

VIETNAMESE STUDY SYSTEM FOR JAPANESE

Architecture Design

Project Code: Veazy

Document Code: Veazy\_ Architecture Design\_EN

**Ha Noi, 31/05/2016**

Record of change

\*A - Added M - Modified D - Deleted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Effective Date | Changed Items | A\*M, D | Change Description | New Version |
| 07/07/2014 | Component diagram | M | Modified | v1.1 |
| 11/08/2014 | Activity diagram | M | Modified | v1.2 |
| 16/08/2014 | Use-case view | M | Modified | v1.3 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table 1: Record of change

SIGNATURE PAGE

**ORIGINATOR:** Dao Thanh Tung 31/05/2016

Technical Leader

**REVIEWERS:** Nguyen Ngoc Minh 31/05/2016

Project Manager

**APPROVAL:**  Nguyen Van Sang 31/05/2016

Supervisor

TABLE OF CONTENTS

[1 Introduction 6](#_Toc459314547)

[1.1 Purpose 6](#_Toc459314548)

[1.2 Scope 6](#_Toc459314549)

[1.3 Definitions, Acronyms and Abbreviations 6](#_Toc459314550)

[1.4 References 6](#_Toc459314551)

[1.5 Overview 7](#_Toc459314552)

[2 Choice of Architecture design 7](#_Toc459314553)

[2.1 MVC Model 7](#_Toc459314554)

[**1.2.1** **MVC Model overview** 7](#_Toc459314555)

[2.2 SpringMVC framework 9](#_Toc459314556)

[**2.2.1** **Overview** 9](#_Toc459314557)

[**2.2.2** **Advantages & Disadvantages** 11](#_Toc459314558)

[**2.2.3** **The reason of choosing** 12](#_Toc459314559)

[2.3 HibernateORM framework 12](#_Toc459314560)

[**3.2.1** **Overview** 12](#_Toc459314561)

[**3.2.2** **Advantages & Disadvantages** 13](#_Toc459314562)

[**3.2.3** **The reason of choosing** 14](#_Toc459314563)

[2.4 AngularJS framework 14](#_Toc459314564)

[**4.2.1** **Overview** 14](#_Toc459314565)

[**4.2.2** **Advantages & Disadvantages** 15](#_Toc459314566)

[**4.2.3** **The reason of choosing** 16](#_Toc459314567)

[3 Architectural Representation 17](#_Toc459314568)

[4 Architectural Goals and Constraints 18](#_Toc459314569)

[5 Use-Case View 19](#_Toc459314570)

[5.1 Guest 19](#_Toc459314571)

[**1.5.1** **Register account** 19](#_Toc459314572)

[**1.5.2** **Learning** 20](#_Toc459314573)

[**1.5.3** Testing 21](#_Toc459314574)

[5.2 Member 22](#_Toc459314575)

[**2.5.1** Private Account Management 22](#_Toc459314576)

[**2.5.2** Testing 23](#_Toc459314577)

[**2.5.3** Content Reporting 24](#_Toc459314578)

[5.3 Editor 25](#_Toc459314579)

[**3.5.1** Private Account Management 25](#_Toc459314580)

[**3.5.2** Lesson Management 26](#_Toc459314581)

[**3.5.3** Question Management 27](#_Toc459314582)

[5.4 Administrator 28](#_Toc459314583)

[**4.5.1** Private Account Management 28](#_Toc459314584)

[**4.5.2** System Account Management 29](#_Toc459314585)

[6 Logical View 30](#_Toc459314586)

[6.1 Overview 30](#_Toc459314587)

[6.2 Architecturally Significant Design Packages 31](#_Toc459314588)

[7 Process View 39](#_Toc459314589)

[8 Deployment view 46](#_Toc459314590)

[9 Quality 46](#_Toc459314591)

# Introduction

## Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.

## Scope

The scope of this document is to depict the architecture of the Vietnamese Study System for Japanese developed by Veazy capstone project team.

## Definitions, Acronyms and Abbreviations

|  |  |  |
| --- | --- | --- |
| Acronym | Definition | Note |
| Veazy | Vietnamese Study System for Japanese |  |
| MVC | Model View Control |  |
| IDE | Integrated Development Environment |  |
| Q&A | Question and Answer |  |
| GUI | Graphic User Interface |  |
| TDD | Test-Driven-Development |  |

*Figure 1‑1 Definitions table*

## References

* Veazy\_ SRS\_v1.4\_EN.docx
* <http://en.wikipedia.org/wiki/Model-view-controller>
* <http://www.tutorialspoint.com/mvc_framework>
* <http://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm>
* <http://www.tutorialspoint.com/hibernate/hibernate_overview.htm>
* http://www.tutorialspoint.com/angularjs/angularjs\_overview.htm

## Overview

The Software Architecture Document contains the following subsections:

* Section 1: Provide an overview of entire Software Architecture Document.
* Section 2: Choice of Architecture Design
* Section 3: Architectural Representation
* Section 4: Architectural Goals and Constraints
* Section 5: Use-Case View
* Section 6: Logical View
* Section 7: Process View
* Section 8: Deployment view.
* Section 9: Quality.

# Choice of Architecture design

## MVC Model

The architecture of Veazy system is structured based on MVC model combined with layered architecture, Spring framework and AngularJS framework.

### **MVC Model overview**

*Model–view–controller (MVC)* is a software [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) for implementing [user interfaces](https://en.wikipedia.org/wiki/User_interface) on computers. It divides a given software application into three interconnected parts: the model, the view, and the controller, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. The model consists of application data, business rules, logic and functions. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as bar chart for management and a tabular view for accountants. The controller mediates input, converting it to commands for the model or view.

The MVC design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects playing in the application but the way objects communicate with each other as well. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an application sometimes referred to as a layer—for example, model layer.



Figure 2‑1 MVC Model

In addition to dividing the application into three kinds of components, the MVC design defines the interactions between them:

* **A controller:** can send commands to its associated view to change the view's presentation of the model. It can also send commands to the model to update the model's state.
* **A model:** stores data that is retrieved according to commands from the controller and displayed in the view.
* **A view:** requests from the model the information that it needs to generate an output representation to the user.

#### **Advantages and disadvantages of MVC model**

* Advantages:
* The MVC model demonstrates professionalism in programming and design analysis. It is divided into independent components to help develop applications faster, simpler, easier upgrades and maintenance.
* Many MVC vendor framework tool kits are available.
* Multiple views synchronized with same data model.
* Easy to change or plug in new interface views, allowing updating of interface views with new technologies without overhauling the rest of system.
* Very effective for deployment if graphic, programming and database development professionals are working in a team in a designed project.
* Disadvantages:
* For small projects that apply MVC model caused cumbersome, time consuming in development process.
* Time consuming to transits data between components.
* Not suitable for agent-oriented applications such as interactive mobile and robotics applications.
* Multiple pairs of controllers and views based on the same data model make data model change expensive.

#### **The reason of choosing MVC model**

* Easily manage the complexity of application by dividing the application into three components: model, view and controller.
* Better support for TDD.
* It is a good support for application built by project team that has many developers and designers but still managed application features.
* Veazy system is an incomplete system for now. Veazy is built in the way that towards extensibility and maintainability in the future.

## SpringMVC framework

### **Overview**

*Spring’s web MVC framework* is, like many other web MVC frameworks, request-driven, designed around a central Servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications. Spring’s DispatcherServlet however, does more than just that. It is completely integrated with the Spring IoC container and as such allows you to use every other feature that Spring has.

The Spring’s web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

* The Model encapsulates the application data and in general they will consist of POJO.
* The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The Controller is responsible for processing user requests and building appropriate model and passes it to the view for rendering.



*Figure 2‑2 SpringMVC process flow*

1. *DispatcherServlet* receives the request.
2. *DispatcherServlet* dispatches the task of selecting an appropriate controller to *HandlerMapping*. *HandlerMapping* selects the controller which is mapped to the incoming request URL and returns the (selected *Handler*) and *Controller* to *DispatcherServlet*.
3. *DispatcherServlet* dispatches the task of executing of business logic of *Controller* to *HandlerAdapter*.
4. *HandlerAdapter* calls the business logic process of *Controller*.
5. *Controller* executes the business logic, sets the processing result in *Model* and returns the logical name of view to *HandlerAdapter*.
6. *DispatcherServlet* dispatches the task of resolving the *View* corresponding to the *View* name to *ViewResolver*. *ViewResolver* returns the *View* mapped to *View* name.
7. *DispatcherServlet* dispatches the rendering process to returned *View*.
8. *View* renders *Model* data and returns the response.

### **Advantages & Disadvantages**

* Advantages
  + No need to reinvent the wheel: It utilizes some of the well-known technologies, ORM frameworks, logging frameworks, JEE, JDK timers, Quartz and so on
  + Dependency Injection gears up testability.
  + Inversion control and APIs: Spring framework provides inversion control and APIs to translate technology-driven exceptions, specifically thrown by JDBC, Hibernate or JDO, into unchecked and consistent ones.
  + Modularity: Spring makes it easy for the developers to know which packages or classes are to be used and which one should be ignored.
  + Consistent Transaction Management: With the help of consistent transaction management interface, Spring framework easily scale down or scale up local as well as global transactions.
  + Spring AOP also brings tons of benefits: There is no need for a developer to have a separate compilation unit or a separate class loader.
* Disadvantages
  + Complex: One of the major criticisms faced by the Spring framework is that it is complex.
  + Longer Learning Curve: it would be quite difficult to learn Spring framework. The main reason behind this is a whole host of new programming methods and detailing require understanding how to set up the Spring XML configuration file.
  + All the Spring applications require a lot of XML: the applications developed using Spring framework often require a huge amount of XML
  + Tons of parallel mechanisms frustrate developers: It makes developers to spend lots of understanding each of them and choose the best one among them.
  + Lack of Guidelines: No clear guidance on cross-site scripting attacks and cross-site request forgery attacks in Spring MVC documentation.

### **The reason of choosing**

* + Spring provides a very clean division between controllers, JavaBean models, and views.
  + Spring’s MVC is very flexible, it is entirely based on interfaces.
  + Spring provides interceptors as well as controllers, making it easy to factor out behavior common to the handling of many requests.
  + Spring Controllers are configured via IoC. This makes them easy to test, and beautifully integrated with other objects managed by Spring.
  + No ActionForms. Bind directly to domain objects.
  + Spring has a well defined interface to business layer.

## ****HibernateORM framework****

### **Overview**

*Hibernate* is an implementation of the Java persistence API (JPA) specification, it can easily used in any environment supporting JPA including Java SE applications, Java EE applications, Java EE application servers, Enterprise OSGi containers and so on. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.



*Figure 2‑3 Hibernate ORM*

### **Advantages & Disadvantages**

* Advantages
  + Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
  + Provides simple APIs for storing and retrieving Java objects directly to and from the database.
  + If there is change in Database or in any table then the only need to change XML file properties.
  + Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
  + Hibernate does not require an application server to operate.
  + Manipulates complex associations of objects of your database.
  + Minimize database access with smart fetching strategies.
  + Provides simple querying of data.
* Disadvantages
  + Slower than JDBC: Hibernate is slower than pure JDBC as it is generating lots of SQL statements in runtime.
  + Not suitable for Batch processing: It is advisable to use pure JDBC for batch processing.
  + Not suitable for Small projects : For small project having few tables, it is useless to work with hibernate.
  + Does not allow multiple inserts : Hibernate does not allow some type of queries which are supported by JDBC.
  + Generates complex queries with many joins : For complex data, mapping from Object-to-tables and vise versa reduces performance and increases time of conversion.

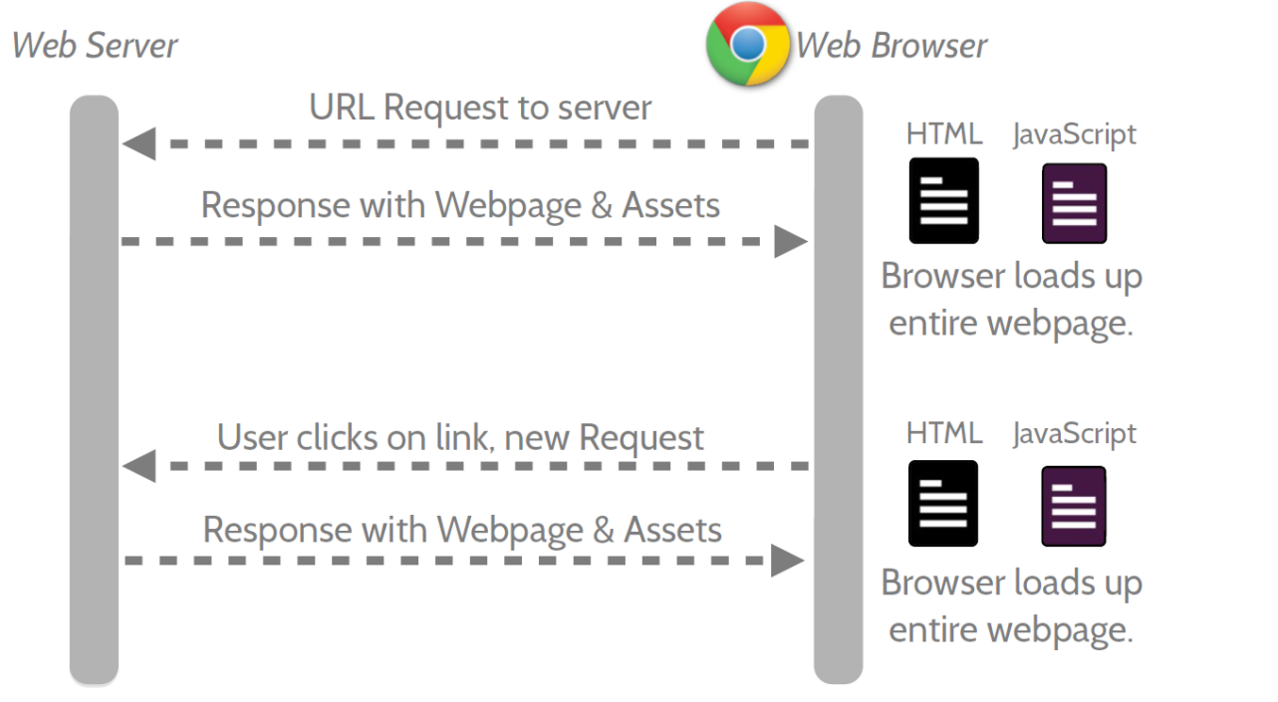
### **The reason of choosing**

* + Productivity: It helps developers get rid of writing complex and tedious SQL statement, no more need of JDBC APIs for result set or data handling. It makes developers more concentrate on the business logic and increase the project’s productivity.
  + Maintainability: It helps reduce the lines of code, makes system more understandable and emphasizes more on business logic rather than persistence work (SQLs).
  + Portability: It abstracts our application away from the underlying SQL database and sql dialect. Switching to other SQL database requires few changes in Hibernate configuration file.

## AngularJS framework

### **Overview**

*AngularJS* is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.



*Figure 2‑4 Multi-page flow*



*Figure 2‑5 Single-page flow*

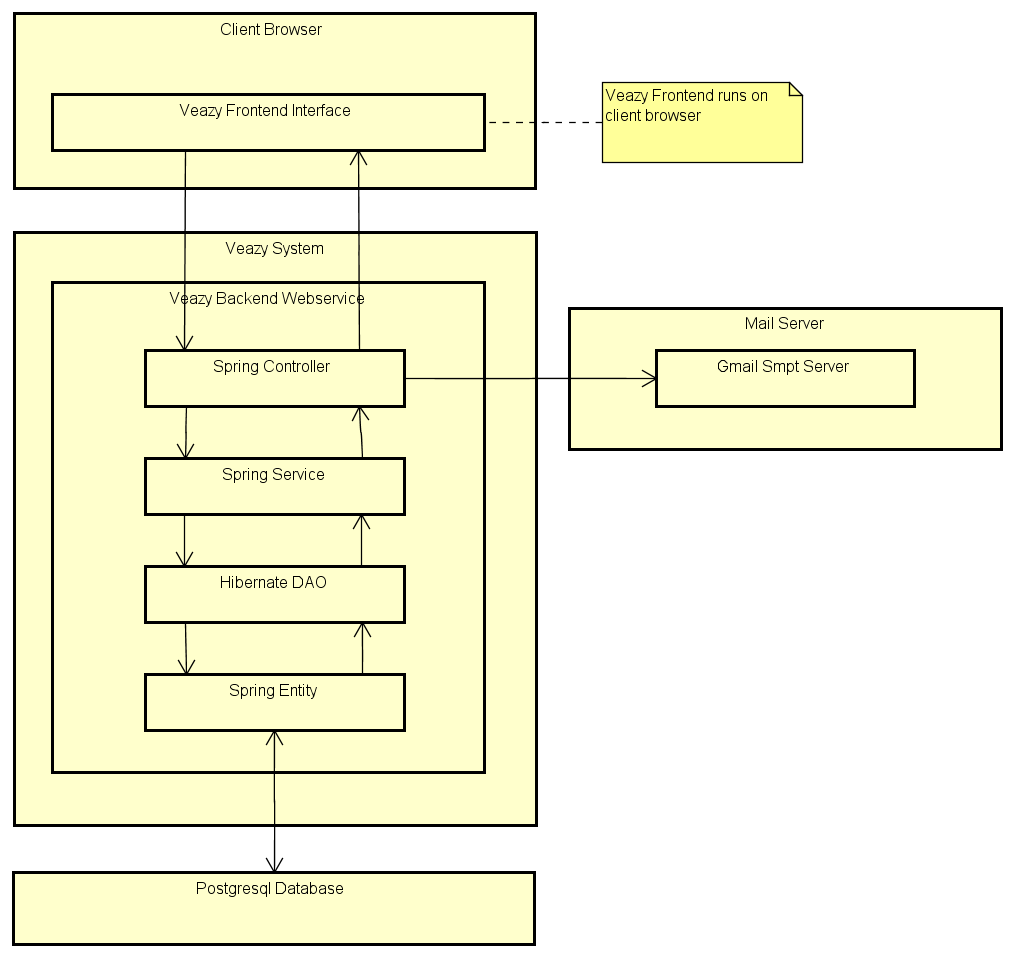
### **Advantages & Disadvantages**

* Advantages
  + AngularJS provides capability to create Single Page Application in a very clean and maintainable way.
  + AngularJS provides data binding capability to HTML thus giving user a rich and responsive experience
  + AngularJS code is unit testable.
  + AngularJS uses dependency injection and make use of separation of concerns.
  + AngularJS provides reusable components.
  + With AngularJS, developer write less code and get more functionality.
  + In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.
* Disadvantages
  + Not Secure − Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
  + Not degradable − If your application user disables JavaScript then user will just see the basic page and nothing more.

### **The reason of choosing**

* + Expressive User Interface: Angular employs HTML to define application user interface. HTML carries special attributes explaining which controllers to use for elements.
  + Less code: By ignoring writing of own pipeline, use of html, simple data model and use of filters, angular makes it coding free for developers.
  + Testability: AngularJs has been developed keeping in mind the testing process of coding and developing at each and every stage of application development so as to avoid wastage of time.

# Architectural Representation



*Figure 3‑1 Veazy System overview*

We follow MVC architecture to implement the Veazy project. MVC offers architectural benefits over jQuery and AngularJS — it helps us write better-organized and therefore more maintainable code.

**Model** is where the application’s data objects are stored. A model object is in charge of encapsulating application state and one object could be related to other objects establishing a one-to-one or one-to-many relationship. When a model changes, typically it will notify its observers that a change has occurred. As with any data object it contains instance variables and getter/setter methods.

**DAO** is an abbreviation of Data Access Object which provides an abstract [interface](https://en.wikipedia.org/wiki/Interface_(computer_science)) to some type of [database](https://en.wikipedia.org/wiki/Database) or other persistence mechanism. By mapping application calls to the persistence layer, DAO provide some specific data operations without exposing details of the database.

**Service** defines set of available operations in regards with interfacing client layers i.e. encapsulates the application's business logic.

**Controller** is the decision maker and the glue between the model and view; it handles user actions and gestures, and responds to user events.

# Architectural Goals and Constraints

**Availability:** The application must be available 95% of time. Users can access to it everywhere from there .Web browser with internet connection.

Maintainability:

* Coding standards and naming conventions
  + Output of the project must include coding standards and naming conventions documentations. Implementation code must be easy to maintain.
  + All code must be clearly commented, including class, method documentations.
  + If some components are reused, the documentations of those components must also be included.
* Design
  + The design of the system must be loosely coupled that chances on some module will not affect others.
* Logging
  + All the errors should be logged, supporting for bug fixing and maintenance.
  + All strange or sensitive situations should also be logged.

**Usability:** Usability Requirements support the following from the perspective of its primary users:

* *Efficiency of use***:** user can complete each function in less than 8 actions.
* *Intuitiveness***:** all help/error messages are simple to understand; user can know exactly how to do each feature after one time using it.

Capacity and scalability:

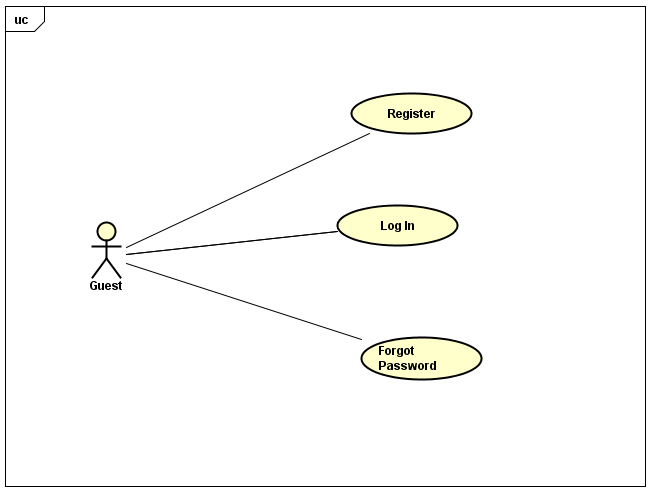
* Throughput, storage and growth requirements.

# Use-Case View

The use-case view consists of 4 parts according to 4 user roles in the system: guest, member, editor and admin.

## Guest

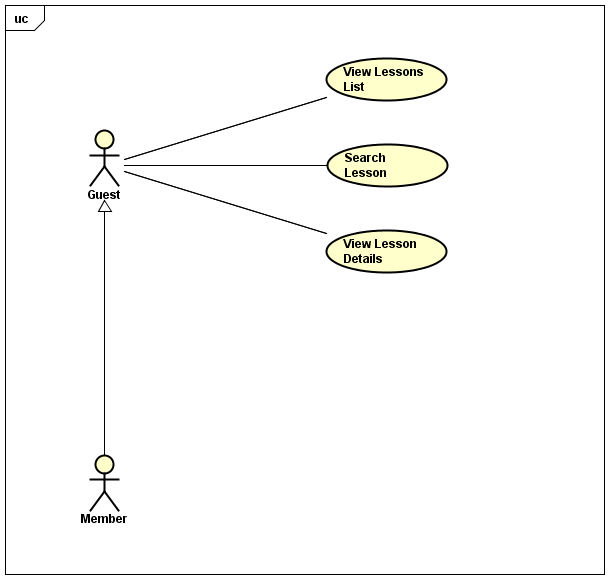
### **Register account**



*Figure 5‑1* ***Register account***

|  |  |  |  |
| --- | --- | --- | --- |
| No | Use-case name | Actor | Description |
| 1 | Register | Guest | Registering new account to log into the system |
| 2 | Log In | Guest | Logging into the system |
| 3 | Forgot Password | Guest | Resetting user’s password |

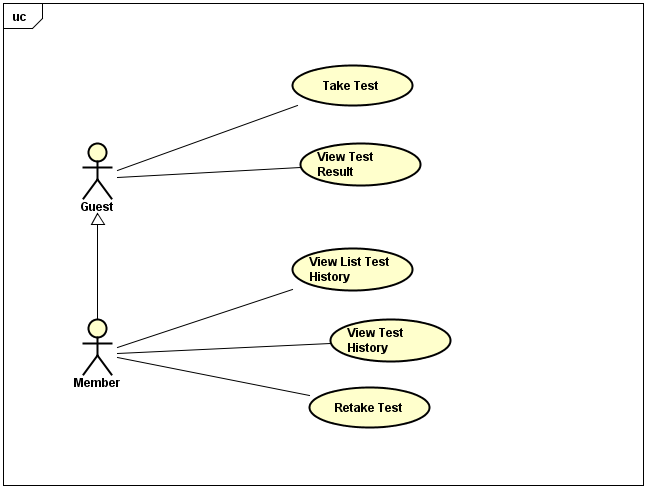
### **Learning**



*Figure 5‑2 Learning*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Lesson List | Guest, Member | Viewing lesson list of courses |
| 2 | Search Lesson | Guest, Member | Searching lesson |
| 3 | View Lesson Details | Guest, Member | Viewing detail of lesson |

### Testing

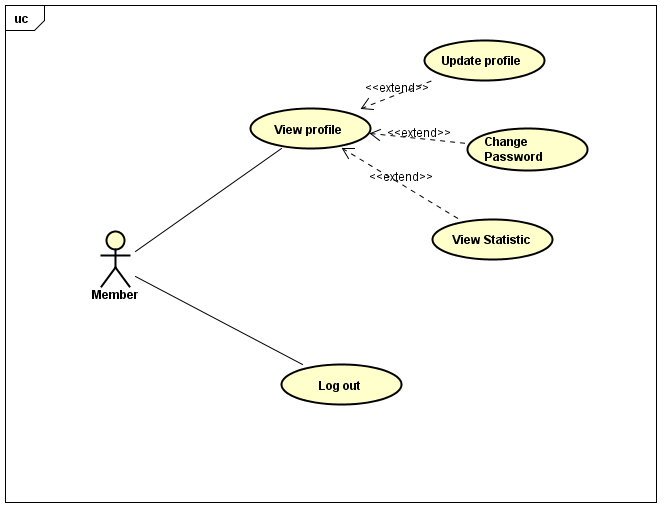


*Figure 5‑3 Tesing*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Take Test | Guest, Member | Taking a multiple-choice test |
| 2 | View Test Result | Guest, Member | Viewing result of test |

## Member

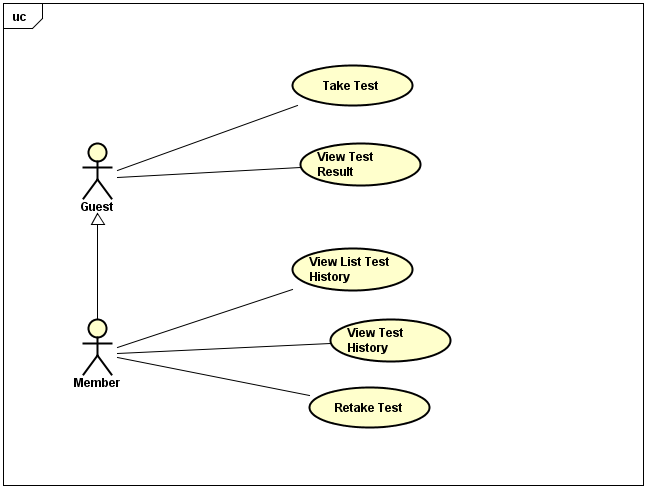
### Private Account Management



*Figure 5‑4 Private Account Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Profile | Member | Viewing detail information of their profile |
| 2 | Update Profile | Member | Updating user’s profile |
| 3 | Change Password | Member | Changing user’s password |
| 4 | View Statistic | Member | Viewing statistics of user |
| 5 | Log Out | Member | Logging out of system |

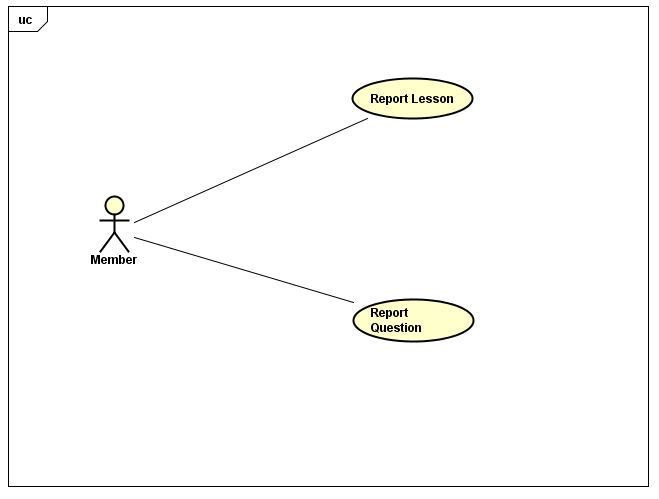
### Testing



*Figure 5‑5 Testing*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View List Test History | Member | Viewing list of test history |
| 2 | View Test History | Member | Viewing detail of test history |
| 3 | Retake Test | Member | Retaking a test in list test history |

### Content Reporting

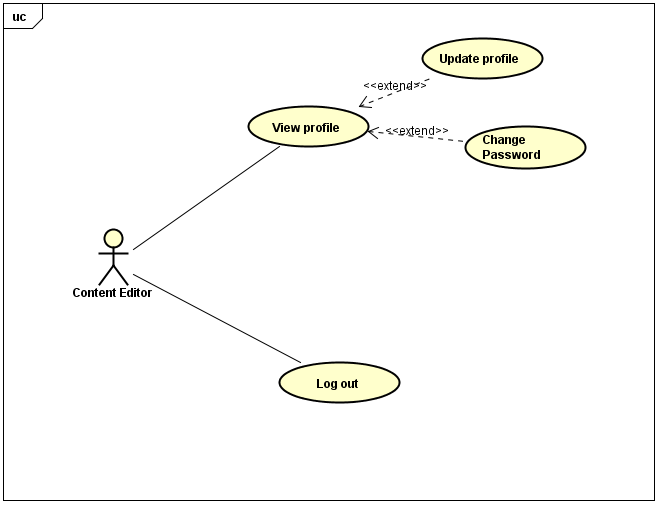


*Figure 5‑6 Content Reporting*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | Report Lesson | Member | Reporting content of lessons that having wrong content |
| 2 | Report Question | Member | Reporting content of questions that having wrong content |

## Editor

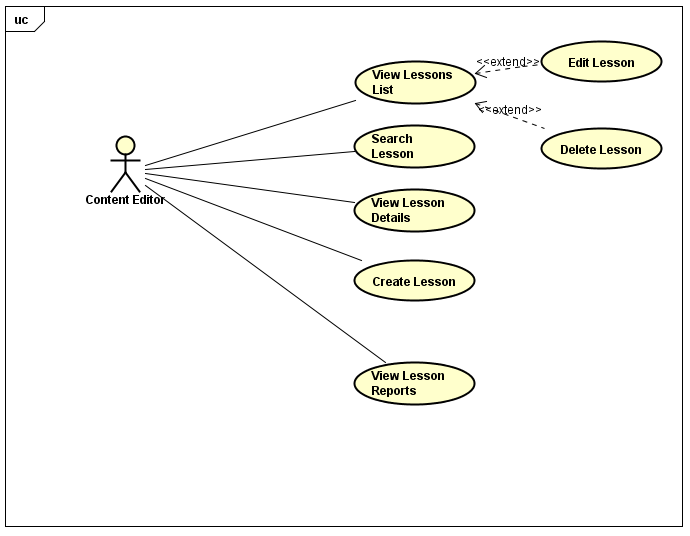
### Private Account Management



*Figure 5‑7 Private Account Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Profile | Content Editor | Viewing detail information of their profile |
| 2 | Update Profile | Content Editor | Updating user’s profile |
| 3 | Change Password | Content Editor | Changing user’s password |
| 4 | Log Out | Content Editor | Logging out of system |

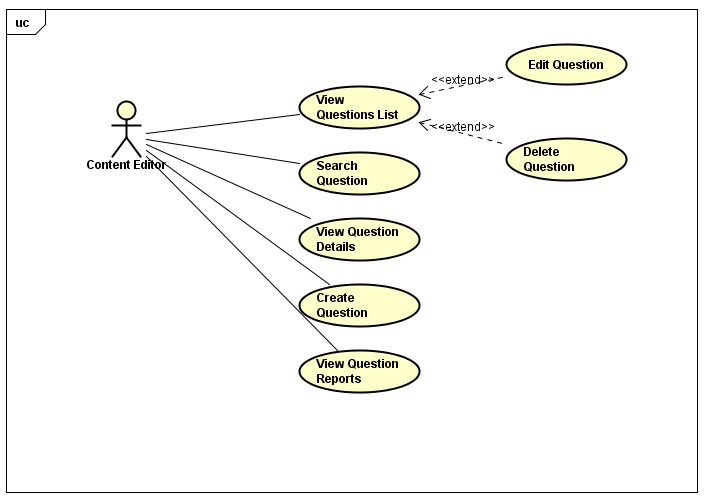
### Lesson Management



*Figure 5‑8 Lesson Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Lessons List | Content Editor | Viewing lesson list of courses |
| 2 | Search Lesson | Content Editor | Searching lesson |
| 3 | View Lesson Details | Content Editor | View detail of lesson |
| 4 | Edit Lesson | Content Editor | Editing content of lesson |
| 5 | Delete Lesson | Content Editor | Deleting lesson |
| 6 | Create Lesson | Content Editor | Creating new lesson of course |
| 7 | View Lesson Reports | Content Editor | Viewing list report about wrong content from member |

### Question Management

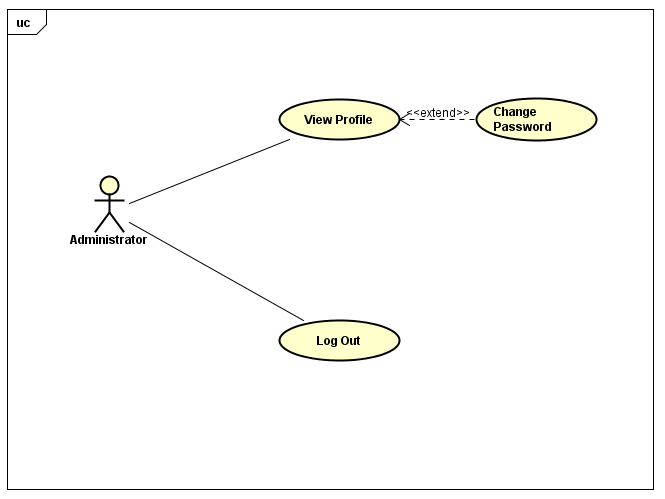


*Figure 5‑9 Question Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Questions List | Content Editor | Viewing question list |
| 2 | Search Question | Content Editor | Searching question |
| 3 | View Question Details | Content Editor | View detail of question |
| 4 | Edit Question | Content Editor | Editing content of question |
| 5 | Delete Question | Content Editor | Deleting question |
| 6 | Create Question | Content Editor | Creating new question of course |
| 7 | View Question Report | Content Editor | Viewing list report about wrong content from member |

## Administrator

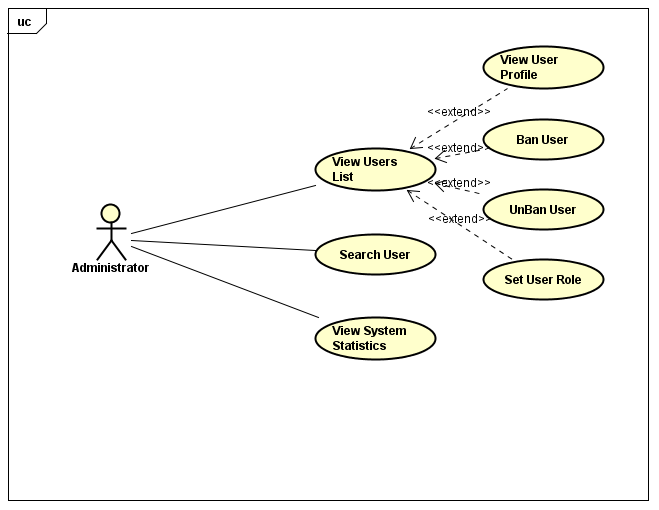
### Private Account Management



*Figure 5‑10 Private Account Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Profile | Administrator | Viewing detail information of their profile |
| 2 | Change Password | Administrator | Changing password |
| 3 | Log Out | Administrator | Logging out of system |

### System Account Management



*Figure 5‑11 System Account Management*

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Use-case name | Actor | Description |
| 1 | View Users List | Administrator | Viewing list of user |
| 2 | Search User | Administrator | Searching user |
| 3 | View User Profile | Administrator | Viewing detail information of user |
| 4 | Ban User | Administrator | Blocking user of system |
| 5 | Unban User | Administrator | Unblocking user |
| 6 | Set User Role | Administrator | Setting role for user |
| 7 | View System Statistics | Administrator | Viewing statistics of system |

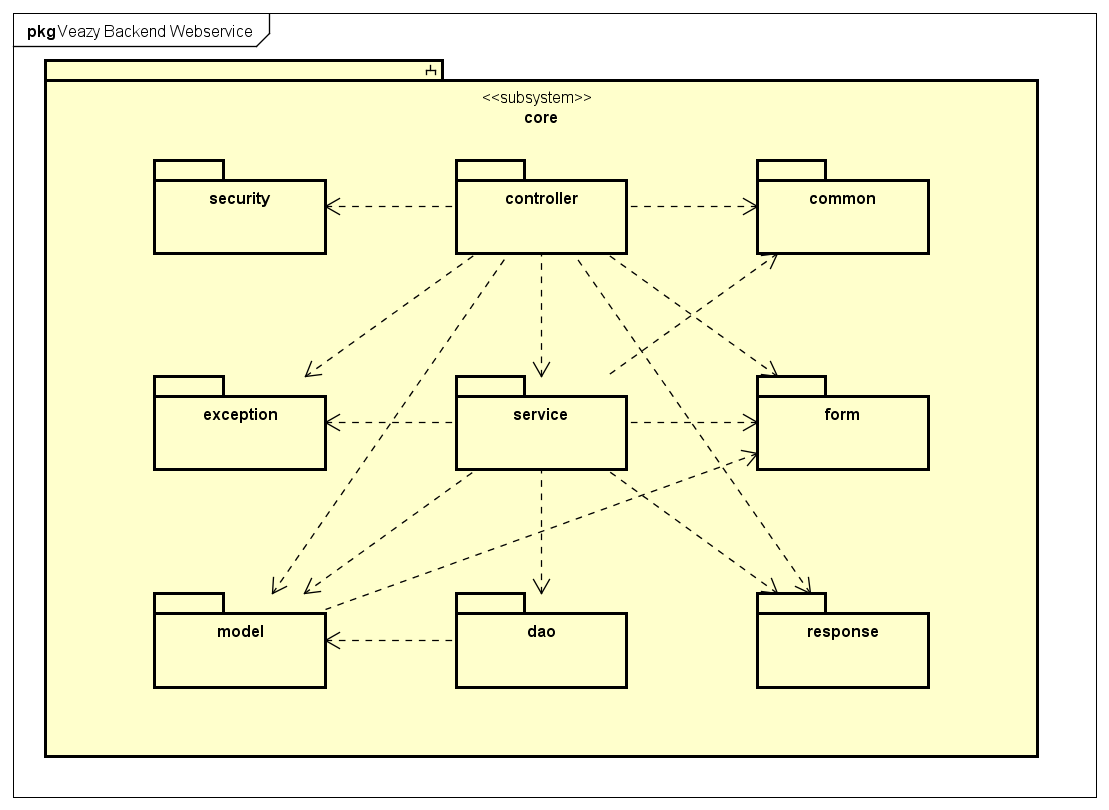
# Logical View

## Overview

Logical View includes Package diagram and Class diagram. Package diagram describes the organization of packages and elements. Class Diagram provides an overview of the target system by describing the objects and classes inside the system and the relationships between them. It provides a wide variety of usages; from modeling the domain-specific data structure to detailed design of the target system.

* Controller contain the interface between
* Associated models
* Associated views
* The input devices (e.g., keyboard, pointing device, time).
* Send commands to the model to update the model's state.
* Model is:
  + The domain-specific software simulation
  + Or implementation of the application's central structure.
* View deal with everything graphical
* Requests data from their model
* Display the data

## Architecturally Significant Design Packages



*Figure 6‑1 Package diagram*

* **Controller**

|  |  |  |
| --- | --- | --- |
| No. | Controller class | Role |
| 1 | CoreController | Processes common core api requests |
| 2 | CourseController | Processes api requests related to course |
| 3 | TestController | Processes api requests related to test |
| 4 | LessonController | Processes api requests related to lesson |
| 5 | QuestionController | Processes api requests related to question |
| 6 | ReportController | Processes api requests related to report |
| 7 | StatsController | Processes api requests related to statistic |
| 8 | UserController | Processes api requests related to user |

* **Service**

|  |  |  |
| --- | --- | --- |
| No. | Service class | Role |
| 1 | CourseServiceImpl | Contains business logical functions for processing course data |
| 2 | TestServiceImpl | Contains business logical functions for processing test data |
| 3 | LessonServiceImpl | Contains business logical functions for processing lesson data |
| 4 | QuestionBankServiceImpl | Contains business logical functions for processing question bank data |
| 5 | QuestionServiceImpl | Contains business logical functions for processing question data |
| 6 | ReportServiceImpl | Contains business logical functions for processing report data |
| 7 | StatsServiceImpl | Contains business logical functions for processing statistic data |
| 8 | UserServiceImpl | Contains business logical functions for processing user data |

* **Model**

|  |  |  |
| --- | --- | --- |
| No. | Model class | Role |
| 1 | AnswerModel | Stores data of a single answer of a question |
| 2 | BasicModel | Contains common fields of model |
| 3 | CourseModel | Stores data of a course (as known as a level) |
| 4 | TestAnswerModel | Stores data of a single answer of a question in a generated test |
| 5 | TestModel | Stores data of a test |
| 6 | TestQuestionModel | Stores data of a question in a generated test |
| 7 | LessonModel | Stores data of a lesson |
| 8 | LessonVersionModel | Stores data of a version of lesson |
| 9 | QuestionModel | Stores data of a question in the questionbank |
| 10 | ReportModel | Stores data of a report |
| 11 | UserModel | Stores data of an user |

* **DAO**

|  |  |  |
| --- | --- | --- |
| No. | DAO class | Role |
| 1 | HibernateCourseDao | Provides methods to manage course data in database |
| 2 | HibernateTestDao | Provides methods to manage test data in database |
| 3 | HibernateLessonDao | Provides methods to manage lesson data in database |
| 4 | HibernateLessonVersionDao | Provides methods to manage versions of lesson in database |
| 5 | HibernateQuestionDao | Provides methods to manage question data in database |
| 6 | HibernateReportDao | Provides methods to manage report data in database |
| 7 | HibernateUserDao | Provides methods to manage user data in database |
| 8 | HibernateAnswerDao | Provides methods to manage answers data in a question in database |

* **Exception**

|  |  |  |
| --- | --- | --- |
| No. | Exception class | Role |
| 1 | CorruptedFormException | Exception throwed when user submitted a corrupted form i.e. form that does not have enough expected data |
| 2 | EmailExpectedException | Exception throwed when submitted form does not contain email |
| 3 | InvalidEmailException | Exception throwed when submitted form contains invalid email |
| 4 | PasswordExpectedException | Exception throwed when submitted form does not contain password |
| 5 | PasswordIncorrectException | Exception throwed when submitted form contains incorrect username or password |
| 6 | UsernameExpectedException | Exception throwed when submitted form does not contain username |
| 7 | NullOrEmptyContentException | Exception throwed when submitted data is null or empty |

* **Form**

|  |  |  |
| --- | --- | --- |
| No. | Form class | Role |
| 1 | AnswerForm | Answer form |
| 2 | ChangeRoleForm | Form for requesting change user role |
| 3 | ChgpwdForm | Form for requesting change password |
| 4 | CreateTestSinglePartForm | Form for generating a single-skill test |
| 5 | CreateLessonForm | Form for creating a test |
| 6 | TestPartForm | Form for generating a test |
| 7 | FileUploadForm | Upload file form |
| 8 | LoginForm | Login form |
| 9 | QuestionForm | Question form |
| 10 | RegisterForm | Register form |
| 11 | ReportForm | Report form |
| 12 | SubmitTestForm | Test form for submittion |
| 13 | SubmitQuestionForm | Question form for test submittion |
| 14 | SubmitAnswerForm | Answer of a question form for test submittion |
| 15 | UpdateLessonForm | Lesson information form for updating a lesson |
| 16 | UpdateUserForm | User information form for updating an user |

* **Response**

|  |  |  |
| --- | --- | --- |
| No. | Response class | Role |
| 1 | AddQuestionResponse | Response for adding a new question api |
| 2 | AnswerResponse | Response for an answer in a question |
| 3 | BriefAnswerResponse | Response for a brief answer in a brief question in a generated test |
| 4 | BriefLessonResponse | Response for a brief lesson |
| 5 | BriefQuestionResponse | Response for a brief question in a generated test |
| 6 | CreateLessonResponse | Response for creating a lesson api |
| 7 | TestAnswerResponse | Response for an answer of a question in a generated test |
| 8 | TestPartResponse | Response for a generated test |
| 9 | TestQuestionResponse | Response for a question in a generated test |
| 10 | TestResultResponse | Response for test result |
| 11 | TestSinglePartResponse | Response for generating a single-skill test api |
| 12 | GetCourseResponse | Response for getting list courses api |
| 13 | GetTestResponse | Response for getting a taken test api |
| 14 | GetLearnerTestsResponse | Response for getting learner tests api |
| 15 | GetLessonResponse | Response for getting a lesson api |
| 16 | GetLessonVersionResponse | Response for getting a lesson version api |
| 17 | GetListUsersResponse | Response for getting list users api |
| 18 | GetUserResponse | Response for getting a user api |
| 19 | LoginResponse | Response for login api |
| 20 | QuestionResponse | Response for a question |
| 21 | Response | Response form for all api |
| 22 | ResponseCode | Return code for each api |
| 23 | StatsCourseAvgResponse | Response for getting average result of tests via course statistic api |
| 24 | StatsLastTestResponse | Response for getting last test statistic api |
| 25 | StatsLessonsResponse | Response for getting lesson statistic api |
| 26 | StatsQuestionsResponse | Response for getting questions statistic api |
| 27 | StatsSkillAvgResponse | Response for getting average result of tests via skill statistic api |
| 28 | StatsUsersResponse | Response for getting users statistic api |
| 29 | UploadFileResponse | Response for uploading api |

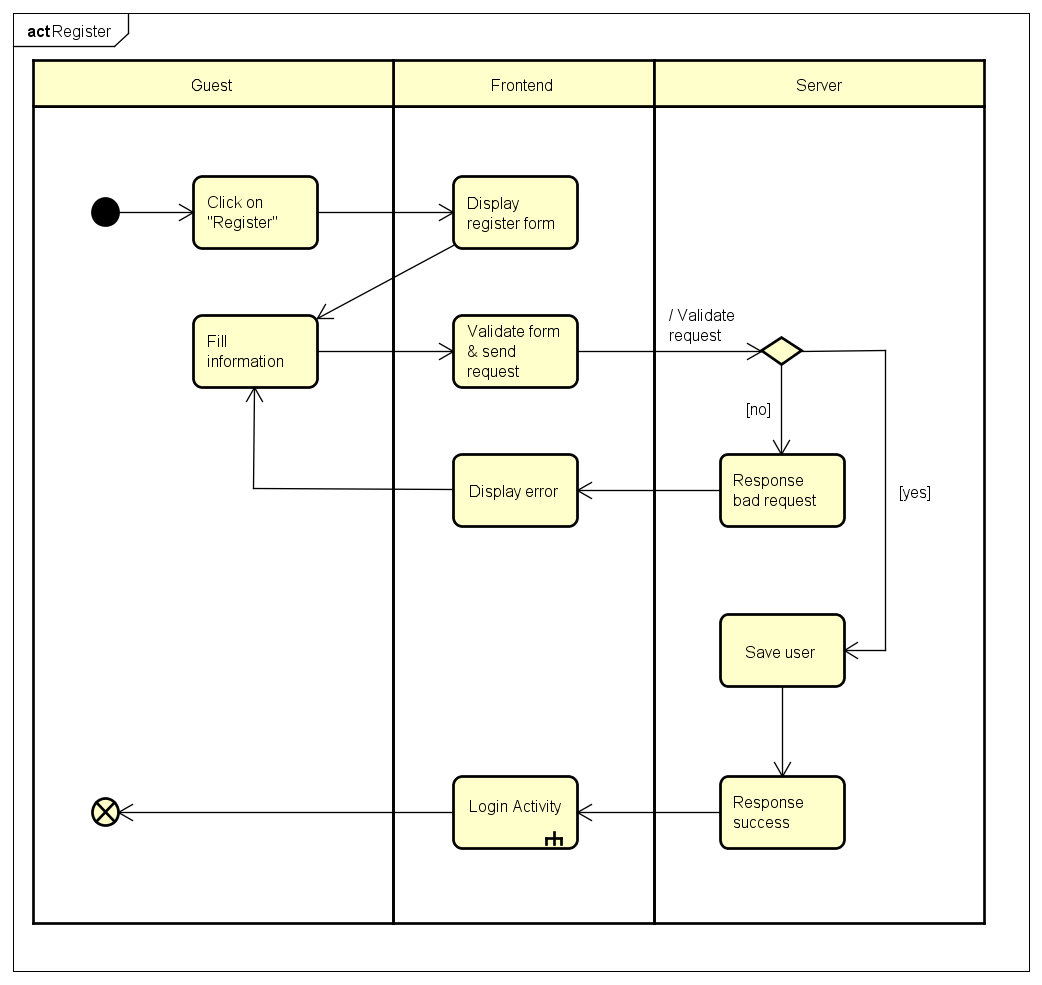
* **Security**

|  |  |  |
| --- | --- | --- |
| No. | Security class | Role |
| 1 | Veazy403Handler | Handles 403 forbidden error |
| 2 | VeazyAuthenEntryPoint | Handles anonymous requests |
| 3 | VeazyAuthenFailureHandler | Handles authentication failure requests |
| 4 | VeazyAuthenFilter | Filters authentication requests |
| 5 | VeazyAuthenProvider | Provides authentication mechanism |
| 6 | VeazyAuthenSuccessHandler | Handles authentication success requests |

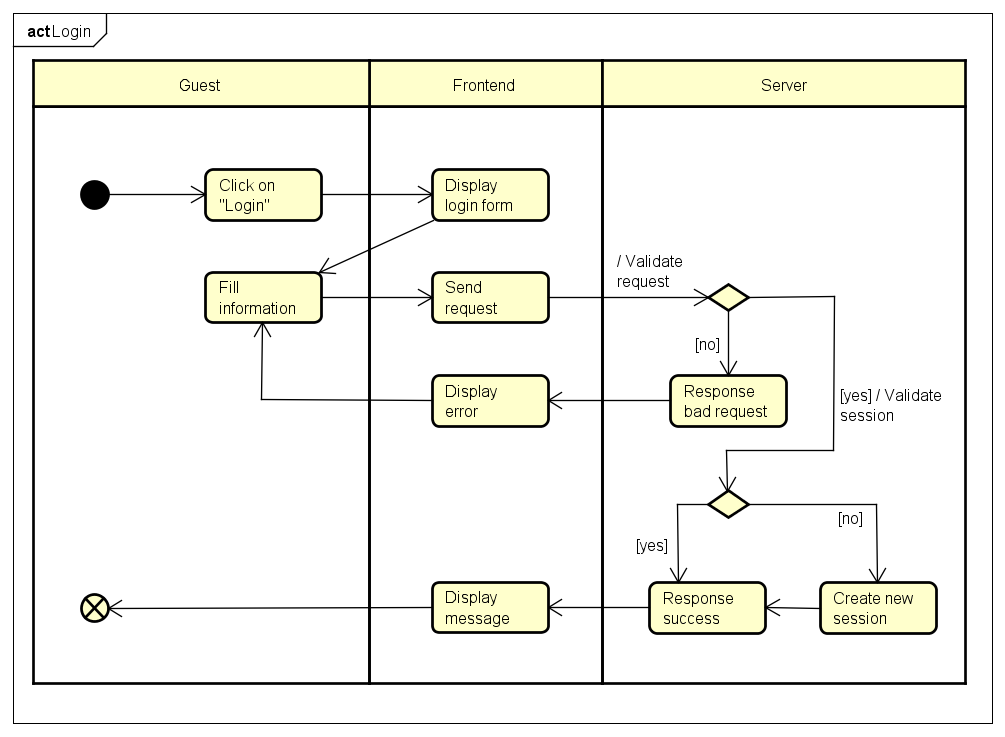
* **Common**

|  |  |  |
| --- | --- | --- |
| No. | Common class | Role |
| 1 | Const | Contains common system constants |
| 2 | Utils | Contains common utility functions |
| 3 | JsonUtils | Contains json utility functions |
| 4 | HttpUtils | Contains http utility functions |
| 5 | HtmlUtils | Contains html utility functions |

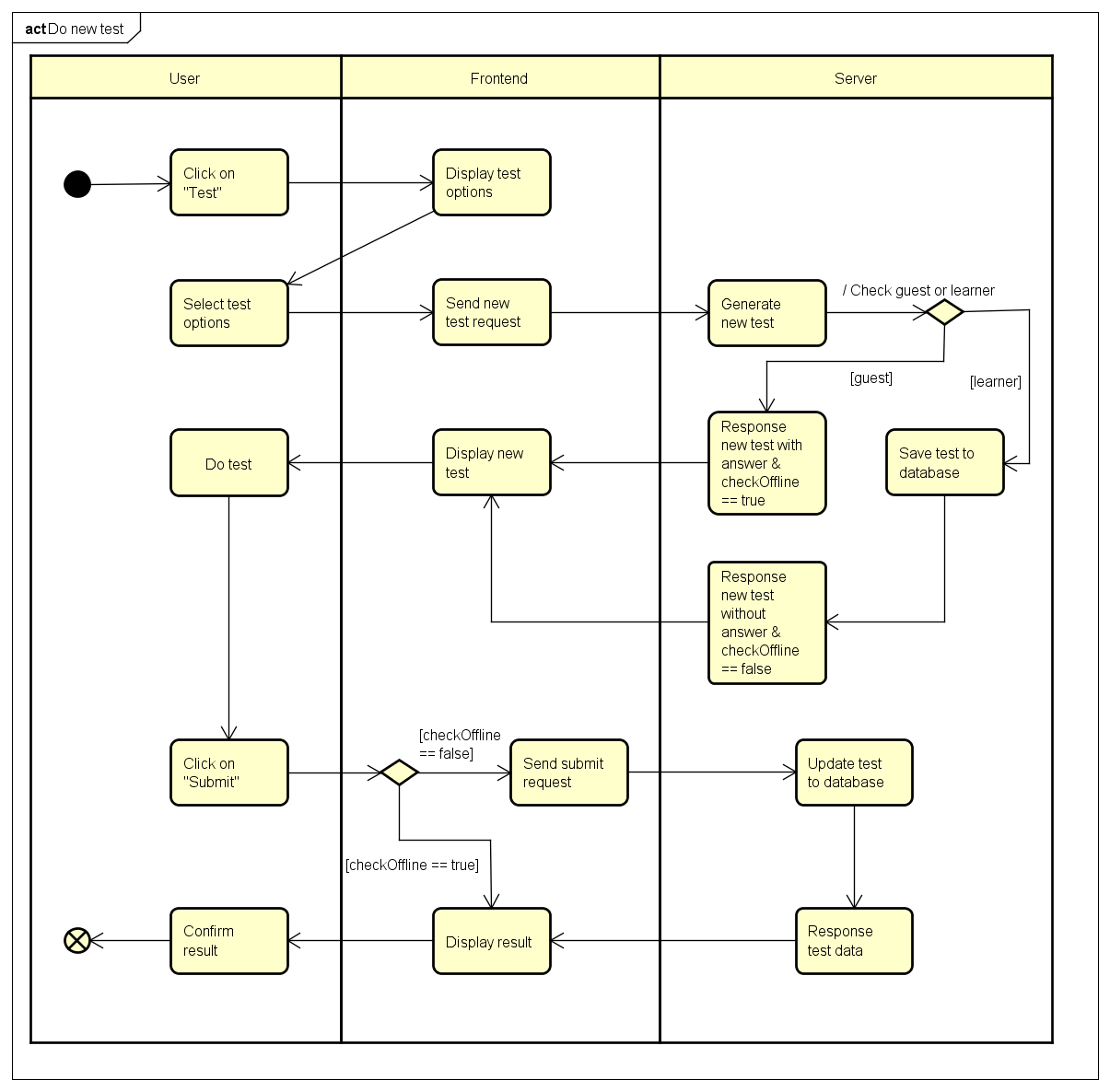
# Process View



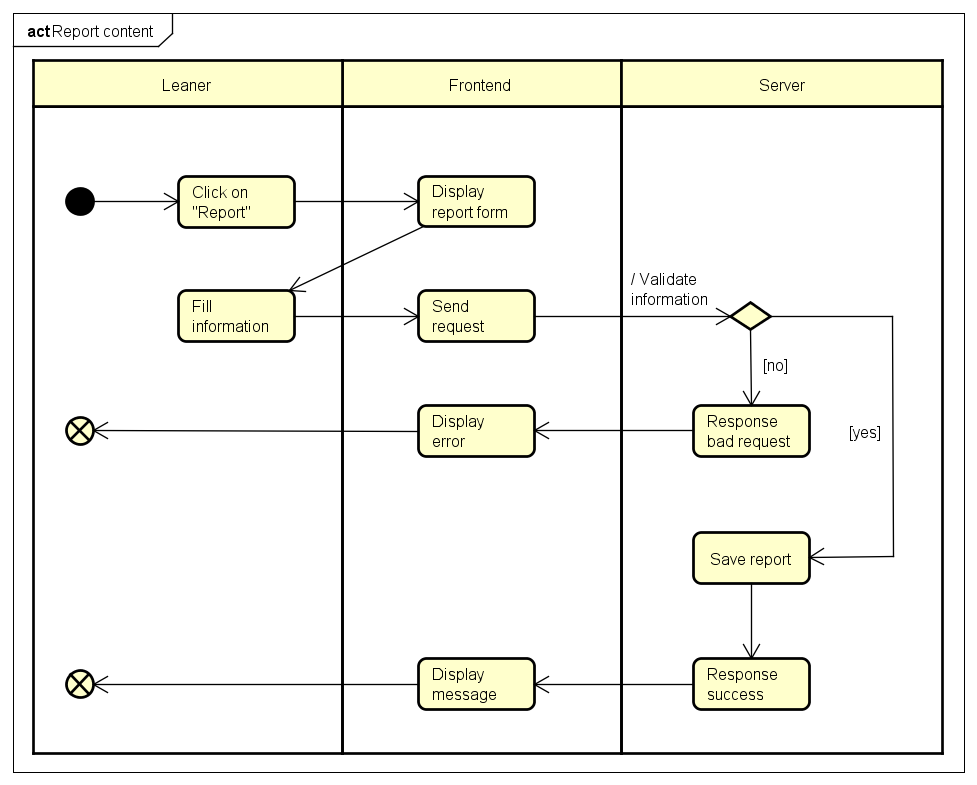
*Figure 7‑1 Register activity*



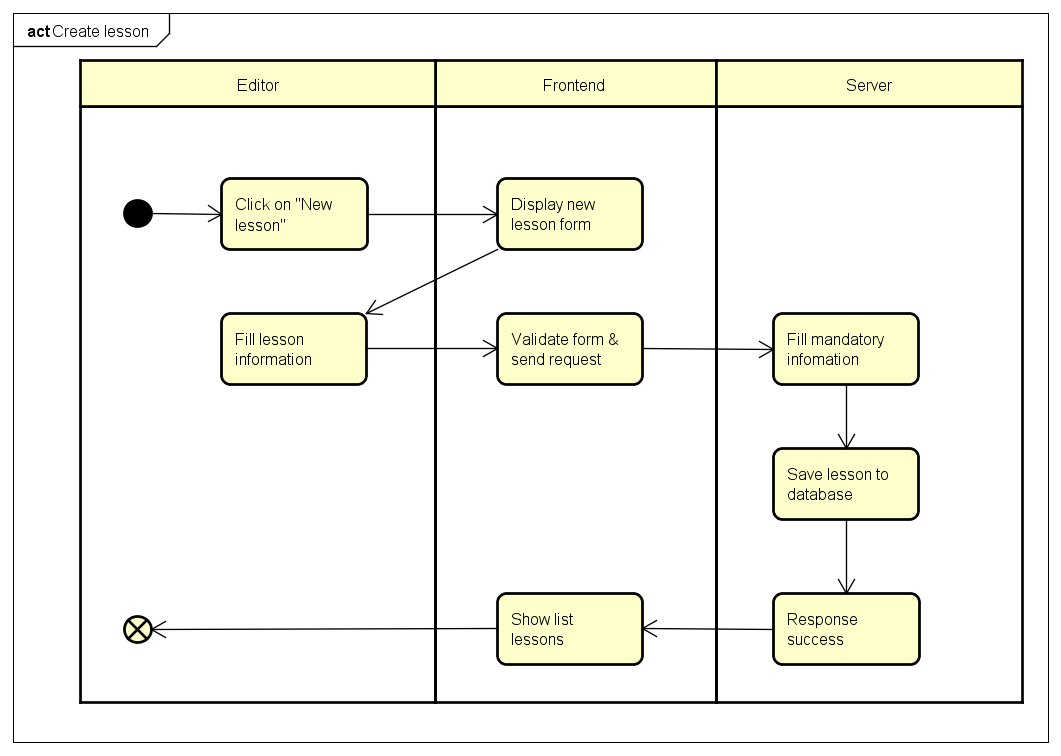
*Figure 7‑2 Login activity*



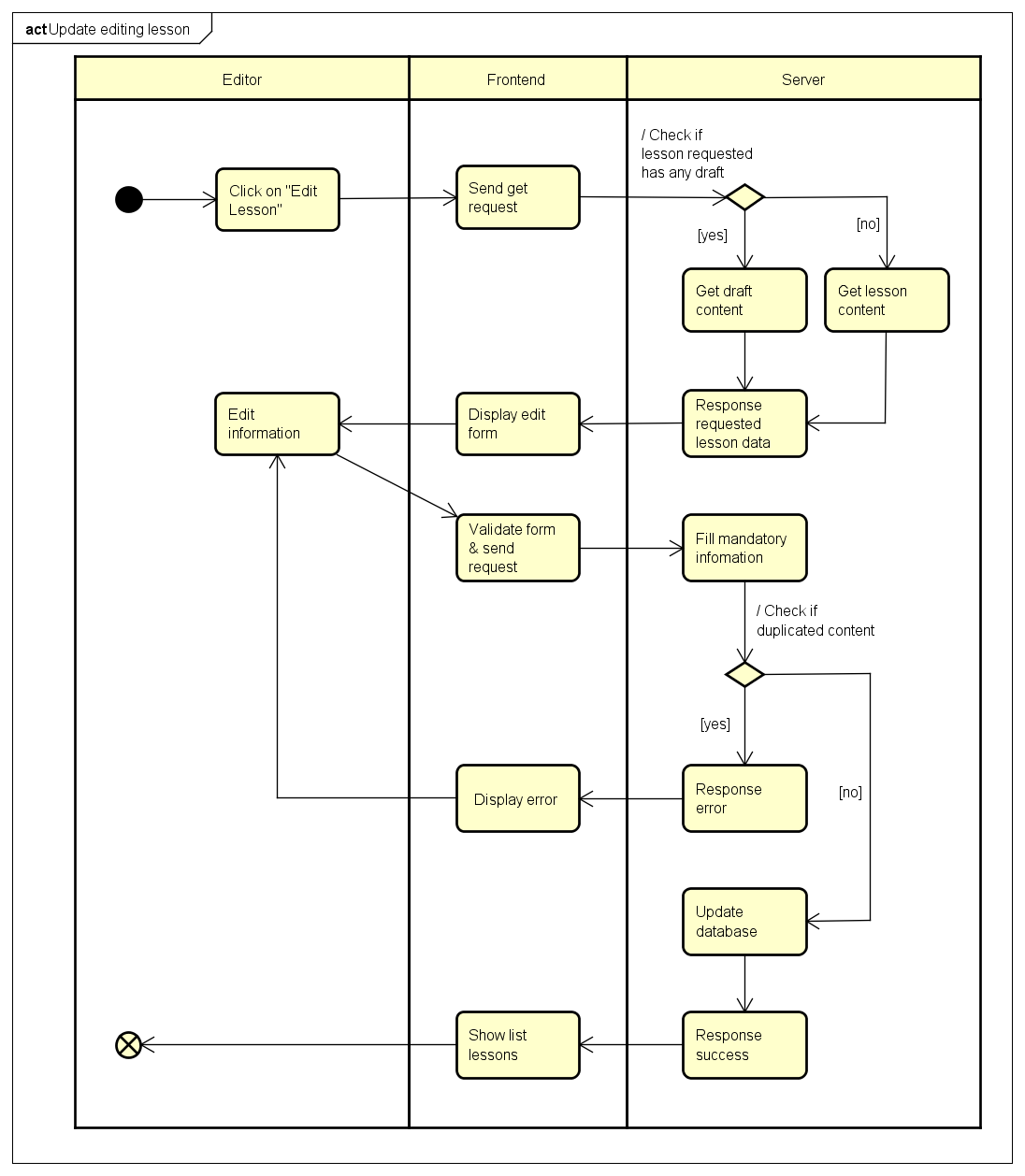
*Figure 7‑3 Do test activity*



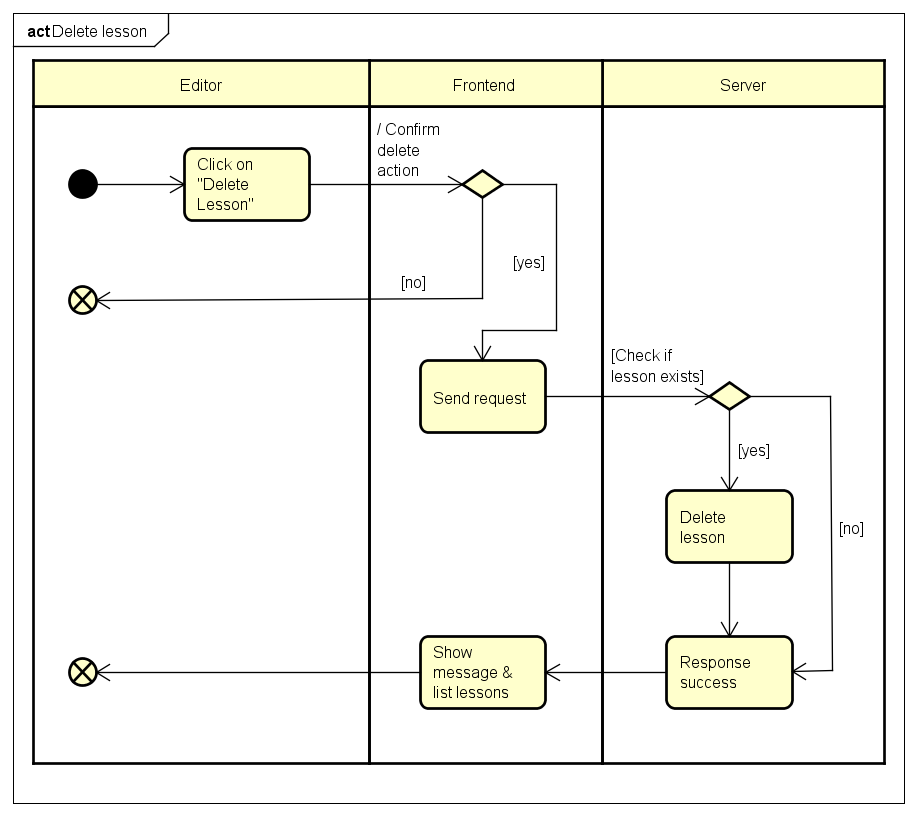
*Figure 7‑4 Report content activity*



*Figure 7‑5 Create lesson activity*



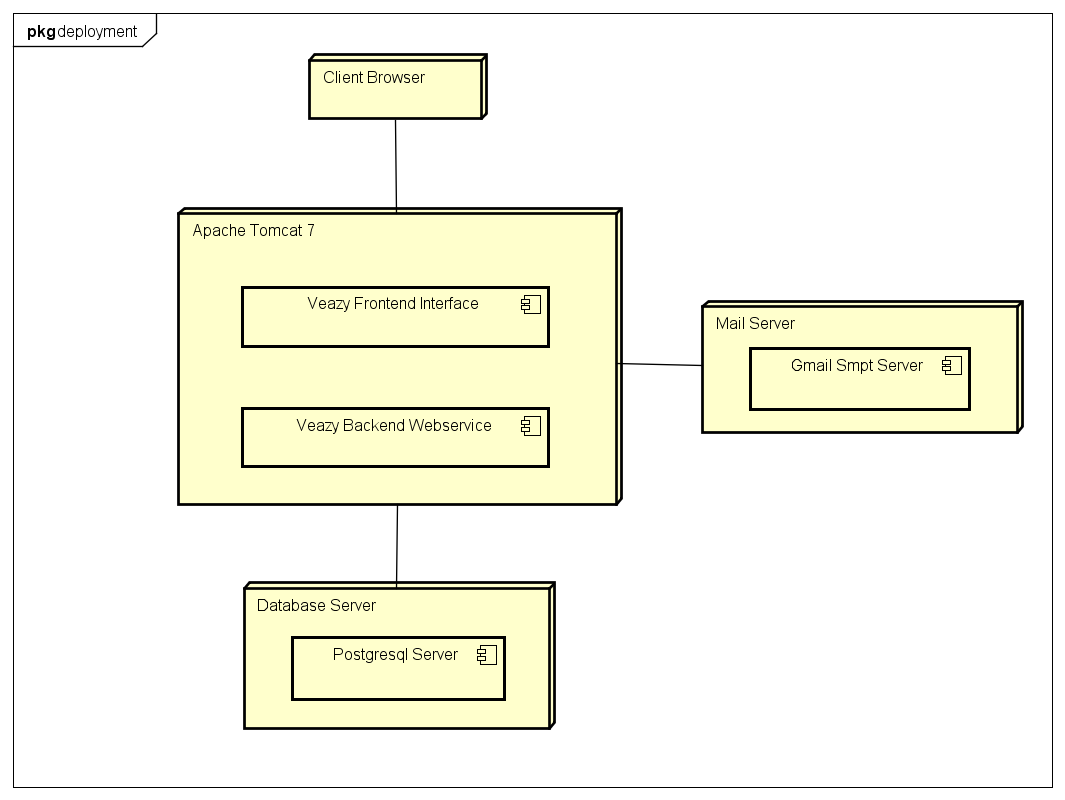
*Figure 7‑6 Update lesson activity*



*Figure 7‑7 Delete lesson activity*

# Deployment view

Deployment view of Veazy system



*Figure 8‑1 Deployment diagram*

|  |  |  |
| --- | --- | --- |
| No. | Name | Description |
| 1 | Postgresql Server | Postgresql server to persist data. Postgresql 9.4 or higher |
| 2 | Mail Server | Mail server to send email to user |
| 3 | Web Server | Web server hosting system |
| 4 | Client Browser | Web browser to display system. Firefox 20, Chrome 36 or higher. |

# Quality

Reference to: “*Veazy\_SRS\_v1.2\_EN.docx*”