Software Design Specification
(AWP)

Ali ÖZTÜRK , Okan EKE , Muhammed Ali DOĞAN

Software Design Specification

1. Introduction

This document consists design specifications of Allaamus Web Portal.

1.1) Purpose

AWP (Allaamus Web Portal) is a bridge between users and experts. AWP will provide on-line appointments and question asking mechanism to users in a domain such as medicine, psychology, programming etc.

1.2) Scope

AWP will provide text chat, voice call and video call for appointments. Users will have chance to ask questions to experts for their problems or researches.

1.3) Definitions, Acronyms and Abbreviation

AWP: Allaamus Web Portal
Vuex.js: Vuex is a library that helps you implement the Flux architecture in your Vue application. By enforcing the principles described above, Vuex keeps your application data in a transparent and predictable state even when that data is being shared across multiple components.
Docker: Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.
API: the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an application like Facebook, send an instant message, or check the weather on your phone, you're using an API.
Node.js: A platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
CRUD: In computer programming, **create, read, update, and delete** (**CRUD**) are the four basic functions of persistent storage.
Falcon: A bare-metal **Python** web API framework for building high-performance micro services, application back-ends, and higher-level frameworks.
Flask: Python based web framework.

1.4) References

* https://opensource.com/resources/what-docker
* https://medium.com/js-dojo/vuex-for-the-clueless-the-missing-primer-on-vues-application-data-store-33fa51ffc3af
* https://www.mulesoft.com/resources/api/what-is-an-api

* https://nodejs.org/en/docs/
* https://en.wikipedia.org/wiki/Create,_read,_update_and_delete
* https://github.com/falconry/falcon

2. Design Consideration

This section will provide preliminary information about AWP.

2.1) Assumptions

We assume that user will have fundamental knowledge of usage of browser. We will provide portal only as web application. Additionally, application is compatible with all browser. The system is not accessible unless there is a Wi-Fi or Ethernet connection.

2.2) Considerations

The system is implemented using JavaScript, HTML, CSS. Any browser is needed to use the application.

2.3) System Environment

Application will be developed on Linux (Operating System) while using VS Code. Each user will need a mobile device, computer or tablet with any browser installed.

3. System Design

This section represents the meat of your document. Be as detailed as time allows.

3.1) Architectural System Design

In this project Vuex.js will be used for front-end side. For back-end AWP will have micro-service architecture by using docker containers. There will be 5 services as Controller, Database API, Machine Learning, Streaming, Chat. In this containers will some frameworks such as Node.js, flask, falcon.

3.1.1. Controller

This container will manage other containers or services.

3.1.2. Database API

This container will be a restful API for database CRUD operations.

3.1.3. Machine Learning

This container will have service for search engine of portal.

3.1.4. Streaming

This container manages voice and video streaming operations.

3.1.5. Chat

This container will manage live text chat.

3.2) Class Diagrams

3.2.1.User

This class contains generic information about users.

3.2.2. Expert

Inherited from User class and contains additional parts for Experts

3.2.3. Inquirer

Inherited from User class and contains additional parts for Inquirer

3.2.4. Question

Information about questions. Both Experts and Inquirers has questions.

3.2.5. Appointment

This class contains information about appointments and has two users.

3.2.6. Wallet

This class contains money information and users has one wallet.

3.2.7. CV

This class contains information about experts and experts has one CV.

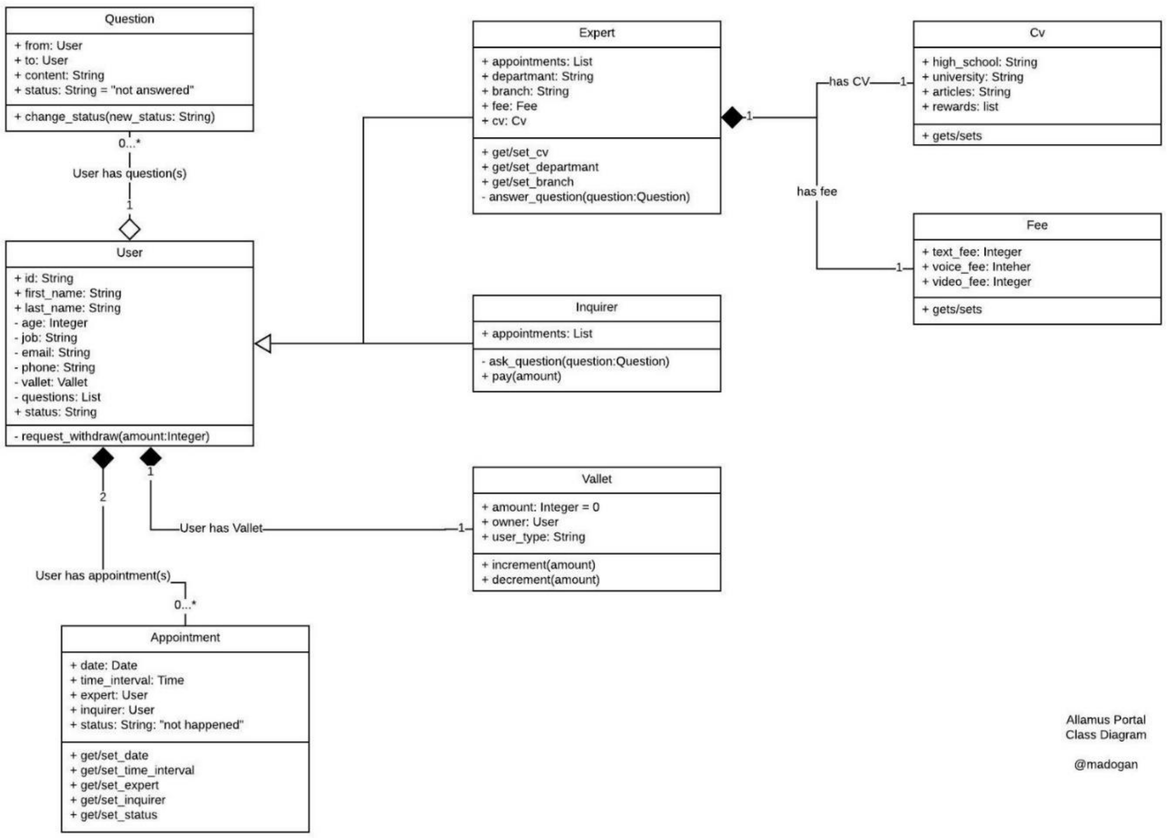
3.2.8. Fee

This class contains information about fee of experts for text, voice and video appointments. Experts has one fee object.

4. Conclusion

As a result, we will design a system that enables users to achieve information in a direct manner from experts. For this we will use machine learning for search engine, and web frame works such as Node.js, Flask and Falcon for question asking and live appointment mechanisms.

5. **Class Diagram**


Question
+ from: User
+ to: User
+ content: String
+ status: String = "not answered"
+ change_status(new_status: String)
0...*
User has question(s)
1
User
+ id: String
+ first_name: String
+ last_name: String
- age: Integer
- job: String
- email: String
- phone: String
- vallet: Vallet
- questions: List
+ status: String
- request_withdraw(amount:Integer)
2
1
User has Vallet
1
User has appointment(s)
0...*
Appointment
+ date: Date
+ time_interval: Time
+ expert: User
+ inquirer: User
+ status: String: "not happened"
+ get/set_date
+ get/set_time_interval
+ get/set_expert
+ get/set_inquirer
+ get/set_status
Expert
+ appointments: List
+ departmant: String
+ branch: String
+ fee: Fee
+ cv: Cv
+ get/set_cv
+ get/set_departmant
+ get/set_branch
- answer_question(question:Question)
1
has CV
1
has fee
1
Cv
+ high_school: String
+ university: String
+ articles: String
+ rewards: list
+ gets/sets
Fee
+ text_fee: Integer
+ voice_fee: Inteher
+ video_fee: Integer
+ gets/sets
Inquirer
+ appointments: List
- ask_question(question:Question)
+ pay(amount)
Vallet
+ amount: Integer = 0
+ owner: User
+ user_type: String
+ increment(amount)
+ decrement(amount)
Allamus Portal
Class Diagram
@madogan