RingBase

Software Requirements Specification

Feb 11, 2014

Group Name: RingBase

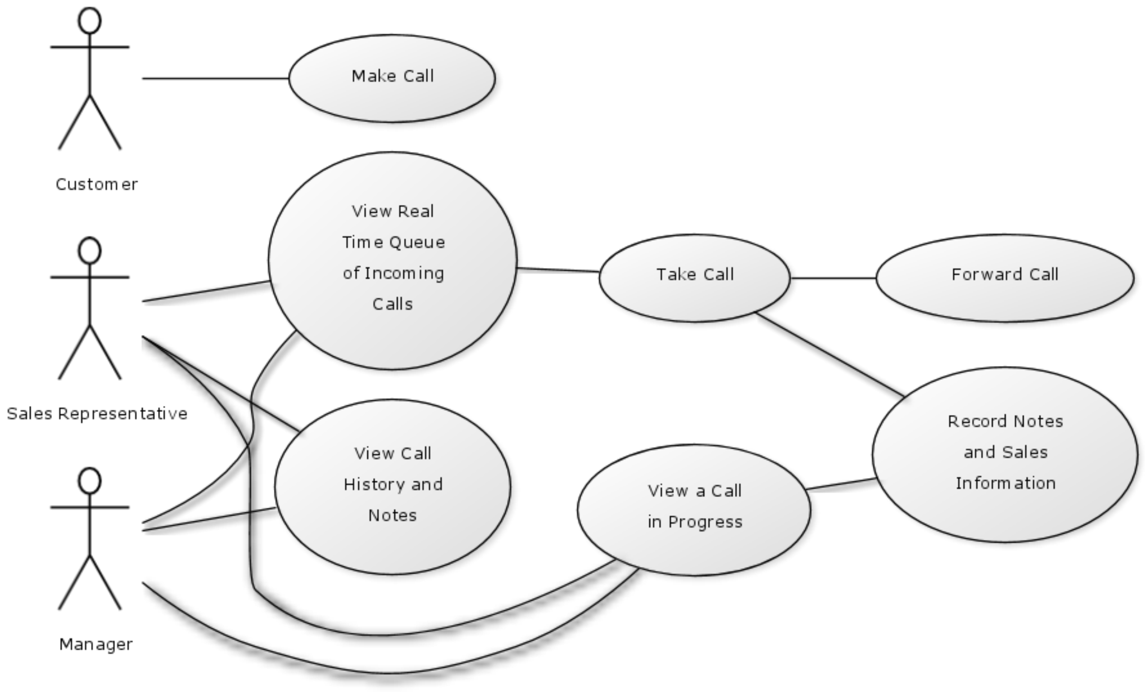
Instructor	Chandra Krintz
Course	CS 189A
Lab Section	Wednesday 6PM
Teaching Assistant	Geoffrey Douglas
Date	Feb 11, 2013
Mentor	Colin Kelley

Andrew Berls	4719696	andrew.berls@gmail.com
Pete Cruz	5226196	petesta@live.com
Alex Wood	4960381	awood314@gmail.com
Shervin Shaikh	5074901	shervinater@gmail.com
Nivedh Mudaliar	4875266	nivedh.mudaliar@gmail.com

Table of Contents

1. Overall Description

1.1 Product Perspective

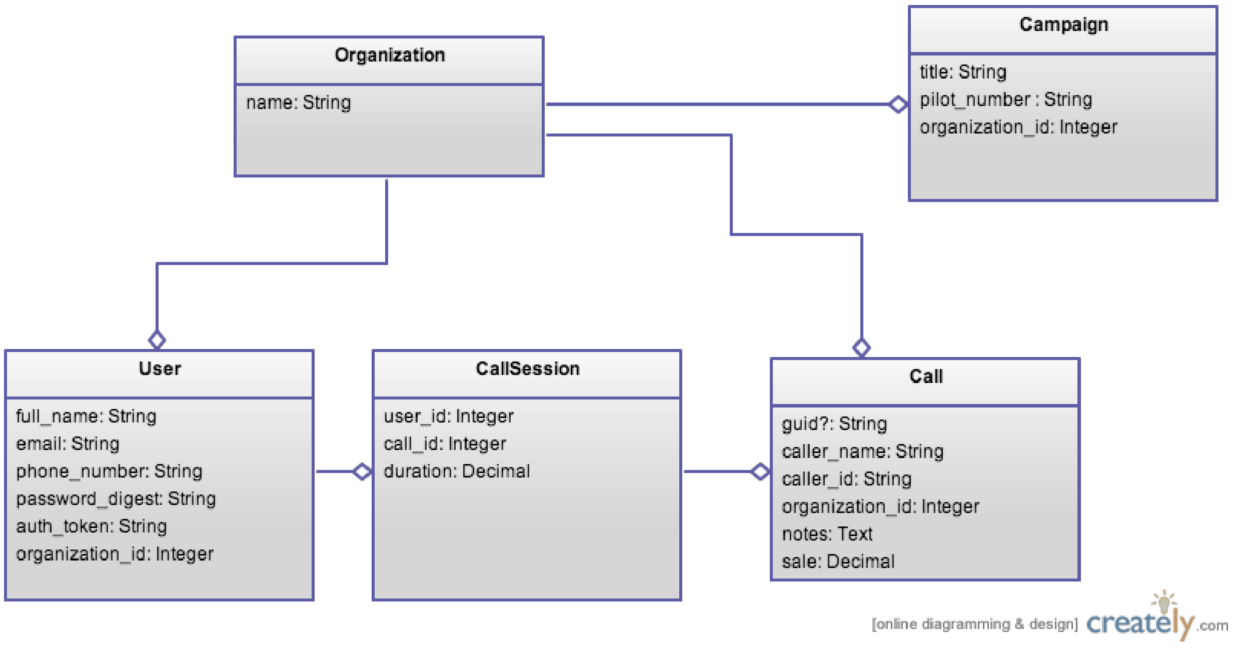

Customer
Make Call
Sales Representative
View Real Time Queue of Incoming Calls
Take Call
Forward Call
Manager
View Call History and Notes
View a Call in Progress
Record Notes and Sales Information

http://yuml.me/edit/02f3c231

1.2 Product Functionality

The software will enable a business to set up an ad-hoc call center for its employees. It will track basic information about each employee, such as their name, email, phone number, and gravatar. It will contain a dashboard where a manager can see incoming calls (caller ID, city, state, name). Agents can claim a call (either through their telephone or WebRTC) and once accepted that agent sees a shared view for adding notes about the call & dollar amounts. The platform will update in real-time for all viewers so there is no need to refresh the page. Agents can pass calls around, either back into the pool or directly to another agent. The software will keep and display a call log of calls and their notes over time.

Database Schema



1.3 Users and Characteristics

RingBase targets three types of users: Agents, Managers, and Customers.

- Agent - is an employee who is overseeing the incoming calls through a dashboard and either answers the call or forwards the call to another suitable agent.
- Manager - is an employee but also has an additional role of overseeing all agents and has access to statistics such as individual/aggregate agent performance and sale numbers from each call. They are able to access all of the data generated by all employees.
- Customer - makes a call that is then displayed on the agents dashboard and interacts with an agent over the line.

1.4 Operating Environment

Our operating environment will consist of the following:

- **Amazon EC2** - virtualized on-demand servers
- **Ubuntu 12.04** - server OS
- **Ruby 2.1.0** - dynamic server-side language
- **Rails 4.0.0** - MVC web application framework
- **nginx 1.2** - High-performance HTTP server / reverse proxy
- **Unicorn 4.6.0** - Rack application server for Rails
- **PostgreSQL** - Relational database
- **Apache Cassandra** - Distributed NoSQL database, for storing call data
- **EventMachine / Goliath** - Asynchronous framework and server for socket broker
- **WebSockets** - Protocol/API for long-lasting browser connections
- **AMQP/RabbitMQ** - Pub/Sub messaging protocol for communication between socket broker and external Invoca API
- **AngularJS 1.2** - Javascript MVC Framework

1.5 Design and Implementation Constraints

Our platform will consist of several services running concurrently on virtualized Amazon EC2 instances. This gives us the ability to scale infrastructure up and down on demand, using a variety of available instance sizes.

The app requires a rich frontend experience, with real-time updates based on data from the server, and views shared between multiple people for adding notes, etc. The frontend will utilize the Angular MVC framework to provide unified code structure and modularity.

Our backend includes a minimal Rails application, which handles the basics of authentication (using bcrypt for password hashing), protection against XSS/CSRF, and page serving. Most of the platform's functionality will be part of a separate 'broker' service, which will be an independent server running EventMachine/Goliath and acting as a broker between Invoca's real-time telephony API and our clients (managed over WebSockets). The only communication layer between the Rails app and the Goliath broker is a shared PostgreSQL database.

This architecture enables a clean separation of concerns into separate services/servers. The majority of our backend code will be written in Ruby, while frontend functionality will be implemented in JavaScript (CoffeeScript). The uniformity will allow for code that can be understood by all members of the team as well as shared coding standards.

2. Specific Requirements

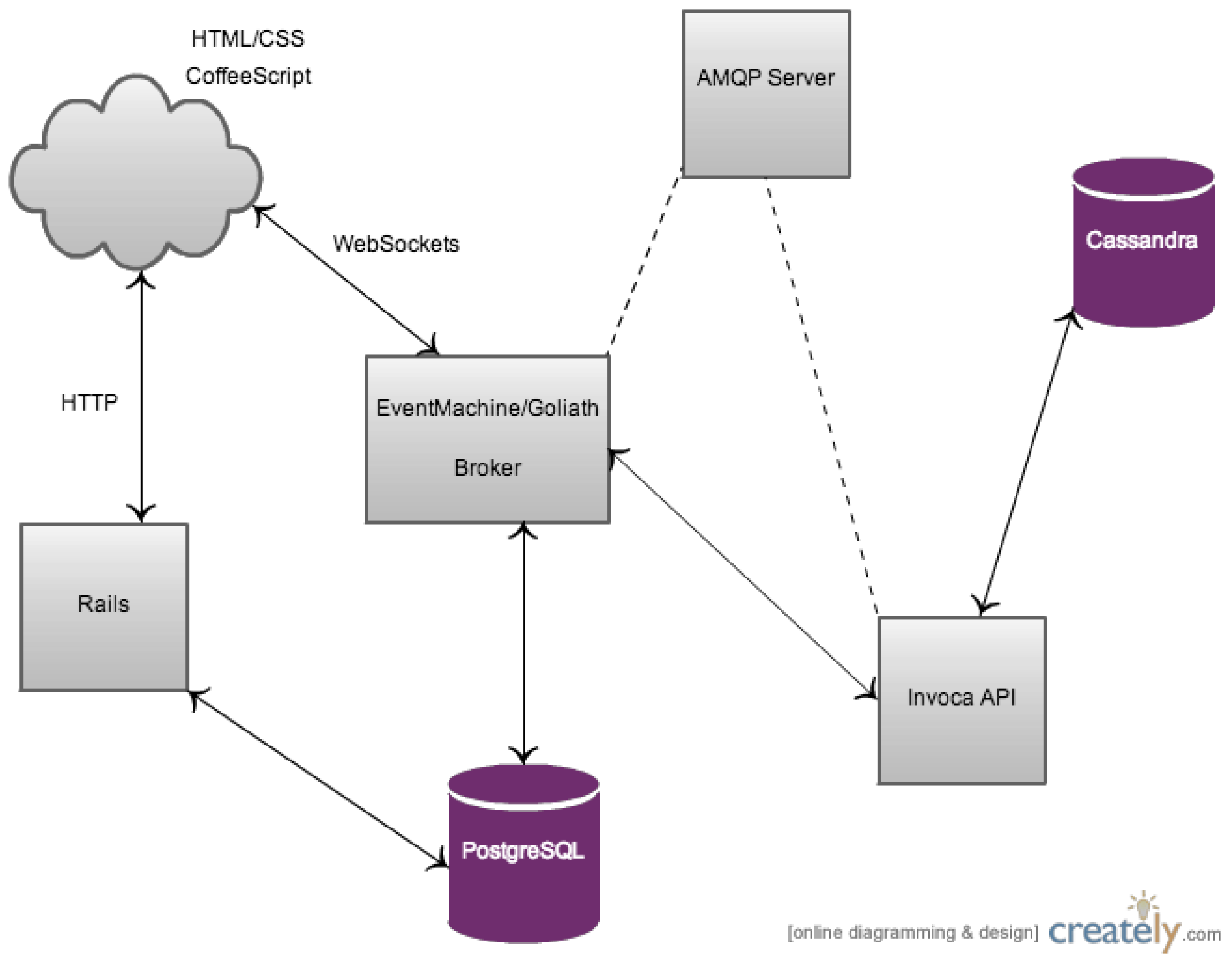
2.1 External Interface Requirements

The platform will consist of responsive pages that update in real-time, showing tge manager incoming calls and agents have claimed them. There will be a common input view with a textarea for allows agents to enter notes and annotations as well as an input to enter in dollar amounts (positive number) for their call. This view will have buttons that allow agents to pass their phone call back into the pool and a selector where they can choose a specific agent to pass their call onto. There will also be a view that allows users to view the call log and note history.

2.2 Hardware Interfaces

As the platform is a web application, there are no specialized hardware interfaces. Users will interact with the app through their browsers (Chrome, Firefox, Android Browser, Android Chrome, iOS Safari), although once a call is accepted it may be forwarded to their physical telephone (all devices - landline, cell phone, etc).

2.3 Software Interfaces



- Web UI (HTML/SCSS, CoffeeScript)
- Ruby on Rails (MVC architecture)
- JSON (data serialization)
- PostgreSQL
- Apache Cassandra
- EventMachine / Goliath (async server)
- AMQP (messaging protocol)

2.4 Communications Interfaces

Communication between client browsers and our servers will happen over HTTP, and leverage the WebSockets API for persistent connections and information transfer in real-time (using JSON as a serialization format). Backend services (such as our messaging broker and Invoca's API) will communicate using the AMQP protocol, using a system such as RabbitMQ. JSON will also be used on the backend for serializing data.

2.5 Non-Functional Requirements

Safety & Security

Data integrity and security is of paramount importance. As our platform is a hosted offering, our database will store valuable business data for many different clients, and any "leakage" of data between clients or unauthorized external access would have very negative consequences. Fortunately, Ruby on Rails provides built-in protection against many common web security vulnerabilities such as cross site request forgery (CSRF), cross-site scripting (XSS), and SQL injection attacks. Combined with rigorous access-checking in our application code and a robust test suite, we can have a high degree of confidence in the security of our application's data and business rules.

To enable users to log in to the platform, we store password data hashed/salted with the BCrypt library, which is also included as part of Rails. BCrypt hashing is an "expensive" operation, making brute-force or dictionary attacks infeasible.

Performance

Most interactions on our platform are required to happen in "real-time", and so it is crucial that our servers be able to handle load and rapidly handle requests. Jakob Nielsen of the Nielsen Norman user experience research group observes that 100ms is the limit for a user to feel that an event has occurred instantaneously, and that 1 second is the limit for noticing a delay and being interrupted. Thus, we will strive to respond to all requests within 100-300ms, with 1 second being the maximum time tolerated. To accomplish this, we will seek to finely tune SQL queries and long-running operations, utilize caching techniques, and so on.

Software Quality

Our platform consists of many different services and components working together, and it's important for us to have a shared coding standard to maximize readability and make it easy for any developer to understand any section of code. Rails prescribes some defaults regarding coding and naming conventions, and we will use published style guides for Ruby, CoffeeScript, etc. as a standard when writing code.

3. User Stories

- **Manager** - head of the travel agency
- **Agent** - an employee at the travel agency
- **Customer** - one who calls the travel agency to arrange itinerary/flight plans

Travel Agency MVP:

As a manager/agent, I can log in and log out (EASY)
- Acceptance: I can visit a page, enter my credentials into a form and be taken to the main dashboard

As a manager I can create an account (EASY)
- Acceptance: I can visit a page, enter my information into a form and be redirected to my newly-created platform account

As a manager, I can create an organization (EASY)
- Acceptance: I can visit a page, enter my organization name, and be taken to the newly-created organization dashboard

As an agent, I can browse a history of completed calls and see their notes (MEDIUM)
- Acceptance: I can see a table or list of calls that let me drill-down to see their notes and information

As a manager with an organization, I can send an email to agents prompting them to create an account (MEDIUM)
- Acceptance: I can visit a page, enter my agents emails, and have them receive an email with a link to join

As an agent receiving an invitation from a manager, I can create an account (EASY)
- Acceptance: I can click link from an invitation email and enter my information to create my platform account

As an manager/agent, I can see incoming call information such as caller ID, city, state, and name (HARD)
- Acceptance: When a call comes in to my organization, I see an entry added to a list displaying the relevant call fields

As an agent, I can be notified as soon as a call comes in (MEDIUM)
- Acceptance: I see an obvious visual notification as soon as a call comes in

As an agent, I can claim a call on the platform and have it directed to my telephone (HARD)
- Acceptance: I can click a button on an incoming call and receive the call on my physical phone

As an agent, I can see a shared view for adding notes and payout information when I'm on a call (MEDIUM)
- Acceptance: After accepting a call, I'm taken to a view where I can see relevant information for the call as well as a form for me to enter notes

As an agent, I can edit shared notes for a call in real-time (HARD)

- Acceptance: I can enter notes into a form, which all other agents viewing the call see update in real-time

As an agent, I can transfer my call to another agent in the organization (HARD)

- Acceptance: When on a call, I can click a button that will prompt me to choose an agent and select one to transfer the actual call

As an agent who receives a call transfer, I can see any notes on the call from the previous agent (MEDIUM)

- Acceptance: After a call is transferred to me, I can see the same shared notes view that the other agent saw