RealOffice System Architecture Document

CS14B023, Rahul Kejriwal
CS14B007, Suhas
CS14B045, Malireddi Sunil Kumar
CS09B043, Shanker Lal Sharma
CS09B031, Chandrakanth

Introduction

RealOffice is an application software intended for use in the CSE department office to simplify routine procedural tasks. In essence, it shall allow office staff to schedule different kinds of meetings, track meeting requirements, manage room allotments, file meeting reports and also cancel arrangements for cancelled meetings. It shall also remind the staff regarding upcoming events that need attention and facilitate routine backups as replacement for hard paperwork. It shall also integrate data from the CSE department calendar about meetings scheduled from other platforms.

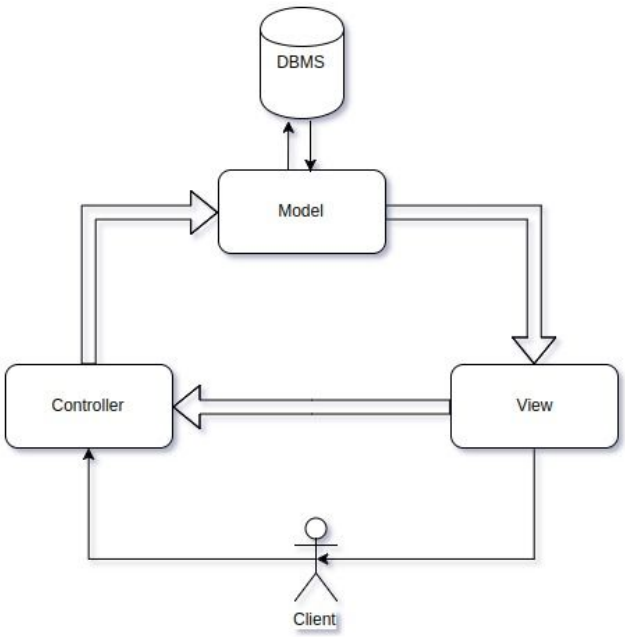
This document provides the architectural pattern and views used for the system along with the rationale behind various design decisions. The design tries to reuse existing software products to bootstrap product implementation like using Django, Django-Rest-Framework, PostgreSQL, jQuery etc.

The partitioning of the system into modules aimed to group relevant tasks in the most efficient manner to satisfy the nonfunctional system requirements.

Use cases and scenarios are not included in this document as they were included in earlier documents.

Architectural Pattern

The RealOffice system builds on the MVC (model - view - controller) architectural pattern in a Client-Server framework. The system consists of a Server providing the models part and the view generator/renderer. The Client displays the generated view. The controller is divided into 2 parts - the frontend controller and the backend controller running at the Client and server respectively.



The system is based on Django framework in python which intrinsically uses the MVC framework. The decision to use Django was taken because it is in line with our software design and allows us to quickly setup the system using PostgreSQL for our database management system (DBMS). Django is also highly robust and has many packages for extensions to quickly integrate features like Authentication, REST APIs etc.

The system uses a DBMS for data storage rather than a flat filesystem based storage to allow quick retrieval of data from large datasets and allow easy scaling along with transaction and recovery management. Using SQL based relational DBMS also allows quick merge and filter operations for generating relevant views.

PostgreSQL was chosen as the DBMS as it is highly efficient and robust for industrial production usage and provides high availability. The development phase uses SQLite as the DBMS to quickly share DB changes and forego DBMS setup and configuration. SQLite is provided with python and is good for rapid prototyping.

Nginx was chosen as the server software as it highly reliable and easy to setup.

Architectural Views

1. Logical View

The purpose of the logical view is to specify the functional requirements of the system. The main feature being the Design Model which consists of functional behavior of the system. The logic View consists of 3 parts :

I. Static Structures :

The notion of class is put forward and we construct classes based on the ER diagram. The Class Diagram is the result and it provides the static logical structure to the project

The Class diagram (also ER diagram) has already been submitted in the System Modeling document of RealOffice.

II. Interactions :

Interactions are described by use-case scenarios and for each use-case scenario, we have an associated Interaction Diagram which describes the flow of control in the scenario.

There are many ways to represent Interaction diagrams, most used method is the Sequence Diagrams method. We have submitted the Sequence Diagrams for various use-cases in our System Modeling document of RealOffice.

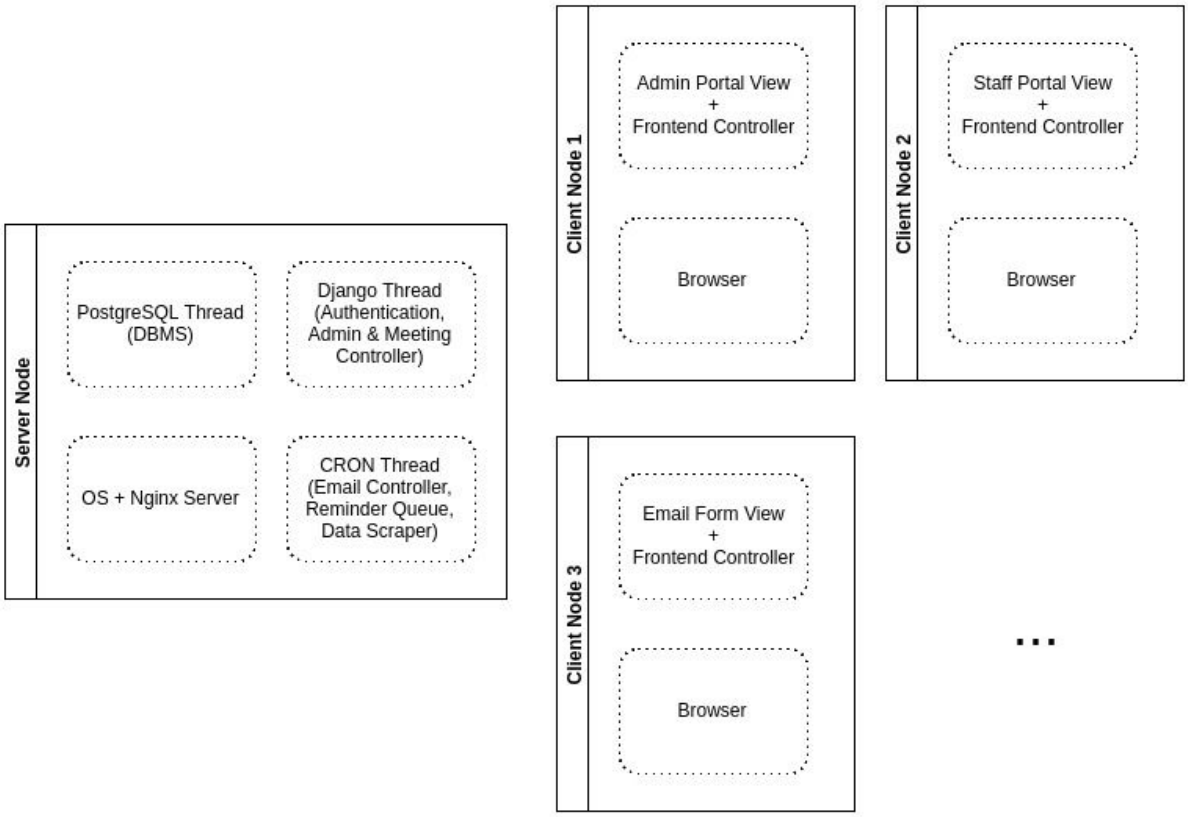
III. Dynamic Behavior :

The dynamic behavior in the program is captured by the State Transition Diagrams which describe the transition from one state to the other based on an event and show the various functions/procedures which will be called as a result of it.

One of the ways to do this is via Activity diagrams which capture the flow of control over various activities. The activity diagrams for various use-cases of RealOffice were submitted in the System Modeling Document of RealOffice.

2. Process View

The different processes running can be seen as threads distributed among various nodes in the following manner:



The activity diagrams for various scenarios (which map to different processes) was included in the earlier system modelling diagram.

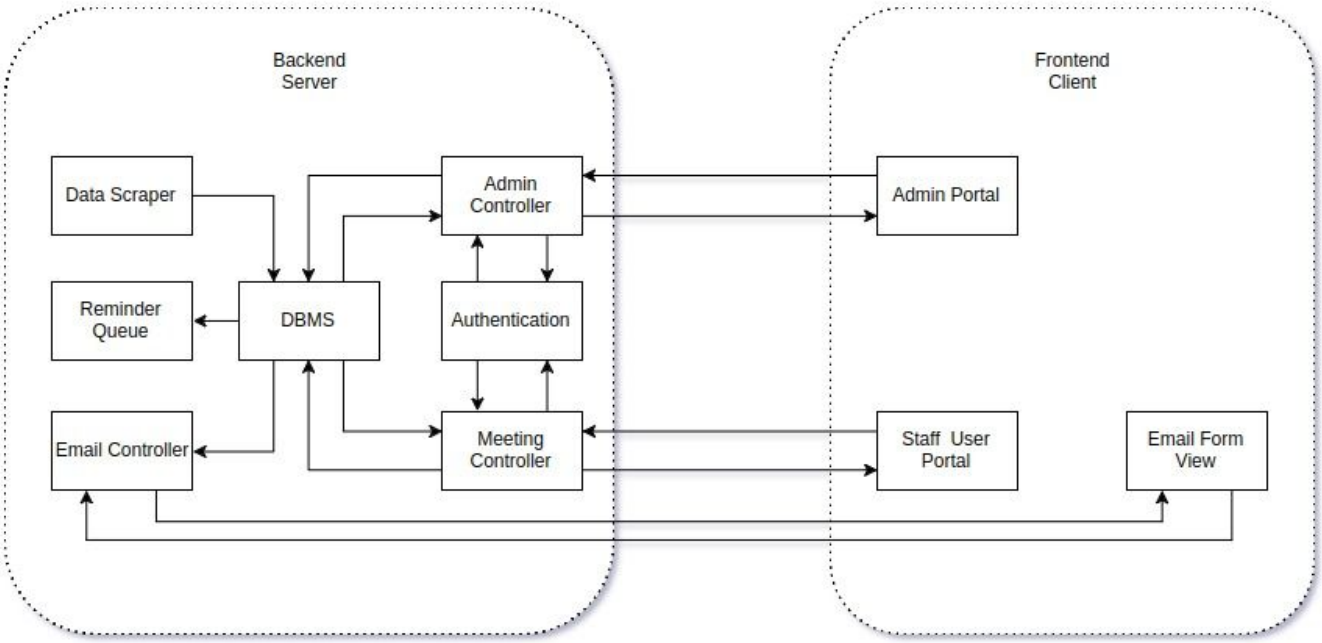
The ‘Add User’ & ‘Delete User’ scenario activity diagrams map to the Django process (backend) and Admin Portal View + Controller process (frontend).

The ‘Add Meeting’, ‘Add Meeting Requirements’, ‘Reschedule Meeting’, ’Cancel Meeting’, ‘Generate Backup’ and ‘Generate Report’ scenario activity diagrams map to the Django process (backend) and Staff Portal View + Controller process (frontend).

The ‘RSVP Reminders’ and ‘Request Requirements’ scenario activity diagrams map to the CRON process (backend) and Email Form View + Controller process (frontend).

3. Development View

The system was decomposed into the following modules for the purpose of development:



The system consists of two broad modules - the frontend and the backed.

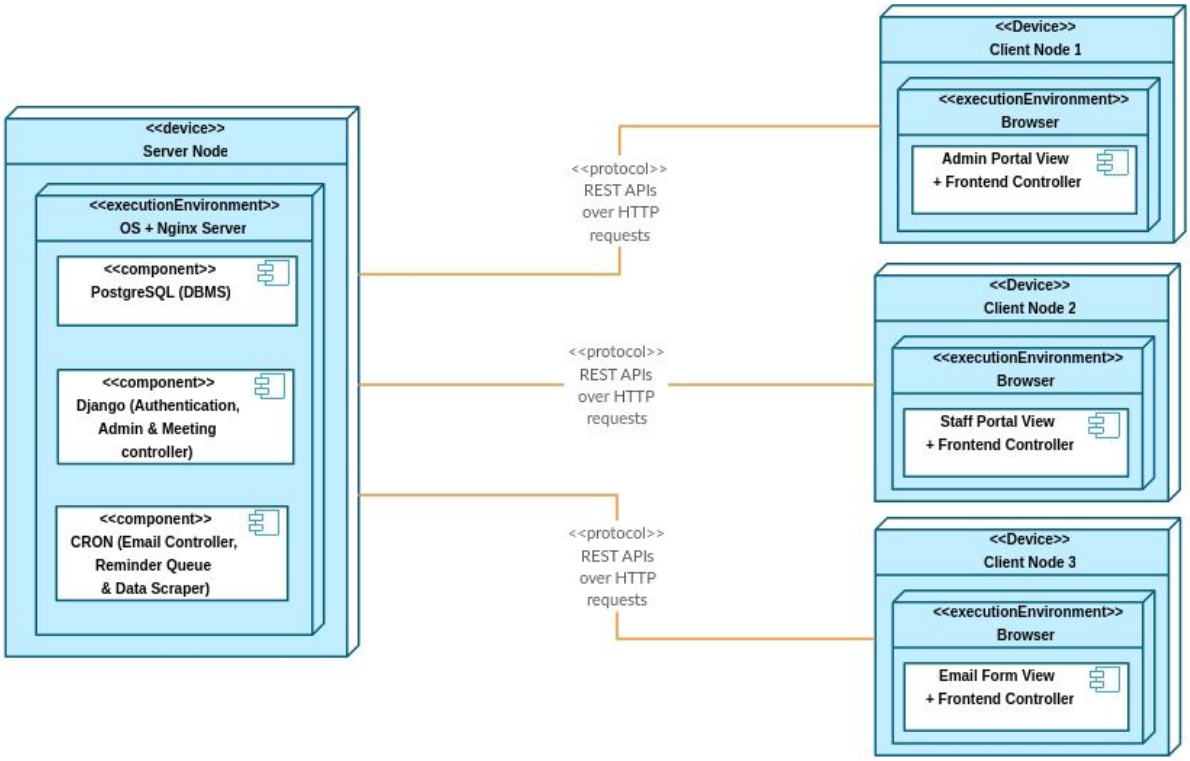
The frontend basically consists of a module for each type of frontend interface - admin or staff user or email form. The frontend modules implement one of the views and its corresponding frontend controller.

The backend consists of 3 types of modules:

1. The first ones are those that respond to client initiated requests and generate relevant responses - Admin Controller, Meeting Controller and (partly) Email Controller.
2. The second type provide helper or base functionality - DBMS and Authentication.
3. The third type executes periodically and performs periodic functions - Data Scraper, Reminder Queue, (partly) Email Controller.

4. Physical View

The system components are distributed physically on different nodes and processors in the following manner during deployment/production:



The system runs on a Server-Client framework with one Server node servicing multiple Client Nodes.

Each Client node is an instance of a View (Admin or Staff or Email Form) and corresponding controller running atop a Browser environment.

The Server runs on top of a OS (probably Linux) and Nginx Server. The Server has 3 main threads running parallely on different processors / hyper-threads - PostgreSQL, Django & CRON.