TRANSITION - FINAL DOCUMENT

Lake Mountain Leprechauns

RELEASE NOTES

== 0: Introduction ==

Hi! I have some instructions on how to configure the environment for The Con Planner 75 website server, and also some possible resources you can use to configure this environment for yourself.

== 1: Tested Versions ==

Con Planner 75 was developed on Ubuntu 14.04 with Ruby 2.2.3 and Rails 4.2.4, but has been tested with various different versions of things as listed below.

Operating Systems: Ubuntu 14.04, Mac OSX Yosemite, Windows 7
Ruby Versions: 2.1.5, 2.2.3
Rails Versions: 4.2.4

If you have some combination of these environments configured for yourself, that is great. If not, however, I can provide some general resources for getting a working installation and also notes more specific to configuring for use with our project.

== 2: Setting up Ruby and Rails and other dependencies ==

= 2.1: Windows 7 (and possibly later versions also) =

The easiest way to do this is to download RailsInstaller from "http://www.railsinstaller.org/en". You should get the Ruby 2.1 version for Windows. Once you have downloaded this proceed to run the installer.

The default installation settings are generally good. Installing Git is optional, but will provide you with a Unix terminal emulator, so if you prefer that to Windows-style command line go ahead. Chill out for a minute because the installation will take some time.

Next you have to install Node.js, which is as easy as heading over to their website and installing whatever the latest version is. After this you should pretty much be good to go.

The original version of our project used PostgreSQL, but to make installation easier on Windows we have a second version running SQLite. You can try to set up PostgreSQL on your Windows machine, but I do not suggest it. The functionality is the exact same, but you lose the hassle of setting up Postgres in this environment.

= 2.2: Ubuntu Linux =

You should go to this site (https://gorails.com/setup/ubuntu/14.04) and follow the instructions. Remember your username and password for Postgres later because you will need to put them your database.yml file. This site provides options for different versions of ubuntu, so change your version if you need to.

= 2.3: Mac OSX =
Mac OSX users have two options. gorails.com has tutorials on how to install rails on Mac that are similar to their Ubuntu tutorials. RailsInstaller also releases a Mac version of their product. Choose whichever of these options seems most reasonable to you.

== 3. Setting up our project ==
Once you have a working ruby installation, open up your terminal program and navigate inside the folder for our website (whatever version you are using). Run the command "bundle install" in order to download all of the gems needed to run a server for our product.

If you are not using the SQLite version, you will need to put your Postgres username and password in the "database.yml" file, located in the "config" directory within the application's folder.

When you are done, run the command "rake db:setup" in order to populate the database with an initial set of values. This is just included so that the database is not empty at start and you can look at examples of conventions.

== 4. Running our project ==
(Windows only) Start your Node.js application.

Open a terminal and navigate to the site's directory.

run "rails s"

browse the site

press <ctrl>-<'c'> from terminal when finished to shut down server.

== 5. Miscellany ==
You will have to create your own user (or multiple users) on the site, but once you create them you will no longer have any way to find or reset their passwords. This would be implemented in a true release of the product, but was not a core feature that we were trying to develop that related to the unique functionality of our project.

You can, however, clear the entire database. Running "rake db:reset" will accomplish this for you. Unless you want to get into the nitty-gritty of the rails command line you should use this to

remove things from the database if you mess something up. You will lose all of your data, but it can be well worth it if you have filled your application with lots of testing conventions.

== 6. Mobile Application ==

Our mobile application is downloadable and installable as an Android .APK file. It's been tested on Android 5.1.1 and 5.0 and has no explicit compatibility restrictions or dependencies. It should be able to run in any given Android OS, but Android 5 is recommended.

Installation is simple. Download the .APK (app-release.apk) and double tap to run/install the application. You must allow for third party app installation in your phone settings.

In order to connect to the server, first connect your phone to your computer. Open Google Chrome and type: "chrome://inspect/" into the top bar. You should see your phone in the device list. Select the "port forwarding" button, then add "3000" in the port field, and "127.0.0.1:3000" in the IP address and port field. Check off "enable port forwarding" at the bottom and you should be all set! You can test that the connection is sound by opening up a web browser on your phone and going to "localhost:3000". You should be able to see the Con Management Site as it appears on your computer.

From there, you can navigate the app and use the search functionality to pull information from the web server on your computer. :^)

FINAL TEST RESULTS

First test results

Test Case	Result	Comments	Automated?
TC1	Pass	Email inclusion dropped.	Yes
TC2	Pass		Yes
TC3	Fail	Some fields ok empty. Accepts empty important fields. Accepts invalid times	Yes
TC4	Fail	Removed ability to change name. Some fields ok empty. Accepting empty important fields, Accepts invalid times.	Yes
TC5	Fail	New Precondition: At least one host exists b/c host field is a drop down menu. Accepts empty important	Yes

		fields or crashes poorly. Some empty fields are ok.	
TC6	Fail	New Precondition: At least one host exists b/c host field is a drop down menu. Accepts empty important fields or crashes poorly. Some empty fields are ok.	Yes
TC7	Pass		Yes
TC8	Fail	Allows empty and duplicate entries.	Yes
TC9	Pass		No
TC10	Pass		No
TC11	Pass		No
TC12	Fail	If a conflict exists, the site crashes	No
TC13	Pass		Yes
TC14.	Pass		Yes
TC15	Fail	Button to see rooms or hosts not showing up. Event information not showing up.	No
TC16	Pass	Dropped conflict detection.	Yes
TC17	Fail	Button not showing up.	Yes

Second test results

Test Case	**Result**	**Comments**	**Automated?**
TC1	Pass		Yes
TC2	Pass		Yes
TC3	Pass		Yes
TC4	Pass		Yes
TC5	Pass		Yes
TC6	Pass		Yes
TC7	Pass		Yes
TC8	Pass		Yes
TC9	Pass	Cannot easily verify the correct document programmatically.	Partially
TC10	Pass		Yes

TC11	Pass	Due to unpredictability of the scheduler, we test this by hand.	No
TC12	Pass	Due to unpredictability of the scheduler, we test this by hand.	No
TC13	Pass		Yes
TC14	Pass		Yes
TC15	Pass	UI test. Easier to do by hand.	No
TC16	Pass		Yes
TC17	Pass		Yes

Analysis

The bulk of our failed tests were attributed to a lack of validation input. Most of our functionality passed the initial testing. There are some tests that are not feasible to implement and thus some of the test cases were not fully automated but were verified by hand.

Final Statement

We can say that based on these results, our application meets the robosity of our stated standards and is ready for use.

BEST PRACTICES

Project Management Site and Code Repository

Our code repository and project management were handled by using Github. Here is the link to our repository. Our project used 4 branches and 106 commits total. We also used Google Drive to help assist in our project management as it helped organize our design documents.

Coding Standards

Our java standards follow Google's coding standard. Our Ruby standards follow this guide. Our IDE's helped to enforce our code standards as well.

Object Oriented Design

We are representing our convention and the convention assets as objects. This is facilitated by the model part of our Model-View-Controller design offered by Ruby on Rails.

Unit Testing Tools

Rails offers an automated testing suite that we used to test our website. We used JUnit to test our mobile application.

Mock Objects

We used mock objects in our mobile application testing to remove the dependence on database and website connectivity when testing the application's functionality.

Build Tools

We used Gradle to build our mobile application. This comes standard with Android Studio.

STATUS REPORT

Since our last report, we have focused on finishing and polishing our applications for release. On the back-end we made some minor edits to the web pages for readability and more intuitive use, and also made some organizational edits to the database/JSON files being distributed to the front-end.

The front end received a number of UI edits to improve visual consistency and navigation. We also fixed up the personal schedule functionality, adding a flag to Events in the stored/downloaded Convention data that showed whether or not it was a member of the user's personal schedule. We also remade the pages for the personal schedule to fit both this implementation and the rest of the design of the application.

Finally, we ran various tests on the final release versions of our product (see above), wrote up final documentation, and made our final presentation for class.

CONTRIBUTION SUMMARY

Kyle Samson ran automated tests and documented the test results and best practices. Rachel King wrote the final status report and improved the front-end UI. Michael Mortimer made the final release and wrote the release notes. Maggie Borkowski made the final presentation and wrote the contribution summary.

Submitted in separate documents: presentation powerpoint, code repository