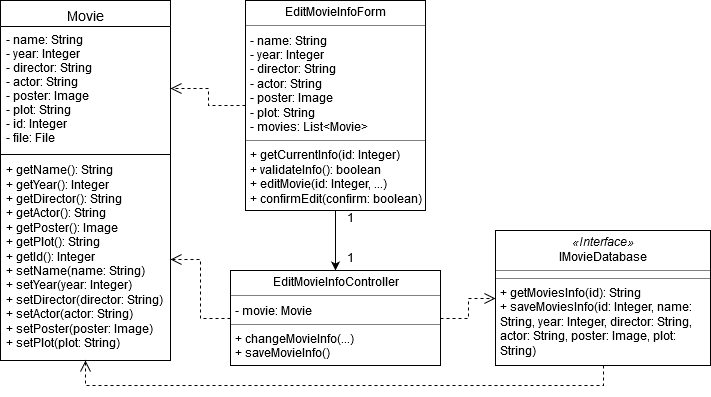


Figure ‑ Design VOPC for the Edit movie’s information use case

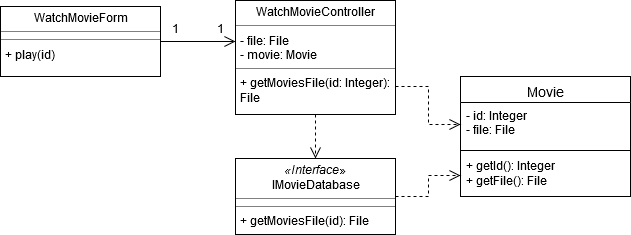


Figure ‑ Design VOPC for the Watch movies use case

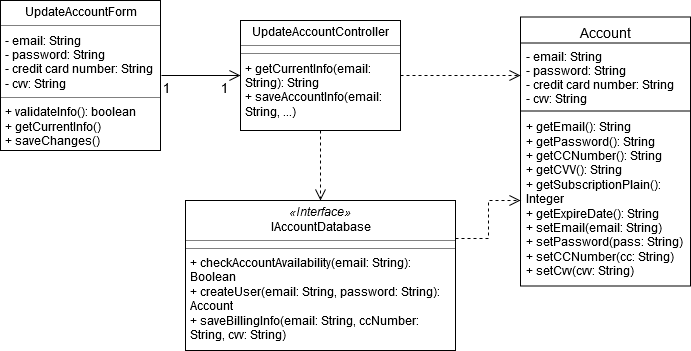


Figure ‑ Design VOPC for the Update account’s information use case

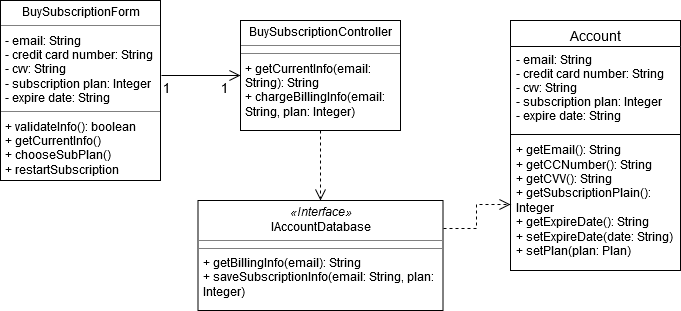


Figure ‑ Design VOPC for the Buy subscription use case

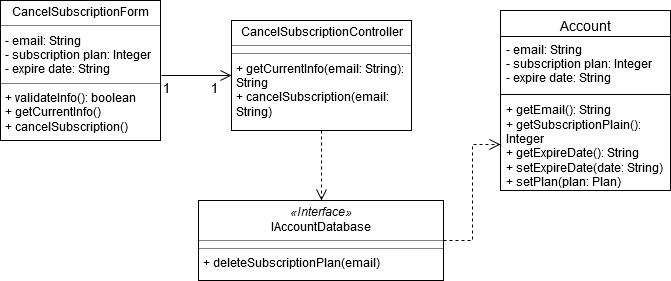


Figure ‑ Design VOPC for the Cancel subscription use case

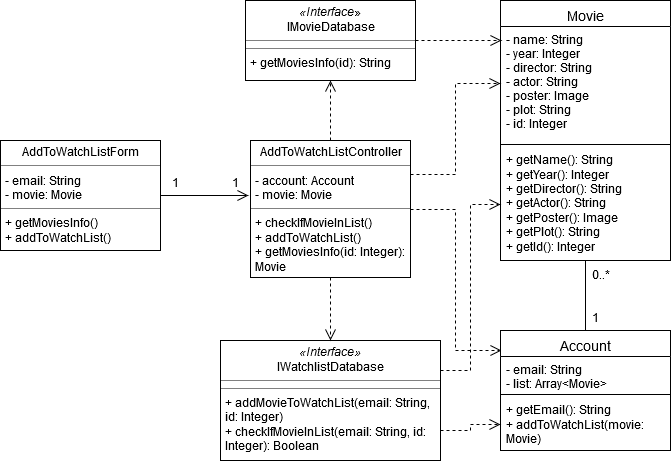


Figure ‑ Design VOPC for the Add movie to watch list use case

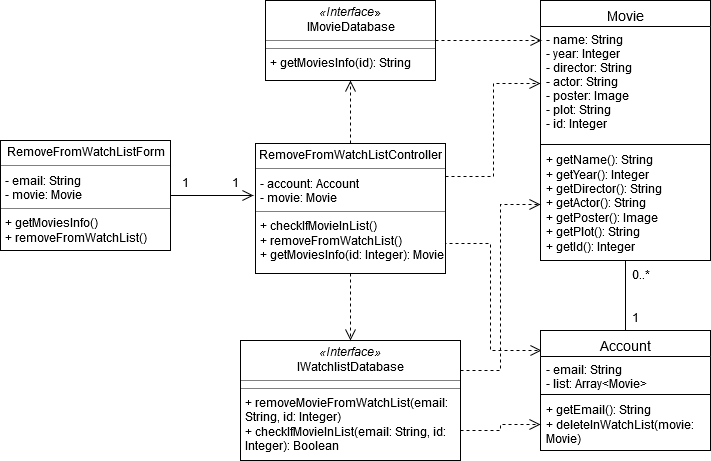


Figure ‑ Design VOPC for the Remove movie from watch list use case

## Database design

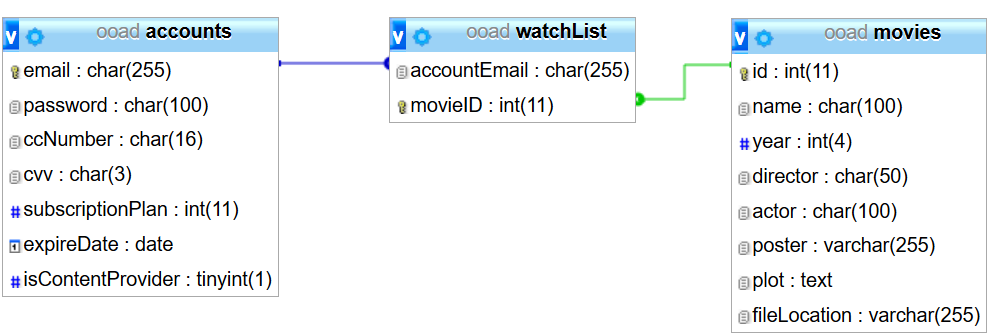


Figure ‑ The relational data model