High level requirement analysis

This system is built to enhance customer support by allowing customer to create tickets and facilitating the communication flow between customers and support agents. This also means that the system must present a nice view to support agents, where they can filter tickets, manipulate them, submit responses and/or questions to users, modifying tickets status, and generating a downloadable tickets report file. The system must have a pleasant design as well, and for that I chose a minimalist design using mostly pale colors and white.

Users

There are three types of users operating the system:

- Customers: users who are able to create new tickets and view status of previous support requests.
- Support Agents: users who are able to process tickets, i.e., change its status and respond it. Also, they must be able to export a PDF report containing all tickets closed in the last month.
- Administrator: user who is able to manage other users, like changing their data or removing them from the system.

Users must be able to register as customer user, login, logout, update, and delete their own account; updating also includes password redefinition. Support users must be created by administrators, and administrators must be created by other administrators or be seeded on database.

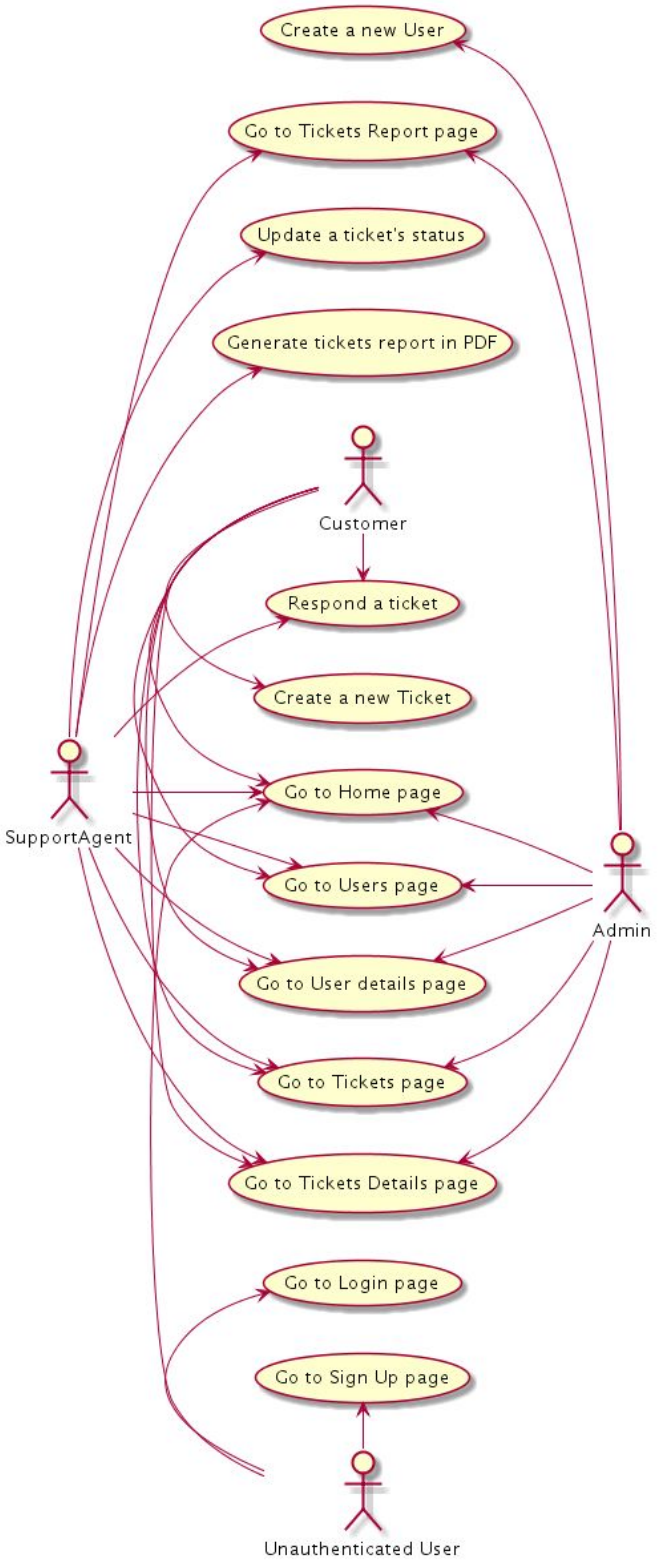
Tickets

Tickets are created by customer, can be updated by the opening customer, and any support agent or administrator. Thus, the tickets are associated with only one customer user, but any customer user must be able to have multiple open tickets at the same time.

Updating the ticket means to comment on it, changing its status, or closing it. Once closed it cannot be reopened. The closing ticket operation also change its status to *closed*. There are two possible status: *open* and *closed*.

Authentication

All users must be able to authenticate to the system. The chosen authentication method to the system is JWT, which allows stateless requests, thus improving the service scalability when compared to other authentication methods.


Create a new User
Go to Tickets Report page
Update a ticket's status
Generate tickets report in PDF
Customer
Respond a ticket
Create a new Ticket
SupportAgent
Go to Home page
Go to Users page
Admin
Go to User details page
Go to Tickets page
Go to Tickets Details page
Go to Login page
Go to Sign Up page
Unauthenticated User

Frontend technology

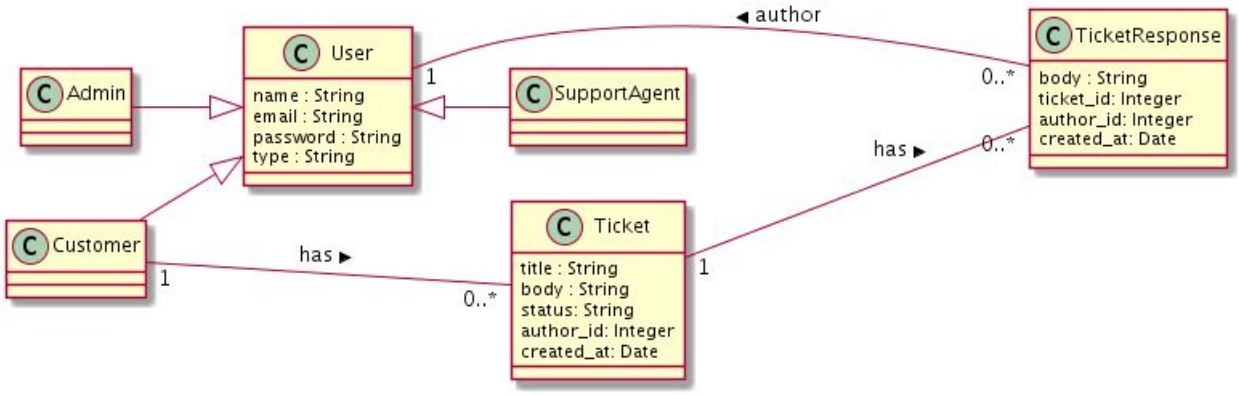
Vue.js is being a rising star in the JavaScript world, being the fastest growing JavaScript repository on GitHub in 2016. This was achievable because Vue.js is a reliable and simple technology which elegantly, and developer friendly, tackles the big challenges we have in frontend development nowadays

Backend technology

The system's backend is an REST API implemented in Ruby on Rails 5.0.1 and powered by MySQL database.

High level presentation of the data model

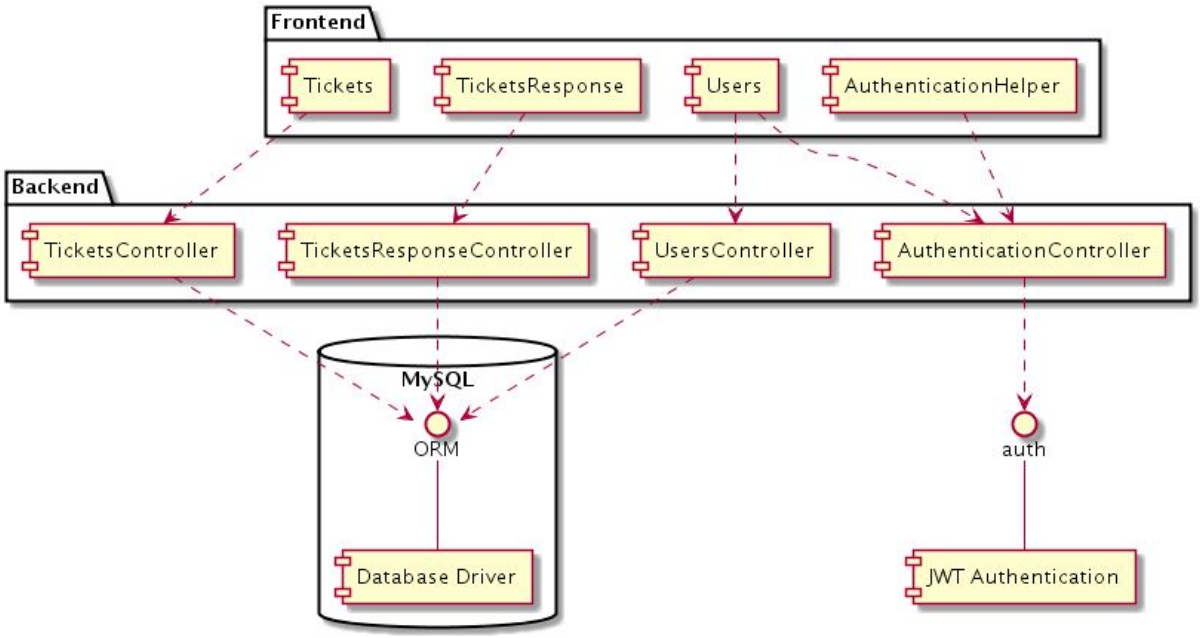
Our data model has six entities as shown below:



We naturally have three types of users: Customer, SupportAgent, and Admin. They all inherits from the User entity, which has a name, email, and password. Also we have the Ticket entity and TicketResponse entity, such that Ticket has an one-to-many possession association with TicketResponse, even though TicketResponse has many-to-one authorship associations with Users. Also, note that only Customer has a one-to-many possession association wich Ticket.

Architecture

As I said in the first section of this document, this customer support system is divided in two services: a frontend and a backend, which we can refer to two packages of specialized components, which along with a Database system and an authentication technology we compose our entire system architecture as demonstrated in the diagram below:



Here, our Frontend components are Vue's view components bult with JavaScript, CSS, and HTML5; while our Backend components are Ruby on Rails' controllers powered by helper modules, Rails components (such as ActiveRecord) and MySQL.

The communication between Frontend and Backend components happens in a HTTP manner, such that all communication always are triggered by Frontend JavaScript code. Basically all the actions are authenticated with exception for the authentication actions, i.e., login and user register.