Joseph Owens

Bradley J Clark

Nathan LaVire

PalermoPOS Installation and Maintenance Guide

# Contents

### Hardware Requirements

### Installation

### Database Schema

### View Flow

### GUI Details

### Dependencies

### Testing

### Recommendations for online use

### How to update the software

# Hardware Requirements

Server: Linux compatible hardware

Ram: 512 MB Minimum, 1 GB+ Recommended

CPU: 2.0 GHZ single core Minimum

Storage: 2 GB Minimum

Client: Google Chrome

Database: MySQL

# Installation

Installation of PalermoPOS requires a working knowledge of Linux, and has been primarily tested on Ubuntu 16.04. It will very likely work on other Linux distros with little or no modification. Knowledge of MySQL or other SQL databases is also recommended. Ruby on rails experience would be helpful as well but may not be necessary if you follow the steps provided here.

This is a runthrough of the various parts of the install.

OS Install

1. Obtain the desired version of Ubuntu from <https://www.ubuntu.com/>
2. Install Ubuntu or Ubuntu server on the intended server hardware. Hard drive encryption is recommended.

Install dependencies

Many of the following commands need to be run in the command line (bash). These commands will be denoted with a $ symbol.

To update apt-get:

$sudo apt-get update

To install git:

$sudo apt-get install git

To install curl:

$sudo apt-get install curl

To get a public key for rvm:

$gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3

OR

$gpg2 --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3

If the first command fails, try the second.

To install RVM:

$\curl -L https://get.rvm.io | bash -s stable --ruby

$/home/username/.rvm/scripts/rvm

Close and reopen bash at this point.

To install node.js:

$sudo apt-get install nodejs

To update gems:

$sudo gem update --system

To install bundler:

$sudo gem install bundler

To install webkit prereqs:

$sudo apt-get install libqt4-dev libqtwebkit-dev

To install rails:

$sudo gem install rails --version=5.0.0.1

Install Project Files

Create a folder for the project wherever you intend to install it.

Navigate to that folder in bash.

$git init

$git remote add origin https://github.com/wmuCS4900Project/PalermoPOS.git

$git pull

$git checkout Release-1.0-Snapshot

The checkout version may change as the application is updated further.

Install Project Required Gems

To install other required gems, navigate back to the installation folder.

$bundle install

Install Database

To install mysql:

$sudo apt-get install mysql-server

To run mysql:

$mysql -u root -p

At this point, create a database in MySQL. This database should be named the same name as the database in /install/config/database.yml under the production tag.

* production:
* <<: \*default
* database: db/production.sqlite3
* adapter: mysql2
* encoding: utf8
* database: palermolive
* username: root
* password:
* host: <%=ENV['IP']%>

The database is named palermolive by default but can be changed to whatever database name you choose. Make sure this fill has the updated username and password for MySQL. No other settings need to be changed, unless you choose another database software.

Now you can run the database initialization scripts we have provided (and some that rails provides).

In the command line:

$rake db:migrate RAILS\_ENV=production

This creates the necessary tables for PalermoPOS to run.

In order to initialize necessary data:

$rake install:all

This command will create the following REQUIRED data:

1. Admin user
2. User roles
3. Palconfig records

It will also optionally import from csvs in the /install/dbimport directory.

1. Categories
2. Products
3. Options
4. Customers

Run Server

At this point the server is ready to run.

Either launch it with the provided launch script (which will need to be pointed at the install directory), or you can run ‘rails s’ in the command line to try it locally in development mode.

# Database Schema

The Database schema is fairly straightforward and can be easily understood from the schema diagram. However, there are a few important points to be made and those are detailed here. See schema.png in the Documents folder for the diagram.

Notes on the schema:

Serializations: Several fields are saved through array serialization, into a varchar in the database. These arrays hold the ids for different objects depending on the table in question. Products:freeoptions, for example, holds an array of option ids which are flagged as being included in the product and are built into that product’s cost automatically. Orders:coupons holds coupon ids applied to the order. Deserialization of these arrays happens automatically when a record is selected from the database, and can be treated simply as an array of ids in the form of @array = [‘1’, ‘2’, ‘3’] when dealing with the object itself within the rails code.

# View Flow

We have included a flowchart of the views included in the program in order to more easily understand the work flow.

# GUI Details

PalermoPOS uses a combination of html and css with bootstrap for formatting.

# Role/Capability Validation

The Rolify gem in combination with the capabilities object provides login-based feature gating. Every feature of PalermoPOS is gated behind a string pair denoting the use of a feature, and can only be accessed by users with roles that have been granted access to those features. The ability-object pairs are created by creating a new capability in the capabilities section of the management menus, or inserted into the database as part of the database setup (or inserted through mysql at a later time). This allows both database admins and a manager with the capability to do so to create new restrictions and allowances.

The pair is straightforward. By default, abilities include view, create, edit, cancel, destroy, and all. Objects include orders, customers, users, products, options, coupons, orderlines, roles, capabilities, configurations, management, and all. Each combination of these abilities and objects has specific applications within the program, but assigning a role with ‘all’ ‘all’ gives them the ability to access every aspect of the application, including deletion of records. This should only be given to an admin login and used with care.

The pairs are enforced in each controller call with the following check:

current\_user.can?("create", "users")

This is typically paired with a check for login and a redirect if not allowed. As an example:

if ( !logged\_in? || !current\_user.can?("create", "users"))

redirect\_to root\_path, :flash => { :danger => "You do not have permission to do this!" }

return

end

Every controller call, with the exception of the log in calls, are gated by these capabilities. New string pairs can be added and used as needed with no further modification of the software, beyond adding the new checks in controller calls (or elsewhere) and adding the string pairs to the database.

# Dependencies

PalermoPOS is dependent on the following ruby gems:

gem 'rails', '~> 5.0.0', '>= 5.0.0.1'

gem 'mysql2'

gem "rolify"

gem 'growlyflash'

gem 'puma', '~> 3.0'

gem 'sass-rails', '~> 5.0'

gem 'uglifier', '>= 1.3.0'

gem 'coffee-rails', '~> 4.2'

gem 'bootstrap-sass'

gem 'jquery-rails'

gem 'jbuilder', '~> 2.5'

gem 'bcrypt', '~> 3.1.7'

gem 'cancan'

Gems for testing and development:

gem 'byebug', platform: :mri

gem 'rspec-rails'

gem 'factory\_girl\_rails'

gem 'capybara'

gem 'rails-controller-testing'

gem 'database\_cleaner'

gem 'headless'

gem 'chromedriver-helper'

gem 'capybara-webkit'

gem 'web-console'

gem 'listen', '~> 3.0.5'

gem 'spring-watcher-listen', '~> 2.0.0'

# Testing

Testing of PalermoPOS is done through the Rspec gem. All tests and test configurations are contained in /PalermoPOS/spec.

Tests can be executed by using ‘bundle exec rspec’ at the command line in /PalermoPOS. Individual tests can be executed by using ‘bundle exec rspec ./spec/features/orders\_spec.rb:42’. Replace the path, filename, and line number as needed.

Test database objects are created with the FactoryGirl gem, defined in /spec/factories. FactoryGirl can be called in a test either with specific object data, or by calling traits defined in the factory files.

Example 1: FactoryGirl.create :user, id: "2", username: "user1", password: "user123", Name: "User1guy"

This call specifically assigns data to fields for the object.

Example 2: FactoryGirl.create :user, :one

This call would refer to the trait :one in the factory file, and create a user with data assigned to fields under that trait.

PalermoPOS feature tests utilize Capybara to test html responses and html page behaviors.

If you go through the effort of getting webkit working, you can also run the test which require javascript. This uses the command:

xvfb-run --auto-servernum rspec rspec/feature/SPECNAME.rb

# Recommendations for online use

The first recommendation is to upgrade the Windows XP computers to an operating system that is supported. All computers will need a reliable antivirus program and up to date firewall. All gems files must be updated regularly. The official website for Ruby on Rails security recommendations in found: <http://guides.rubyonrails.org/security.html>.

Rails has build in session management tools and protections. Rails hashes the ids as long as no long items are saved in the session to prevent session hijacking. To prevent session fixation to always issue new session ids with reset\_session.

Recommended gems to have installed can be found: <https://hakiri.io/blog/ruby-security-tools-and-resources>. Bundler-audit, loofah, and codesake::Dawn are all installed already. Other gems are need if the webapp is to be used for online ordering. And always check and keep all gems updated.

# How to update PalermoPOS

If PalermoPOS is running on a server with internet access, updating is easy! Just shut the server software down, and run the following commands at the install root in bash:

$git pull upstream release

$bundle install

$rake db:migrate

After this, you should be ready to start the server back up.

If the server is NOT connected to the internet, updating is a little more complicated. We will provide a diff patch with only the updated files. These files can be dropped into the install folder and overwrite any that prompt to be overwritten. At this point the following can be run from bash:

$rake db:migrate

In the event that an update requires a gem update via bundle install, or another software dependency, the server will have to be attached to an internet connection or the software can NOT be updated.