CS3300: Introduction to Software Engineering

Design Document for RESTful API

Version 2.0: 06/07/15
Team: RESTful API

Contents

1 Introduction

1.1 Overview (Executive Summary)

This document describes the software being used in order to build an API service for a messaging application. In addition, it explains how clients can use the API to easily communicate between devices and manipulate resources. The API will adhere to the architectural principles and constraints of REST. These properties and how they will be implemented to make a RESTful API are explained in section three. The way in which software components and objects interact are represented by the diagrams in section four.

1.2 Definitions and Acronyms

RESTful: Representational State Transfer

API: Application programming interface.

HTTP: Hypertext transfer Protocol

Hypermedia: An extension to hypertext that supports linking of other formats such as images, PDF, etc.

State: Or application state, is the data needed by the server to fulfill a request. Data varies by client and request.

Resource: Or resource state, is the data that defines the resource representation. Data is stored and remains constant across every client who requests it.

URI: (Uniform Resource Identifier) Each resource in a service will have at least one URI identifying it. The URI uniquely identifies the resource and the body contains the state (or state change) of that resource.

URL: Uniform Resource Locator

Representation: Components communicate by transferring a representation of a resource. A representation is a sequence of bytes, plus representation metadata to describe those bytes.

Metadata: the data providing information about one or more aspects of the data. Provides shared understanding of data types in the form of name-value pairs

Class: a representation of an object that reflects its structure and behavior within the system.
Object: an instance of a class.
GET: Retrieving a representation of a resource
POST: Similar to get but places additional info in the body of the request.
PUT: Upload a resource to the server
DELETE: Delete a resource from the server
Spring: Java Developmental Framework
Maven: Software project management tool
JSON: Javascript Object Notation.
Jackson: JSON parsing library
Hal: (Hypertext Application Language) is a standard convention for defining hypermedia such as links to external resources within JSON.
IntelliJ: Interactive development environment

1.3 References

N/A

2 Design Considerations

2.1 Assumptions

The system was designed with the assumption that clients will be able to use HTTP requests with the JSON mime-type. This format will be used in the manipulation of resources. In order for clients to interact with our service, they will be provided with the information to do so. Client's will have access to class and sequence diagrams. This will define the classes, and inform the clients of class attributes. The class attributes will define the resource representation to be used by its JSON format. If a team cannot make requests with the JSON format, a different data format will need to be agreed upon.

2.2 Constraints

The constraints of the API being designed are:

- The system must provide service to a web client, and to an android phone, tablet and smart-watch client
- The system must allow for client communication
- The system must allow clients to upload/access resources
- The system must allow clients to manipulate resources
- The system must allow clients to transfer resources
- The system must able to handle requests from multiple clients at the same time
- The system must always be running

2.3 System Environment

In order to build the system for a RESTful API, multiple software components are required. The main framework will be provided by Spring. This approach handles HTTP requests with a controller. This feature will be implemented using Jackson and will be represented with JSON. Jackson provides classes that can to convert a Java Object to JSON and back. If hypermedia links are required, HAL is a simple format for hyperlinking between JSON representations. In order to build and deploy the project, Spring will be used with Maven. Maven uses an XML file that contains information about the project's dependencies, build order, package architecture, and required plug-ins. Once Maven builds the project, the system will use AWS to host its web service and SQL database.

3 Architectural (High-Level) Design

3.1 Overview

A web service is RESTful when its structure and functionality adhere to certain constraints and properties. The basis of REST provides a uniform interface for clients to make stateless requests of resources to the server.

The uniform interface of REST encompasses four constraints:

Identification of resources
Every resource is uniquely addressable using a URI.

Manipulation of resources through representations
Components manipulate a resource by using a representation to capture the current or intended state of that resource. The representation is then transferred between components. When a client holds a representation of a resource, it can make requests to manipulate it. Standard HTTP requests are used.

Self-descriptive messages
Each representation includes the information required by the server to process it. This means clients can send HTTP requests to a Resource Request Handler. This method will then be mapped to the corresponding operations of the resource.

Hypermedia as the engine of application state
If needed, the server will provide links contained in the returned representation. The client transitions through states through these links or through the controls identified by the hypermedia representation.

The stateless part of REST means the session state is held within the client. The client must include all information for the server to fulfill the request. If a state must span multiple requests, the client must resend it. When the client is ready to change state, a request is sent to the server. The client transitions through states by choosing from the state transitions in the current set of representations. Lastly, cache constraints require that the data within a response to a request define its cahcheablity. If a response is cacheable, then a client can store that response data for later requests.

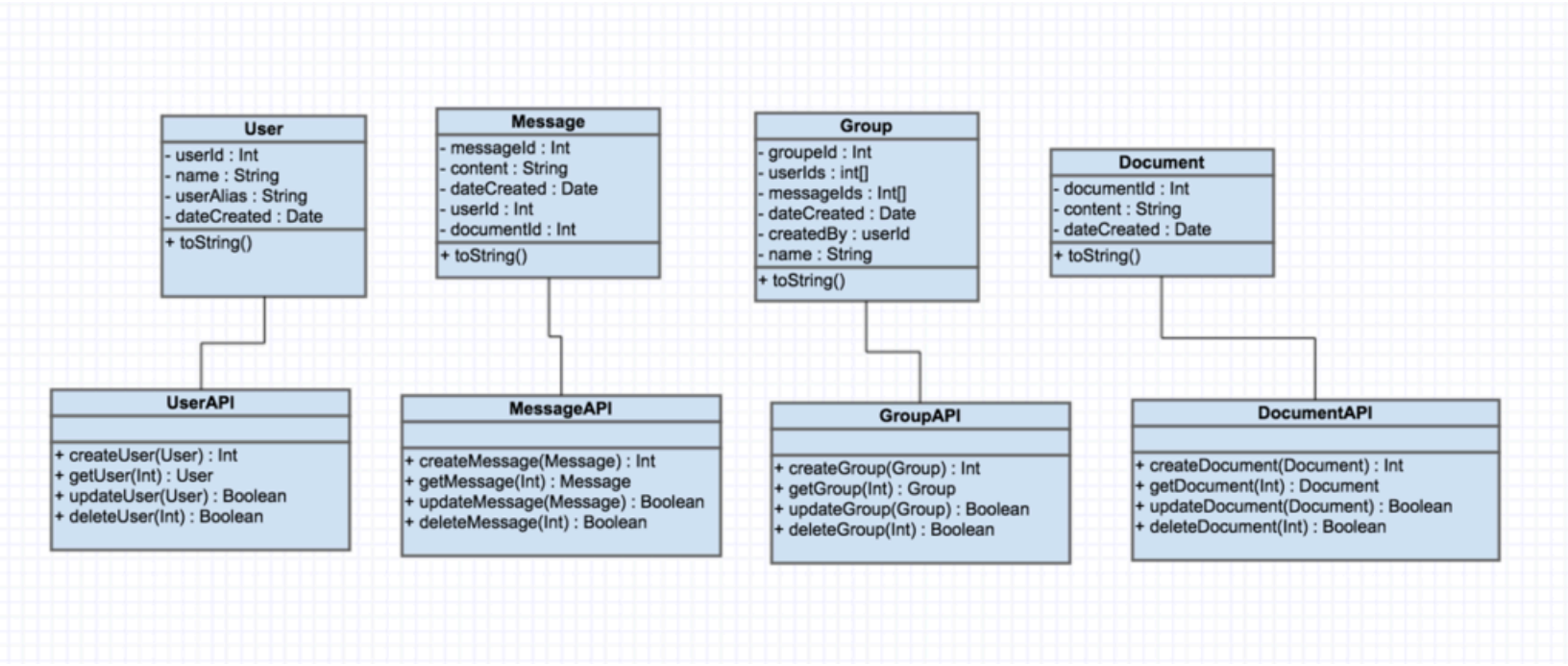
3.2 Rationale

In order to build a service that can handle our client's needs, the API will be RESTful in architectural design. This design allows clients to construct HTTP requests with the URLs in their Web browser. This provides a simple method for clients to manipulate resources and transition to different states. Since it is assumed that the client developers are familiar with JSON, our service will use this as its format. JSON will provide a simple way to read, write and parse the resources. The stateless constraint between clients and server enables greater scalability since the server does not have to maintain the session state. The cache constraints make the system more efficient by only allowing explicitly stated resources to be stored. Hypermedia allows client and server to evolve independently. In addition, it allows resources to manipulated by any client. Through this implementation, the system as whole is simplified and made more efficient.

3.3 Conceptual (or Logical) View

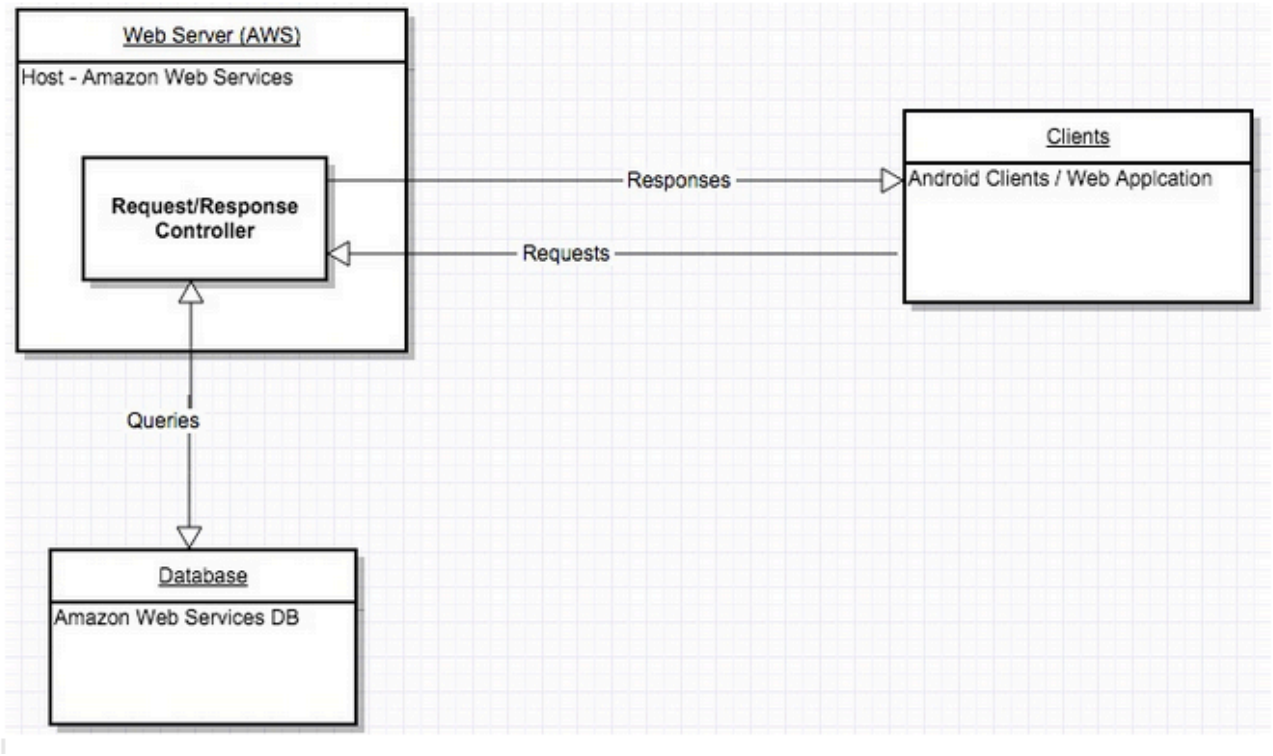
4 Low Level Design

4.1.1 Class Diagram



4.2 Sequence Diagram

4.3 Component Diagram



5 User Interface Design

User Interface is the API Documentation located in our README.md
URL : https://github.gatech.edu/cs3300/p1server/blob/master/README.md