SUMMARY REPORT

HIV WORLD PREVALENCE ON CONTROL

SCORE Contest 2013

Network Fetcher
Tao Cheng, Pucong Han, Shujian Bu, Nanzhu Wang

January 1, 2013

Columbia University
Department of Computer Science
Fu Foundation School of Engineering & Applied Science

www.columbia.edu

ABSTRACT

This document describes the summarized software engineering activities of our project HIV world prevalence on control, which is a participating project for the student contest on software engineering (SCORE) 2013.

CONTENTS

1 Team

We are a team of four graduate students at Columbia University. Our knowledge varies widely among Data Visualization, Data Mining, Graphical Programming, and Social Media. However, we were all inexperienced in project management. None of us had been part of a software development process before. We choose the team name Network Fetcher. The reason for us to pick this name is that we fetch existing data on the Internet, and use the data to tell a story about HIV Prevalence. The members of the team include:

Pucong Han (Major Contact)

- Master of Science Candidate in Journalism and Computer Science
- Columbia University
- Email Address: ph2369@columbia.edu
- Pucong is responsible for the maintenance of the framework, Ruby on Rails. He configures the developing environment and builds the database MongoDB with Tao. He mainly devotes to the back-end development, and also takes part in the controller to connect the view module and model module. Further, he has been the communicator of the team.

Shujian Bu

- Master of Science Candidate in Journalism and Computer Science
- Columbia University
- Email Address: sb3331@columbia.edu
- Shujian, interested in social media and front-end data visualization, has been working for building the website of HIV Prevalence in HTML and CSS, and developed the data visualization of HIV data in pair programming with Nanzhu. Moreover, she has been responsible for keeping the records of team meetings.

Nanzhu Wang

- Master of Science Candidate in Computer Science
- Columbia University
- Email Address: nw2260@columbia.edu
- Nanzhu is responsible for the interaction between the HIV Prevalence website and some public web portals. She also designed and implemented the data visualization: Choropleth Map, Line

Chart, Bubble Chart, in JavaScript in pair programming with Shujian.

Tao Cheng

- Master of Science Candidate in Computer Science
- Columbia University
- Email Address: tc2569@columbia.edu
- Tao's primary interest lies in database technologies and software engineering process, collected the HIV sample data and transferred data to the database MongoDB with Pucong. He is experienced in the software engineering documents writing and testing.

The project is being performed in conjunction with the following academic course:

- COMS W4156: Advanced Software Engineering
- Instructor: Swapneel Sheth
- Email Address: swapneel@cs.columbia.edu

2 Development Process

This section covers the development process including a timeline of real events. The development process explains the used methods and principles in software engineering.

2.1 Intended Development Process

Our development process was based on what we learnt from our Advanced Software Engineering course at Columbia University. We were graduate students lived on campus. It was easy for us to work together on specification, design, implementation and testing. Our team decided to use waterfall model among several development process models, such as the agile development.

The waterfall model was a steadily downwards sequential design process that involves discrete developmental stages, such as specification, design, implementation, testing and maintenance. In a strict Waterfall model, after each phase was finished, it proceeded to the next one. Reviews might occur before moving to the next phase which allowed for the possibility of changes. Though it brought "inflexibility" to the software development, it best fitted our demands and allowed us to develop an application in a short period of time.

2.2 Actual Development Process

Some necessary changes were added into the development process (Table 1): Even though we completed software design before the deadline, there were still some necessary changes based on the feedbacks from the users added into the software design when we were implementing the application.

Furthermore, extreme programming and pair programming in Agile software development process were applied in our actual development process, because we had relatively short time to implement the application.

Step	Start Time	End Time	Note
Requirements Specification	Oct 11, 2012	Oct 18, 2012	-
Software Design	Oct 18, 2012	Oct 31, 2012	Design changed according to feedbacks.
Implementation	Nov 8, 2012	Nov 19, 2012	-
Testing	Nov 29, 2012	Dec 12, 2012	-
Deployment	Dec 20, 2012	Dec 21, 2012	-
Maintenance	Dec 22,2012	-	-

Table 1. Development Schedule

	Scenario #1	Scenario #2	Scenario #3
Title	Browse Interactive Choropleth World Map	Compare Different Factors in a Selected Country	Compare HIV Infection Among Countries
Priority	High	Medium	Low
Difficulty	High	Medium	Medium
Primary Actor	General User	General User interested in a particular country.	General User interested in Comparison
Precondition	A user opens the website.	A user selects a country.	A user selects a Country Comparison Mode
Postcondition	A user browses the Interactive Choropleth World Map.	A user browses the information in a selected country.	A user is able to compare the HIV infection among countries.

Table 2. Use Scenarios

3 Requirements

3.1 Problem Statement

December 1st is the World AIDS Day. In order to raise the public awareness of the HIV/AIDS prevention and find out effective approaches to alleviate its spreading, our team decides to implement an N-tier web application to visualize related data. By mapping groups of data to a graph, our application should be able to tell a story about recent trends of HIV/AIDS and potential correlated factors. By digging into open data, such as The World Bank data and The Open US data, our application is able to promote government transparency and facilitate the policy makers.

3.2 Requirement Elicitation

Our requirements have already been outlined in the document called Project Proposal. And we then gave it more detailed insights into these requirements, and made a new document called Project Requirement in which specific requirements are listed with user cases. The user cases helped us greatly in the process of implementation because they provide some non abstract implementation idea of the application.

3.3 Requirement Specification

Three scenarios are listed in the requirement documents (Table 2), describing how to use our application to view and compare the HIV Prevalence in the world. The following table shows the three basic scenarios for our application while the detailed success scenario steps are not presented in the table.
We also present 5 user cases to describe the user experience for our application, and these user cases cover the aspects of viewing World HIV Infection, interacting with country information, reading related news and pictures, making comparison in different time zones and making comparison among different countries.

4 Architecture

The following section will briefly overview the architecture of our software.

4.1 Overview

Our software has one major component. It is a web application which is split into three tiers (Figure 4.1):

- Models for eight factor data tables and one user table
- Controllers for the eight factor data tables and user management
- Views (Data Visualization for JSON Data)

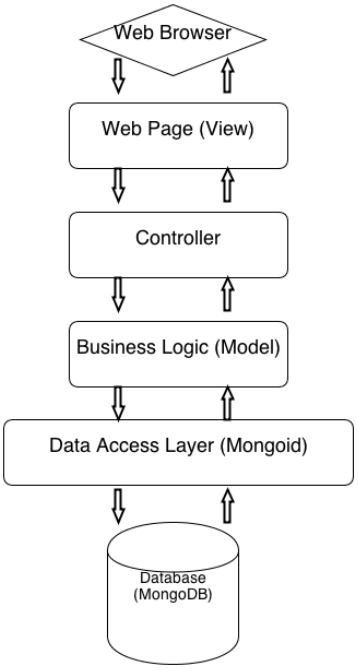

Figure 4.1 An overview of the web application component

Model Layer

Model represents the rules to manipulate data of web application. Each table in a database usually corresponds to one model in the web application. The model will set up the rules of interaction with a corresponding database table, such as restricting the access of table fields and validating variables.

Our application has eight data tables including data for HIV prevalence, improved water source, rural population, improved sanitation facilities, GDP, literacy rate, unemployment rate and health expenditure. These eight models have three attributes, including country name, year and value. These eight models simply define all attributes are accessible.

Our application also has a user table for user validations and managements. This user table has five attributes, including username, email, based country, priority and encrypted_password. All attributes in this model are accessible. This table defines two additional rules. Attributes must be in presence: username, encrypted password and based country. Attributes must be unique: username and email.

Controller Layer

Controller provides connections between models and views and processes the requests from web browsers, controller accesses models for data and passes data to the views for presentation. Controllers include functions of accessing the table and updating the table.

Our application has one controller HIVdatafetcher for the eight data tables. This controller handles requests from the web browser with various parameters and passes data in JSON format.

Our application also has a user controller. It handles user management requests. The controller allows admin user to view, update and delete user account.

Our application uses devise gem to validate user account. The default devise controller allows users to login/logout their account, create new account and update existing account.

View Layer

View represents the user interface with data from the controller. They include HTML files with embedded Ruby code that perform tasks related to the presentation of the data. Our views imported Google Visualization

libraries to visualize the JSON data files passed from the controller. Our team uses the bubble chart, line chart and the google map for our visualizations. Our front-end view also imports Wiki API, Google News API and Twitter API to get related content and information from the Web.

Our web application has a Model-View-Controller architecture. This major component should be able to access our MongoDB database. Our software need to have a data access layer.

4.2 Data Access Layer

In order to create a proper model of the entities and access our MongoDB database, we used mongoid, an object document mapper (ODM) for MongoDB written in Ruby. According the documentation, mongoid provides a familiar API to Ruby developers who have been using Active Record or Data Mapper, while leveraging the power of MongoDB's schemaless and performant document-based design, dynamic queries, and atomic modifier operations.

5 Technologies

Our design was influenced by the technologies we used. In this section, we will discuss the technologies we used and the reason for us to choose them.

Since we were building a multi-layer web application, we considered Ruby on Rails, Python and Django as our main programming languages. As we were all familiar with Ruby on Rails and open source development tools for Ruby are widely available, we decided to develop our web app using Ruby on Rails. Ruby on Rails framework develops web apps with Model-View-Controller architecture. Such multi-layer web application isolates business logic from the user interface and keeps code DRY (Don’t Repeat Yourself). Ruby on Rails allows developers to use yml files to specify conventions. Rails conventions over configuration decrease the number of decisions that developers need to make, gaining simplicity, but not necessarily losing flexibility. Web apps developed using Ruby on Rails are tightly integrated in a number of open source IDEs, such as Aptana.

Our team decided to use MongoDB as our backend database. MongoDB is schemaless. Developers do not need to migrate our database. Data in JSON format can be directly feeded into such schemaless database. One important reason for our team to choose MongoDB is that our front-end visualizations require data in JSON format. Our data access layer can directly get data in JSON format from our MongoDB database. Our database contains eight factor data (HIV prevalence, improved water source, rural population, improved sanitation facilities, GDP, literacy rate, unemployment rate and health expenditure). Our database also has a user table that contains user information.

Our team used the standard web technologies, such as HTML, CSS and JavaScript in the front end. Most of our visualizations are developed using existing libraries in JavaScript, such as Google Visualization.

5.1 Libraries and Frameworks

For the front-end developed we included Google Charts Tool as our visualization li-

brary. Visualizations, including the bubble chart, the line chart and the interactive choropleth world map, are implemented using this existing library. We also imported Google News API, Youtube API and Twitter API to get real time feeds related to HIV. To get real time feeds about a selected country or area, our team also imported Wikipedia Mobile API.

On the backend side we imported many libraries and plugins from the Ruby Gems Organization (http://rubygems.org/). We used the default bundlers to install all included gem plugins and libraries. To make sure that our application works with different application servers with updated libraries, developers can update all included framework components using bundle install command.

5.2 Development Tools

To clean our data source files, our team uses Python and Visual Studio to organize our JSON data files. We also use Sublime to clean unexpected characters from the source files. Completed JSON data files will be feeded into our MongoDB database using the rake db:seed command.

To build our project, our team uses a number of open source integrated development environment, such as Aptana and Netbeans. We also use a number of tools and editors, such as TextWrangler and Sublime. Controllers and models are generated using command line tools, such as rake (ruby build program).

Our team used RSpec for writing unit tests. For generating random feeds and user accounts for testing units, our team used FactoryGirl gem. For the automatic execution of this, our team used Guard.

5.3 Version Controls

Our web application is under GitHub version control (https://github.com/puconghan/HIVDataFetcher.git). Our application also has a remote repository on Stash, one of the git management interfaces (https://ph2369@ase.cs.columbia.edu/stash/scm/TEAM1/project.git).

6 Design

Since our web application visualizes eight factors in about 40 years ranges, we group data in various years to eight different models. An important aspect of the application is reusability and accessibility of these factor models. This requirement leads to the following design of models and controllers.

Factor Models contains information about the eight different factors (HIV prevelence, GDP, improved water source, rural population, improved sanitation facilities, literacy rate, unemployment rate and health expenditure) from 1960 to 2011. These eight models have three attributes, including country name, year and value. The main controller can access any attributes of these eight models after passing the validations.

User Model contains information about the user account. This model has five attributes, including username, email, password, based country and priority.

HIVdatafetcher Controller provides connections between the eight models and views. It handles request from the web browser and respond with JSON data files.

User Controller provides connections among the user model, the user validation view and the user management view.

6.1 Class Diagram

A class diagram in the UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations or methods, and the relationships among the classes. In our HIV Prevalence Overview system, the roles of the model, view and controller are well-defined by the Ruby on Rails MVC framework.

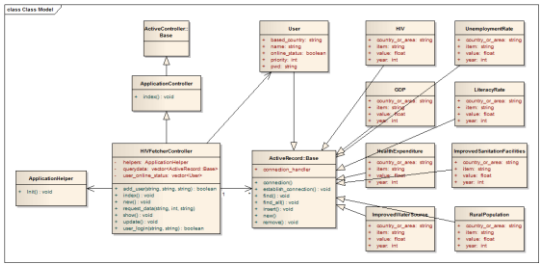

Figure 6.1 Class diagram

In our class diagram (Figure 6.1), the data are stored as a set of classes derived from ActiveRecord::Base, namely HIV / GDP / ImprovedWaterSource / RuralPopulation / ImprovedSanitationFacilities / Literacy Rate / UnemploymentRate / HealthExpenditure. The ApplicationHelper class contains a set of utilities functions, the HIVFetcherController class is the main controller of the system, which handles requests from the client-end and keeps track of user online status. It is derived from ApplicationController, the default controller of a Ruby on Rail application. The ApplicationController class is based on ActiveController::Base. Besides we have the User class to store the user and administrator information.

View

Since our view is mostly HTML, CSS and JavaScript, representing what will be sent to the browser, we have no class for View.

Controller

The controller communicates the models and views together. We have only one controller, because we are designing a single-page application (SPA, a.k.a. SPI). Here's why we choose SPA.

It fits on a single web page with the goal of providing a more fluid user experience akin to a desktop application. In an SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or partial changes are performed loading new code on demand from the web server, usually driven by user actions. The page does not automatically reload during user interaction with the application, nor does control transfer to another page.

Models

Rails comes with a generator for models. The eight model classes are all associated by one-to-one relationship amongst each other. The lines are omitted in order to keep the diagram simple.

Activation Records

We omit ActiveResource::Base since we'll just use it temporarily for connecting to other web services. Also, most of the methods of the classes are not used in our system, and that accounts for our ignoring of those unused methods. We list the necessary approaches that will be implemented.

Based Classes and Variable Types

There are a lot of base classes in Ruby on Rails, each having dozens of methods. Most of the classes like the fixnum, boolean are omitted here. Our MongoDB database will store all information in string.

Password Encryption

Hash functions should be implemented for the Password function. Our application uses devise for user validation. The default devise gem will automatically encrypt all password fields using hash functions.

6.2 Sequence Diagrams

Sequence diagram is kind of interaction diagram shows how objects communicate with each other in terms of a sequence of messages. It indicates the lifespans of objects relative to those messages. The sequence diagram of our web application emphasizes on user interactions with the maps and diagrams, user validations, and user managements.

Check Choropleth Map

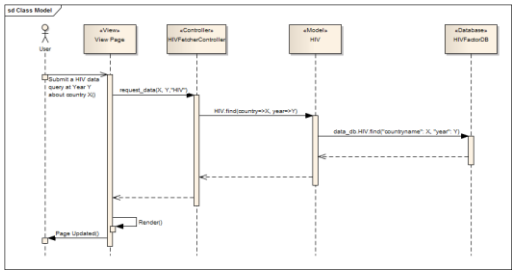

Figure 6.2 Sequence diagram 1

Our eight data factors include HIV, improved water source, rural population, improved sanitation facilities, GDP, literacy rate, unemployment rate and health expenditure. This application only uses HIV data. It requires our application to retrieve HIV statistics, year and country.

User/Admin Login

Normal user login and administrator login are similar to each other. These two login validations retrieve username and password from our user table and check whether the user provides valid login user information.

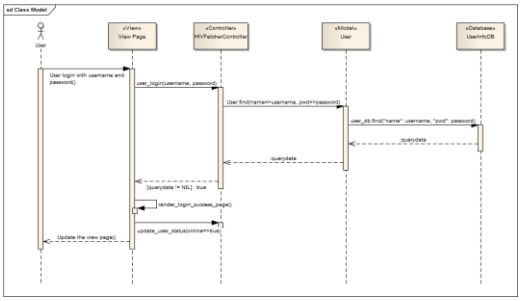

Figure 6.3 Sequence diagram 2

Admin Delete Operation

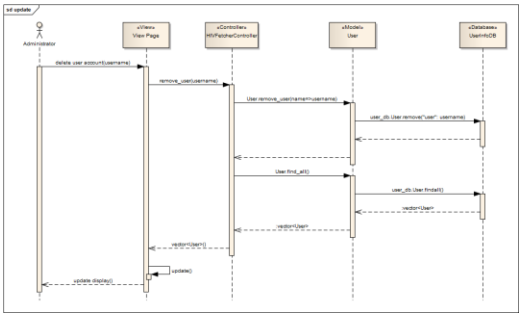

Figure 6.4 Sequence diagram 3

Administrator account has right to delete and update normal user account. Deletion is use-case based. After logging into admin accounts, administrators can have access to user management list. Deletion and update actions will available for administrator. The list of users will be updated.

6.3 State Diagrams

State diagram is a behavior diagram that describes the states and state transitions of the systems. The state diagram of our web application emphasizes on the front-end visualizations.

Browse Interactive Choropleth World Map

This state diagram describes the major Choropleth World Map use-case.

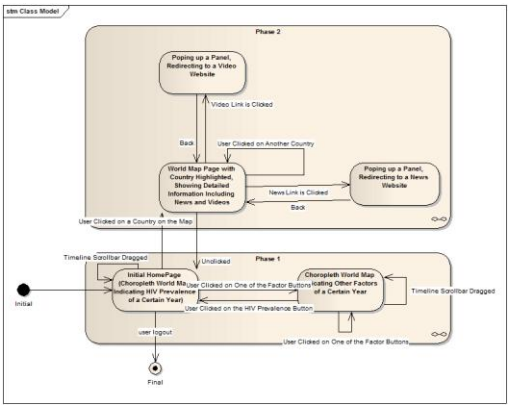

Figure 6.5 State diagram 1

Phase 1 has one homepage state and one state Choropleth World Map Indicating Other Factors of a Certain Year. When a user is in the Choropleth World Map state, he/she can either drag the time sequence bar or click on the factor buttons. These two functions will not change a user's state.

Phase 2 has three states: World Map Page with County Highlighted Showing Detailed Information Including News and Videos, Popping up a Panel Redirecting to a Video Website and Popping up a Panel Redirecting to a News Website. When a user is in the first state (World Map Page with County Highlighted Showing Detailed Information Including News and Videos), he/she can either click the video link to enter the second state (Popping up a Panel Redirecting to a Video Website and Popping up a Panel Redirecting to a News Website) or click the news link to enter the third state (Popping up a Panel Redirecting to a Video Website and Popping up a Panel Redirecting to a News Website). In the second and third states, users can also go back to the previous state. User can also click another country in the first state, and it will not drive user to other states.

Compare Different Factors in a Selected Country

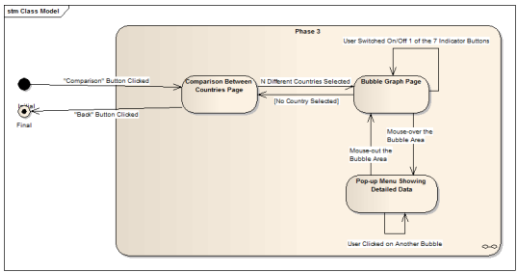

Figure 6.6 State diagram 2

Phase 3 has three states: Comparison Between Countries Page, Bubbled Graph Page and Pop-up Menu Showing Detailed Data. When a user is in the first state (Comparison Between Countries Page), he/she can select n different countries and enter the second state (Bubbled Graph Page). In the second state, a user can either deselect countries to go back to the first state or mouse over the bubbled area to enter the third state (Pop-up Menu Showing Detailed Data). In the third state, a user can either mouse out the bubbled area to go back to the second state or click on another bubble. Clicking another bubble in the third state will not drive user to a new state.

6.4 Use Case Diagram

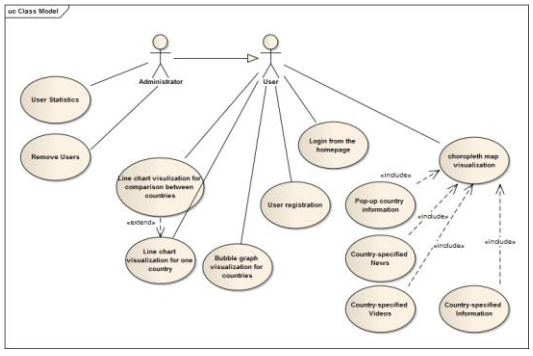

Figure 6.7 Use Case diagram

- Log In (Both admin and user need this first step)

From Users' Perspectives:

- View the choropleth map based on personal preference (based country).
- Click on each country and read the pop-up information.
- Read the news feeds from Google News API.
- Watch the videos from YouTube.
- Read more information about a country from the sidebar.
- Change the factors and view the changing history of a certain factor over the past several years.
- Compare the different trends of several factors.
- View the Bubble chart comparison for multiple countries.

From Administrators' Perspectives:

- Administrators have full accessibility to all functionalities that a normal user has
- Check the statistics of all user activities
- Delete any user accounts

6.5 Component Diagram

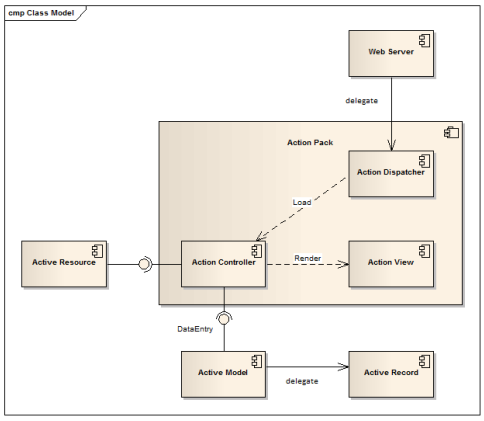

Figure 6.8 Component diagram

In UML, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.

The Action Pack is the central part of Ruby on Rails' MVC framework. The Active Resource and the Active Model are the two major components that we are going to use. Active Resources is a framework for managing the connection between business objects and RESTful web services. Using AR helps us to map different resources (in our case, google APIs and youtube APIs) into objects in Ruby. Active Record is the default Model Component in Rails and is the Base Class for all models that we are going to connect with the Factor database and the User database. It provides ORM, which maps between tables in the database and the classes in the application. In addition, it provides database independence, basic CRUD functionality, advanced finding capabilities, etc.

7 Implementation

During the implementation period we coded the three tiers according to the previous design phase. We created the user interfaces, business logic and database access based on some components that are provided in the Ruby on Rails framework. Introduction of the gems (framework component) and the description of the corresponding services being used in our application are included as follows.

Action Pack

Actionpack is a gem containing three modules - Action Controller, Action Dispatch, and Action View modules that help constructing the View-Controller part of the MVC structure. The three modules corres-

pond to services as Session Management, Redirection Management and HTML template handling. Other services including concurrency, naming service, trader service, scaffolding are also included in Action Pack. This is the major component of the MVC framework served as a web application infrastructure of our system.

Actionmailer

Action Mailer provides mailing services and is used to send emails from the application using a mailer model and views. In our project we created a mailer in app/mailers that inherit from ActionMailer::Base in order to send 'Reset Password' emails to users who forget the password. The mailer has an associated view that appear alongside controller views in app/views.

Activemodel

Active Model provides a known set of interfaces for usage in model classes. They allow for Action Pack helpers to interact with non-ActiveRecord models. Active Model also helps building custom ORMs/ODMs for use outside of the Rails framework. Active Model provides basic data management services like data integrity, transaction management and persistence. In our application we created users and factors based on ActiveModel::Model and we used Mongoid as a non-ActiveRecord mapper. Activemodel constitutes the Model part of our architecture.

Mongoid

Mongoid is an Object-Document-Mapper (ODM) for MongoDB written in Ruby. We used Mongoid for MongoDB's schemaless and performant document-based design, dynamic queries, and atomic modifier operations (CRUD). In our project we used MongoDB as a schemaless object database for data storage and Mongoid as a bridge between active model and the MongoDB. Mongoid provides persistence-as-a-service and data integrity.

Json/multi_json/bson/bson_ext

Mongoid has dependencies on the json, multi-json, bson, bson-ext gems. These four gems provides supportive services (serialization, data management and cache optimization).

Country_select

Country_select provides a simple helper to get an HTML select list of countries. The list of countries comes from the ISO 3166 standard. We rendered country selection controls in HTML pages by using this gem.

Sundawg_country_code

Since countries in our MongoDB database are saved in the ISO3166 Country Code. We used this sundawg_country_code gem to manage ISO 3166 Country Names and Codes, ISO 639 Languages, and ISO 4217 Currencies. We used the translation service to find corresponding country names (for display), alpha-2 codes (Google API) and alpha-3 codes (WorldBank datafiles).

Devise

Devise is a flexible authentication solution for Rails based on Warden. It is Rack based completed MVC solution based on Rails engines. We use device to validate normal users and admin users. Devise allows users to have multiple roles signed in at the same time. Devise is composed of 12 modules, including Database Authenticatable, Token Authenticatable, Confirmable, Recoverable, Registerable, Rememberable, Trackable, Timeoutable, Validatable and Lockable. Besides authentication, devise also provides a bunch of security services including encryption (here we used SHA-1

as the password hashing function), filtering, session management and access control. In our web app, we used five modules including:

- Recoverable - resetting the user password and sending reset instructions.
- Registerable - handling signing up users through a registration process.
- Rememberable - managing generating and clearing a token for remembering the user from a saved cookie.
- Timeoutable - expiring sessions that have no activity in a specified period of time.
- Validatable - providing validations of username and password (default validation is to validate email and password, but we customized it to validate username and password). User will not be able to visit any visualization and user account pages before a successful login.

Sass-rails

This gem provides official integration for Ruby on Rails project with the Sass stylesheet language (CSS).

Jquery-rails

This gem provides jQuery and the jQuery-ujs driver for our Rails 3 application.

Rspec-rails

RSpec is testing tool for the Ruby programming language. It enables a test-driven development with command line programs, textual descriptions of examples and groups (rspec-core), flexible and customizable reporting, extensible expectation language (rspec-expectations), built-in mocking/stubbing framework (rspec-mocks). We installed this gem for testing purposes.

Mongoid-rspec

Mongoid-rspec is a RSpec matcher for Mongoid models, including association and validation matchers. We installed this for testing purposes.

Factory_girl_rails

Factory_girl_rails provides integration between factory_girl and rails 3 (currently just automatic factory definition loading). factory_girl is a fixtures replacement with a straightforward definition syntax, support for multiple build strategies (saved instances, unsaved instances, attribute hashes, and stubbed objects), and support for multiple factories for the same class (user, admin_user, and so on), including factory inheritance. This gem provides naming and life cycle services. We used devise in our project and devise has dependency on this gem.

Tlsmail

This library dynamically replace net/smtp and net/pop with these in ruby 1.9 and enables pop or smtp via SSL/TLS. In our application, tlsmail enables pop or smtp via SSL/TSL for the devise Recoverable module.

Activeresource

Active Resource connects business objects and Representational State Transfer (REST) web services. It implements object-relational mapping for REST web services to provide transparent proxying capabilities between a client (ActiveResource) and a RESTful service. This is part of the default components of our Ruby on Rails Framework.

Activesupport

Active Support is a collection of utility classes and standard library extensions that were found useful for the Rails framework. These additions reside in this package so they can be loaded as needed in Ruby projects outside of Rails. This is part of the default components of our Ruby on Rails

Framework.

Warden

Warden provides a service for authentication in Rack based Ruby applications. It's made with multiple applications sharing within the same rack instance in mind. In our project, devise requires warden gem to be installed for authentication purposes.

I18n

I18n provides internationalization and localization services in Ruby. Features of i18n including translation and localization, interpolation of values to translations, pluralization, customizable transliteration to ASCII, flexible defaults, bulk lookup, lambdas as translation data, custom key/scope separator, custom exception handlers, extensible architecture with a swappable backend. We used pluralization feature, translation and localization feature in our web application.

Our team also used a number of other front-end libraries and web services in our web application:

- JQuery
- Google JavaScript
- Wikipedia API
- Google News API
- YouTube API
- Twitter API

8 The Application

All in one the application provides the following functionality, below is the basic layout.

Figure 8.1 Basic Application Layout Before Login

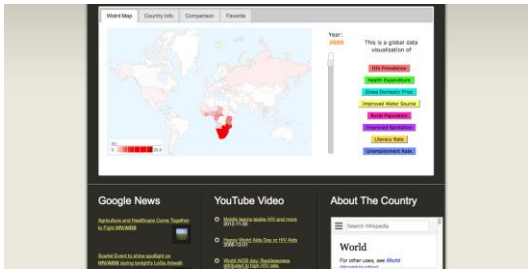

Figure 8.2 Basic Application Layout After Login (HIV in 2005 Default Setting)

Users (Figures 8.1-8.11 Displayed In Order)

- login/logout and get his/her favorite country;
- play around with the choropleth map, which displays every country's 8 factors, such as HIV, GDP, Health Expenditure, Improved Water Source, Rural Population, Improved Sanitation, Literacy Rate, and Unemployment, from 1960 to 2010;
- user can read related Google News, watch YouTube Video and read Wiki Info according to selected country;
- check one country's detailed information on one line chart, and switch between countries;
- compare up to 10 countries on one bubble chart;
- change his favorite country and check the page change.

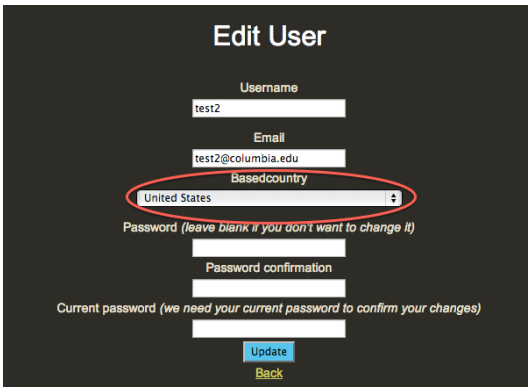

Figure 8.3 User Changing His/Her Profile

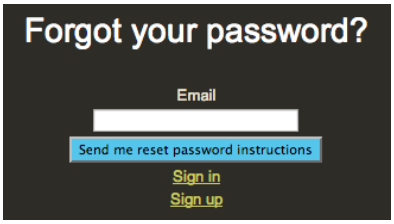

Figure 8.4 Reset One's Password and Retrieve From Email

Figure 8.5 Read News, Watch Videos, and Browse Country Information

Figure 8.6 Read Live Twitter Feed

Figure 8.7 Choropleth Map
(Global Health Expenditure in 2005)

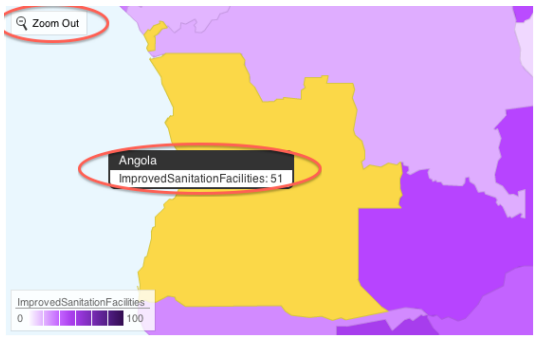

Figure 8.8 Zoom in to Angda
(Improved Sanitation in 2010)

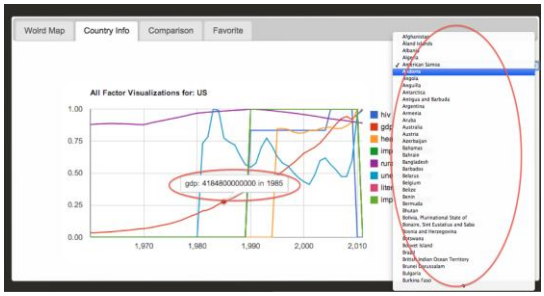

Figure 8.9 All 8 Factors on one Line Chart
(Country U.S.)

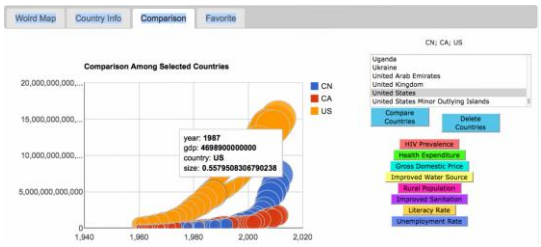

Figure 8.10 Bubble Chart
(Comparison of China, Canada, and United States)

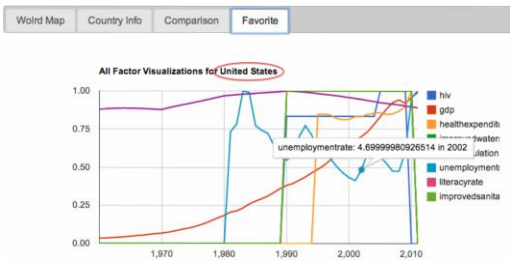

Figure 8.11 Favorite Page

Admins (Figures 8.12-8.14 Displayed In Order)

- admin can create, read, delete and modify users' personal data
- admin can check users' activity log

- admin can raise users' priority, which means change that user from common to admin

Figure 8.12 Administrator's Page

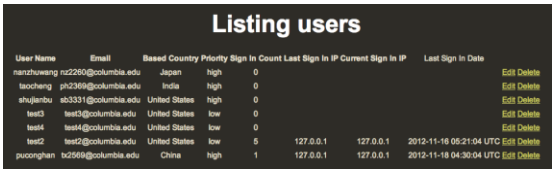

Figure 8.13 List & Manage Users

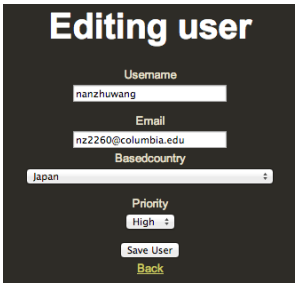

Figure 8.14 Editing Users - Administrator

Others (Figures 8.15-8.16 Displayed In Order)

- There are three static pages - "Data", "Stories", and "Contact" in light of UI/UX. Information including the data source, latest updates is shown on the 'Data' page, as shown below:

Figure 8.15 Data Source

- 'Stories' page keeps track of our findings. Half of our team members are journalists and are interested in digging into the stories of HIV Prevalence. On this page, they can update their latest feature stories. Our team information is listed on the 'Contact' page, as shown below:

Figure 8.16 Stories & Findings Page, and Team Contact Page

9 Verification & Validation

In order to test the correctness of our application, we use Rspec as the Ruby language unit testing tool and JMeter as the stress and load testing tool. To explain the abbreviation in the following text: the User model consists of five string attributes: Username (U), Encrypted_Password (P), Based_Country (C), Email (E), and Priority (R). The HIVDataFetcher Controller consists of four string attributes: Country (C), Year (Y), Value (V), and Factor (F).

9.1 Black Box Testing

For black box testing, we use the equivalence partitioning method and boundary value analysis for creating test cases. The black box tests are implemented using RSpec. For models, our group uses the basic lambda{}.should/should_not raise error test blocks. Inside the bracket, we call ModelName.create!() function, @ObjectName.update_attributes() function, @Object.destroy functions with specific parameters listed in above partitions and boundaries. For controllers, we user GET index tests include response.should be_successful to check whether the json response is valid when passing the above parameters to con-

troller. Our test results, as shown in Figure 9.1, captured one bug: password minimum length is not eight as designed.

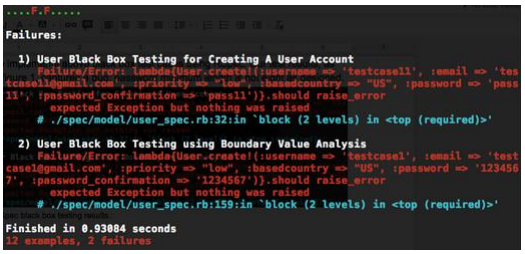

Figure 9.1 RSpec black box testing results

9.2 Metamorphic Testing

Our group implements the first metamorphic test using RSpec lambda{}.should/should_ not raise error test blocks. We use both the create!() function and the update_attributes() function. See from figure 1, the rest 10 tests including the first metamorphic got passed and did not find bugs. We test the second metamorphic test by passing customized input parameters to the URL (hivdatafetcher. json?year=x) to request for JSON data files and check whether the returned JSON file is valid. We found that by incrementing or decrementing year that initially between 1960 and 2011 by 52, the output is invalid JSON files f(x') = 0. The second metamorphic test is passed. No bug found.

9.3 White Box Testing

Our group tests our HIVDataFetcher Controller using the white box testing since it consists a number of if-statement branches in the hivdatafetcher method to handle URL requests with different types of parameters. Our group will execute all if-statement branches of logical decisions by passing various parameters using RSpec.

Before implementing our tests, we implemented user validation initializations in devise.rb and mongoid.rb saved in RSpec folder. These initializations enable the controller to validate user before running white box tests. Otherwise, the test cases cannot pass the user validation. Our group implements four GET index tests with various parameter inputs. The first GET index test has parameters Y == null and F == null and C == null. The second GET index test has parameters Y != null and F != null and C == null. The third GET index test has parameters Y == null and F != null and C != null. The fourth GET index test has parameters Y == null and F == null and C != null. For each of the test, we set the response :format => :json. First of all, our test includes response.should be_successful to check whether the response is valid. After this test we will parse the body from the response and check whether the three fields (country_or_area, year, and value) are included in the body of the json response by using body.should include any test function. These four tests are combined with black box testing in the RSpec tests. As we can see from figure 1, the rest 10 tests including the four white box tests got passed and did not find bugs.

Before implementing our tests, we implemented admin user validation initializations in devise.rb and mongoid.rb saved in RSpec folder. These initializations enable the user model to validate admin users before running white box tests. Otherwise, the test cases cannot pass the user validation. Our group implements the three white box tests in a new RSpec User_Controller.rb file. It checks whether the responses from the three methods including manage, update, and dis-

play succeed. For all three methods, we use response.should be_success test. But, for update and manage, we need to pass parameters (U, E, C, R and P). These three tests are combined with black box testing in the RSpec tests. As we can see from figure 1, the rest 10 tests including the three white box tests got passed. No bug found.

9.4 Security Testing

For authorization of particular actions, test whether users can access methods and functions inside the HIVDataFetcher Controller with and without normal user or admin user validations, such as requesting V by passing different C, Y, and F inputs. We can simply comment out the [before :each] user validations in the RSpec Controller. The Previous RSpec gives an error, as shown in Figure 9.2. Missing user validations, the previous Controller tests fail to request V. Adding it back, all white box test will work as usual. The user validation works.

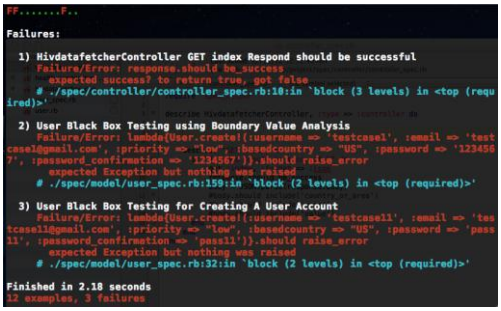

Figure 9.2 RSpec test results without user validation

Apart from the security testing at the server side, we also tested at the client side with security attack approaches, for example, SQL injection. The login page clearly rejected incorrect input and prompted an input error alert. However, if the user tries to input an incorrect URL, the server will return a 500 error rather than covering everything up. It is also possible that the server exception message contains part of the source code, which violates the security requirement of the application.

9.5 Stress Testing

For the stress testing part we use JMeter 2.8, which is a free Java-based tool for load testing and stress testing client-server applications.

We create a Thread Group with various number of users (1, 10, 100, 500, etc) and send HTTP Requests to our server (which is 127.0.0.1:3000, and we post login parameters to the user login page at /users/sign_in). To check the correctness of the response, we assert that the retrieved page has a constant size (4051 bytes). For test cases with threads no more than 100 the application works fine. However, the 500 thread test case shows that some users have not received the complete response (see the following figure), which means they will get a 'The Page Cannot Be Displayed' message.

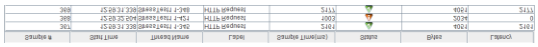

Figure 9.3 Stress Testing JMeter 2.8

The following graph shows the responding time (raw data are random black dots, averages are in blue, deviations are in red) behavior while the number of users grows from 1 to 500.

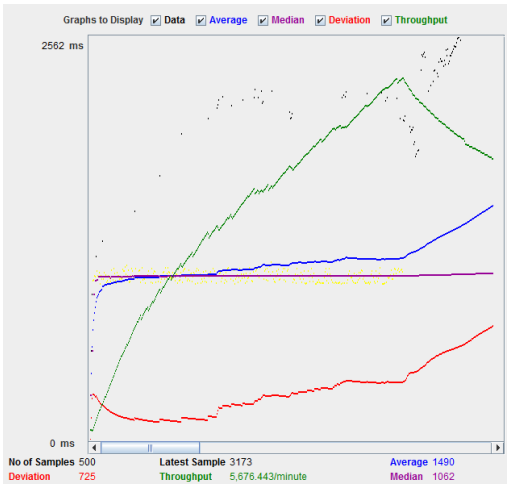

Figure 9.4 Responding Time with User Increase

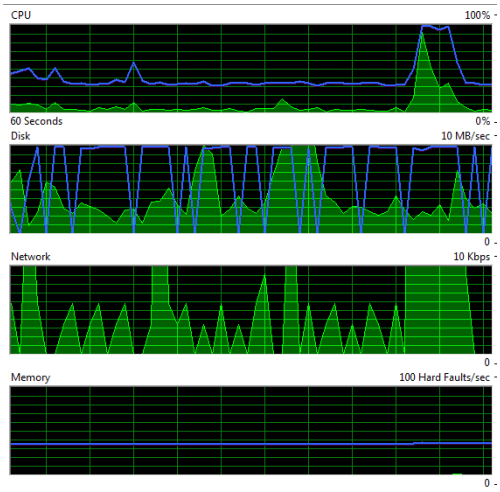

Figure 9.5 Resource Monitor View

Clearly as the number of user exceeds 400, the average responding time and deviation grows rapidly, showing that the latency increases on the server. Despite the network throughput, the memory performance does not change much (see the following resource monitor view, in which the stress testing starts from the last two columns in the grids):

Apart from JMeter, we also tried HP LoadRunner v11.00 (in a 10-day try-out license), the professional stress testing software for our application. However LoadRunner is not designed to test a local server/database and LR itself takes up too much memory and CPU resources, which heavily affects the calibration.

9.6 Mutation Testing

In order to evaluate the quality of our previous RSpec tests, our team designs a number of mutation tests. These test modifies our testing source codes in small ways, such as using wrong operators or changing variable name. Test cases do not detect and reject the mutated code is considered defective.

Mutation tests for checking blackbox password length -> In our Model RSpec test code, we modify the tests of our password length boundary value by increasing the length of the password of the failed case by one. As we can see, from Figure 9.6, RSpec detects and rejects the mutated test cases. These boundary value test cases are not defective.

Figure 9.6 Mutation Test Result - For Boundary Value

Mutation tests for checking controller tests -> In our Controller RSpec test code, we modify our GET index block by changing the action name [get :index, :format => :json] to [get :view, :format => :json]. As we can see from Figure 9.7, RSpec detects and rejects the mutated test cases. The security test case is not defective.

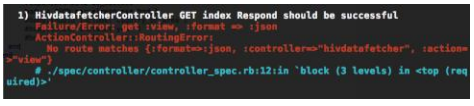

Figure 9.7 Mutation Test Result - For Hivdatafetcher Controller

Mutation tests for checking security uniqueness and presence tests -> In our Model

RSpec test code, we modify our uniqueness and presence tests by passing fields that are not presence, such as based_country, and not unique, such as priority. The mutated test cases are: it {should validate_uniqueness_of (‘based_ country’)} and it {should validate_presence _of(‘priority’)} These two mutated test cases are included in the RSpec user model tests. As we can see from Figure 9.8, RSpec detects and rejects the mutated test cases. These security test cases are not defective.

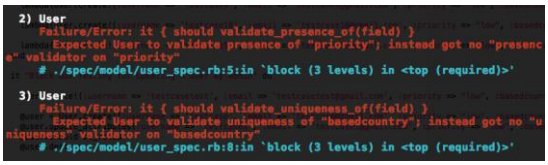

Figure 9.8 Mutation Test Result - For Uniqueness and Presence