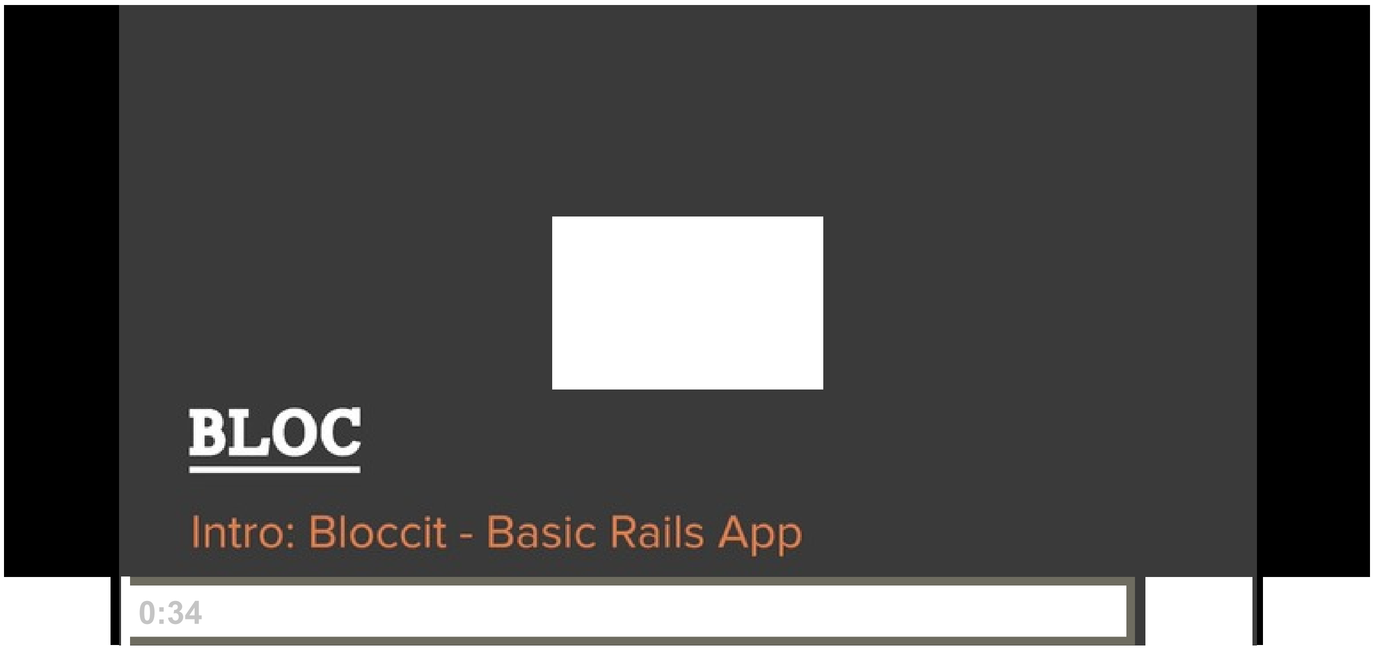
24 Rails: Basic Rails App

> "Rails is the most well thought-out web development framework I've ever used. Nobody has done it like this before."
>
> — **James Duncan Davidson**

Bloccit



In this checkpoint, you'll start a new project, similar to **Reddit**, named Bloccit.

Just like Reddit, Bloccit will be an app where people can post, vote on, share and save links and comments. Bloccit will have many features needed to make it a cool web app, but the first thing you need to do is design a basic user-interface (UI) as a foundation to build on.

Create Bloccit

The first step is to create a new Rails app. Run the `rails new` command in the `code` directory we created earlier:

Terminal

```
$ cd code
$ rails new bloccit -T
```

The app name is `bloccit`. The `-T` option specifies that the app should not be created with standard test packages since we'll be testing our app with `RSpec`.

When we ran the `rails new` command, we should've seen a long output in your console. Among other things, `rails new` creates the Rails app structure. Open the project in the editor to explore the Rails app structure.

We should see a full Rails app structure. We'll explore the various directories as we progress through the Roadmap. We'll begin to make changes, but before you do, we'll want to establish the README, and update the database file and Git repositories

Create a New README

A `README` file should describe what the app or program does. It should also provide directions on how to install it, run tests, or anything else that another developer would need to know.

Rename the `README` to use Markdown:

Terminal

```
$ mv README.rdoc README.md
```

Open `README.md` and update it with the following content:

```
+ ## Bloccit: a Reddit replica to teach the fundamentals of web development and Rails
+
+ Made with my mentor at [Bloc](http://bloc.io).
```

> Feel free to change the style or content of `README.md` as you see fit.

Create the Development Database

Replace the contents of your `Gemfile` with the following:

Gemfile

```
+ source 'https://rubygems.org'
+
+ # Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
+ gem 'rails', '4.2.5'
+
  # #1
+ group :production do
+   gem 'pg'
+ end
+
  # #2
+ group :development do
+   gem 'sqlite3'
+ end
+
+ # Use SCSS for stylesheets
+ gem 'sass-rails', '~> 5.0'
+ # Use Uglifier as compressor for JavaScript assets
+ gem 'uglifier', '>= 1.3.0'
+ # Use CoffeeScript for .coffee assets and views
+ gem 'coffee-rails', '~> 4.1.0'
+ # Use jquery as the JavaScript library
+ gem 'jquery-rails'
+ # Turbolinks makes following links in your web application faster. Read more: https
+ gem 'turbolinks'
```

At **#1** and **#2** we specify different databases for our Development and Production environments. We use `sqlite3` for our Development environment because it is an easy to use database perfect for rapid testing. Heroku only supports Postgres, so we use `pg` in our Production environment.

Because we changed your `Gemfile`, we must update our application with `bundle install --without production`. This command installs everything specified in the `Gemfile` and ensures that all of the gems work harmoniously. The `--without production` option ignores gems in `group :production`. These gems aren't needed or used in our Development environment. Our Production environment will automatically run `bundle install` when we deploy, and will account for gems declared in `group :production` at that point. On the command line, in the root `Bloccit` directory, type:

Terminal

```
$ bundle install --without production
```

Run the following command in your terminal to create the database:

Terminal

```
$ rake db:create
```

This creates a new local database for our app to use. We have to run this command after creating a new app, or after dropping an existing database.

The Asset Pipeline

As stated in the **Rails Guide**:

> The asset pipeline provides a framework to concatenate and minify, or compress, JavaScript and CSS assets. It also adds the ability to write these assets in other languages such as CoffeeScript, Sass, and ERB.

The purpose of the asset pipeline is to make Rails apps fast by default while allowing developers to write "assets" (images, styles, and JavaScript, mostly) in a variety of languages.

Rails 4 requires some minor configuration changes to properly serve assets on Heroku:

Gemfile

```
  ...
  group :production do
    gem 'pg'
+   gem 'rails_12factor'
  end

  group :development do
    gem 'sqlite3'
  end
  ...
```

We added `rails_12factor` to the Gemfile; let's install it in our application:

Terminal

```
$ bundle install
```

Heroku provides a **detailed explanation** of the Rails 4 configuration changes.

Test Locally

Start the Rails server from your command line:

Terminal

```
$ rails s
```

> If you're **using Cloud9**, remember to start the Rails server with the `-p $PORT -b $IP` flags.

Navigate to **localhost:3000** to make sure the app is working locally.

> Starting the web server with `rails s` will leave your terminal in an "open" state. That is, you won't see a command prompt until you stop the server. Open your app on localhost and view it next to the terminal where you started the server. Refresh the page on localhost, and you'll see the server logs update in your terminal. While you're running the Rails web server, the terminal logs all activity in your app.

Git and GitHub

Sign into your **GitHub** account and create a new repo named `bloccit`. You've already created a README, so make sure the "Initialize this repository with a README" is *unchecked*.

Commit and push your code up to your GitHub repo:

> If your Rails server is still running, you can either stop it by pressing `CTRL-C` or leave it running and open a new Terminal tab. Either way, you'll need a Terminal prompt before moving forward.

Terminal

```
$ git init
$ git add .
$ git commit -m 'First commit and README update'
$ git remote add origin git@github.com:<user name>/<repo_name>.git
$ git push -u origin master
```

> Use the URL from GitHub's instructions.

Reload the repo homepage on GitHub. It should display the content from `README.md` at the bottom of the page and you should see all of this repo's files.

Deploying to Heroku

It is time to deploy and share your app with the world. There are many choices for deploying and hosting Rails applications, and Bloc recommends the popular **Heroku** platform. Heroku makes it easy to manage and deploy Rails apps using the command line.

Sign up for a free Heroku account. Then install the **Heroku Toolbelt** for your OS. This toolbelt will allow you to run Heroku commands from the command line.

> If you're using Cloud9, the Heroku toolbelt is already installed.

Log into your new Heroku account:

Terminal

```
$ heroku login
```

After you've logged in, create a new application in Heroku:

Terminal

```
$ heroku create
```

> Because we did not specify a name with `heroku create`, Heroku created one for us.

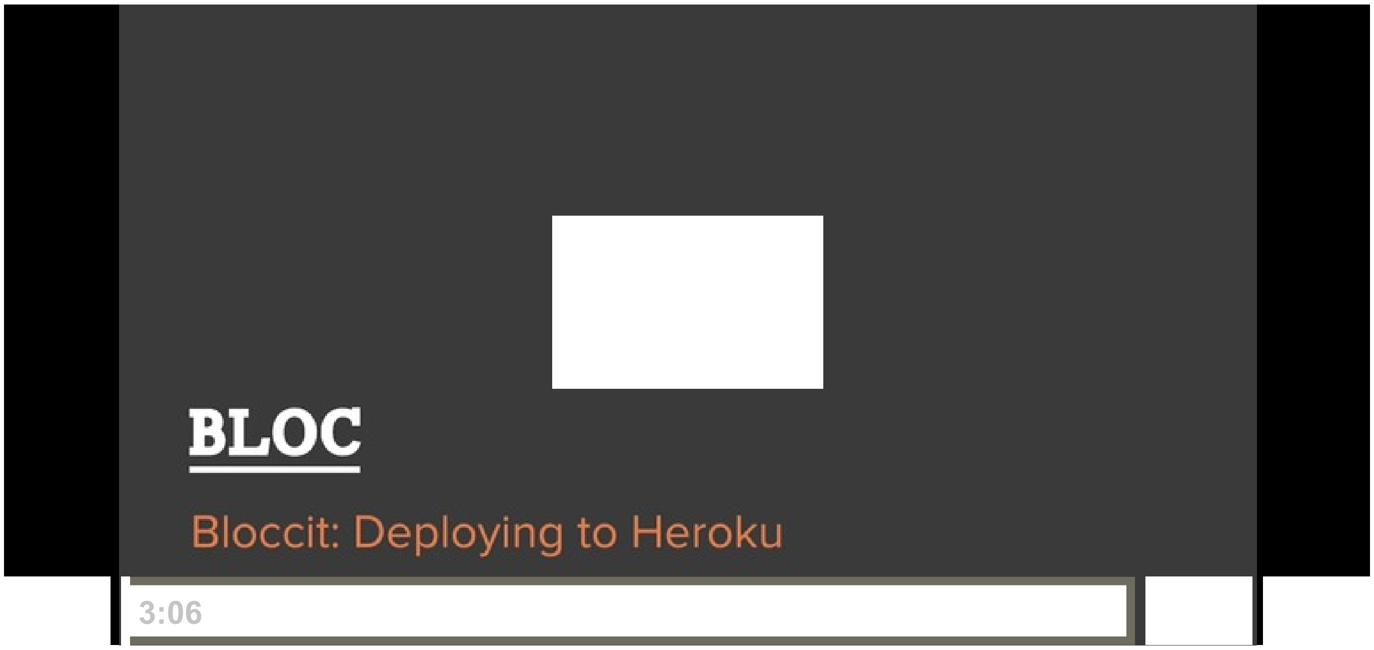
We have a Production environment to `push` our application to. Type this command to push the code from the master branch of your Git repo to Heroku:

Terminal

```
$ git push heroku master
```

> It may take a few minutes for the new application to propagate in Heroku. If you receive an error that says `Permission denied (publickey)`, **go here** to learn how to fix it.

Here is a video recap of how to deploy to Heroku:



You can see the web address for your application in Production by typing the following:

Terminal

```
$ heroku apps:info
```

After this, you need to migrate the heroku!!

$ heroku run rake db:migrate

Congratulations, you've deployed an application to your Production environment. For now, you'll receive an error message when you visit your Heroku URL. This is because the static index page **is not used in production**. We'll fix that soon.

Make sure you've added your **GitHub** account to your Bloc account page. Use the "Submit your work" tab to submit your first Bloccit commit for your mentor to review.

Recap

Concept	Description
`rails new`	`rails new` creates a new Rails application with the entire default Rails directory structure.
README	A `README` is a text file commonly distributed with a program. It contains information that describes what the program does, provides directions on how to install it, run tests, or anything else that another developer would need to know.
Asset Pipeline	The asset pipeline provides a framework to concatenate and minify, or compress JavaScript and CSS assets. It also adds the ability to write these assets in other languages such as CoffeeScript, Sass, and ERB.
`rails server`	The `rails server` command launches a small web server named WEBrick, which comes bundled with Ruby.
Heroku	Heroku (pronounced her-OH-koo) is a platform-as-a-service (PaaS) that enables developers to build and run applications entirely in the cloud.
Gems	Gems are Ruby libraries that can be used to extend or modify functionality within a Ruby application.
Rails Environments	Rails ships with three environments: "Development", "Test", and "Production". These environments are used to tell your app to behave differently in different circumstances, primarily by setting different configuration options and variables.

After starting the Rails server, we can use `Ctrl-C` to shutdown the server. A common issue encountered by Rails developers is starting the Rails server and then accidentally closing the command line window in which it is running. The Rails server is still running in the background, but with the command line window closed, you can no longer use `Ctrl-C` to shut it down. You could restart your computer, which kills all running processes. However, there is a better way, and we'll explore that in this assignment.

1. Simulate losing track of your Rails server by starting it using the `-d` option:

   > If you're using Cloud9, remember to start the Rails server with the `-p $PORT -b $IP` flags.

   Terminal

   ```
   $ rails s -d
   ```

   Using `-d` option starts the Rails server as a **daemon**, a computer program that runs as a background process.

2. Start your Rails server again. You will see an error message:

   Terminal

   ```
   $ A server is already running. Check [local path]/tmp/pids/serve
   $ Exiting
   ```

3. To kill the lost server, find its **process id (PID)** using the `lsof` **command**. `lsof -i :3000` returns the PID of the process using port 3000 (the port Rails server uses).

   > If you're using Cloud9, the server will be running on whatever port number is in the `$PORT` environment variable, which may not be 3000. Instead of lsof -i :3000, use `lsof -i :$PORT` to find the PID.

4. We have the PID for your lost Rails server; terminate it using the `kill -9 PID` **command** using the PID we found with `lsof`.

Once you have completed the steps in this assignment, message your mentor with questions you have about the Rails server. If you have questions about starting the server, stopping the server, server logs, PID, etc. be sure to discuss them with your mentor before moving to the next checkpoint.

Solution

Do not watch this video until after you've attempted to complete the assignment. If you struggle to complete the assignment, submit your best effort to your mentor *before watching a solution video*.

Basic Rails App Solution

assignment completed

?