Design document

Federico M. Iachetti

@:t ::t |:t ^:nil -:t f:t *:t <:t

TeX:t LaTeX:t skip:nil d:nil todo:t pri:nil tags:not-in-toc

High level requirement analysis

In terms of domain model classes, the system is composed of 4 main ones.

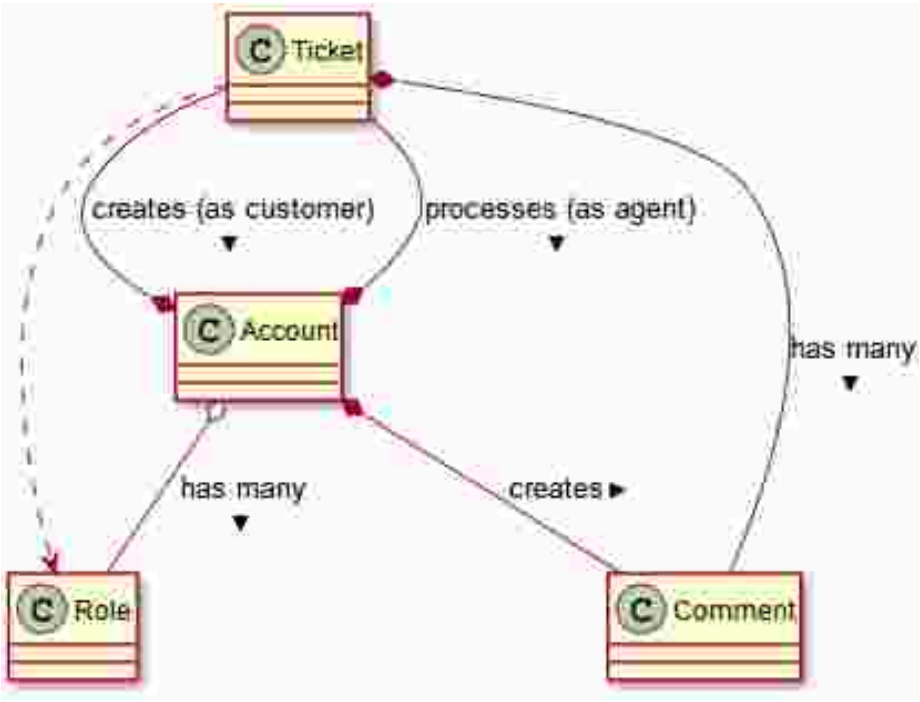
The `Ticket` class represents a ticket in the sistem, that can be created by a customer.

The roles are provided by the `Role` class and owned by the `Account` class.

At the same time, the `Ticket` class checks the account's role in order to know if it's a customer or an agent.

The agents can get tickets assigned

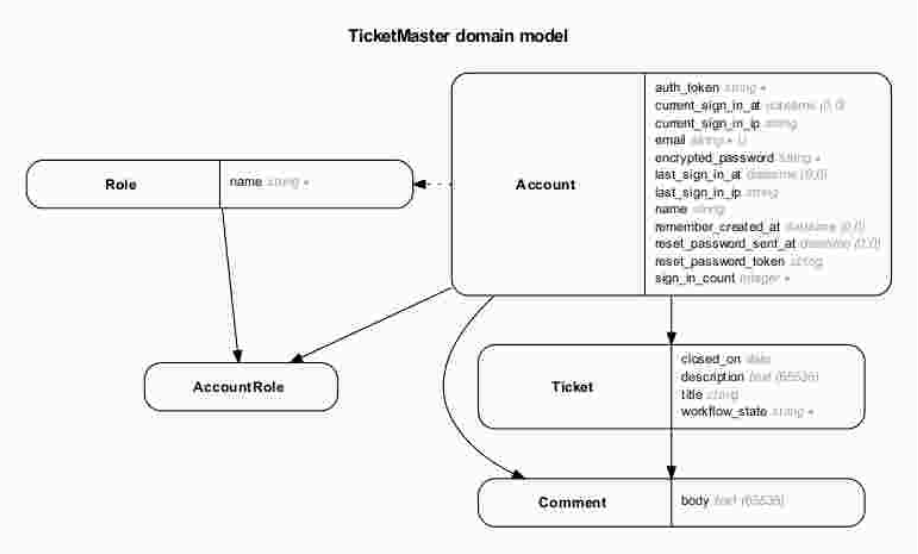
A user can create comments in the tickets he/she either created as a customer or got assigned as an agent.



High level presentation of the data model

The classes on the class diagram has a one-one relation with thir respective database tables.

The only thing to point out here is that we're using an intermediate table for the accounts-roles many to many relation.

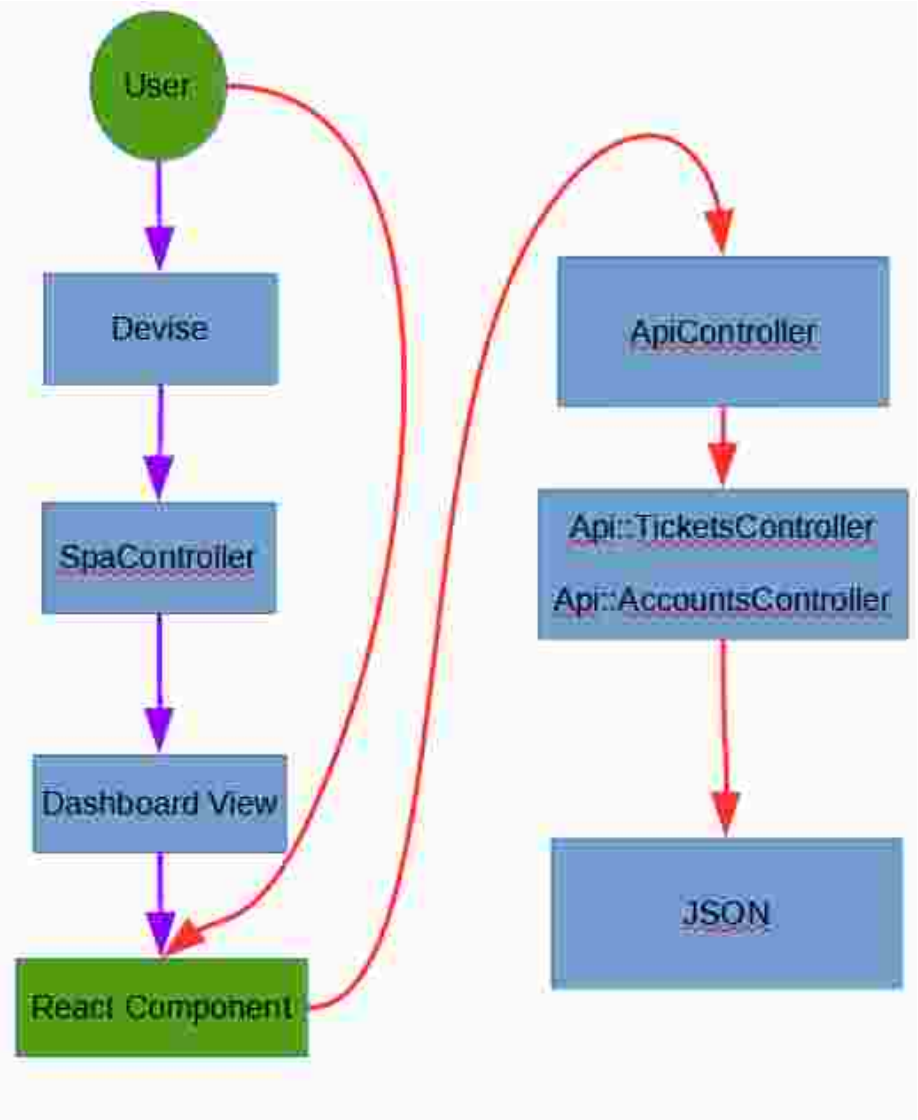


Architecture diagrams describing the composition and working of the system, explaining the component interaction and process, control and data flows.

The system has two main data-flows that work in parallel.

First, we have the main dashboard views (views as in *Rails* views). The data here flows from the user to the server, here, it gets interrupted by `Devise` in order to get authenticated. It then arrives to the `SpaController`, which renders the apropriate React component on to the view, passing the necessary data for it to work.

Once the component is rendered, it triggers the second data flow. It triggers as series of requests to the server through the Api controllers, passing at all times, the apropriate authentication token for the current account. The `ApiController` (from which the rest of the Api namespaced controllers inherits), authenticates the account by comparing the email and token and delegates the request to the target controller and action. From there, the request is processed and a json document is rendered using the *Rabl* gem.



All the controllers are plain *Rails* controllers. The `ApiController` inherits from `ApplicationController` and adds the api authentication method to every action using a `before_action` callback. The api authentication method was inmplemented from scratch and consists of four of steps:

1. On creation, the user accounts get an `auth_token` field, generated by a secure random function (and that is compatible with browsers)
2. When a component is rendered, the auth token for the current account gets send throug it's props
3. When a component needs data from the server, it issues a request passing

the `auth_token` along.

4. The `ApiController` checks that the received token corresponds to the current loged in account and a. If it does, passes the request along b. If it doesn't, it returns an error message along with a `403 Forbidden` HTTP status.

All the components on the SPA are `React.js` components.

The components are separated into two categories:

Component In charge of rendering information on the browser

Container In charge of performing the data requests and passing it along to the corresponding components.