Table of Contents

[XAMPP 2](#_Toc516665581)

[Directories 4](#_Toc516665582)

[GitHub 6](#_Toc516665583)

[Composer 7](#_Toc516665584)

[Propel 9](#_Toc516665585)

[PHP Pass 11](#_Toc516665586)

[Slim 12](#_Toc516665587)

[Combining All Elements 13](#_Toc516665588)

[Digital Ocean 15](#_Toc516665589)

[Works Cited 16](#_Toc516665590)

# XAMPP

**Overview**

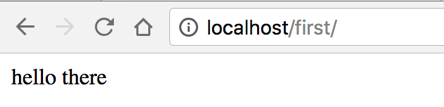
XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use [1].

**Installation**

Download and install XAMPP from <https://www.apachefriends.org/index.html>, version < 8 is recommended for now since the new versions ask for extra permissions. Once XAMPP installs go the *Manage Servers* tab and turn on *MySQL Database* and *Apache Web Server*, then go to [http://localhost](http://localhost/) and you should see the XAMPP welcome page. Dig around your files and find where XAMPP was installed (/Applications/XAMPP/xamppfiles/htdocs/ in mac) and delete index.php. reload localhost and you should now see the directory structure. Make a new directory in htdocs/ named first/ and make an index.php file inside it, with a simple PHP script:



Output:



**Updating PHPMyAdmin DB using SQL File**

- Go to phpmyadmin dashboard & select the database you want to change (left column with “New”)

- Delete the old database

- Click Operations tab and click "Drop the database (DROP)"

under Remove database section

- Confirm by clicking ok

- create a new database with the same name as the database you just deleted

- Click New

- Fill in the name input box

- Click create

- Import the SQL file

- Click import tab

- Click choose file and select the SQL file (should be in data/ directory)

- Scroll down and click Go

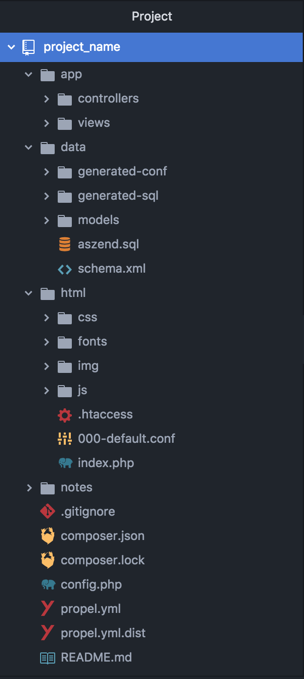
- Go back to project (cd with terminal) and run composer dump-autoload -o

# Directories

**Overview**

The structure that all projects must follow for readability and consistency.

**Structure**



**Structure explanation**

* *app* directory: contains the slim controllers and the views
  + *controllers* directory: Slim route organization (not necessary for small projects)
  + *views* directory: all php views
* *data* directory: contains files concerning the database
  + *generated-conf*directory: contains propel’s config.php with connection configurations
  + *generated-sql*directory: contains default sql dump (no insertions)
* *html* directory: contains files accessible directly to the user (anything that doesn’t require protection)
  + *css/fonts/img/js*directories: files that correspond to css, font, images, and javascripts files in that order
  + *.htaccess*file: used to remove index.php from the url
  + *index.php*file: main entrance point
* *notes* directory: contains any updates of work or explanation
* *vendor* directory: contains dependencies downloaded with composer (not shown in the pic)
* *composer.(json/lock)*file: created by composer
* *propel.yml[.dist]* file: created by propel to define a connection
* *README.md* file: short project description
* *config.php* file: contains the array needed for slim settings, and other functions that should be available globally

**Directory notes**

Window uses \ as a path delimiter, while Unix-based machines use /, for the following commands concerning paths (such as *vendor/bin/propel init*) make sure to use the path separator corresponding to your machine. Keep in mind that the directory structure is used to keep projects organized, and is subject to tweaks, but the main structure will stay relatively similar from project to project.

# GitHub

**Overview**

GitHub is a web-based hosting service for version control using git. It is mostly used for computer code. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features [2].

**Installation**

To install git on your machine follow the instructions at <https://git-scm.com/downloads>. Once it’s installed, run the command *git* in the terminal, and you should receive instructions on how to use git.

**Important git commands**

|  |  |
| --- | --- |
| Command | Explanation |
| git init . | Initialize a Git repository in the current directory |
| git clone repo\_url.git | Create a local copy of a remote repository |
| git status | Check status since last pull |
| git add [file] | Add a file to the staging area |
| git add –A | Add all new and changed files to the staging area |
| git commit –m “commit message” | Commit your messages |
| git push -u origin [branch name] | Push a branch to your remote repository (almost always master) and remember the branch |
| git push | Push changes to remote repository of remembered branch |
| git pull | Update local repository to the newest commit |
| git remote add origin repo\_url.git | Add a remote repository |
| git rm –r [file] | Remove a file or directory |

**Gitignore file**

Create a .gitignore file in the root of the local repository. Git will look at the contents of this file and decide what items to ignore, such as large files not suitable for pushing. For example, a .gitignore file that excludes the large vendor folder from being pushed:



Make sure to **never push vendor/**, as it will cause headaches and reduce efficiency.

**Github notes**

These commands will be mostly used while working in a group project, if you encounter a problem or would like to learn more, go to <https://try.github.io/>. Unlimited private remote repositories are available through a fee, or by registering as a student, which is recommended. Merge conflicts can be a pain to deal with, and therefore should be resolved with atom/sublime or be avoided completely. Make sure to **always pull before you push**, and therefore obligating you to commit any changes.

# Composer

**Overview**

Composer is an application-level package manager for the PHP programming language that provides a standard format for managing dependencies of PHP software and required libraries [3].

**Installation**

Composer can be installed following the guide at <https://getcomposer.org/doc/00-intro.md>, make sure you install composer as a global installation. Once it’s installed, run *composer* in the terminal, and you should receive the composer screen. In case of an error, make sure you follow the steps correctly with admin privileges (e.g. using sudo on a Unix machine).

**Important composer commands**

|  |  |
| --- | --- |
| Command | Explanation |
| composer require package/library | Will update composer.json with the new dependency or create a new composer.json and composer.lock if they don’t exist |
| composer install | Installs the vendor packages according to composer.lock (or creates composer.lock file if not present) |
| composer update | Will regenerate composer.lock with the new composer.json dependencies and versions, no matter if composer.lock exists or not |
| composer dump-autoload -o | Regenerates the list of all classes that need to be included in the project (vendor/composer/autoload\_classmap.php). |

**Example composer.json file**



*Explanation of composer.json:*

**require**: {…} lists the dependencies required for the project (propel, slim, etc...)

**autoload**: {…} lists extras to be included when *composer dump-autoload –o* is ran. classmap allows directories to be included in the autoload process. The psr-4 autoload is used to define the mapping from namespaces to directories. The example filename would be app/hello.php containing an App\Hello class.

**Composer notes**

Make sure that any of the composer commands are ran in the same directory as the composer.json, or the commands will fail. Composer will download the dependencies into a vendor/ directory, which it will create if it doesn’t exist. Every dependency will have a directory, for example, propel will be inside vendor/propel. *composer dump-autoload –o* will **not** work in the previously shown composer.json example if data/models and app/ directories don’t exist. If you want to test it, simply remove everything except require: {…}. **Always run *composer dump-autoload –o* after installing a new dependency** in order to load up the new files, or the new dependencies will go unused.

# Propel

**Overview**

Propel is a free, open-source object-relational mapping toolkit written in PHP. It is also an integral part of the PHP framework Symfony and was the default ORM up to, and including version 1.2 [4].

**Installation**

Require it in composer.json:



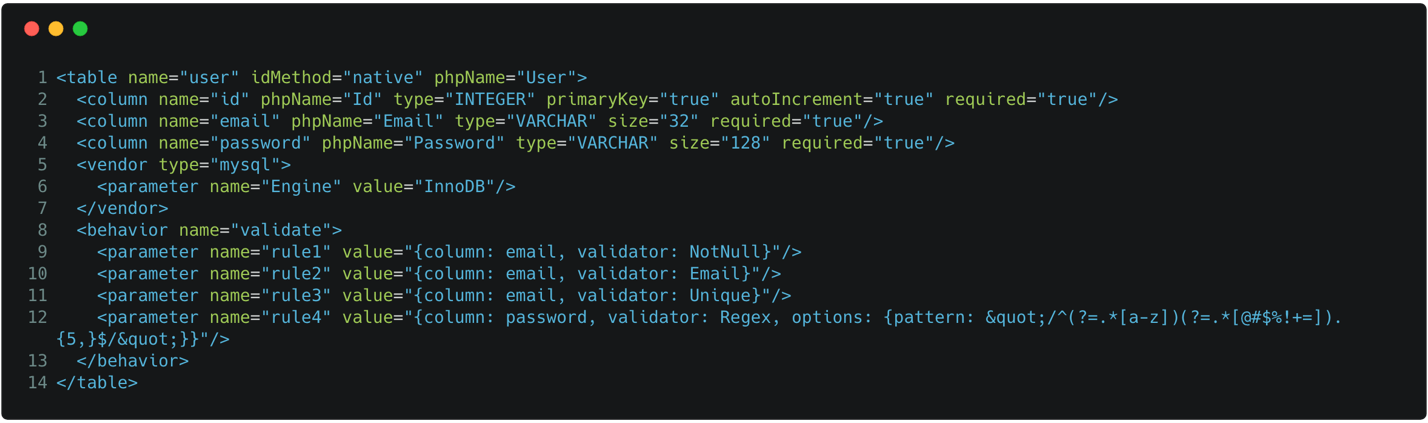
Run composer update to download propel and its dependencies. The dependencies will be downloaded into the vendor/ directory; if you see a propel/ directory in vendor/ you’re set.

**Important Propel commands**

|  |  |
| --- | --- |
| Command | Explanation |
| vendor/bin/propel init | Initialize propel installation |
| vendor/bin/propel reverse | Create a new schema.xml with the updated database information (inside generated-reversed-database/ directory) |
| vendor/bin/propel model:build | Rebuild the models according to schema.xml |

**Validate functionality**

The schema.xml file lives in data/models/ directory, and has the structure of the database. One of the most important additions to schema.xml is the validate behavior:



The validate behavior must go after the vendor section and before the end of the table which you’re adding the validation rules. Once the validation rules are set in the schema.xml, save the file and run *vendor/bin/propel model:build* to rebuild the models according to the modified schema, and then run *composer dump-autoload -o* to load up the new models. The user model will now have new functions, such as validate and getValidationFailures (validate returns true if all rules were followed, and getValidationFailures returns an array of errors if validate failed). If you want to know more about validate you can look at <http://propelorm.org/documentation/behaviors/validate.html>.

**Propel notes**

When running *vendor/bin/propel init* make sure you have your apache server on, or propel won’t be able to establish a connection to the database; an example run of *vendor/bin/propel init* can be found at <https://goo.gl/ej4nt5>. Once you initialize Propel, two new folders will be created in the current directory (generated-conf/ and generated-sql/), move them into the data/ directory to keep the [directory structure](#_Directories_1) consistent. To learn more about propel and it’s important functions look at <http://propelorm.org/documentation/>.

# PHP Pass

**Overview**

PHP Pass is an open source PHP password library designed to ease the tasks associated with working with passwords in PHP. It is capable of generating strong cryptographic password hashes, verifying supplied password strings against those hashes, and calculating the strength of a password string using various algorithms [5].

**Installation**

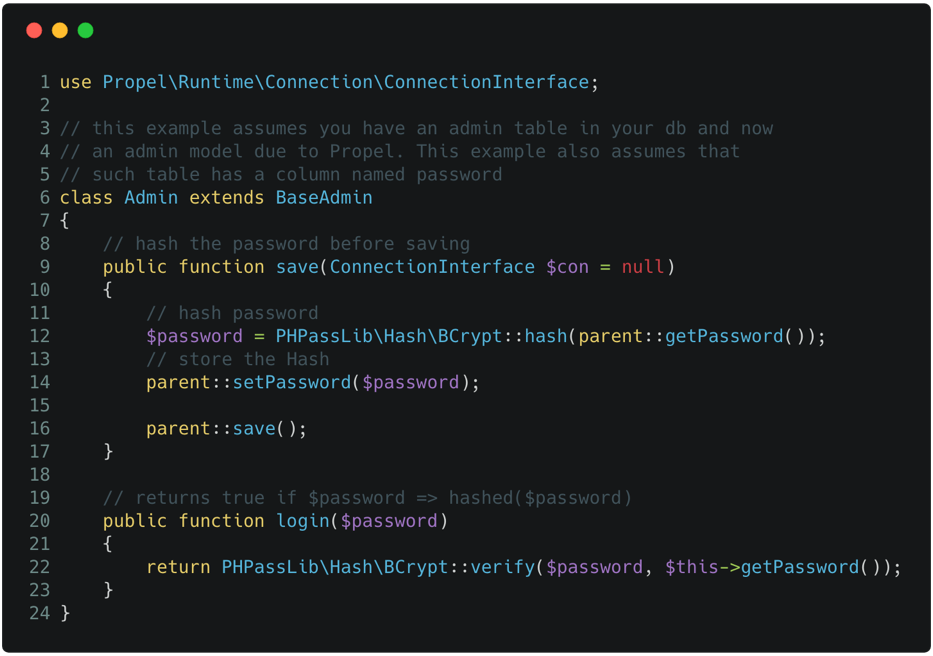
Require it in composer.json:



Run composer update to download phpass and its dependencies. The dependencies will be downloaded into the vendor/ directory; if you see a rych/ directory in vendor/ you’re set.

**Important functionality**

The following is an example of how to use phpass in a model created by Propel:



The two main functions are BCrypt::hash and BCrypt::verify.

# Slim

**Overview**

Slim is a PHP micro framework that helps you quickly write simple yet powerful web applications and APIs. At its core, Slim is a dispatcher that receives an HTTP request, invokes an appropriate callback routine, and returns an HTTP response. That’s it [6].

**Installation**

Require it in composer.json:



Run composer update to download slim and its dependencies. The dependencies will be downloaded into the vendor/ directory; if you see a slim/ directory in vendor/ you’re set.

**Getting started**

Create an index.php file and type the following:



If the code above works, you’re set with slim.

# Combining All Elements

**Overview**

Using all the previous mentioned technologies to create a project as a team.

**Slim setup**

Unlike the [slim setup](#_Slim_1) previously mentioned, we will use the power of slim for more than just writing strings to the browser. Slim will be set up to render views that we will construct, and have middleware in charge or allowing certain visitors to see only certain views (depending on status). In order to render views with slim you must have *slim/php-view* installed. The following snippet of PHP demonstrates render:



**Propel + Slim**

In order to start using Propel, be sure to read the [propel guide](#_Propel_1). Once you have done so, we have to tell composer to add the models to the autoload process. The finalized composer.json file will look similar, if not identical, to this:



Make sure to run *composer dump-autoload –o* to reload the dependencies into the autoload mechanism. Once the dependencies are loaded and slim is configured to show PHP view files you can start using propel with slim. The [slim example](#all_slim_setup) configured to show views will be modified to integrate propel, like so:



And by doing so the home.php view can use the variables *users* and *books*:



# Digital Ocean

**Overview**

DigitalOcean provides developers cloud services that help to deploy and scale applications that run simultaneously on multiple computers.

**Connecting URL**

Go to namecheap.com and buy URL. To link to digital ocean go to “manage domains” in namecheap, then click manage to the domain you want to manage. Scroll down to nameservers, click the dropdown and select “custom DNS”. Type in “ns1.digitalocean.com, ns2.digitalocean.com, ns3.digitalocean.com” in the 3 inputs, then click the green checkmark to save.

**Creating a droplet/server**

Change to the Aszend team droplets by clicking on your avatar on the top right. To create a new droplet click the green “Create” button next to avatar, and select droplets. Under “Choose an image” click “One-click apps” and select “LAMP on vv.vv”. Choose size (1 GB default), and choose a datacenter (San Francisco default). Create an SSH key and add it: run the command “ssh-keygen” and don’t enter a phrase when prompted, then cat the contents of the public key and paste it in the new ssh key input in digital ocean. Change the host name to something simple, and finalize by clicking long green create button.

**Connecting droplet to URL**

Click the 3 dots on the far right the droplet and click “add a domain”. Fill in the domain example.com and submit. When creating new record, type the “@” symbol in hostname. “Will redirect to” must be changed to the url ip, and then submit by pressing “create record”. Once that’s done, go to the url to verify when a blue shark thing.

**Accessing the server (SSH)**

Copy IP-address from droplet. Open the terminal and cd into the directory where your ssh keys are located. Type *ssh –i key\_name root@ip\_address.* Get your SQL root password run *cat /root/.digitalocean\_password* copy it and run *mysql\_secure\_installation* which will prompt you to enter the password. After entering the password a set of prompts will appear:

- VALIDATE PASSWORD PLUGIN…: no

- Change the Password for root? : no

- Remove anonymous users? : yes

- Disallow root login remotely? : no

- Remove test database and access to it?: yes

- Reload privilege tables now? Yes

cd into /var/www/html and delete info.php (*rm info.php*)

**FTP into server (FILEZILLA)**

Install Filezilla. To connect to server you need to import the **private** ssh key: Open Filezilla settings, and navigate to SFTP (settings -> connection -> FTP -> SFTP). Click Add key file and import private key file, and click Ok. On Filezilla main page file in the inputs as followed:

- Host = sftp://ip\_address

- username: root

- Click Quickconnect

- Trust it

Html directory = /var/www/html

**Connecting to MySQL in the server**

Install Sequel PRO. SSH into server. Open Sequel PRO and click the SSH tab, then fill in the following information:

- MySQL Host: ip\_address

- Username: root

- Password: get from /root/.digitalocean\_password

- SSH host: Same as MySQL Host

- SSH User: Same as Username

- SSH Password: Click Key icon and import private key

- Test connection

- Add to favorites

- Connect

To import SQL:

- Click dropdown “Choose Database” on top left corner and select “Add Database”

- File -> import -> select SQL file

**Composer usage in server**

Download the latest composer.phar from <https://getcomposer.org/download/> (Scroll down to manual download and click latest version). Once the file is downloaded, remove the .phar extension. Make sure FileZilla is up and connected, and drag the new composer file into /usr/local/bin. Try running composer on the server, if permission denied run ***chmod 777 /usr/local/bin/composer***.

**Uploading website**

Make sure to run “apt-get update”, “sudo apt-get install php7.0-curl”, and “sudo apt install zip unzip php7.0-zip” in the server. Delete vendor folder. Drag and drop folder items into /var/www (html folder should be part of items you drop in). Once it’s done, run composer install in the directory where composer.json is located. If you encounter an error in which Classes can’t be found make sure your composer.json file has the namespace App\\ connected to “app/” (don’t forget the /). If internal error 500, make sure to run “sudo tail /var/log/apache2/error.log” to view error log, if error log shows “invalid command 'RewriteEngine” as the error run “sudo apt-get install apache2”, “sudo a2enmod rewrite”, and then “sudo apache2 restart” to restart the server.

**SSL**

Go to <https://certbot.eff.org/lets-encrypt/ubuntuxenial-apache> and select “Apache” and “Ubuntu version”, and follow along.

# Works Cited

[1] Apache Friends. March 27, 2018. Apache Friends RSS. <https://www.apachefriends.org/index.html>

[2] Wikipedia. March 23, 2018. Github. <https://en.wikipedia.org/wiki/Composer_(software)>

[3] Wikipedia. March 24, 2018. Composer. <https://en.wikipedia.org/wiki/Composer_(software)>

[4] Wikipedia. March 12, 2018. Propel (PHP). <https://en.wikipedia.org/wiki/Propel_(PHP)>

[5] Github. March 27, 2018. rchouinard/phpass. <https://github.com/rchouinard/phpass>

[6] Slim Framework. March 26, 2018. Documentation. <https://www.slimframework.com/docs/>