Storm Cloud Development

ProjectCM
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
28/Nov/2013	1.0	Initial software architecture description	Storm Cloud Development

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This Software Architecture Document provides an architectural overview of the ProjectCM web application. This Document has been partly generated directly out of Ruby on Rails.

1.3 Definitions, Acronyms, and Abbreviations

CRUD	Create, read, update, delete
MVC	Model-View-Controller
SRS	Software Requirements Specification
SAD	Software Architecture Document

1.4 References

SRS	Software Requirements Specification.pdf

1.5 Overview

The SAD shows how the architecture of the software is. So we will start with the general representation and then show the goals and constraints. Afterwards the use cases and the different views are explained. These views are the logical view, the process view, the deployment view and the data view. In the end there is a paragraph about size and quality.

2. Architectural Representation

The software architecture is organized as MVC. So this document contains information about model, view and controller.

3. Architectural Goals and Constraints

3.1 Security

All important data e.g. the password has to be stored as a hash. So it should not be possible to use this data when someone who should not have access to it gains access. Also all authorization checks must be done by the server to avoid manipulations by the client. It should be possible fetch emails with SSL from other servers to the ProjectCM server and in production environments all communication between client and server must be via SSL.

3.2 Distribution / Reuse / Dependencies

The application is on a single instance when we start with it, but it should be possible to scale it on different servers. Also as much code as possible should be reused. There are no external dependencies.

4. Use-Case View

See SRS

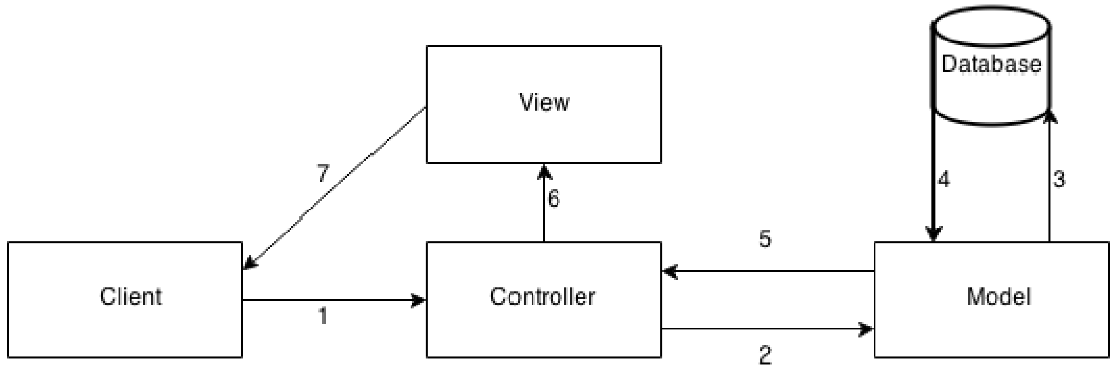
4.1 Use-Case Realizations

See SRS

5. Logical View

This section explains how the logical architecture of the application is. For this purpose an overview of the architecture is given and the class diagrams of model and controller are included and shortly explained.

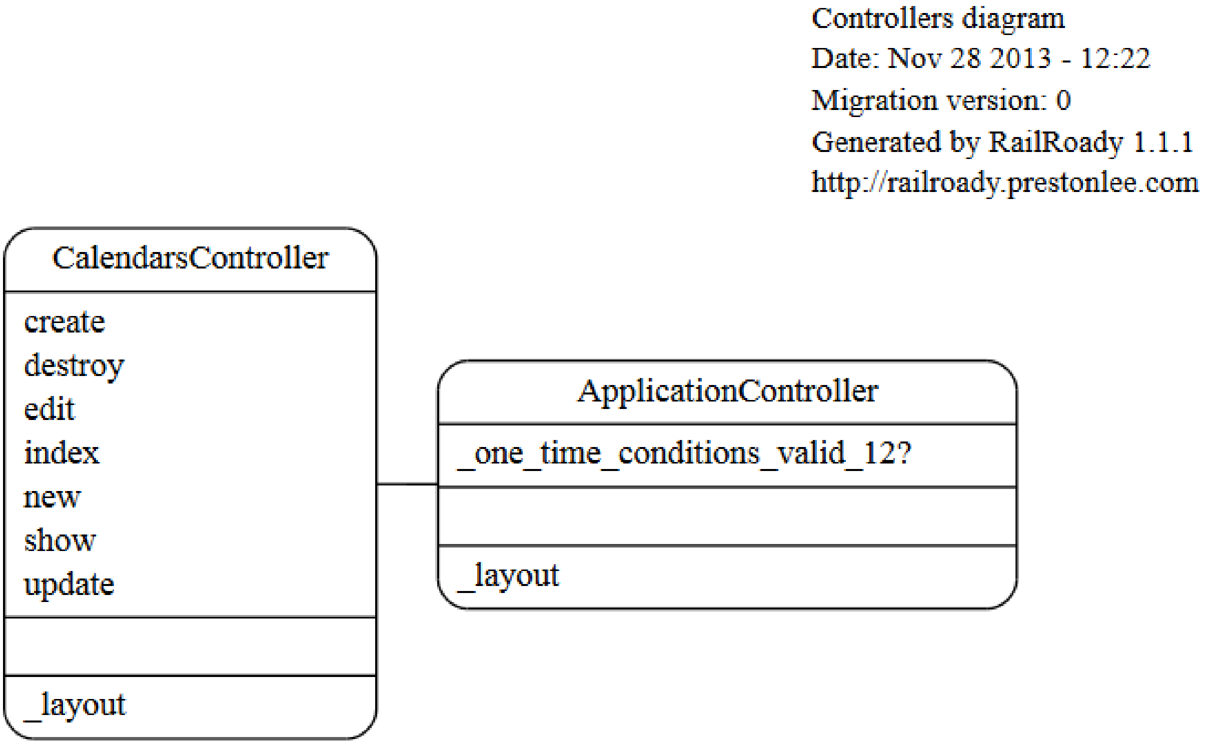
5.1 Overview



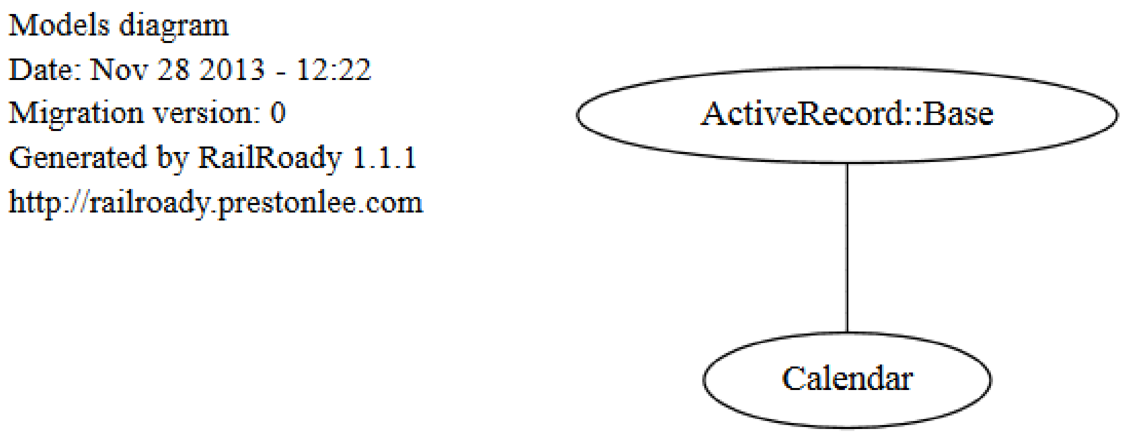
The client sends a request to the Controller (1). The controller calls the model to prepare the data (2). The model fetches the data from the database (3, 4), prepares it and returns the prepared data to the controller (5). The controller chooses the view to show the data with and sends the data to the view (6). The view shows the page to the client (7).

5.2 Architecturally Significant Design Packages

Below are the already implemented classes of the use case manage calendar (CRUD). There is no View class as the views are only templates to display the data. The CalendarsController controller class implements the methods for the CRUD.


Controllers diagram
Date: Nov 28 2013 - 12:22
Migration version: 0
Generated by RailRoady 1.1.1
http://railroady.prestonlee.com

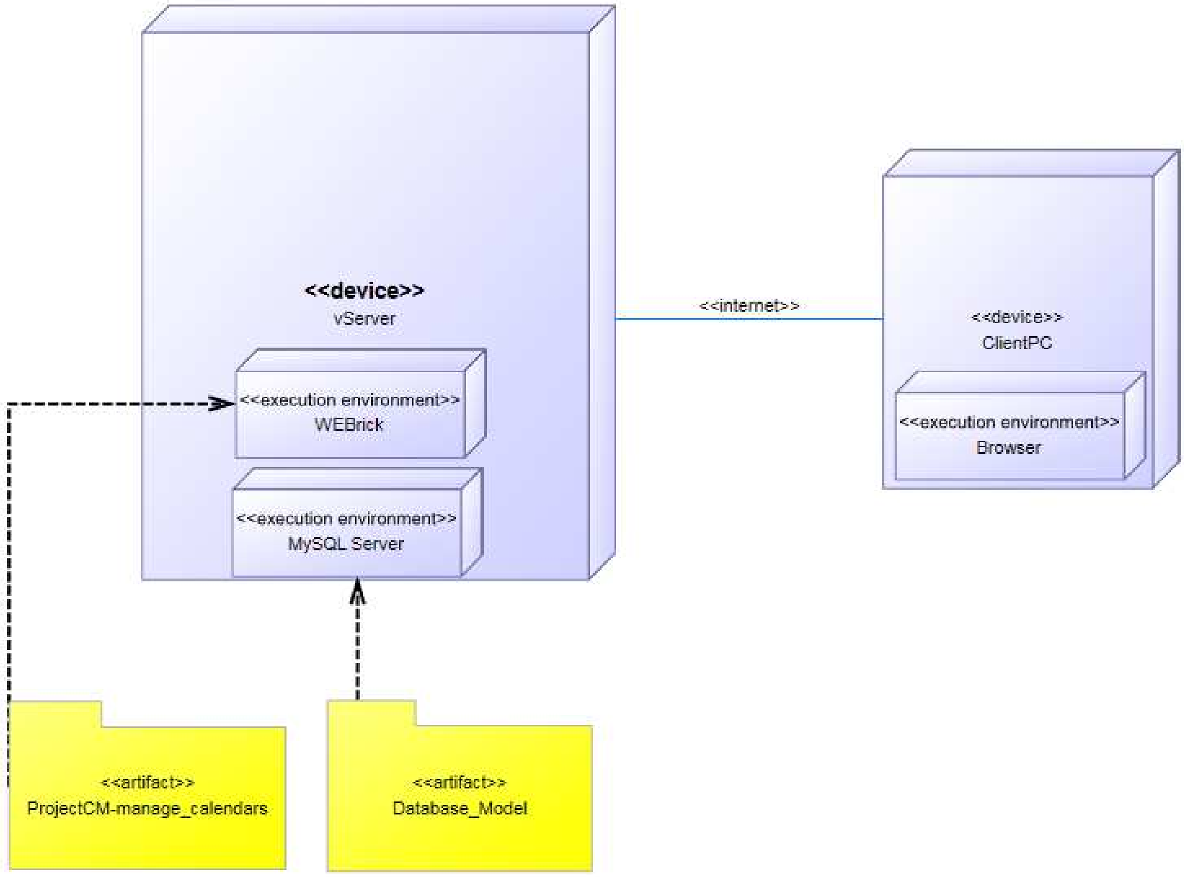
The Calendar model consists of a class which derives from the ActiveRecord class for database connection functionality.


Models diagram
Date: Nov 28 2013 - 12:22
Migration version: 0
Generated by RailRoady 1.1.1
http://railroady.prestonlee.com
ActiveRecord::Base
Calendar

6. Process View

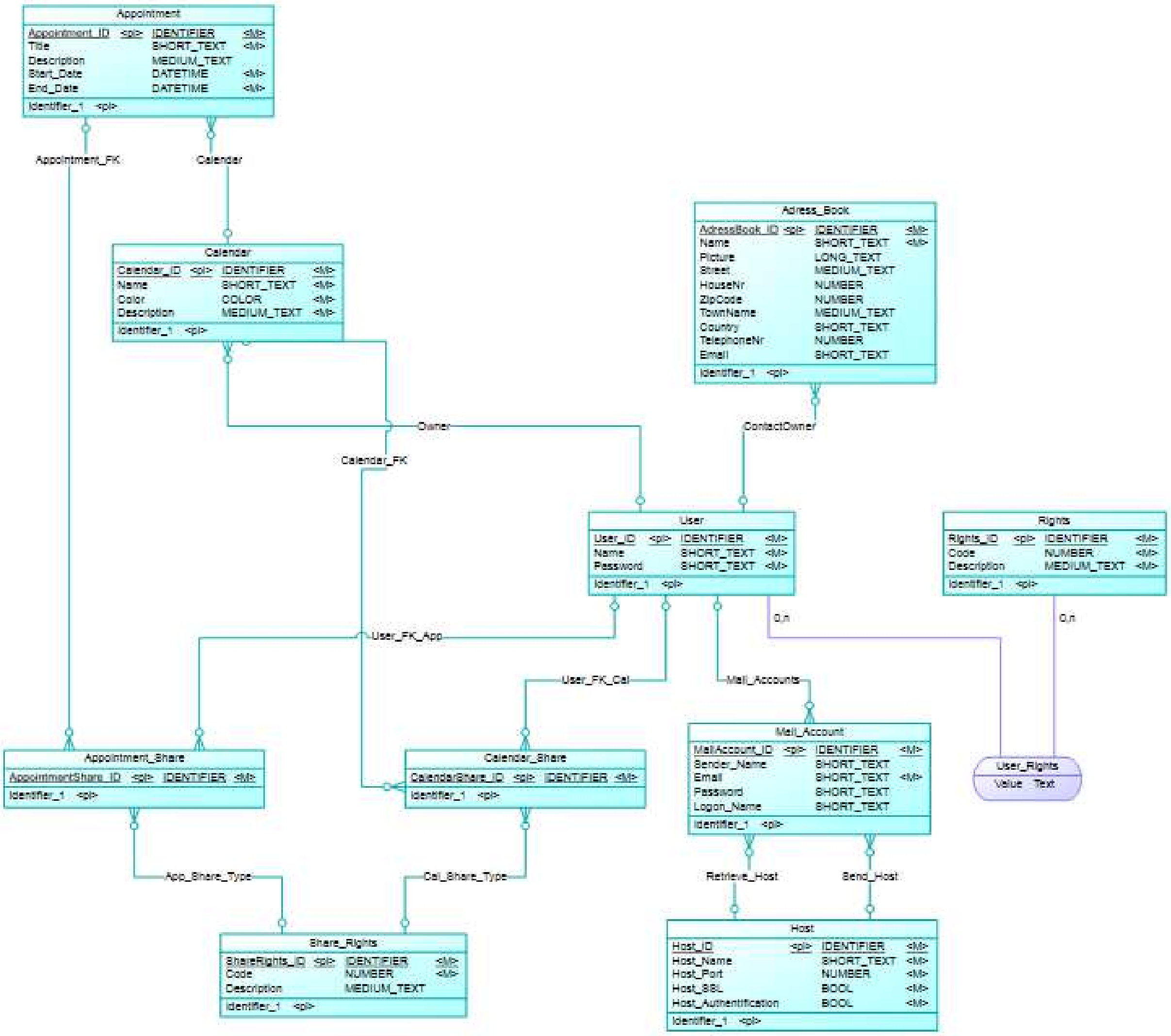
n/a

7. Deployment View



8. Data View

The database model below is the complete database model, planned for the finished application.



9. Size and Performance

n/a

10. Quality

n/a