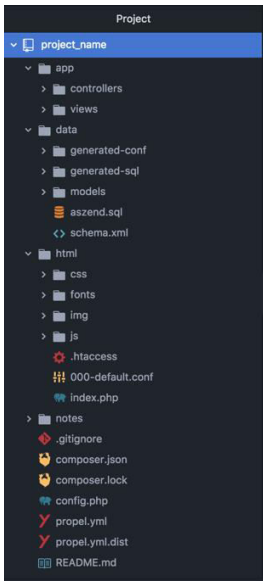
## Table of Contents

# 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

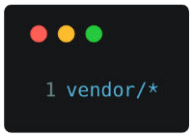
**Overview**
GitHub is a web-based hosting service for version control using git. It is mostly used for computer code. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features [1].

**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 [2].

**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**

```
{
  "require": {
    "propel/propel": "~2.0@dev",
    "rych/phpass": "3.0.*@beta",
    "phpmailer/phpmailer": "^6.0",
    "league/oauth2-client": "^2.2",
    "slim/slim": "^3.9",
    "slim/php-view": "^2.2",
    "paypal/rest-api-sdk-php": "^1.13",
    "ckeditor/ckeditor": "dev-full/4.3.x"
  },
  "autoload": {
    "classmap": [
      "data/models/"
    ],
    "psr-4":{
      "App\\": "app"
    }
  }
}
```

*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 [3].

**Installation**

Require it in composer.json:

```
{
  "require": {
    "propel/propel": "~2.0@dev"
  }
}
```

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:

```
<table name="user" idMethod="native" phpName="User">
  <column name="id" phpName="Id" type="INTEGER" primaryKey="true" autoIncrement="true" required="true"/>
  <column name="email" phpName="Email" type="VARCHAR" size="32" required="true"/>
  <column name="password" phpName="Password" type="VARCHAR" size="128" required="true"/>
  <vendor type="mysql">
    <parameter name="Engine" value="InnoDB"/>
  </vendor>
  <behavior name="validate">
    <parameter name="rule1" value="{column: email, validator: NotNull}"/>
    <parameter name="rule2" value="{column: email, validator: Email}"/>
    <parameter name="rule3" value="{column: email, validator: Unique}"/>
    <parameter name="rule4" value="{column: password, validator: Regex, options: {pattern: &quot;/^(?=.*[a-z])(?=.*[@#$%!+=]).
  {5,}$/&quot;}}"/>
  </behavior>
</table>
```

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 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 [4].

## Installation

Require it in composer.json:

```
{
  "require": {
    "rych/phpass": "3.0.*@beta"
  }
}
```

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:

```
use Propel\Runtime\Connection\ConnectionInterface;

// this example assumes you have an admin table in your db and now
// an admin model due to Propel. This example also assumes that
// such table has a column named password
class Admin extends BaseAdmin
{
    // hash the password before saving
    public function save(ConnectionInterface $con = null)
    {
        // hash password
        $password = PHPassLib\Hash\BCrypt::hash(parent::getPassword());
        // store the Hash
        parent::setPassword($password);

        parent::save();
    }

    // returns true if $password => hashed($password)
    public function login($password)
    {
        return PHPassLib\Hash\BCrypt::verify($password, $this->getPassword());
    }
}
```

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 [5].

**Installation**
Require it in composer.json:

```
{
  "require": {
    "slim/slim": "^3.9",
    "slim/php-view": "^2.2"
  }
}
```

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:

```
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require '../vendor/autoload.php';

$app = new \Slim\App;
$app->get('/hello/{name}', function (Request $request, Response $response, array $args) {
    $name = $args['name'];
    $response->getBody()->write("Hello, $name");

    return $response;
});

// index.php/hello/oscar => Hello, oscar
$app->run();
```

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 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:

```php
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

// require autoload classes from vendor
require '../vendor/autoload.php';
// require propel's config file to establish connection with db
require '../data/generated-conf/config.php';
// require file that has slim settings and global functions
require '../config.php';

$app = new \Slim\App(["settings" => $config]);

// set the view index of container to be a phpRenderer object,
// this in order to be able to render views from php files
$container = $app->getContainer();
$container['view'] = new \Slim\Views\PhpRenderer("../app/views/");

// home
$app->get('/home', function (Request $request, Response $response, array $args) {
    // just render the home.php view and thats it
    return $this->view->render($response, 'home.php');
});
```

**Propel + Slim**
In order to start using Propel, be sure to read the propel guide. 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:

```json
{
  "require": {
    "slim/slim": "^3.0",
    "rych/phpass": "3.0.*@beta",
    "propel/propel": "~2.0@dev",
    "slim/php-view": "^2.2"
  },
  "autoload": {
    "classmap": [
      "data/models/"
    ]
  }
}
```

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 configured to show views will be modified to integrate propel, like so:

```php
// ... setup code ...
// examples
$app->get('/home', function (Request $request, Response $response, array $args) {
    $all_users = \UserQuery::create()->find();
    $user = \UserQuery::create()->findOneById(1);
    $books = \BookQuery::create()->filterByUser($user)->find();

    return $this->view->render($response, 'home.php', ['users'=>$all_users, 'books'=>$books]);
});
```

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

```php
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <?php foreach ($users as $user) {
      var_dump($user);
    } ?>

    <?php foreach ($books as $book) {
      echo $book->getTitle();
    } ?>

  </body>
</html>
```

# Works Cited

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

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

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

[4] Github. March 27, 2018. rchouinard/phpass. https://github.com/rchouinard/phpass

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