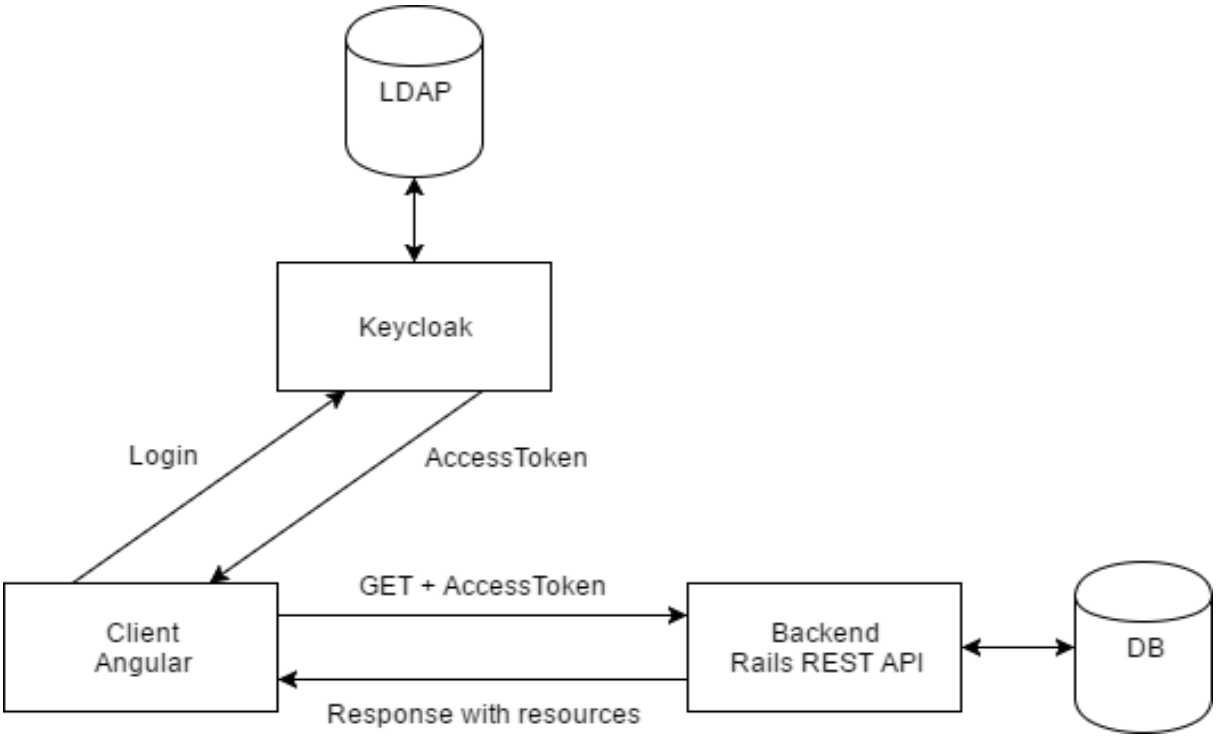
System architecture

Web applications are specific, because when user accesses the application running on the server, he does it via his web browser. Data are transferred over HTTP/HTTPS between the server and the browser, so the application can be split into two parallel programs one running on the server (server-side) and the second in the browser (client-side).

Client-side (Angular)

Angular is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time.

To support single sign-on, the Client is integrated with Keycloak. Keycloak comes with a client-side JavaScript library that can be used to secure HTML5/JavaScript applications and after the user is authenticated, the application can make requests to RESTful services secured by Keycloak.



Server-side (System backend - Rails REST API)

Backend part of the application is implemented in Ruby on Rails. Communication between the Client and the Backend will be done via REST interface. Rails is a framework written in the Ruby language. It's designed to make programming easier, it uses two major guiding principles:

- DRY (Don't repeat yourself)
- Convention over configuration

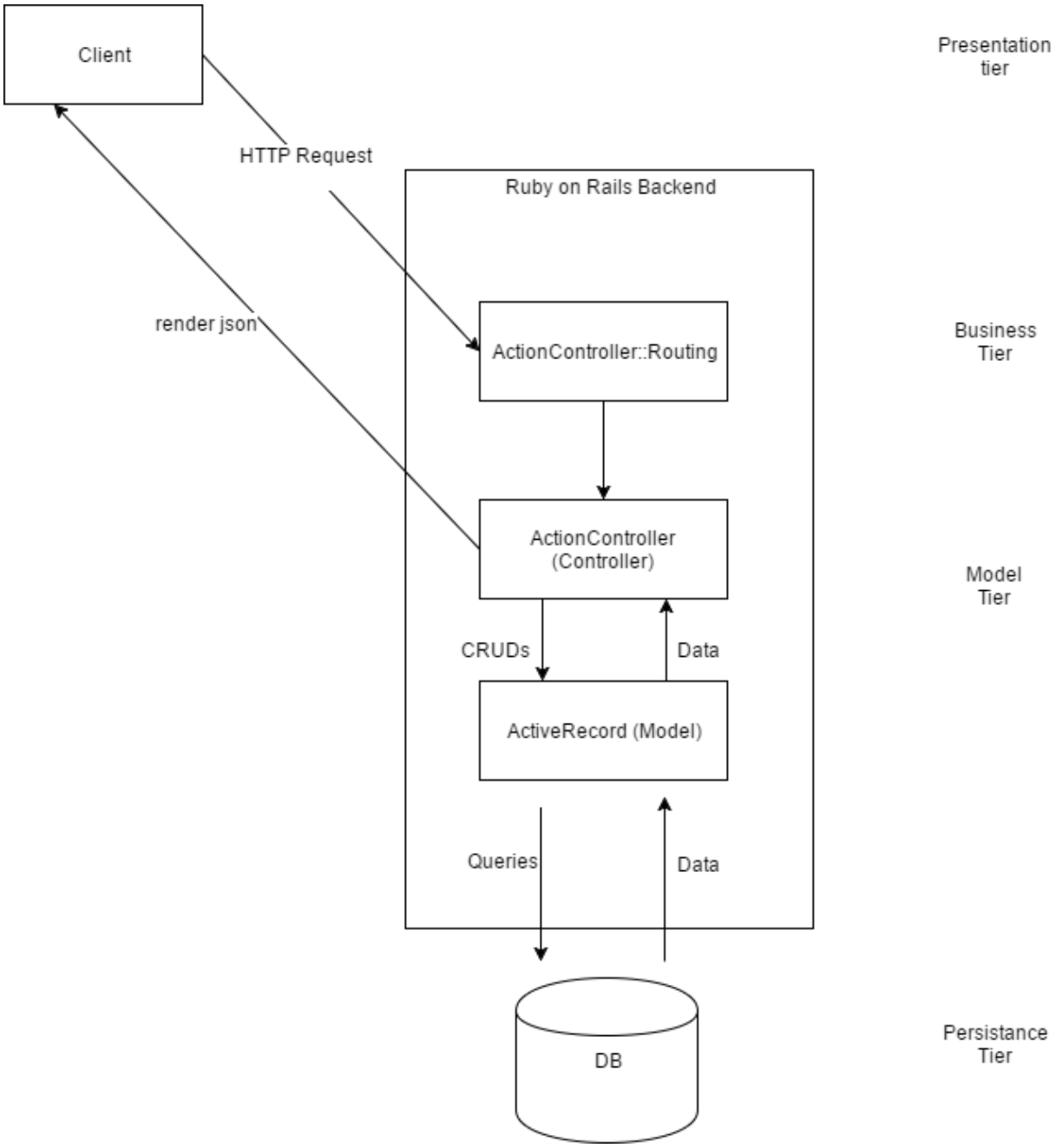
The reason to use Rails is that it provides a set of defaults that allows developers to get up and running quickly, without having to make a lot of trivial decisions. Rails provides several things out of the box, that can be used well for an API application:

- Development/Test mode
- Logging
- Parameter parsing
- Resourceful routing
- Header and redirection responses
- Basic, digest and token authentication
- Generators
- Plugins

Main modules of the Rails backend are:

1) ActionPack
   a) Dispatcher/Routing
      Handles routing of web browser requests. It parses the request and does advanced processing around HTTP, such as handling cookies, sessions, request methods and forwards the request to the specified controller.
   b) Controller
      ActionController contains actions to control the model and to render the responses. It makes data available as needed which are to be rendered and sent back to the user, or it saves and updates the user data to the model.

2) ActiveRecord
   ActiveRecord is an architectural pattern used to manage data in relational databases through objects. In Ruby on Rails the Active Record module provides object-relational mapping to classes. This module builds the Model layer that connects the database tables with its representation in ruby classes. Rails provide tools to implement the CRUD functionality with zero-configuration. CRUD allows creating, reading, updating and deleting records from the database through ruby objects. An object represents each row in a database tables. Additionally, it also provides advance search capabilities and the ability to create relationships or associations between models. Active Records relies heavily on conventions on how the classes should be named, the tables in the database, the foreign keys and primary keys.


Client
Presentation tier
HTTP Request
Ruby on Rails Backend
render json
ActionController::Routing
Business Tier
ActionController (Controller)
Model Tier
CRUDs
Data
ActiveRecord (Model)
Queries
Data
DB
Persistance Tier