# 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.

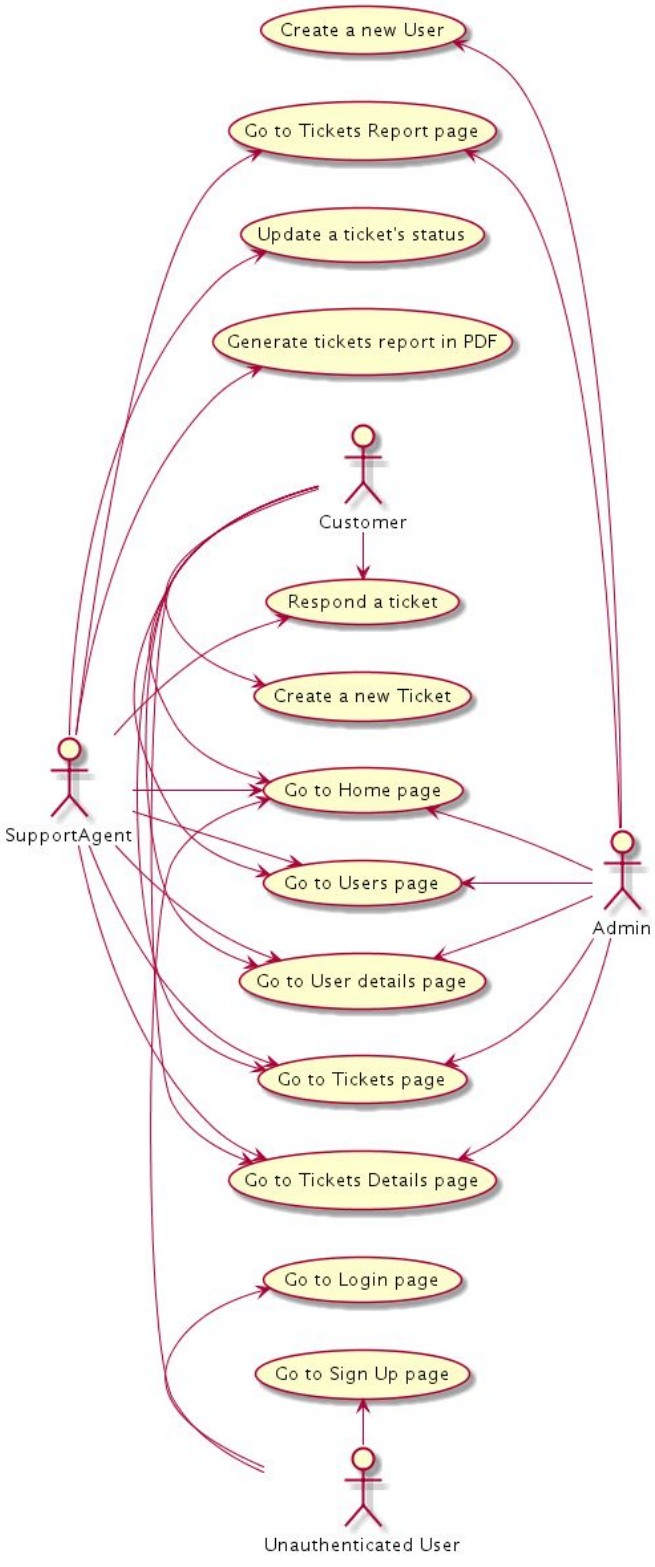
## Technologies

The system was required to be powered by Ruby on Rails and MySQL, as well as some JavaScript frontend framework of my choice. I have chosen Vue.js.
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

I have also used the most recent version of Ruby on Rails, 5.0.1, as an REST API which provides data and handles the authentication and provisioning of the system data through a nice

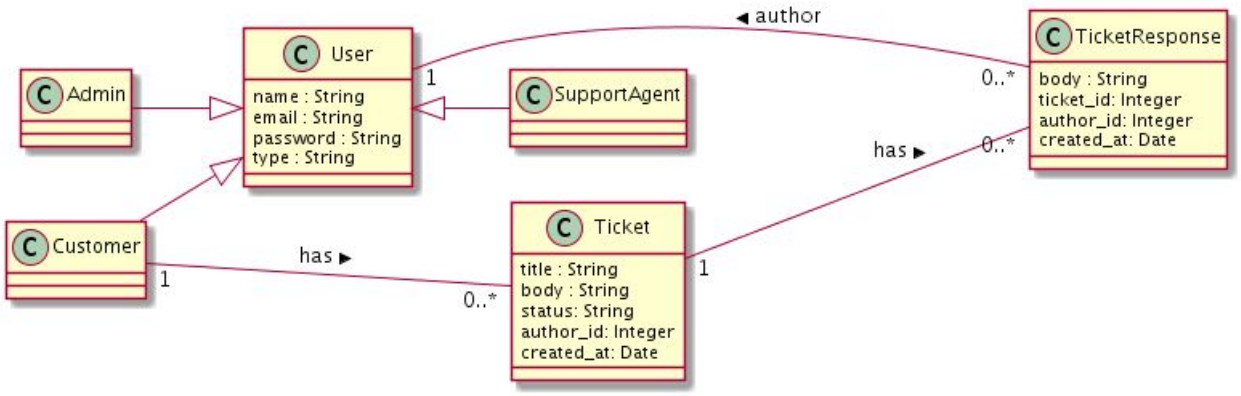
REST API. Underneath the Ruby on Rails layer we have a MySQL database which handles our data persistency.

For generating our UML diagrams I have chosen PlantUML, which does a really nice job for most of diagrams. Below we can se the use cases diagram



# High level presentation of the data model

Our data model has six entities as shown below:

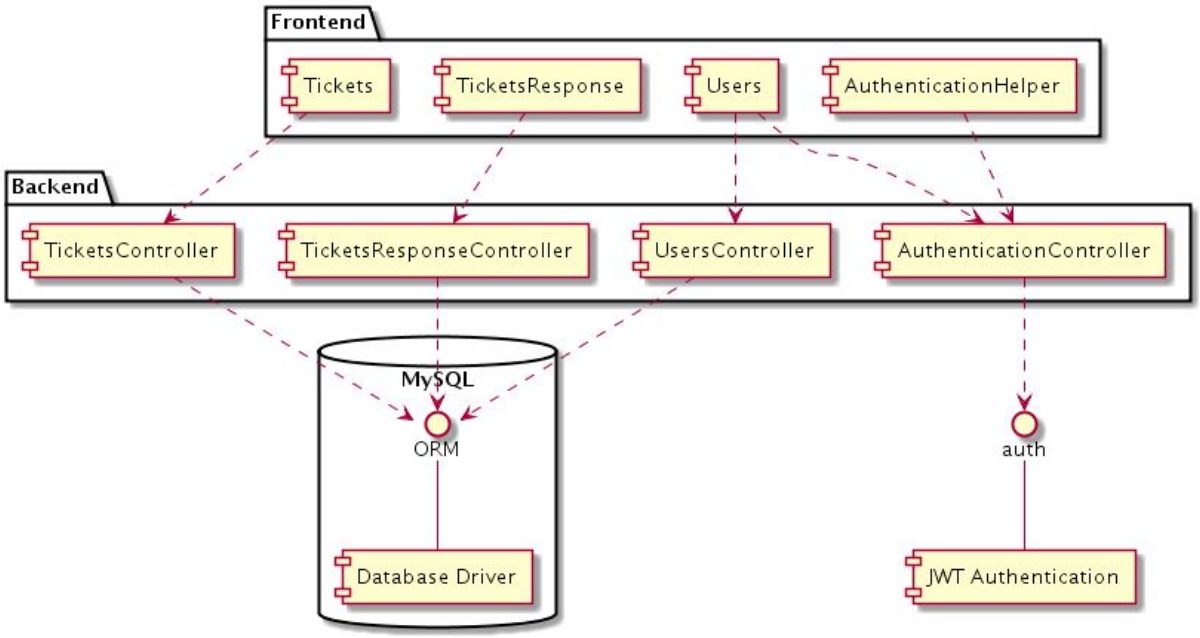



We naturally have three types of users: Customer, SupportAgent, and Admin. They all inherits from the User entity, which has a name, email, and password. Also we have the Ticket entity and TicketResponse entity, such that Ticket has an one-to-many possession association with TicketResponse, even though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