Project Links

Wiki Table of Contents
Development
How to Use Phabricator
Deployment
Backups
Reading Resources
Authentication

Deployed Instances

- Production | Status Page
- Development

Tools and Technologies

- Phabricator (Code Review)
- Travis CI (Build/Testing/Deployment)
- Cloudflare (DNS)
- Digital Ocean (VPS)
- Nginx (Webserver)
- PM2 (Process Manager)

Getting Started

We use Git (hosted on GitHub) to manage source code for this project. There are three protected branches that are mirrored in the three stages of deployment. The master branch is mapped to production, the staging branch is mapped to staging, and the development branch is mapped to development.

To make any changes to the project you will need to clone the repository by running

`https://github.com/buie/KipsWarehouse.git`

For this project we are using Node.js version v6.9.4 and NPM version 3.10.10 so install those if you don't already have them installed.

Within the newly cloned repository, run the following commands.

`npm install` - installs needed dependencies from the node package manager. Required packages for this project can be viewed in `package.json`

We have set up the dev environment with Babel and Webpack to enable support for ES6 features (like arrow functions) and async/await. While developing, run the following command to transpile as code changes are made.

`webpack --progress --colors --watch`

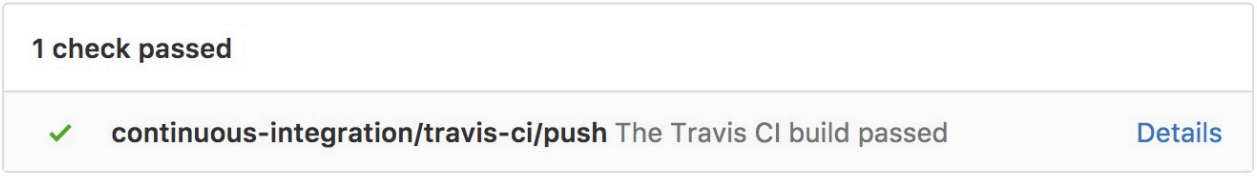
The project is currently configured to connect to our remote development MySQL database. The credentials for accessing that have been kept off of Git for security purposes but can be obtained by contacting one of the developers of this project.

With the credential configuration files in place, you can run the project on your local machine by running

`bin/www`

in the KipsWarehouse directory. This will start the application at localhost:3000.

To add a new feature please create a new branch from development with the title of that feature as the branch title. Then make your changes on that branch, code review the changes using Phabricator, and then make a pull request to the development branch. Wait for the Travis CI build to pass and you see the status update as shown below.



Assign the pull request to @buie, @esonmez, @charleschengxu, or @jayyay. Once the pull request has been approved, the changes will be automatically deployed to the development site at dev.kipswarehouse.com. After a set of features have been completed, the changes will be propagated through the development pipeline stages Dev->Staging->Production.

Install

1. Install Arcanist. *Make sure you remember to add `arc` to your path.*
2. Point `arc` to Phabricator:

```
arc set-config default http://kips-warehouse.phacility.com
arc install-certificate
```

3. Configure `nano` (or your editor of choice) with Phabricator by running: `arc set-config editor "nano"`
4. Try running `arc` or `arc diff --preview` to make sure everything is working.

Here's a handy Phabricator Install Guide.

Development Cycle

All code merged into `development` should be reviewed using Phabricator.

Building a New Feature/Bug Fix/Improvement

1. Create a new branch for your feature (or bug fix) off of `development`.
2. Develop the feature, pushing changes to your feature branch.
3. After the feature is complete & tested, make sure your development branch is up-to-date and create a diff off of development. Running the command below will create a differential between *your feature branch* and `development`. Make sure your test plan and summary are thorough. *Always test your code according to your test plan before submitting a diff.*

```
arc diff development
```

4. Assign a reviewer.

Needs Review

1. Make any necessary changes in your feature branch, mark requested changes as complete on Phabricator, and then re-run `arc diff development` or `arc diff development --preview` to update your diff to reflect the new changes.
2. Repeat Step 1 as necessary until the reviewer approves the diff.

Ready to Land

1. After your review has been accepted, run the following command to land your branch onto the

`development` branch.

`arc land --onto development`

This will merge your branch into `development` and delete your local copy of your branch. It will not delete your remote feature branch (these must be deleted manually).

Release Cycle

We will only use Phabricator in development and will cut releases from `development` to `staging` or `staging` to `master` using Github.

Resources

- Short summary of useful ES6 features
- Async/Await - allows you use the keywords `async` and `await` to write synchronous looking code that gets transpiled (using Babel) to callback-style code
    - The long road to async/await in javascript
    - Async generators
- React
    - React official tutorial: https://facebook.github.io/react/docs/tutorial.html
    - Learn React: http://jamesknelson.com/learn-raw-react-no-jsx-flux-es6-webpack/ - specific benefits of using React compared to writing vanilla Javascript for UI code
    - React howto: https://github.com/petehunt/react-howto - broader look at the React ecosystem
- Javascript, The Good Parts
- Airbnb Javascript Style Guide
- The Art of Node

Eslint

Our project is configured with eslint using Airbnb's JavaScript Style Guide. All code pushed should conform to our eslint rules. If you modify legacy code that triggers eslint warnings or errors, please fix the warnings and/or errors before pushing.

Setup

> npm install

> npm install -g grunt-cli

Then, install the Atom plugin linter-eslint. Restart Atom for the changes to take effect.

Before Pushing New Code

1. If using Atom, make sure no eslint errors exist in new code.
2. Run `grunt eslint` in command line. Make sure no eslint errors exist in files with modified code.
3. If errors exist in files with modified code, fix before pushing.

Sources

- Configuring Atom editor with ESLint and the AirBnB style guide rules
- Getting Started with ESLint using Grunt
- Airbnb JavaScript Style Guide

All /api paths are authenticated.

Login or Sign Up Your Own Account

Post {host}/login/admin/initialadminpassword0987654321 to login as admin first, and copy the token from the response

In postman, for every /api request, add key: "Authorization" and value: "JWT {token}" (with {token} being the token you copied) for every /api request. Without the presence of this field in header, response will be "unauthorized". If passed the authentication, you'll find that you can use req.user.id to retrieve the currently-logged-in user's id.

Post /api/users/:name/:password with :name and :password being your own choice will create a new account that you can use to login so that you don't need to use admin. In a SQL client you can change your permission level from User to Admin in the Users table.