

Thank You, PhoneGap!



[Masa Tanaka](#)

[Aug 19, 2020](#) · 3 min read

～PhoneGap, お疲れ様でした～



PhoneGap and Cordova

The Adobe PhoneGap has recently [announced](#) that they will sunset their tools and services (PhoneGap and PhoneGap Build) on October 1, 2020. PhoneGap will not be actively maintained but will remain free and open sourced whereas the PhoneGap Build service will no longer be available.

[Adobe donated the PhoneGap codebase](#) to the Apache Software Foundation for incubation in late 2011. The project was originally named Apache Callback and later renamed to Apache Cordova.

PhoneGap is a distribution of Apache Cordova. You can think of Apache Cordova as the engine that powers PhoneGap, similar to how WebKit is the engine that powers Safari.

Since the donation of Cordova from Adobe to Apache, the OSS community has formed with lots of participation and contribution from great individuals and companies. The Apache Cordova project has become the de facto standard for HTML5 hybrid application development and plays an important role to bridge between web development and native APIs for major platforms such as iOS and Android.

We, Monaca, have been involved in Apache Cordova and share the same vision since the beginning. Our development members have been playing a significant role in the community. Some of the

highlight features and projects that the team has been working on recently which are worth mentioned are:

- Supporting and releasing of Cordova core libraries, tools, and CLI — Cordova 10, WKWebView integration, Adaptive icon support...
- Cordova Electron platform to provide support for desktop applications on Windows, macOS, and Linux.
- Supporting and releasing of major mobile platforms (Android and iOS)

Cordova, as the pioneer of Hybrid mobile development technology, will continue to grow as an open source project supported and loved by the community.

What is Monaca?

[Monaca](#) is a development environment for Cordova provided by us. Monaca makes PhoneGap/Cordova development simple and easy. It is ready to be immediately plugged into your existing workflow and development environment.

Our tools are designed to utilize the cloud to supercharge your app development in two ways. The complete cloud development environment gives you flexibility with no setup required. And the cloud-synced local development enables you to use your own environment but enjoy features like device live-sync and remote build. It's the best of both worlds.

From [Cloud IDE](#), [CLI](#), [Localkit](#) to [debugger](#) and Continuous Integration and Continuous Deployment ([CI/CD](#)), everything you need for your hybrid app development is [here](#).

Complete Migration in Minutes

To all PhoneGap users, you can continue to develop your application with the technology you are familiar with. In response to the end service of PhoneGap Build, we have compiled a step-by-step migration [guideline](#) to ensure a smooth transition from PhoneGap to Monaca. We will be providing a free **2-month** coupon code **THANKPHONEGAP** for new users, to help you [get started](#).

This coupon is only available for "Pro" plans. You can enter this coupon code during the signup process after confirming your email address.

Please feel free to reach out to us if you have any questions or problems.

Conclusion

We would like to express our gratitude to PhoneGap and Adobe for paving the path for us in the field of hybrid mobile app development. We at Monaca will continue contributing to the Apache Cordova project and web technology development in overall.

Happy Coding!

<https://medium.com/the-web-tub/thank-you-phonegap-b3f4102aa6bb>

Introducing Cordova 10 support in Monaca

Starting this week, Monaca has started supporting Cordova 10.

With this support, WKWebView will be installed as standard on Android 10 and iOS. Also, newly created projects will be created for Cordova 10.

The versions of Android and iOS provided by Monaca on Cordova 10 are:

Android updates

- Cordova Android Platform: updated to “9.0.0” (API level 29)
- Gradle version: updated to “6.5”

iOS updates

- Cordova iOS Platform: updated to “6.1.1”
- Xcode: “11.3”
(Xcode12 support will be announced separately.)

For more information, please refer to [Monaca Docs](#).

How to upgrade the project

You can upgrade your project to Cordova 10 by following the steps below.

1. Open the target project
2. Click “Settings” in the header menu
3. Click Manage Cordova Plugins
4. Click “Upgrade to 10.0.0” in the figure below



index.html

Cordova Pl... x



Cordova Plugins

Cordova Version:

CLI Version ?

9.0.0 [Upgrade to 10.0](#)

[Show Details](#) ▼

Choose CLI version and

Plugin Search

Project downgrade

Cordova10 Upgraded projects cannot be downgraded to a previous version.

A backup project of the previous version will be created automatically at the time of upgrade, so please use that.

Warning in Monaca Debugger

The “Monaca Debugger” for Cordova 10 is version 10.0.0 or later.

If the Cordova version installed in the debugger and the Cordova version of the project is different, the application may not work properly. In that case, please use “Custom Build Debugger”.

For information on how to get the custom build debugger, please refer to the following document.

[Android Custom Monaca Debugger](#)

[iOS Custom Monaca Debugger](#)

Monaca Debugger for Android



Debugger for Android

ver X.X.X

 demo@monaca.mobi



	Monaca Debugger (Store Version)	Custom Monaca Debugger
Description	Monaca Debugger available at the store	Monaca Debugger built from Monaca Cloud IDE
Installation	<ul style="list-style-type: none"> • Play Store • App Store 	Refer to Build and Install Custom Monaca Debugger
Cordova Plugins	Core and some third-party Cordova plugins are automatically included.	In addition to the core and third-party Cordova plugins, user submitted plugins (of the current project) are included.
App ID (Android:PackageNa mobi.monaca.debugger me)		App ID set by a user
Version Name (Android:versionNa Fixed me)		Display version name set by a user
App Version (Android:versionCo Fixed de)		Version set by a user
USB Debugging	Available (Chrome Dev Tools)	Available (Chrome Dev Tools)
Network Install	Available	Available

The following cordova plugin for push notification can not be used in the store version and the custom build debugger.

You can use these plugins in debug build or release build.

- MonacaBackend plugin
- NIFCloudMB plugin
- Other cordova plugin for push notification Example:
 - phonegap-plugin-push
 - onesignal-cordova-plugin
 - cordova-plugin-firebase

If the Cordova version of your project is lower than 5.2, it might not work properly with Monaca Debugger 5.X.X. There are two ways to fix this issue:

- [upgrade Cordova version](#) of your project
- use [Custom Monaca Debugger](#).

Cordova Plugins

In Monaca Debugger, core and third-party Cordova plugins are automatically included.

Monaca includes Core cordova plugins which are a minimal set of APIs such as Battery Status, Camera, Contacts, Device and so on. For a complete list of core Cordova plugins, please refer to [Core Cordova Plugins](#).

Monaca also includes some third-party Cordova plugins such as Statusbar, DatePicker, BarcodeScanner and so on. For a complete list of currently included third-party Cordova plugins, please refer to [Third-party Cordova Plugins](#).

While developing your project, you may need to add other third-party or [custom Cordova plugins](#) to your project. The standard Monaca Debugger (Store Version) doesn't have these newly added plugins. For this reason, your project might not run properly in the debugger. Therefore, you need to use Custom Monaca Debugger. Custom Monaca Debugger is a debugger which is built from a Monaca Project within Monaca Cloud IDE. Please refer to [Build and Install Custom Monaca Debugger](#).

USB Debugging

Monaca Debugger for Android supports USB debugging functions with Google Chrome browser such as:

- console debugging: using console to display message.
- DOM inspection: viewing and modifying DOM structure with live updates.
- JavaScript debugging: profiling JavaScript performance, setting breakpoint and execution control.

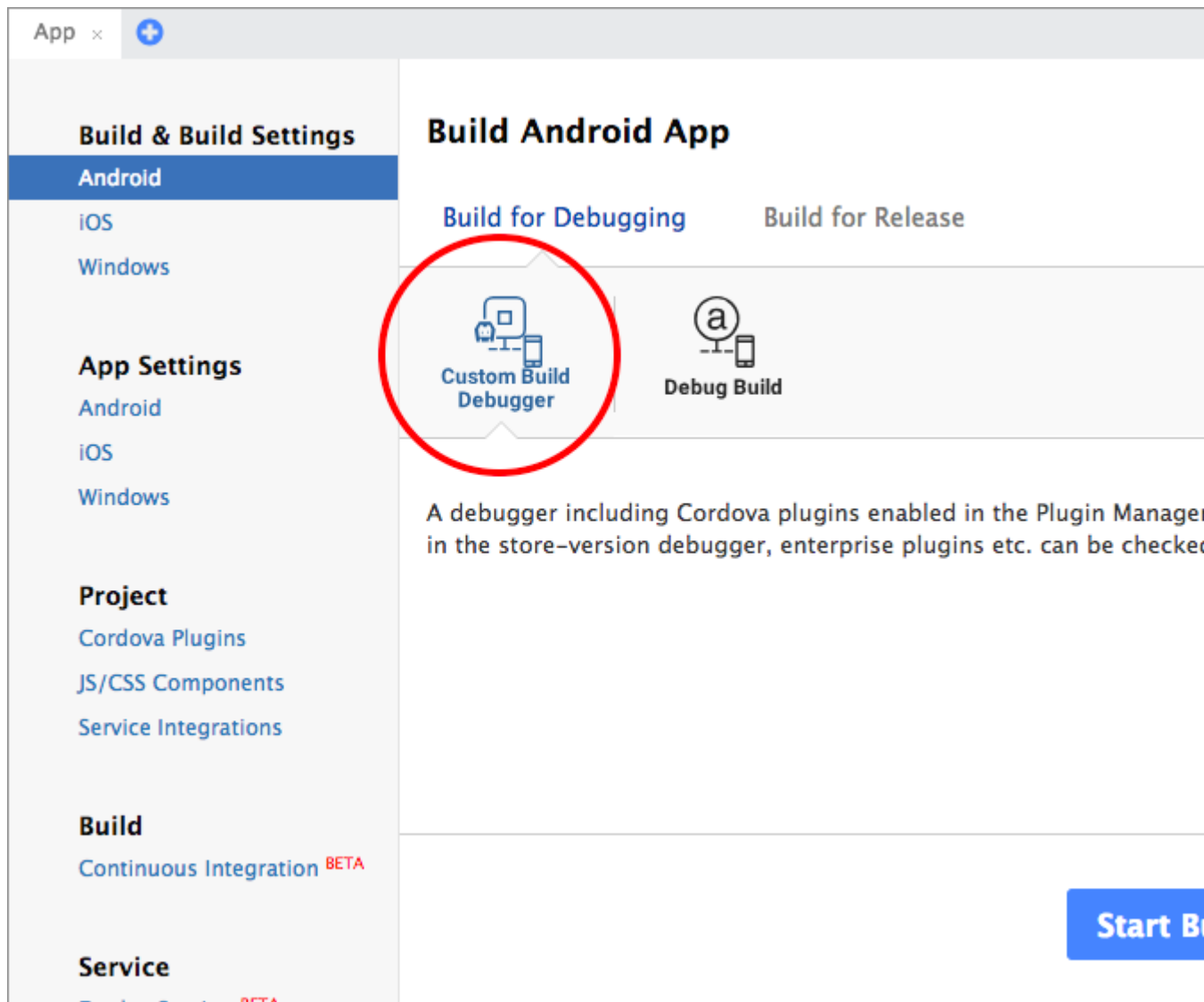
For more information, please refer to [USB Debugging with Monaca Debugger for Android Apps](#).

Network Install


Network Install is a feature provided by Monaca Debugger for Android allowing you to install the built app (debug build only) using the debugger. For more information, please refer to [how to use Network Install feature](#).

Build and Install Custom Monaca Debugger


1. From Monaca Cloud IDE menu, go to **Build** → **Build App for Android**.
2. Select Custom Debugger Build and click on Start Build.





3. This may take sometimes until the build is completed. The following screen will appear after the build is successfully completed. Then, you can use the QR code to install the debugger on your device or download the built file to your PC.

 **Monaca** Photo Sharing App Template

Build




Android Debugger Build
Project v1.0.0 / Debugger v7.0.2


 a few seconds ago
 May 1 17:14:29

Congratulations!

Your build is successfully finished. Please [download](#) and install the app on the device.

A successful build does not guarantee that your application will pass the regulation.


Download to Local PC


Install via QR code

Do you want to set this build as publicly downloadable?

Monaca Debugger for Android Emulator

In this page, we will describe how Monaca Debugger works on AVD (Android Virtual Device) called Android Emulator. With this, you can simulate Android apps on a PC. Moreover, it is useful when you want to test your apps on more than one Android device.

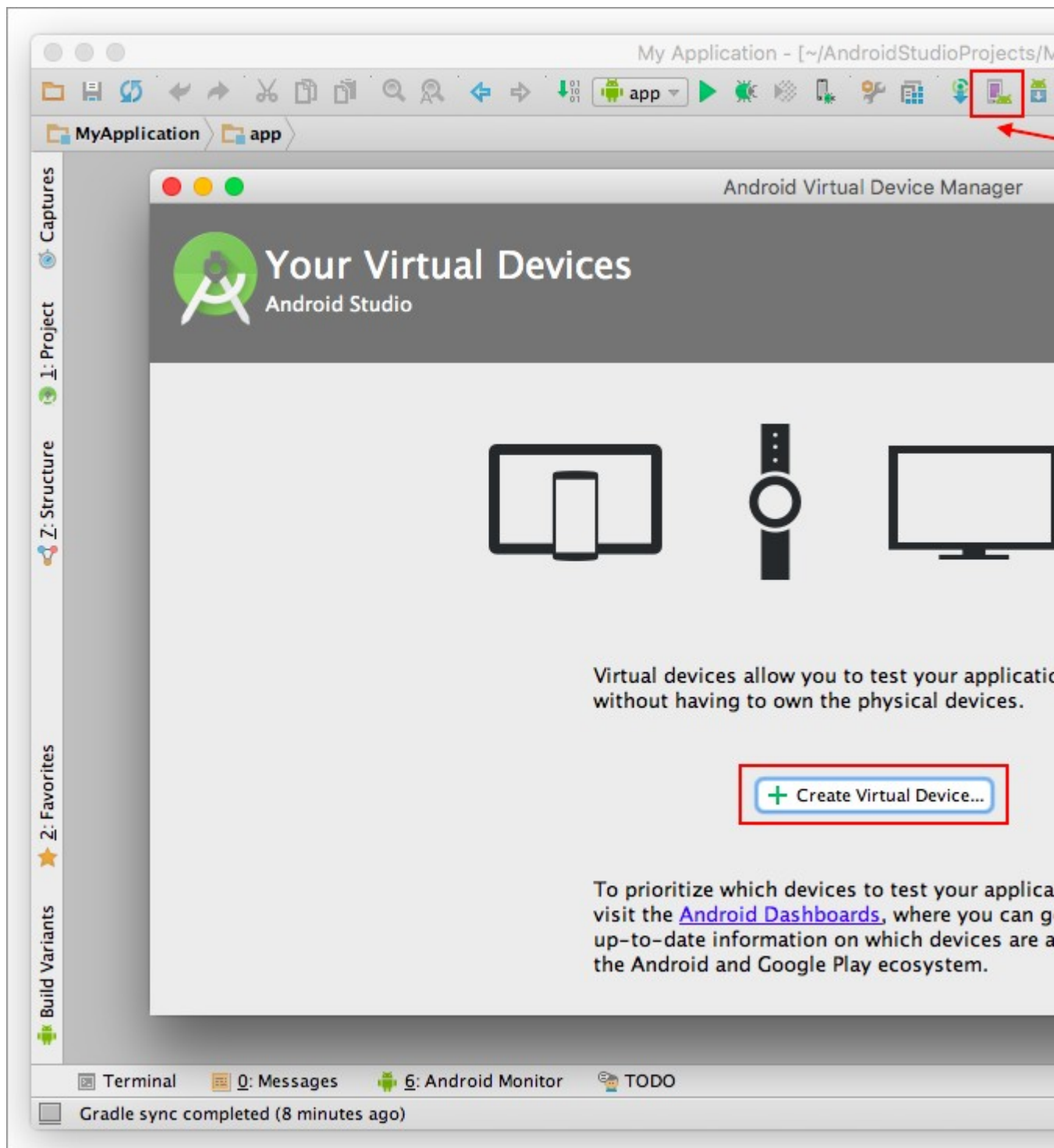
Monaca Backend plugin is not included in Monaca Debugger.

In this page, the instruction is made on a Mac OS X. Internet is needed in order to run Monaca Debugger on emulator.

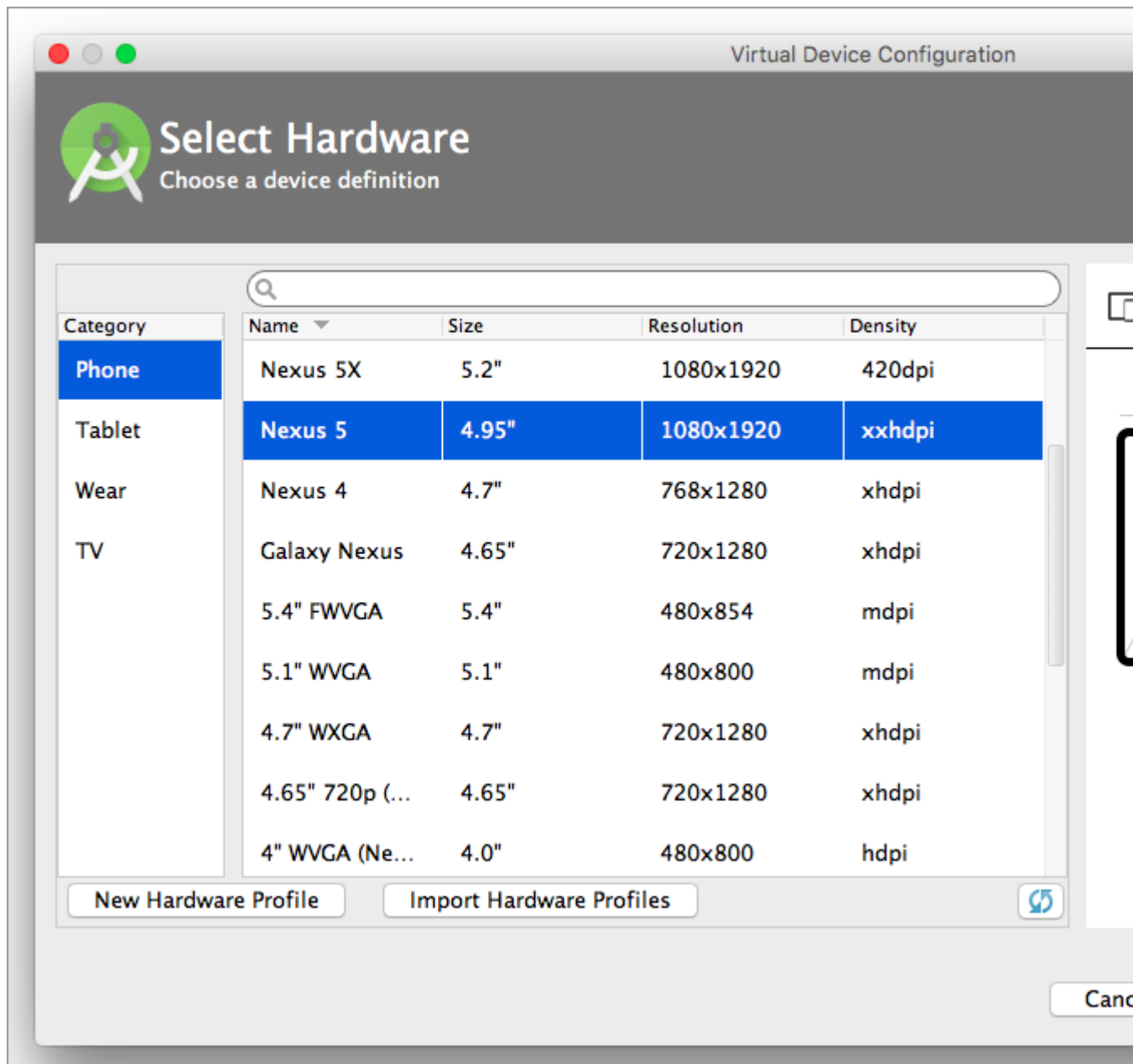
Step 1: Setting up an Android Virtual Device

1. Download and install [Android Studio](#).
2. Run Android Studio.
3. Create a new project or open an existing one.

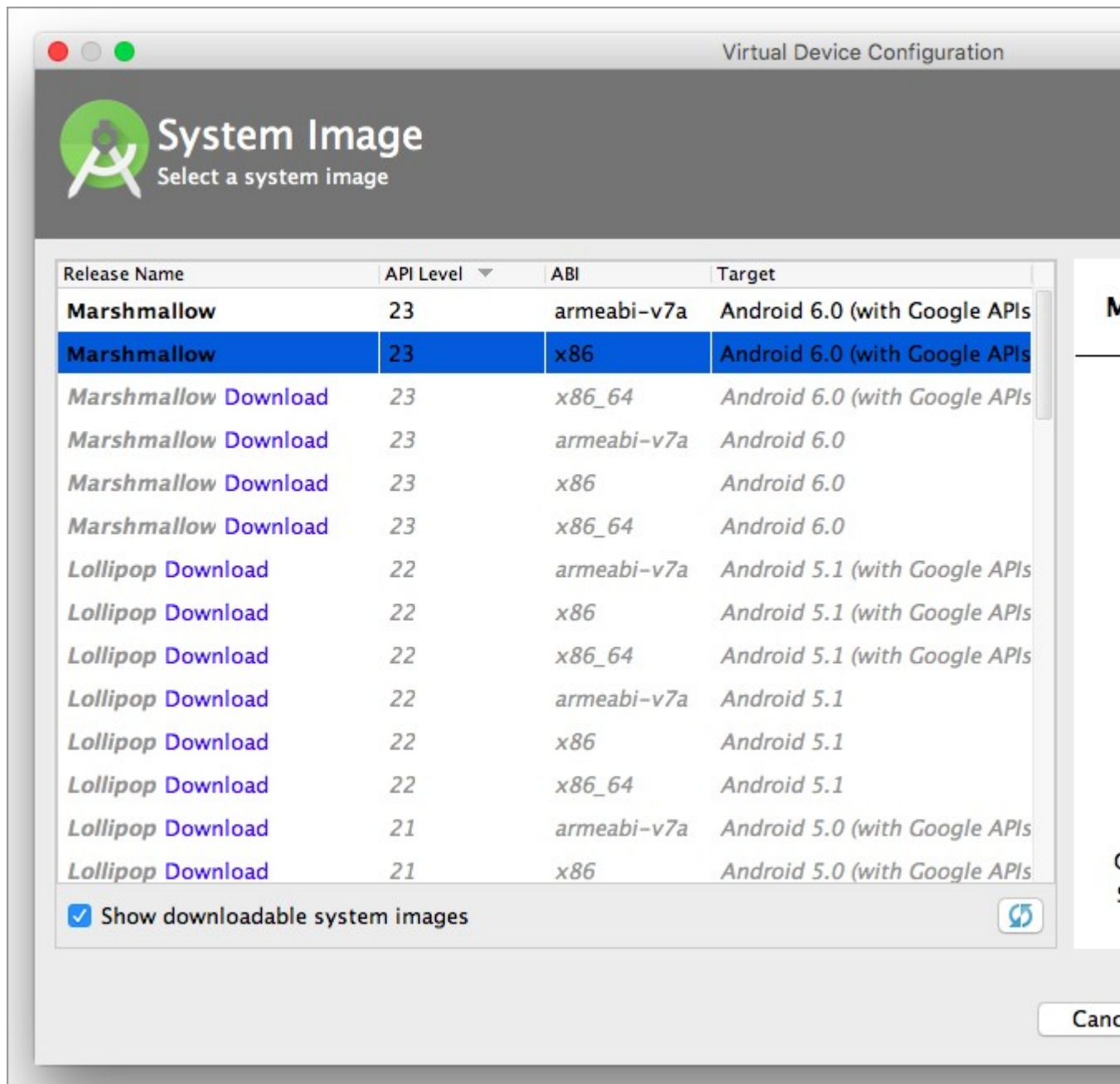
- Click on AVD Manager. Then, Android Virtual Device Manager dialog will appear.



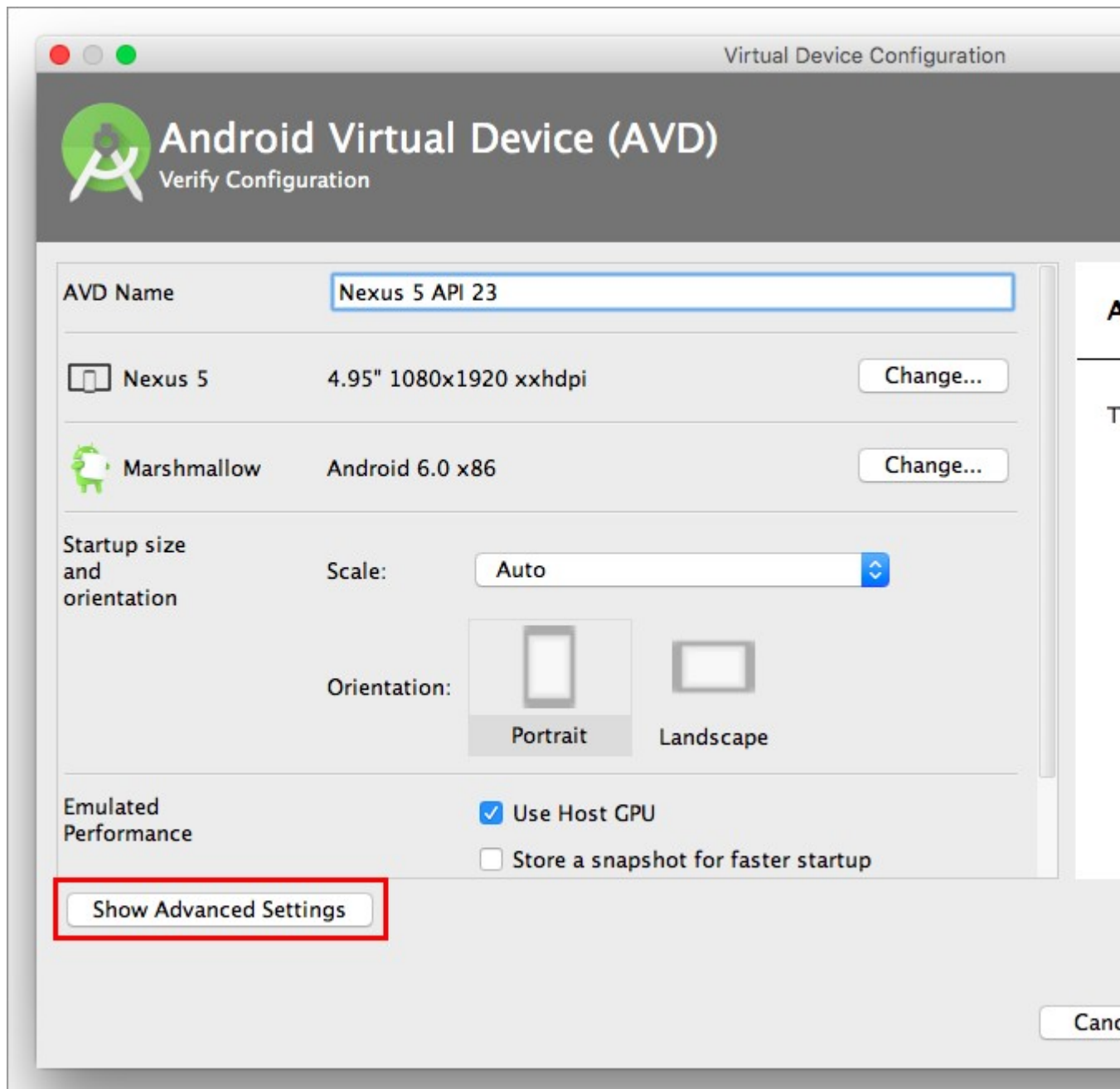
- Click + Create Virtual Device.
- Select an Android device and click Next.



7. Select a system image (you may need to download it first) and click Next.



8. Make the virtual device's configurations. Select Show Advanced Settings for other settings such as Memory and Storage, Device Frame and Keyboard.





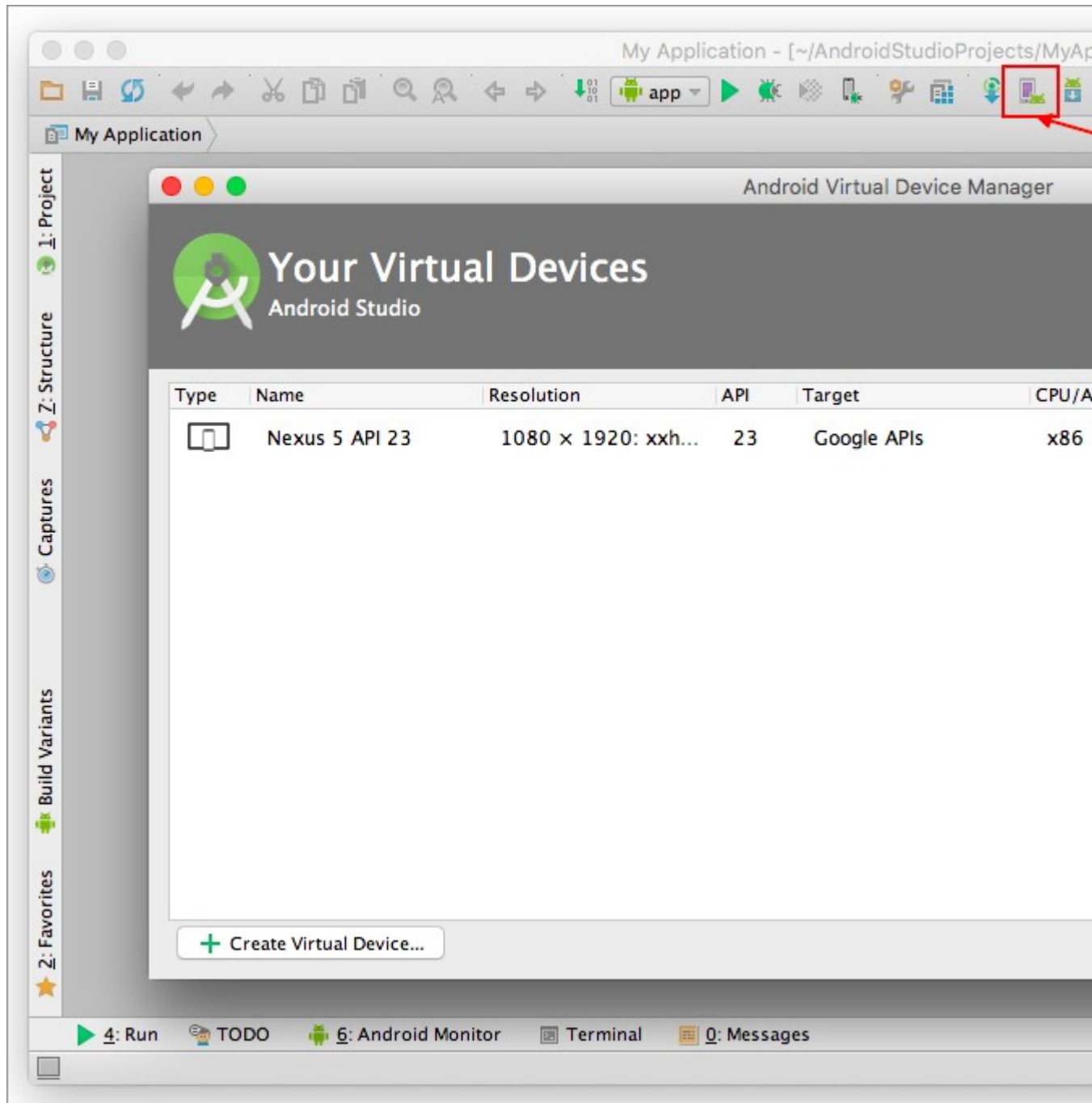
9. After completing the configurations, click Finish.

Step 2: Building Custom Monaca Debugger for Android

1. Go Monaca Cloud IDE and build custom Monaca Debugger for Android. For more information, please refer to [Build and Install Custom Monaca Debugger](#).
2. Download the debugger file and upload it to any file hosting services such as Google Drive, Dropbox and so on.
3. Get a download link of the uploaded file. You will need to use this link later in the virtual device.

Step 3: Installing Monaca Debugger

1. Go to AVD Manager and launch the virtual device.



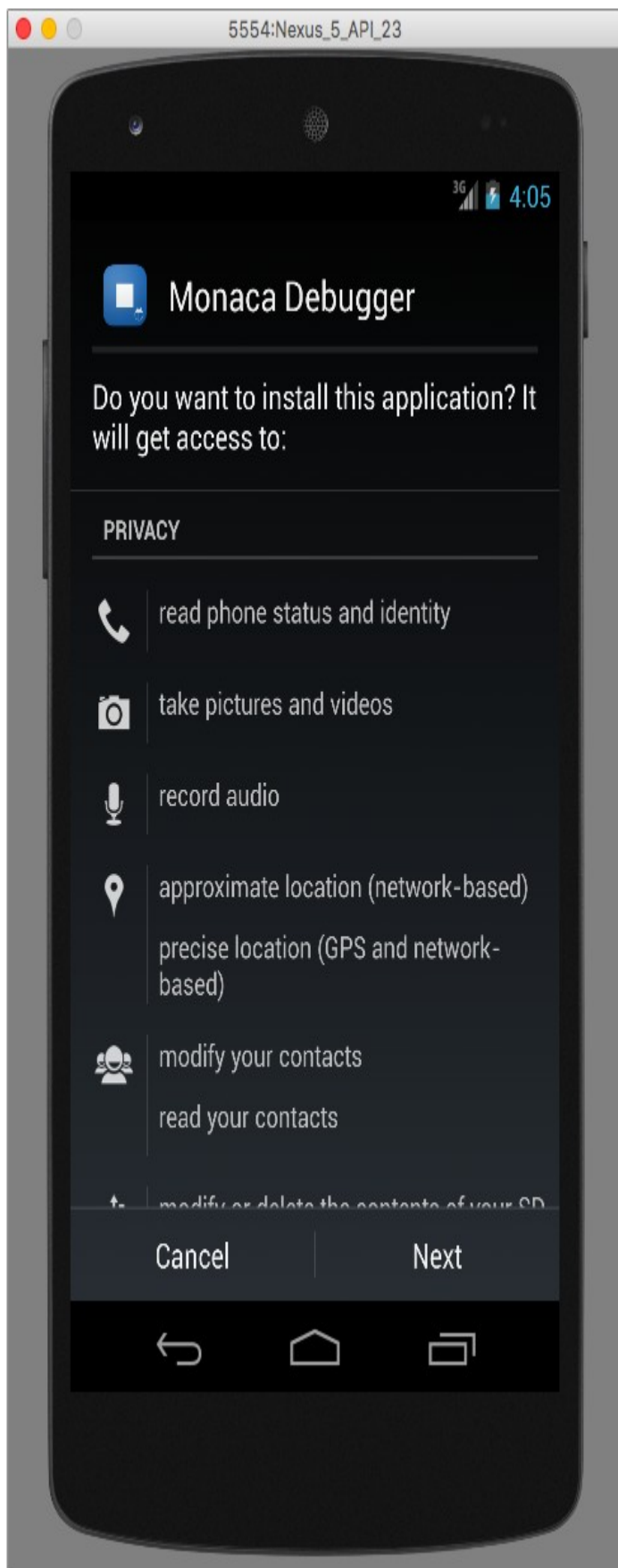
2. From your virtual device, open a browser and go to the download link you created in [Step 2](#) to download custom Monaca Debugger for Android.



3. After the download is completed, click on the downloaded file to start installing the debugger.



4. Follow the installation wizard.



5. Once the installation is completed, you can find Monaca Debugger in your apps page.

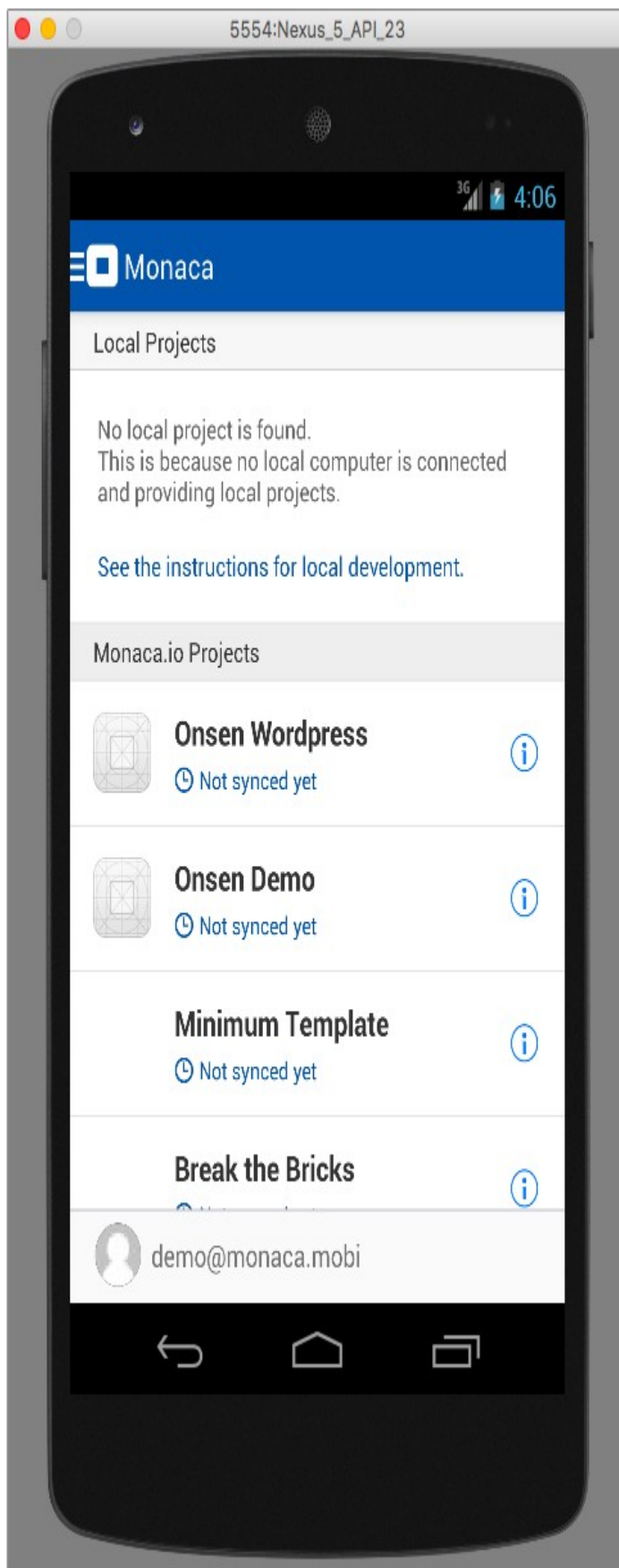


Step 4: Running a Project on Monaca Debugger

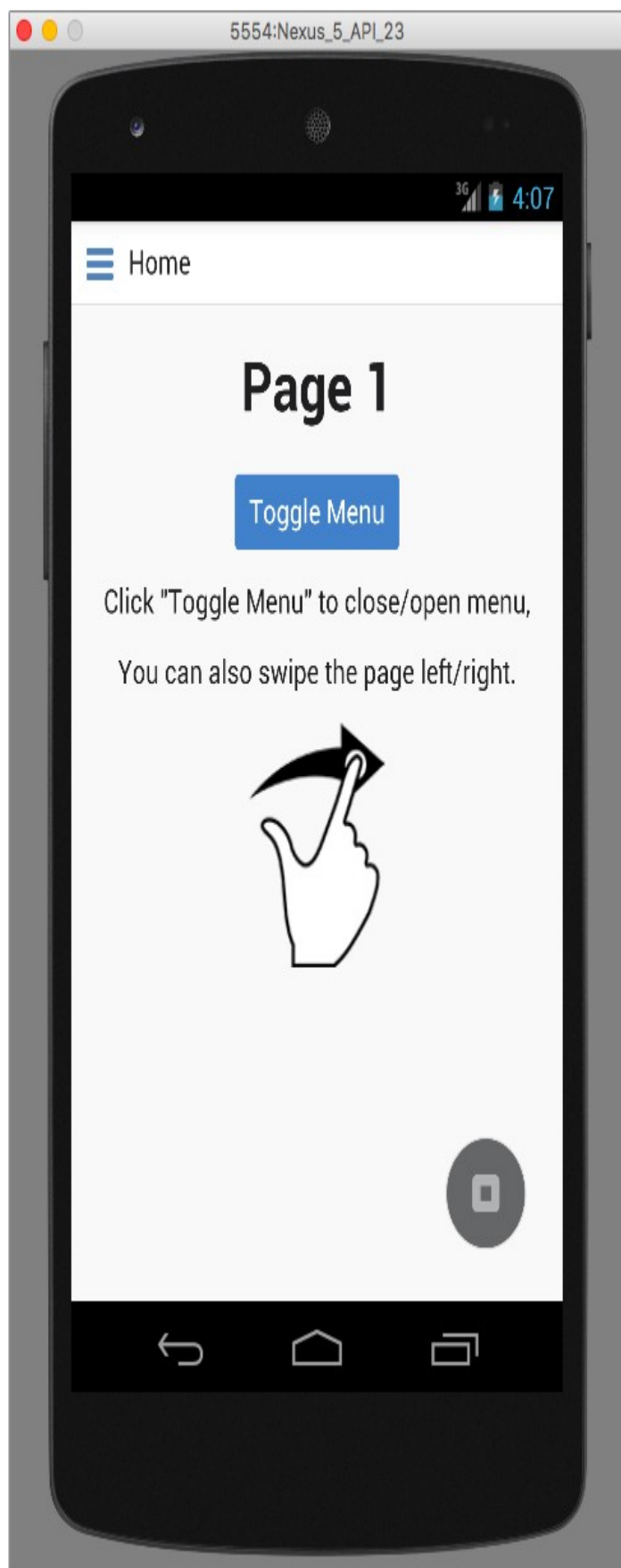
1. Open Monaca Debugger and sign in with your Monaca account.



2. Click on a project you want to run in Monaca Debugger.



3. Now, your project is running. You can start testing your project. Please refer to [Functionalities](#) and [Usage](#) on what you can do with Monaca Debugger in order to help enhancing your app development processes.



Usage

In this page, we will describe how to use Monaca Debugger with Monaca Cloud IDE and other Monaca local development tools such as Monaca Localkit and Monaca CLI :

- [Monaca Debugger with Monaca Cloud IDE](#)
 - [Monaca Debug Panel](#)
 - [Console Debugging](#)
 - [DOM Inspection](#)
 - [USB Debugging](#)
 - [Safari Remote Debugging \(for iOS and Mac only\)](#)
 - [Chrome Remote Debugging \(for Android with Google Chrome Browser\)](#)
- [Monaca Debugger with Monaca Local Development Tools](#)
 - [Prerequisite for USB Debugging with Monaca](#)
 - [USB Debugging with Monaca Local Development Tools](#)

Before starting, please install Monaca Debugger on your device or emulator. Please refer to [How to Install Monaca Debugger](#) for more information.

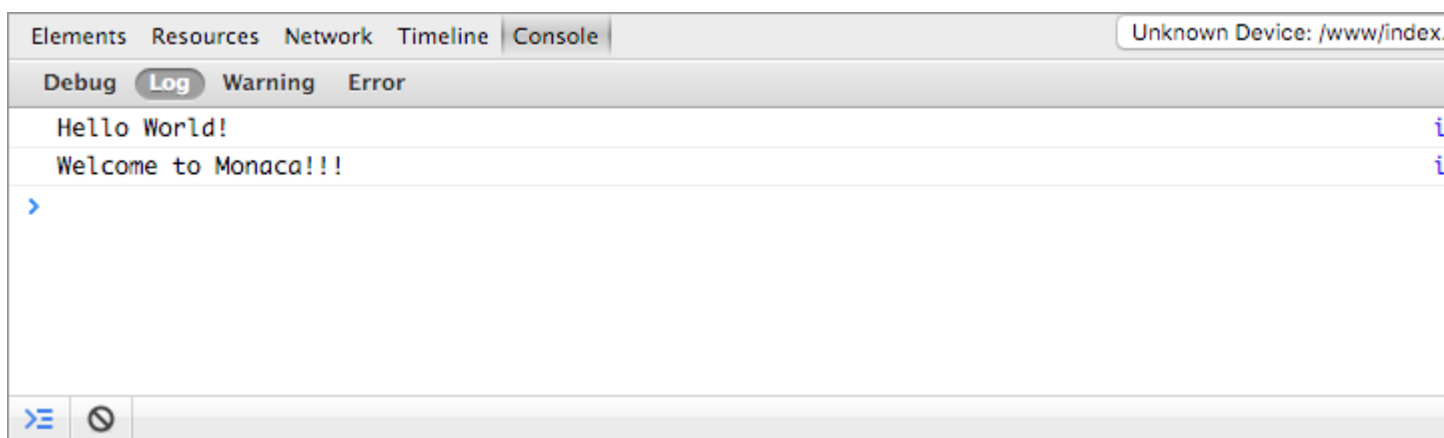
Monaca Debugger with Monaca Cloud IDE

There are 2 main debugging mechanism can be used to debug Monaca apps with Monaca Cloud IDE such as:

- [Monaca Debug Panel](#): DOM inspection and console debugging.
- [USB Debugging](#): DOM inspection, console and JavaScript debugging.

Monaca Debug Panel

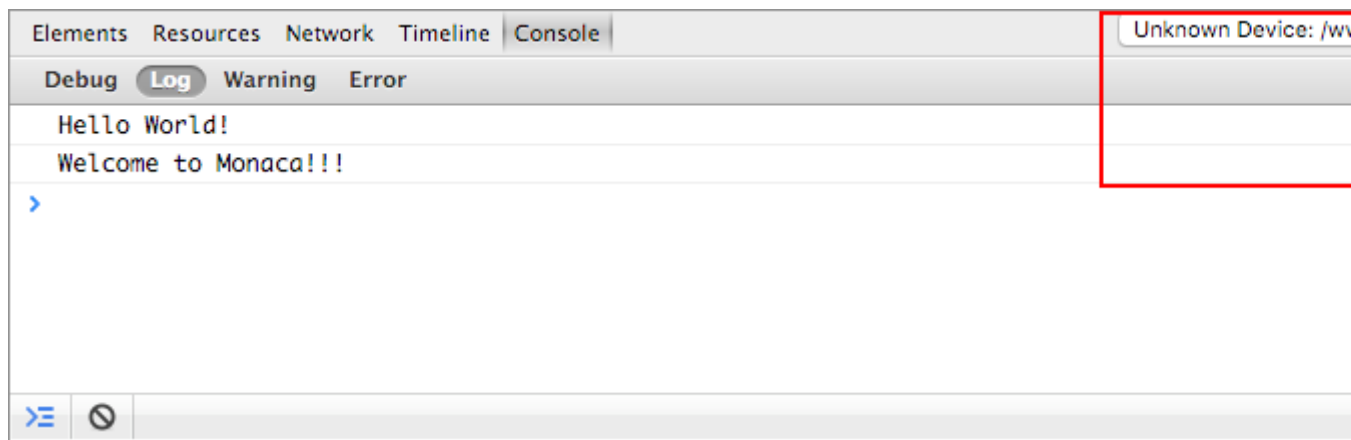
A popular Web debugging tool, [Weinre \(WEb INspector REmote\)](#), is embedded in debug panel of Monaca Cloud IDE. This tool allows you to debug your app using console debugging and DOM inspection.



In order to start debugging your app by using Monaca Debugger with Monaca Cloud IDE, please follow the following instruction:

1. Open a project in Monaca Cloud IDE.
2. Run the project in Monaca Debugger.

3. Make sure your device is connected with the IDE. For example, if your device appears in the debug panel, it is successfully connected to the IDE (see the screenshot below). Otherwise, please refresh the IDE or debugger until the connection is successfully made. After your device is connected to the IDE, you can start debugging your app.



Console Debugging

Console API allows you to write/display message to the Console using Javascript.

Here are some common used Console APIs:

- `console.log()`: displays a message to the console.
- `console.debug()`: displays a message as debug level (you can see the message in the debug tab).
- `console.warn()`: displays a message with yellow warning icon.

For more information about Console APIs, please refer to [Console API references](#).

Here is an example of using Console APIs:

1. Copy and paste the following code into the `index.html` file.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, height=device-
height, initial-scale=1, maximum-scale=1, user-scalable=no">
    <script src="components/loader.js"></script>
    <link rel="stylesheet" href="components/loader.css">
    <link rel="stylesheet" href="css/style.css">
    <script>
      var a = 1;
      var b = 2;

      function debug(){
        var c = a + b;
        console.log("debug() function is executed!");
        console.log("executed! variable c is " + c);
      }

      debug();
    </script>
  </head>
  <body>
```

```
<h1>Hello World!</h1>  
</body>  
</html>
```

2. Save the code and connect Monaca Debugger with Monaca Cloud IDE. Run the project in Monaca Cloud IDE. Now you can see those messages in the debug panel in Monaca Cloud IDE and in the app log inside the Monaca Debugger.

index.html

```
1 <!DOCTYPE HTML>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1">
6     <script src="components/loader.js"></script>
7     <link rel="stylesheet" href="components/loader.css">
8     <link rel="stylesheet" href="css/style.css">
9     <script>
10      var a = 1;
11      var b = 2;
12
13      function debug(){
14        var c = a + b;
15        console.log("debug() function is executed!");
16        console.log("executed! variable c is " + c);
17      }
18
19      debug();
20    </script>
21  </head>
22  <body>
23    <h1>Hello World!</h1>
24  </body>
25 </html>
```

Elements Resources Network Timeline Console


Unknown Device: /w

Debug Log Warning Error

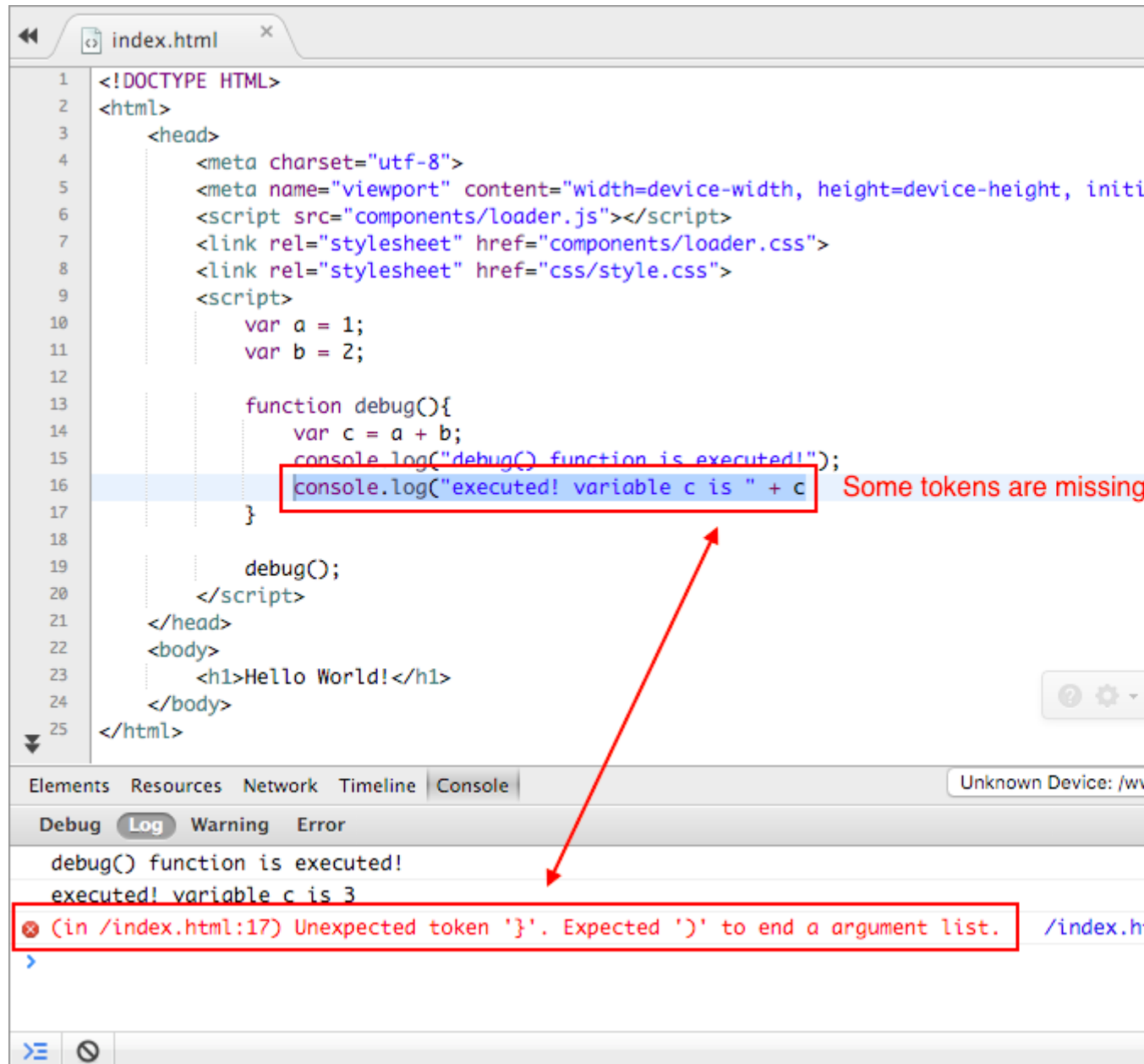
debug() function is executed!
executed! variable c is 3

>

>=

	App Log	Delete
	debug() function is executed!	
	executed! variable c is 3	

3. This debug panel also allows you to see the error log of your app as well.

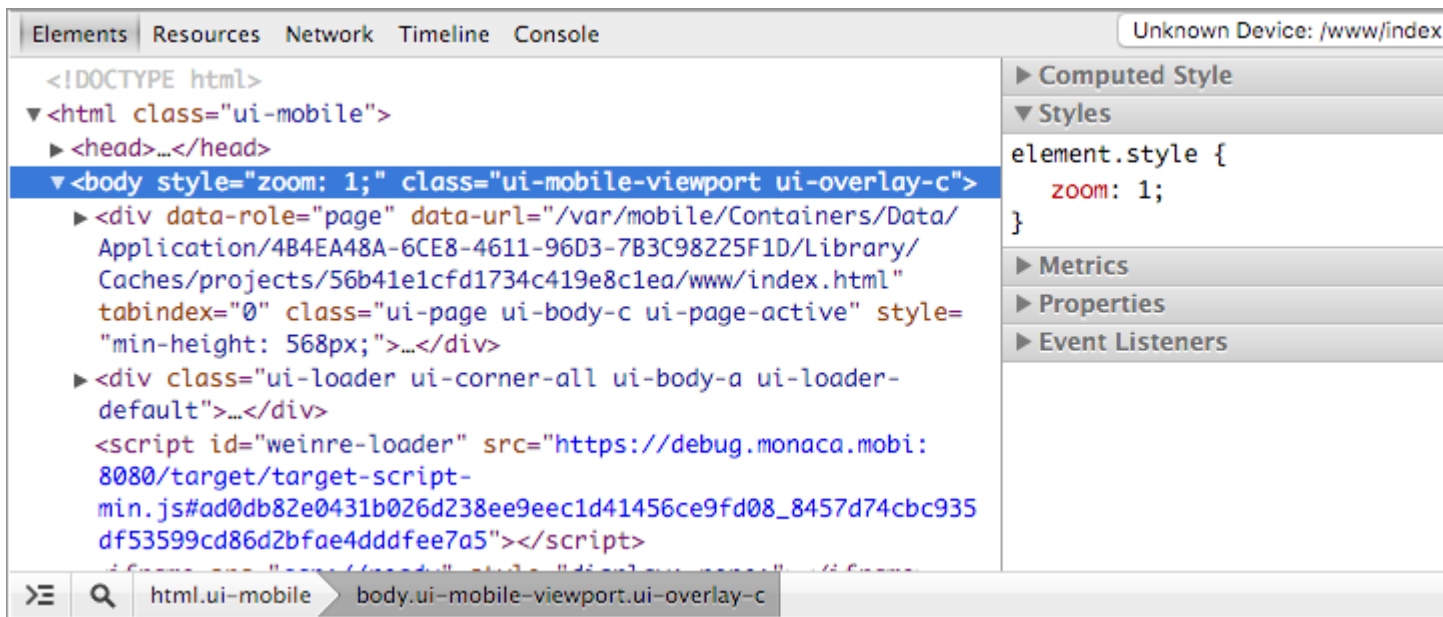


DOM Inspection

DOM (Document Object Model) Inspection allows you to:

- view DOM structure of the currently active page.
- modify the DOM structure as well as CSS of the page with live update.

For more information, please refer to [DOM Inspection and Style Editing](#).



USB Debugging

With USB debugging, you can:

- Console debugging: uses console to display messages and set debugging sessions.
- DOM inspection: views and modifies DOM structures with live updates.
- JavaScript debugging: profiles JavaScript performance, sets breakpoint and execution control.

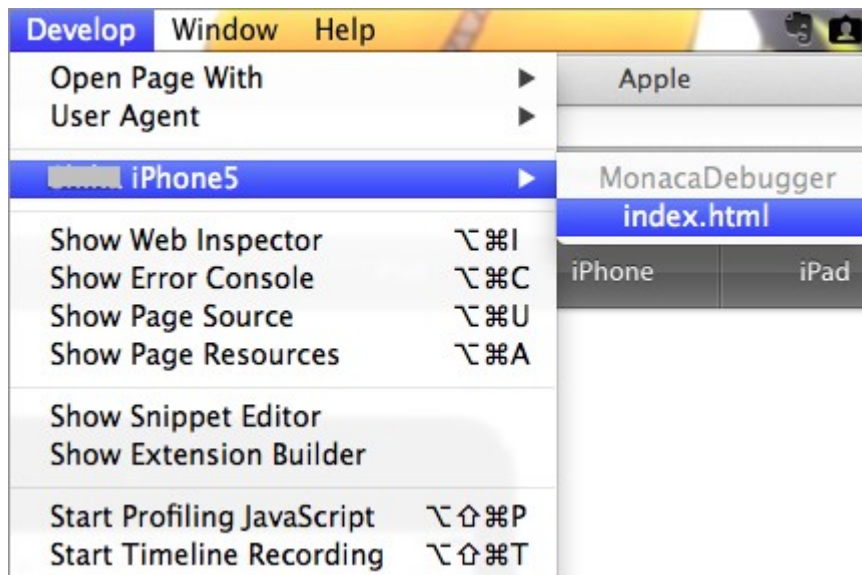
There are two ways to implement USB debugging depends on what kind of device you use:

1. If you are using iOS device, you can use [Safari Remote Debugging](#).
2. If you are using Android device, you can use [Chrome Remote Debugging](#).

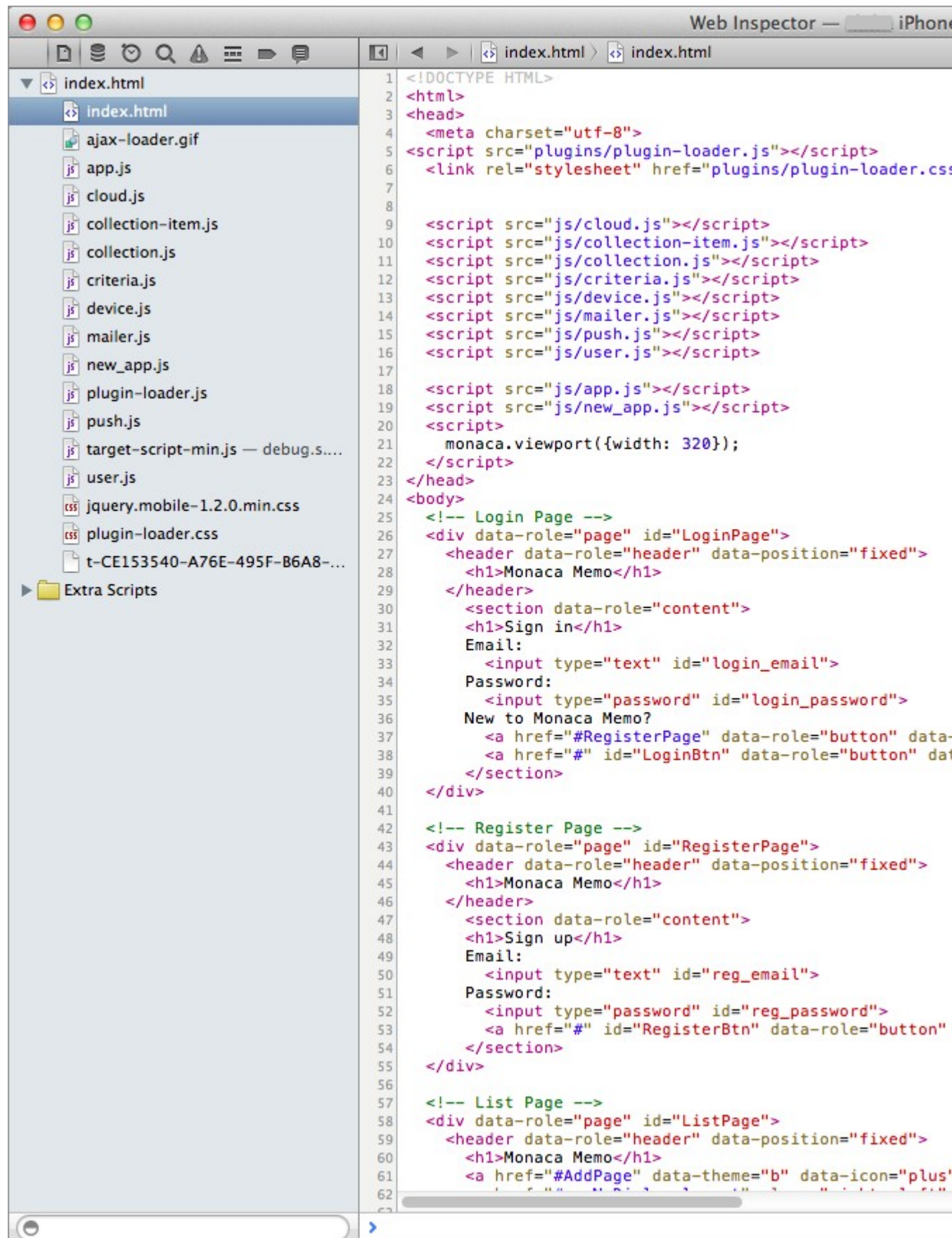
Safari Remote Debugging (for iOS and Mac only)

You are required to do some setups before using USB debugging with Monaca. Please refer to [Prerequisite for USB Debugging with Monaca](#).

1. Connect your iOS device to your Mac via a USB cable.
2. Run your Monaca project in your Monaca custom built debugger.
3. Open Safari app and go to Develop menu. Your iOS device's name should be shown in the list. Then, you can select each available page of Monaca app from a submenu belonged to your device's name.



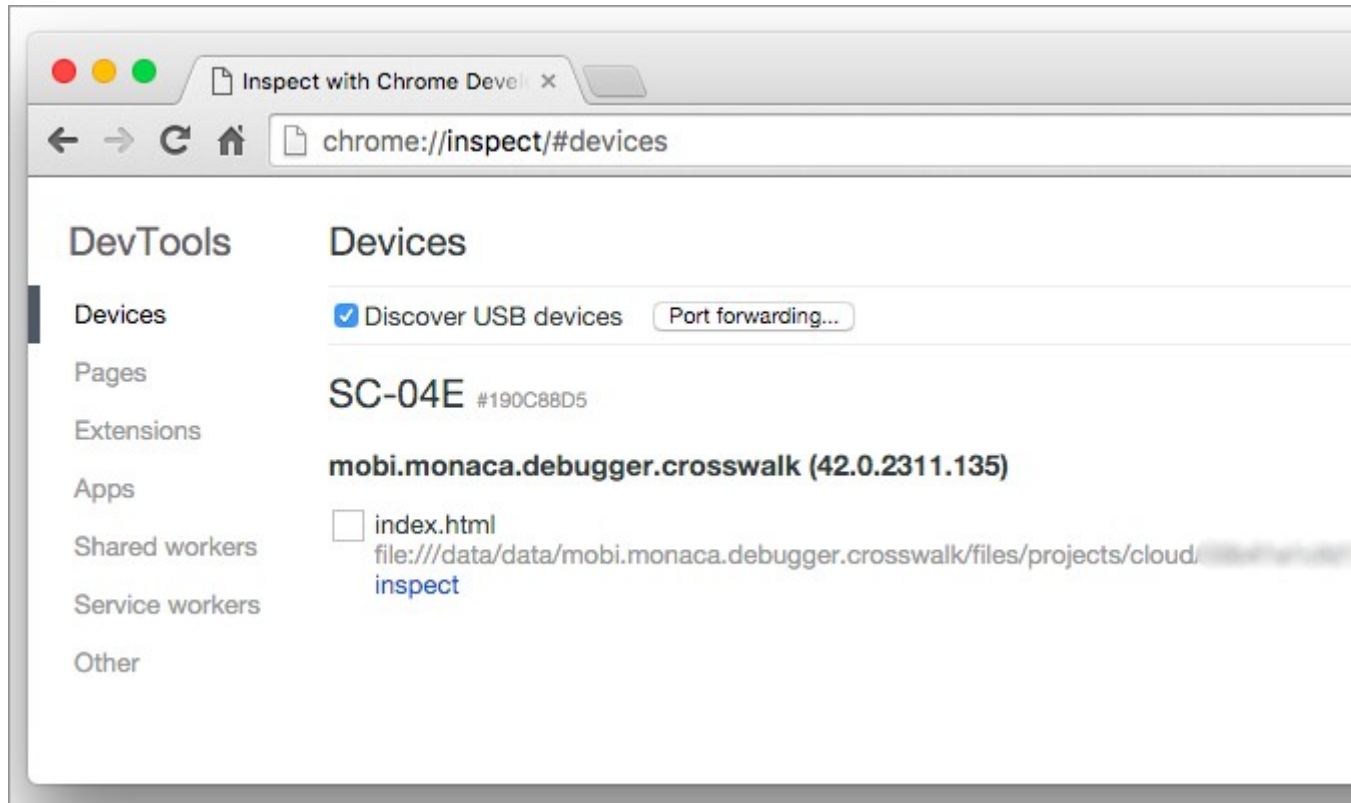
4. Then, the Web Inspector window will appear. In this window, you can use timing HTTP requests, profiling JavaScript, manipulating the DOM tree, and more. In order to learn how to use Web Inspector, please refer to [Safari Web Inspector](#).



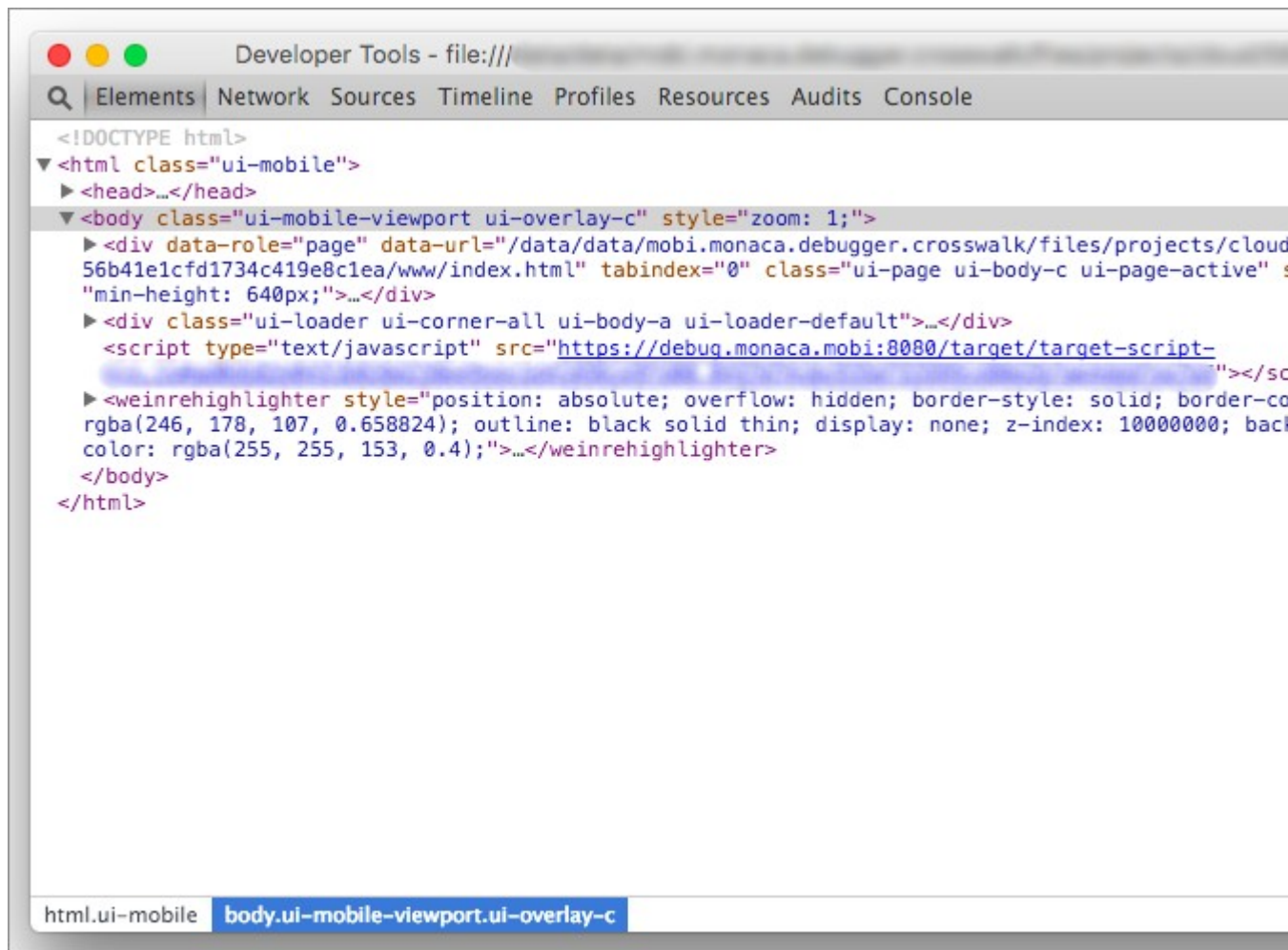
Chrome Remote Debugging (for Android with Google Chrome Browser)

You are required to do some setups before using USB debugging with Monaca. Please refer to [Prerequisite for USB Debugging with Monaca](#).

1. Connect your Android device to your PC via a USB cable.
2. Run your Monaca project in Monaca Debugger.
3. In Chrome address bar, enter `chrome://inspect/`.
4. Then, the Devices page appears as shown below. Your connected Android device should be shown there. Click inspect belonged to your device.



5. Then, the Chrome Inspection page should be appeared. Now you can start debugging your Monaca app. For more information, please refer to [How to Use Chrome DevTools](#).



Prerequisite for USB Debugging with Monaca

Platform	iOS	Android
Monaca Debugger	Custom built Monaca Debugger only	Either store version or custom built Monaca Debugger
Install Driver	For Mac OS X, necessary drivers should be already installed. Enable Web Inspector in iOS device:	For Windows, you need to check the device manufacturer to find the appropriate driver for the device. For Mac OS X, the system will automatically find the device without any installation. Enable USB debugging in Android device:
Enable USB Debug	<ul style="list-style-type: none"> Go to Settings ► Safari Scroll down and select Advanced. Switch on Web Inspector. 	<ul style="list-style-type: none"> Go to Settings ► More Select Developer options. Tick USB Debugging.
Trust Connection	The connected device should display if you trust the host	The connected device should display if you trust the host computer. Please trust the

computer. Please trust the
computer in order to get
connected.

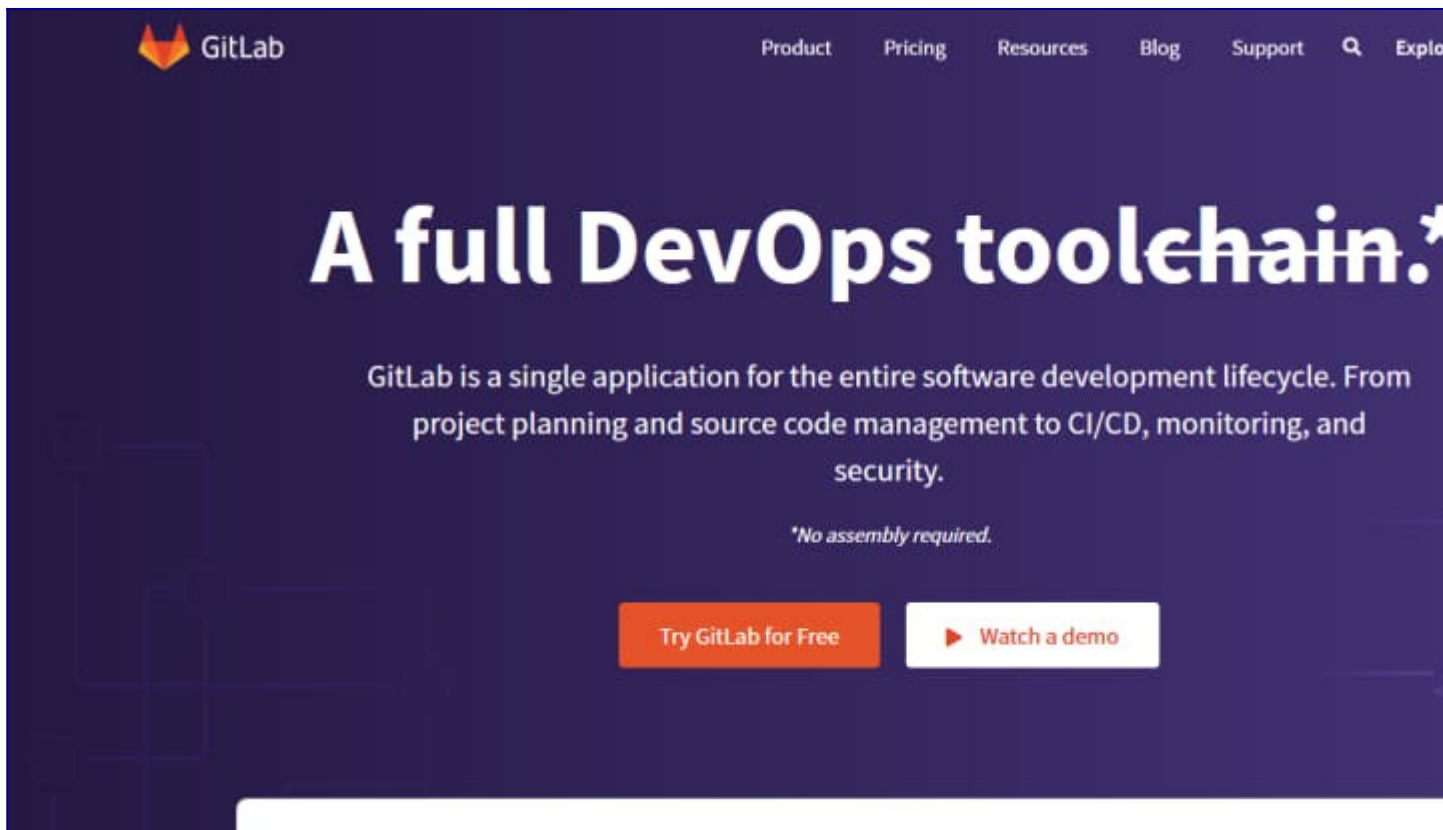
computer in order to get connected.

GitLab Pages

GitLab is one of the places where you can host your code and deploy it. It offers unlimited private and public repositories.

GitLab provides a single application for [the entire software development and operations lifecycle](#). GitLab provides everything you need to [Manage](#), [Plan](#), [Create](#), [Verify](#), [Package](#), [Release](#), [Configure](#), [Monitor](#), and [Secure](#) your applications.

-[GitLab](#)



[GitLab Pages](#) is one of the services that GitLab offers.

Host your static websites on [GitLab.com](#) for free, or on [your own GitLab instance](#).

-[GitLab](#)

Contents

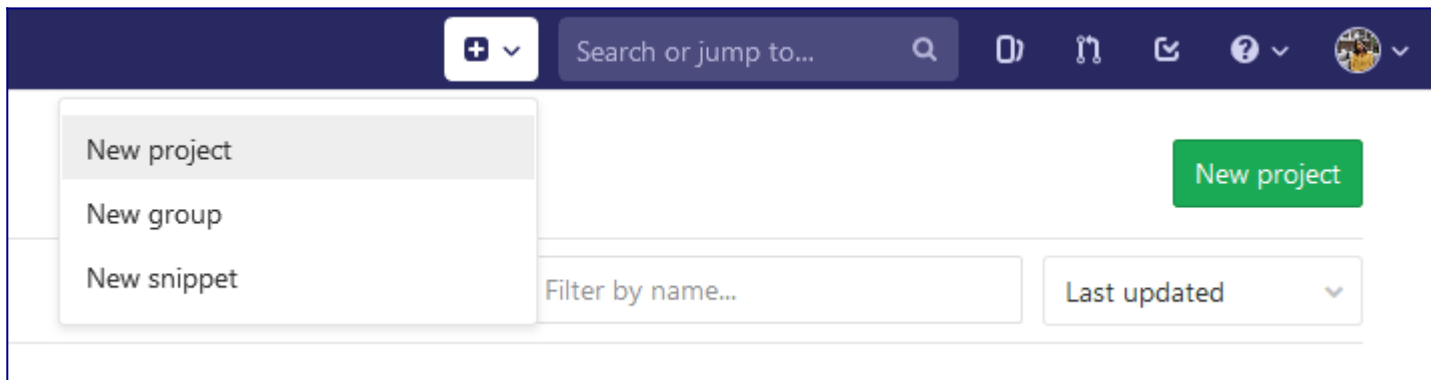
1. [Create a GitLab Account](#)
2. [Create a Repository](#)
3. [Upload your files](#)
4. [Deploy your website using GitLab Pages](#)

1. Create a GitLab Account

If you don't have a GitLab account yet, you may create one [here](#).

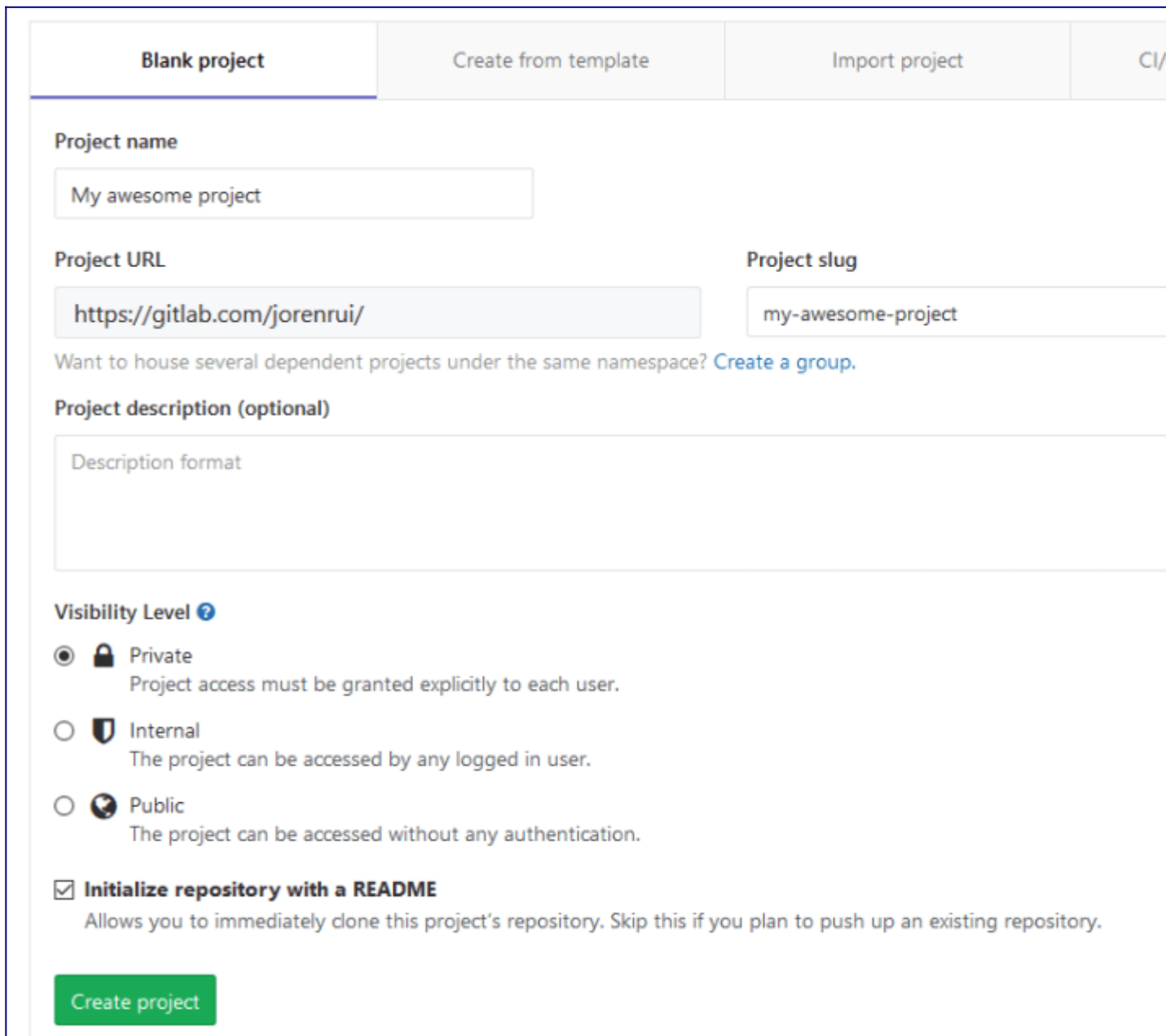
2. Create a Repository

In the navigation on the upper right corner, click the *New Project* under the plus icon. Or you can press the green *New Project* button on the right.



The screenshot shows the top navigation bar of the GitLab interface. On the left, there is a plus icon with a dropdown menu that is currently open, showing three options: 'New project', 'New group', and 'New snippet'. To the right of the dropdown is a search bar with the placeholder text 'Search or jump to...'. Further right are several icons for navigation and help. On the far right, there is a green button labeled 'New project'.

Then, fill out the details. After that, you may click *Create Project*.



The screenshot shows the 'Create Project' form in GitLab. At the top, there are four tabs: 'Blank project' (which is selected), 'Create from template', 'Import project', and 'CI/CD'. Below the tabs, the form contains several fields and sections:

- Project name:** A text input field containing 'My awesome project'.
- Project URL:** A text input field containing 'https://gitlab.com/jorenrui/'.
- Project slug:** A text input field containing 'my-awesome-project'.
- Project description (optional):** A large text area with the placeholder text 'Description format'.
- Visibility Level:** A section with three radio button options:
 - ☒ **Private**: Project access must be granted explicitly to each user.
 - ☐ **Internal**: The project can be accessed by any logged in user.
 - ☐ **Public**: The project can be accessed without any authentication.
- Initialize repository with a README:** A checkbox that is checked, with the text 'Allows you to immediately done this project's repository. Skip this if you plan to push up an existing repository.'

At the bottom left of the form is a green button labeled 'Create project'.

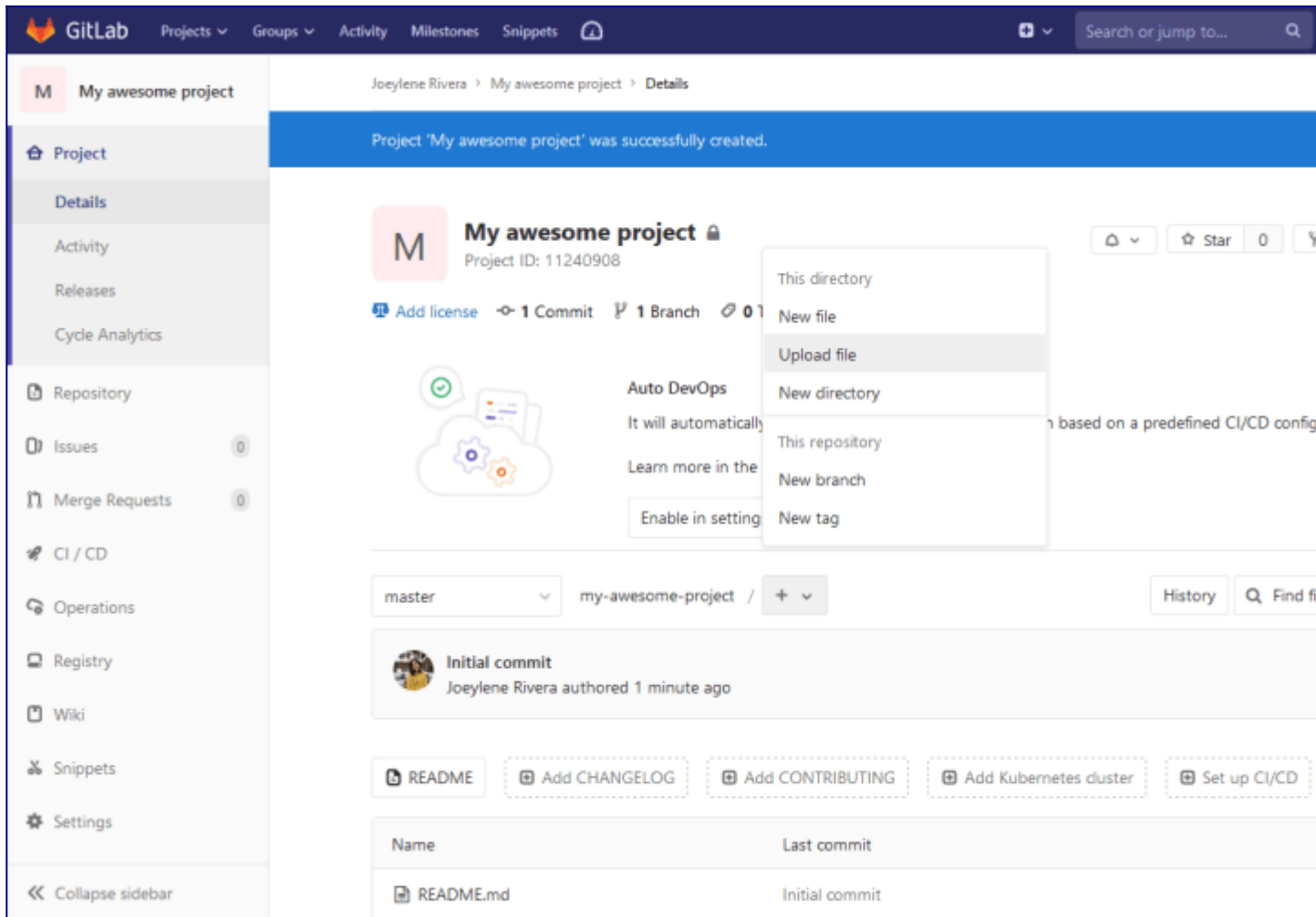
3. Upload your files

There are two ways you can add files to your repository:

- Using Git
- Via file upload

As for the file upload, click the plus icon then pick *Upload file*

Note: by default you are in the **master** branch of your repository.



Then you can drag or upload your file.

Upload New File

Attach a file by drag & drop or [click to upload](#)

Commit message

Upload New File

Target Branch

master

Upload file

As for the *commit message*, type something that describes what you did. For example, if you added an About Page then you can type *Add About Page*.

4. Deploy your website using GitLab Pages

To get started, click the *Set up CI/CD*.

master my-awesome-project / +

History Find file

Initial commit

Joeylene Rivera authored 13 minutes ago

README

Add CHANGELOG

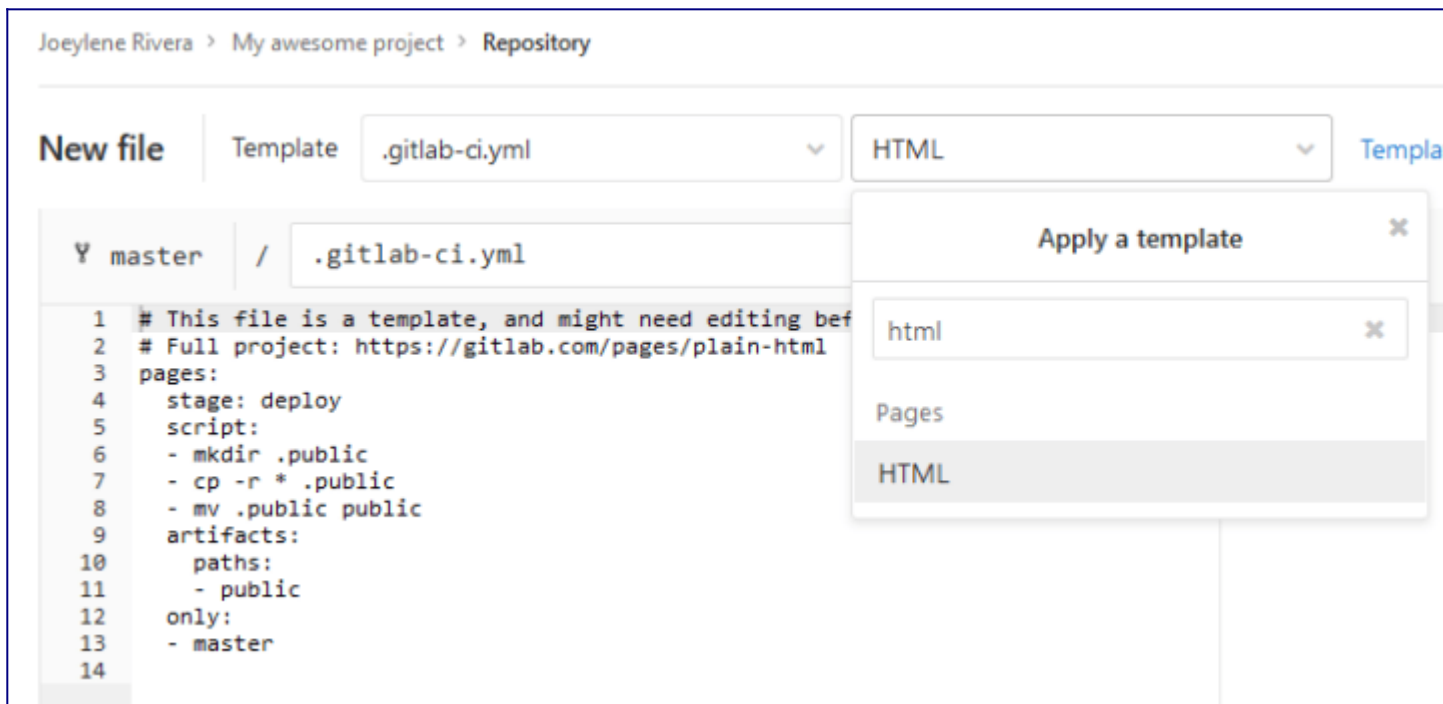
Add CONTRIBUTING

Enable Auto DevOps

Add Kubernetes cl

Set up CI/CD

This will then create a configuration file for your deployment. You can pick a template. For simple projects, just pick *HTML*.



Then press *Commit changes*.

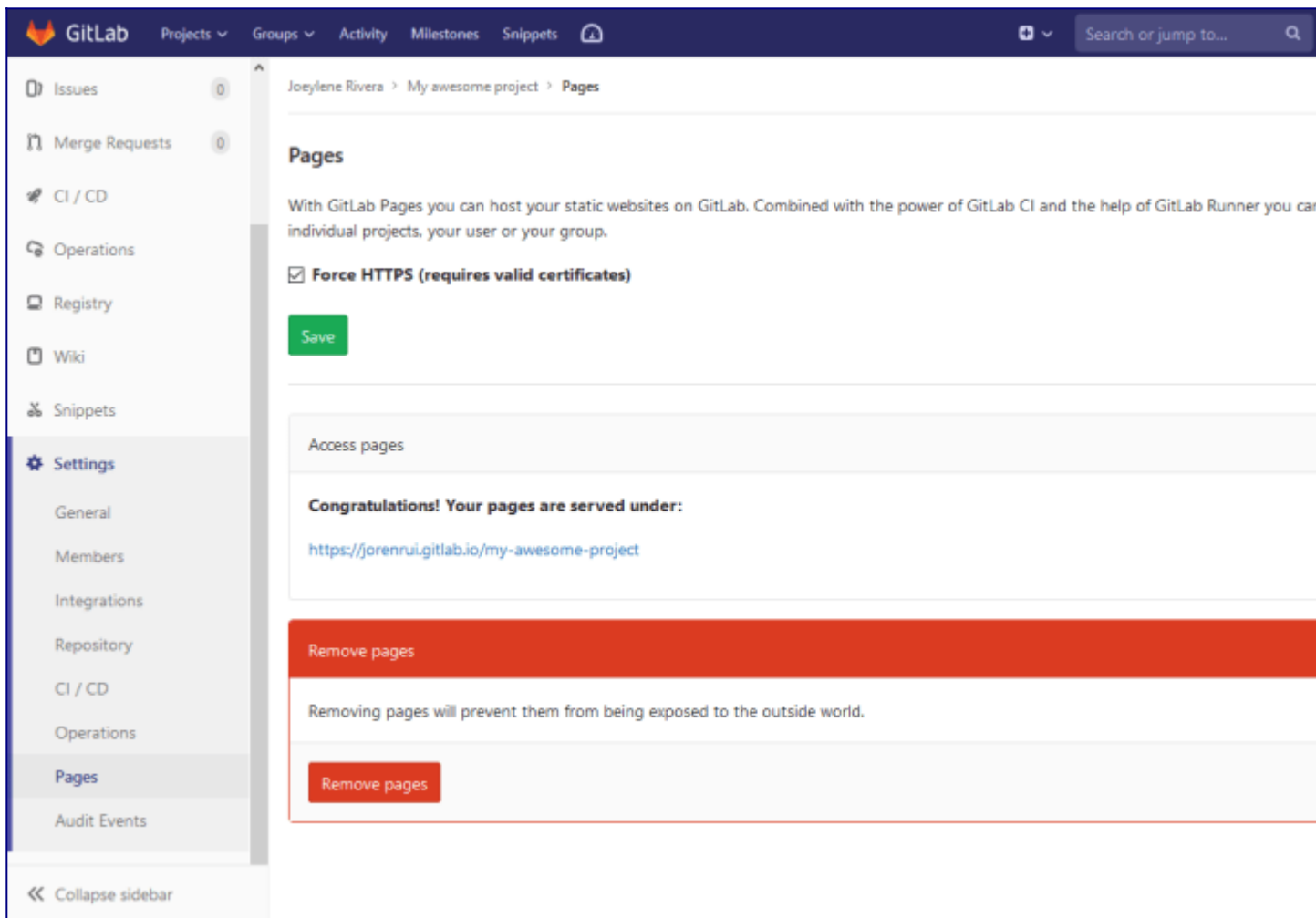
Commit message	Add .gitlab-ci.yml
Target Branch	master
<button>Commit changes</button>	

This will create a `.gitlab-ci.yml` to your project's root folder that contains:

```
pages:
  stage: deploy
  script:
    - mkdir .public
    - cp -r * .public
    - mv .public public
  artifacts:
    paths:
      - public
  only:
    - master
```

The GitLab CI/CD will then build and deploy your website using GitLab Pages.

In the sidebar, go to *Settings* then *Pages*. You'll find your website's URL there which is `https://<your-username>.gitlab.io/<repository-name>`.



You may see a *404 Error* for now. But don't worry, it just needs some time before your website is up and running. Try to check it again after a few minutes.

If you want to use `https://<your-username>.gitlab.io` rather than `https://<your-username>.gitlab.io/<repository-name>`, you just need to name your repository `<your-username>.gitlab.io`. This is suitable for portfolio websites.

With this, your website is now live. Congrats

For more information, you can visit [GitLab Pages](#).

[Return to TOC](#)

<https://dev.to/jorenruigitlab/6-ways-to-deploy-your-personal-websites--php-mysql-web-apps-for-free-4m3a#gitlab>

How to deploy PHP/HTML website with Gitlab CI

Posted by [Caleb](#) on August 23, 2016 [Leave a comment](#) (2) [Go to comments](#)

Setting up Gitlab-CI to deploy your PHP & HTML (or any other ‘static’ site) is quite simple, but poorly documented. After a few weeks of fiddling around I found the following solution. I’d love to hear everyone’s feedback.

Requirements:

1. Your webserver must have root access (to install the runner).
2. Your site/codebase should be rather small and not too complex.
 - This example uses rsync to move/sync two folders, so sites that can self-update their DB like wordpress should be OK, but I have not tested it.
 - We’re going to use rsync to move/sync two folders, so sites that self-manage media content, like wordpress, will lose that media content. You will need to tweak the rsync script to handle this yourself.
3. A gitlab install or repo you have permission to add runners to.
4. These instructions are for Linux (Ubuntu 14.04 specifically) but they should be applicable to any OS supported by the runner.

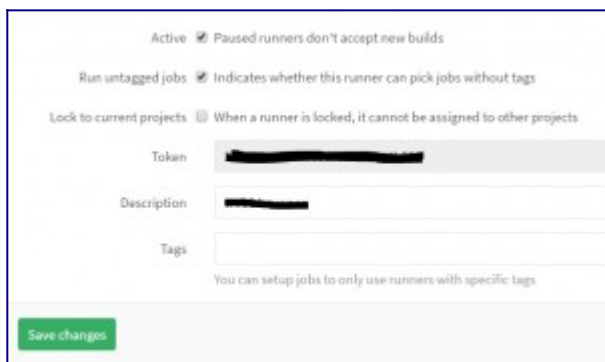
Step 1 – Install the Runner

Install the [Gitlab Runner](#) on your web server.

I’m not going to go over this one in much detail (none actually) as the documentation for getting the runner installed is pretty good.

Step 2 – Setup the Runner

If you’re like me and you run your own private Gitlab install, but are not *that* familiar with git and git tagging you might become confused about how the runner tagging works, since you can assign tags to entire repos in gitlab.



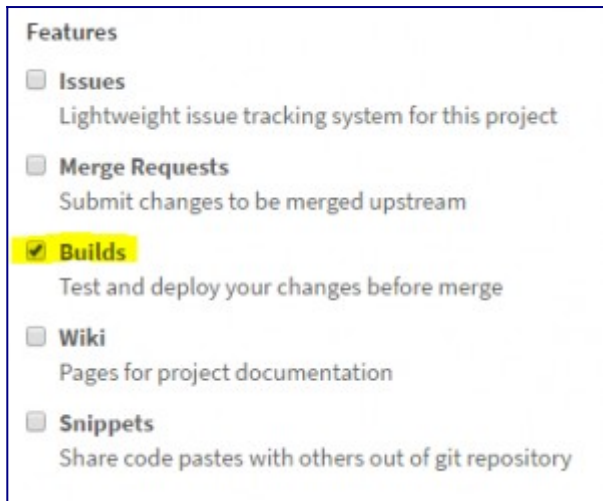
The Gitlab Runner settings screen

On the Runner settings, found in the admin section under Runners, then clicking on the runner token number, you have an option to set which tags will fire this runner. This corresponds to the tags on individual commits, **and not** the tag set to the repo. If you want specific repo’s to always use specific runners, such if you have different webserver, you will need to manually assign the runner to that repo. You can do that from the runner’s setting screen, or from the repo. Unless you

want to the madness that is [Git Tagging](#) I suggest you leave the “Tags” field empty and the “Run untagged jobs” box checked.

Step 3 – Configure the Repo

3.1 – Enable CI Builds



Make sure “Builds” is checked.

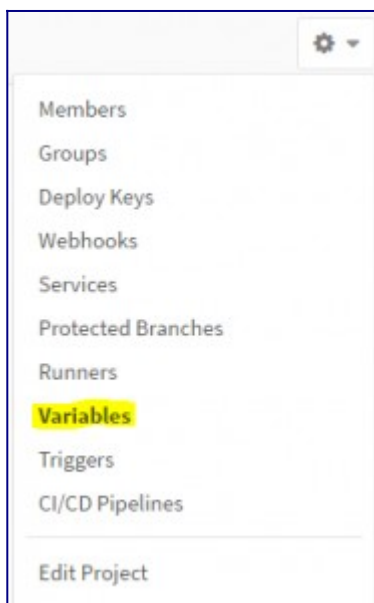
Go to your repo settings. In Gitlab 8.11 that can be done by clicking the gear icon in the upper right corner of the screen when looking at any other “Project” page .

Gear -> Edit Project

Scroll down until you see “Features” and make sure that “Builds” is checked.

In this case I don’t need any of the other features so I uncheck them. You don’t have to, but if you aren’t going to use those features it will clean up the Project overview screen a good bit if you turn them off.

3.2 – Set Project Variables



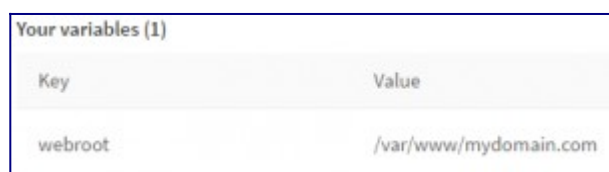
The options you see here will vary depending on what features you have enabled, and other settings in your project.

This one is sort of optional, but if you have multiple small projects you can use variables in your build script (you'll see it in a couple of steps) that allows you to re-use the same build script in multiple projects and set the few unique variables from the project itself.

You get to the variables settings page from any other page in the project by clicking on the Gear icon in the upper right.

Gear -> Variables.

Set the Key to "webroot" and the value to the web path on your webserver where your site's files reside. For example, it's normally /var/www/, so I set "/var/www/mydomain.com/". The trailing slash is important here.



Key	Value
webroot	/var/www/mydomain.com/

You should see this once you've added the new variable.

Step 4- Add build Script

You might see some errors around the project indicating that builds are not configured or working correctly. The specific message seems to be to change from one Gitlab version to another, and from one screen to another, but all of them are indicating that you have builds enabled for your project, but no build script.

Create a file named ".gitlab-ci.yml" in the root of your project. Note the leading period is important. If you're trying this from Windows it might give you grief about it, I ended up kidnapping a .htaccess file from another project and renaming it.

Open the file in your preferred text editor and set this as the contents:

```
?
  stages:
1 - deploy
2
3 pages:
4   stage: deploy
5   script:
6     - if [ -z "$webroot" ]; then echo "Need to set webroot" && exit
7 1; fi
8     - sudo rsync -rv ./ $webroot --exclude '.git' --exclude
9     '.gitlab-ci.yml' --exclude '.gitignore' --delete
10    - sudo chown user:group $webroot -R
11  only:
    - master
```

NOTE: WordPress likes to strip leading spaces. Every line except "stages:" and "pages:" are indented 2 spaces.

So whats going on here? This is a Gitlab CI build script, and it basically instructs the CI to do various tasks. Normally this is used to compile code, check it for simple mistakes, etc. We're going to re-purpose its ability to execute bash/batch commands (even on windows) to push our files where we want.

You can read the full documentation [here](#). I'll break down my script for you if your eyes got a little glossed over after trying to ingest the official help documents.

- Line 1 tells CI that we're declaring our stages. You can have as many stages as you want, but they must all be either "build", "test" or "deploy". Since we're not compiling or testing code, we just declare a deploy on Line 2.
- Line 4 is your stage name. You are free to call this whatever you want. Since I was working off the official "Gitlab Pages" build instructions, my build is called "pages".
- Line 5 declares that this is our deployment stage.
- Line 6 declares that the following lines will be our batch script.
- Lines 7-9 are batch scripts, executed sequentially. Note that on linux systems these should be *bash* scripts, while on Windows you should use *batch* syntax and commands.
 - Line 7 is a sanity check for our variable. If it isn't set, this check dies and returned a 1 exit code, which causes the entire build to stop and fail. This prevents rsync from doing anything funny if you forgot to set the variable, and if you have build notifications enabled, you'll be prompted with the text "Need to set webroot".
 - Line 8 is the real meat & potatoes of this whole thing. Since this commands run from the temp folder the runner makes for itself, this grabs the entire contents of the current folder "." and rsyncs it to our webroot folder. You did set that variable, right?
 - The three exclude flags tell rsync to ignore the git & gitlab specific files. You're free to add more if your project needs it.
 - The -delete flag tells rsync to delete anything in the webroot folder that isn't in the new temp folder. If you delete a file from your repo, this step makes sure its also cleared from your webroot.
 - If you like to keep your web-facing code someone else other than the root of your repo (such as /pub_html or some such) you will need to adjust the "." to be "./pub_html" or w/e you use.
 - Line 9 changes the ownership of the new files in the webroot to be whatever owner & group you want. This isn't strictly needed, if your project is strictly read-only from the web server side of things this step can be skipped. I like to have this set to same owner & group as if I had uploaded the files, in the event I need to SSH in and make a quick change to (zomg, testing in prod?) something real quick to appease the Powers-That-Be before committing the change to the repo.
 - Lines 10 & 11 makes it so that this job will only fire for changes to the 'master' branch. You could use this to have multiple branches deploy to different folders if you wanted, but you would need to extend the variables and/or adjust the rest of the script.

If you want to keep the 'sudo chown' command in there, you'll need to make a sudoers change to allow it. On Ubuntu & debian:

2

```
1sudo visudo
```

add at the bottom:

[?](#)

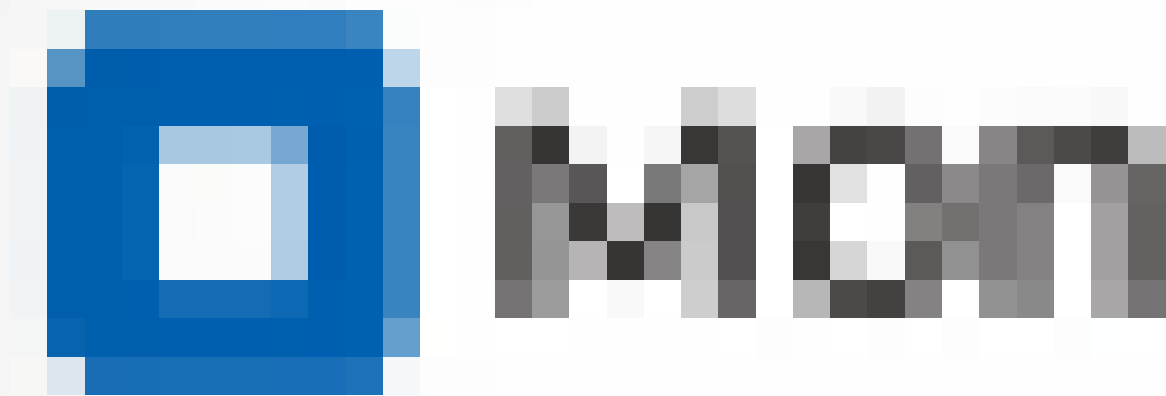
```
1gitlab-runner ALL=(ALL) NOPASSWD: ALL
```

This will allow the gitlab runner to run any sudo command. It is a potential security issue being wide open like that, and if it concerns you you can [lock it down a bit tighter](#).



[MPyK](#)

[Sep 30, 2019](#) · 3 min read





The Cordova team has recently launched the new Cordova Electron platform which gives Monaca users the ability to expand their smartphone application to the desktop market as a **desktop application**.

Note for Monaca Users: The Electron platform is now available for all plans including the Free Plan.

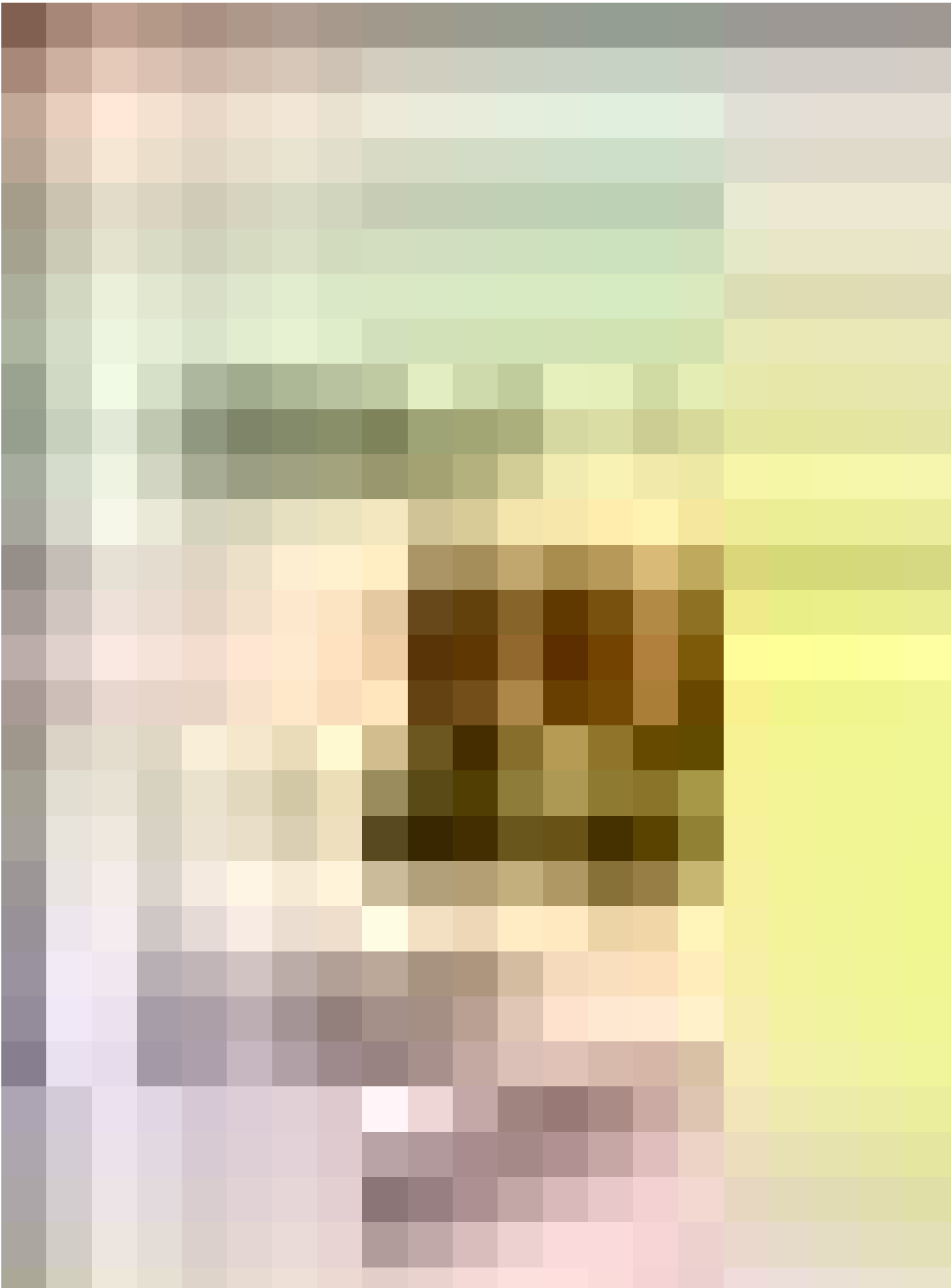
In this article, we will illustrate how we can build desktop applications with Monaca. If you want to find out more about Cordova Electron, we have the right blog just for you [here](#).

Creating an Electron Application

First of all, we will create a TODO application with an already prebuilt template provided by Monaca. The best way to start locally is by using the [Monaca CLI](#) tool to create a brand new Monaca project with the prebuilt template. We will also use this tool to perform various development and build tasks.

Monaca CLI is the full stack development tool for hybrid apps. It is compatible with Cordova CLI and makes the development much easier and faster.

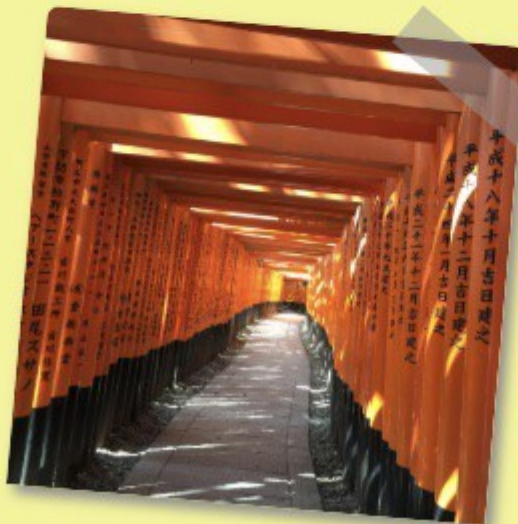
If you haven't installed it to your environment already, you can install it by running `npm install -g monaca` in your favorite terminal. Then you can run the following commands to create and preview the sample electron application.



things to do



check!



visit F



edit



delete

The TODO application template that we are using is unique from other traditional TODO apps. It has a special feature that allows users to insert a picture along with the todo memo. The functionality uses the device's built-in camera for capturing images. For more implementation details of this feature, please refer to this [tutorial](#).

Building an Application

To build the application, I suggest using the Cloud IDE. You can quickly access the Cloud IDE Build page from the terminal by running the following command.

After the project has uploaded to the cloud successfully, a new browser tab will be open as below.



Build & Build Settings

Android

iOS

Windows

macOS

Linux

Web

App Settings

Android

iOS

Windows

macOS

Linux

Web

Project

Cordova Plugins

JS/CSS Components

Service Integrations

Build

Build Environment Settings

Continuous Integration **BETA**

Service

Deploy Services **BETA**

Build History

Build Android App

Build for Debugging

Build for Release


Custom Build
Debugger


Debug Build

An Android app with a dummy signature will be built.

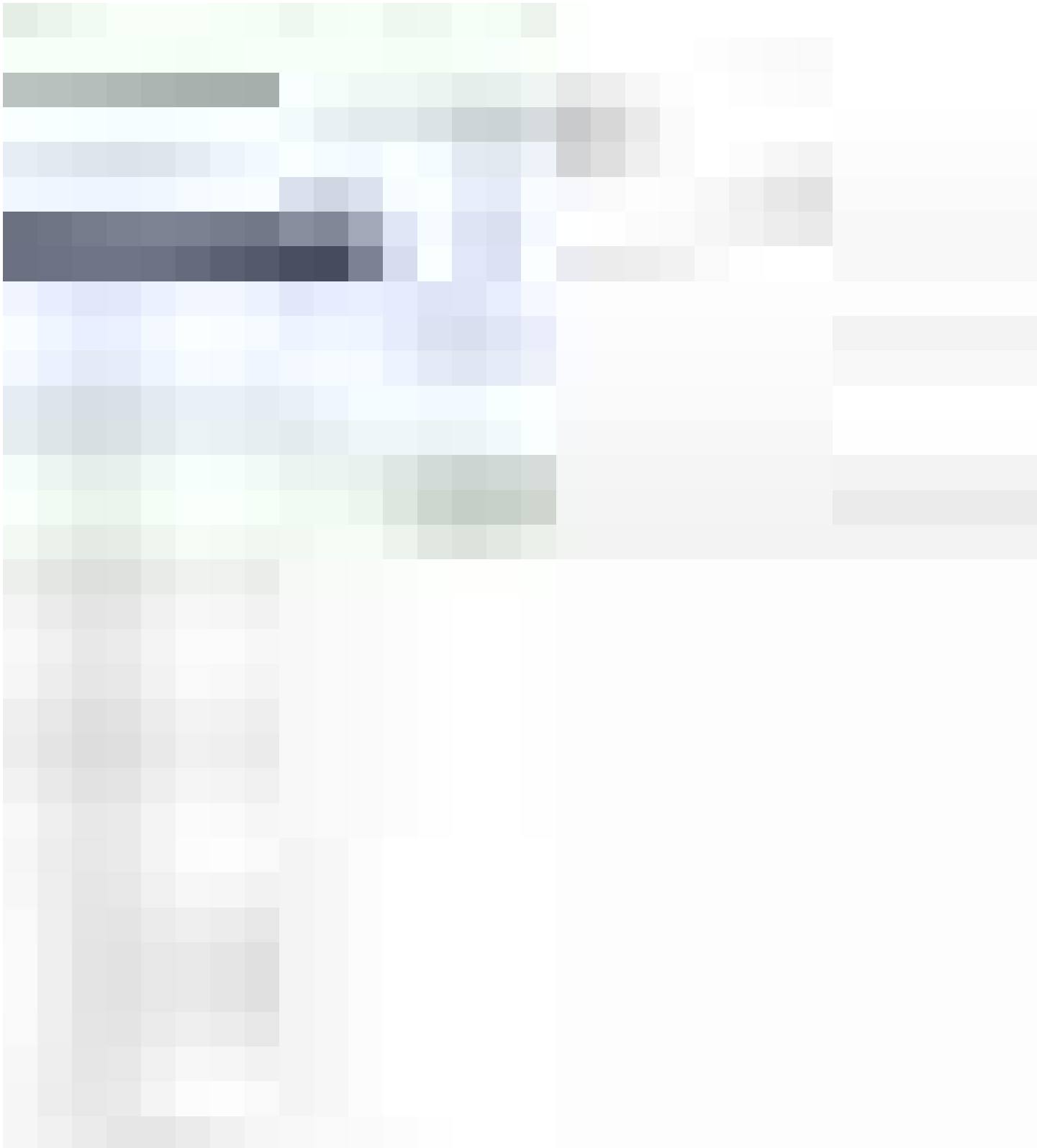
A debug-build application has no signature, so it cannot be registered in Google

From this page, we can see that Monaca supports various platforms. The currently supported platforms are Android, iOS, Windows, macOS, Linux, and PWA.

In this example, we will build a releasable macOS application. The build page itself is very simple and straightforward.

- Select “macOS” form the left panel.
- Choose “Build for Release” as the build type.

- Click the “Start Build” button.



Build & Build Settings

Android
iOS
Windows

macOS

Linux
Web

App Settings

Android
iOS
Windows
macOS
Linux
Web

Project

Cordova Plugins
JS/CSS Components
Service Integrations

Build

Build Environment Settings
Continuous Integration **BETA**

Service

Deploy Services **BETA**
Build History

Build macOS App

Build for Debugging

Build for Release



**Mac OS
Release Build**

An Electron macOS application will be built.

App Settings

App icon, splash screen, file version number etc. can be set in the macOS App Set

It may take a few minutes for the project to build based on the project size, your account subscription plan, and how many builds are in the queue. Once the build is successful, you can download the package from the build result page.

Build & Build Settings

Android
iOS
Windows
macOS
Linux
Web

App Settings

Android
iOS
Windows
macOS
Linux
Web

Project

Cordova Plugins
JS/CSS Components
Service Integrations

Build

Build Environment Settings
Continuous Integration **BETA**

Service

Deploy Services **BETA**
Build History

Build



Electron macOS Release Build

Project v1.0.0

Your build is successfully finished. Please

The same source code could be built for different platforms as well. For more detail about building other platforms, please refer to the guideline [here](#).

Other Useful Resources

For more tutorials with Electron platform, we compile some useful tips and tricks [here](#).

Monaca is built on top of the Cordova framework, Therefore, any Cordova projects can be imported to Monaca. However, if you are coming from a different framework/platform, please refer to this migration [guideline](#).

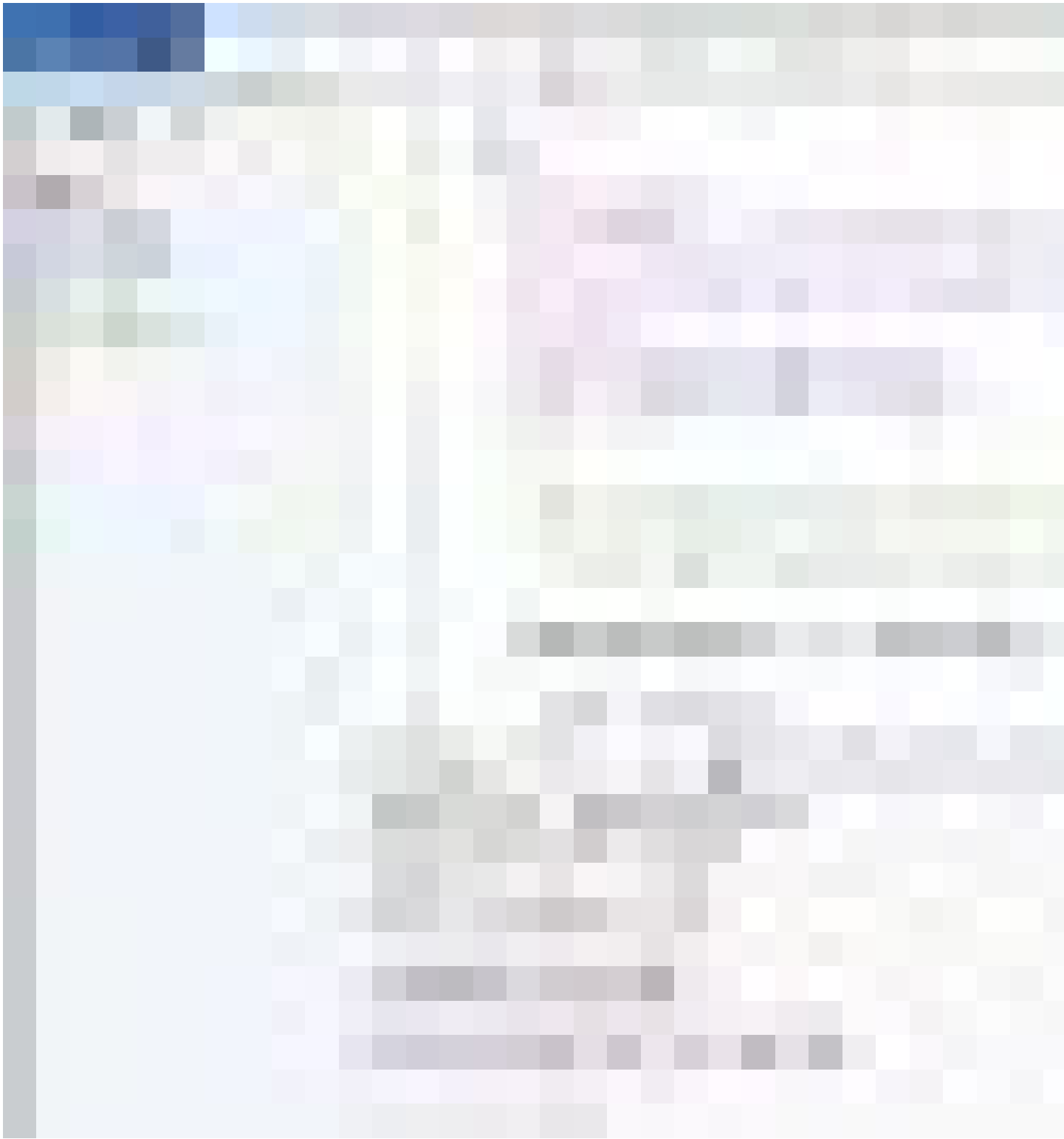
Getting the Most out of the New Monaca Cloud IDE with the integrated terminal



[David Alvarez](#)

[May 31, 2018](#) · 4 min read

Hello developers! As you may know, we've recently released the brand new Monaca Cloud IDE. Today, we are going to show you what is great about and how you can make the most out of it!

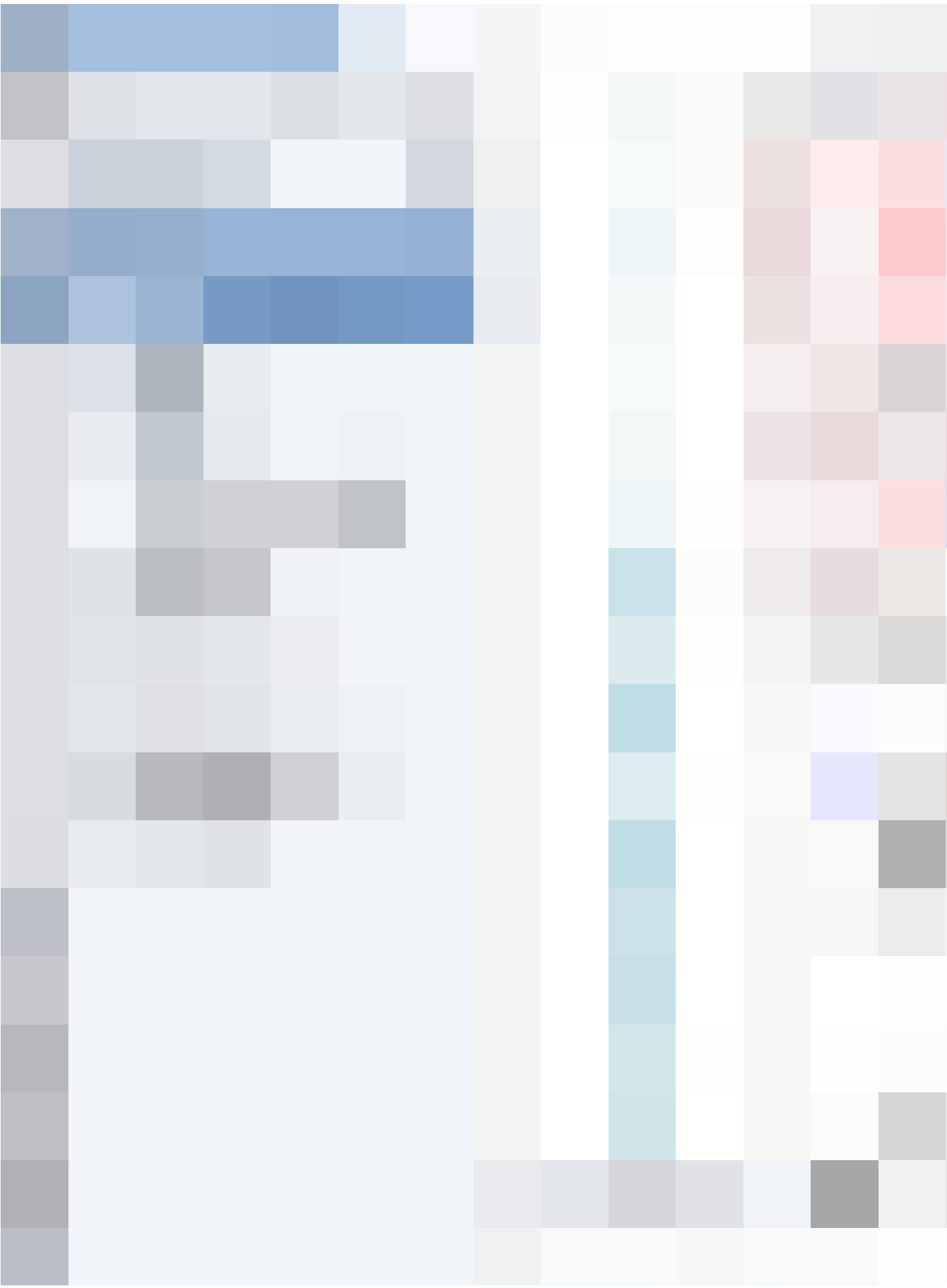


We have worked for months on this to offer a better development experience. The most important improvements are the followings:

1. **Integrated Terminal:** the embedded terminal with transpiling & hot-reloading support
2. **Flexible Golden Layout:** personalize IDE's interface as you like
3. **Multiple Previewers:** speed up testing
4. **Monaco Editor:** new and improved code editor

1. Integrated Terminal

We are very excited to release this new addition to Monaca Cloud IDE. With Integrated Terminal, Monaca Cloud IDE now supports the highly coveted **transpiling** feature. In addition, Hot Reloading and File Watch are also available.



Transpiling support

Integrated Terminal makes it possible to to create, manage and develop **transpilable projects** (such as React, VueJS, Angular, and more) right within Monaca Cloud IDE.

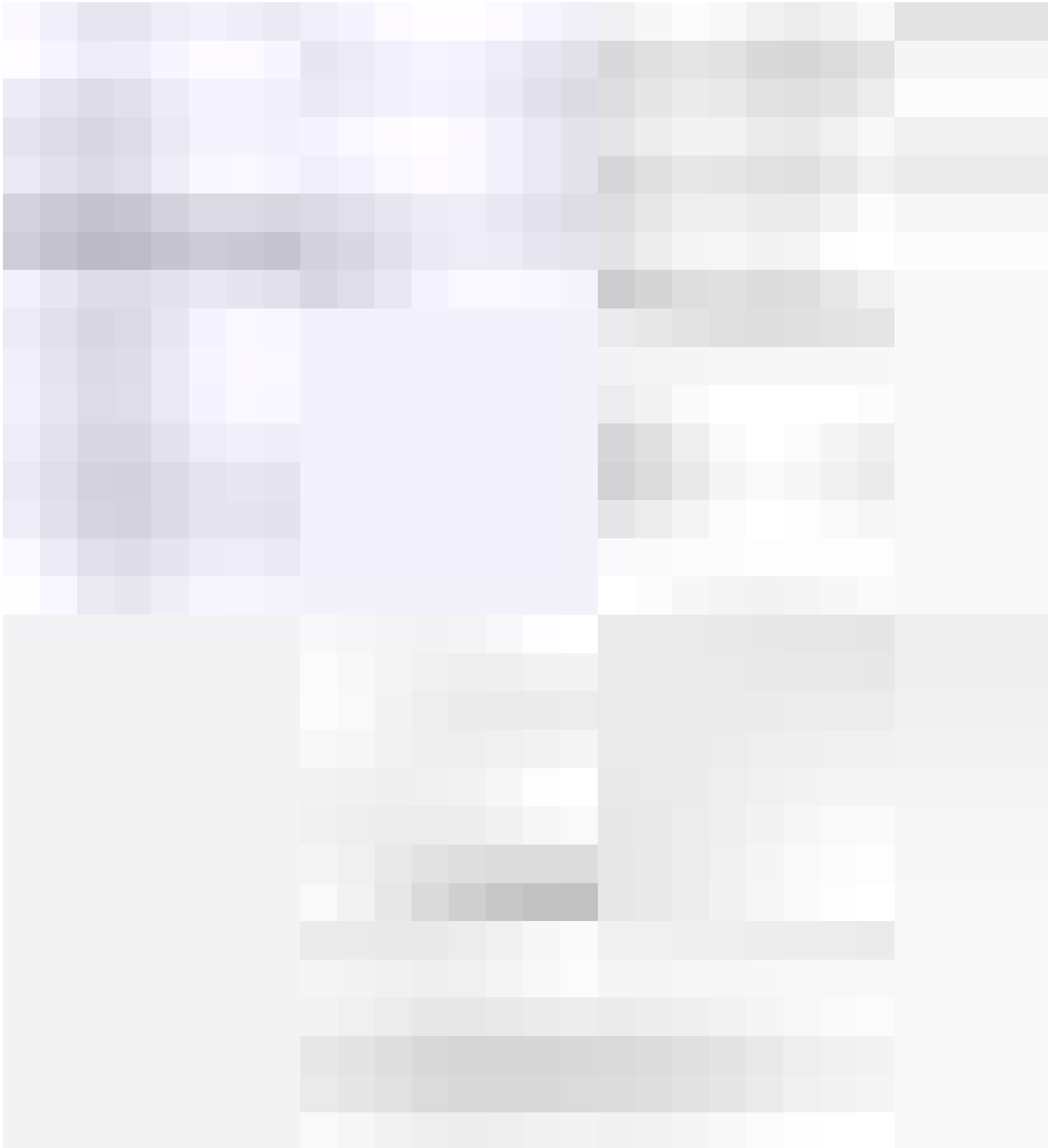
Prior to this release, transpiling projects could only be run with Monaca CLI or Localkit.



Hot reloading for transpilable projects

Every time you made a change and saved it in the old IDE, the Previewer refreshed the whole application showing the index page. Unlike the old IDE, the Previewer **only refreshes the current page** in the new IDE. This feature is called *Hot Reloading*. It allows you to see the instant update of the current page you are working on without having to start from the beginning every time you make changes.

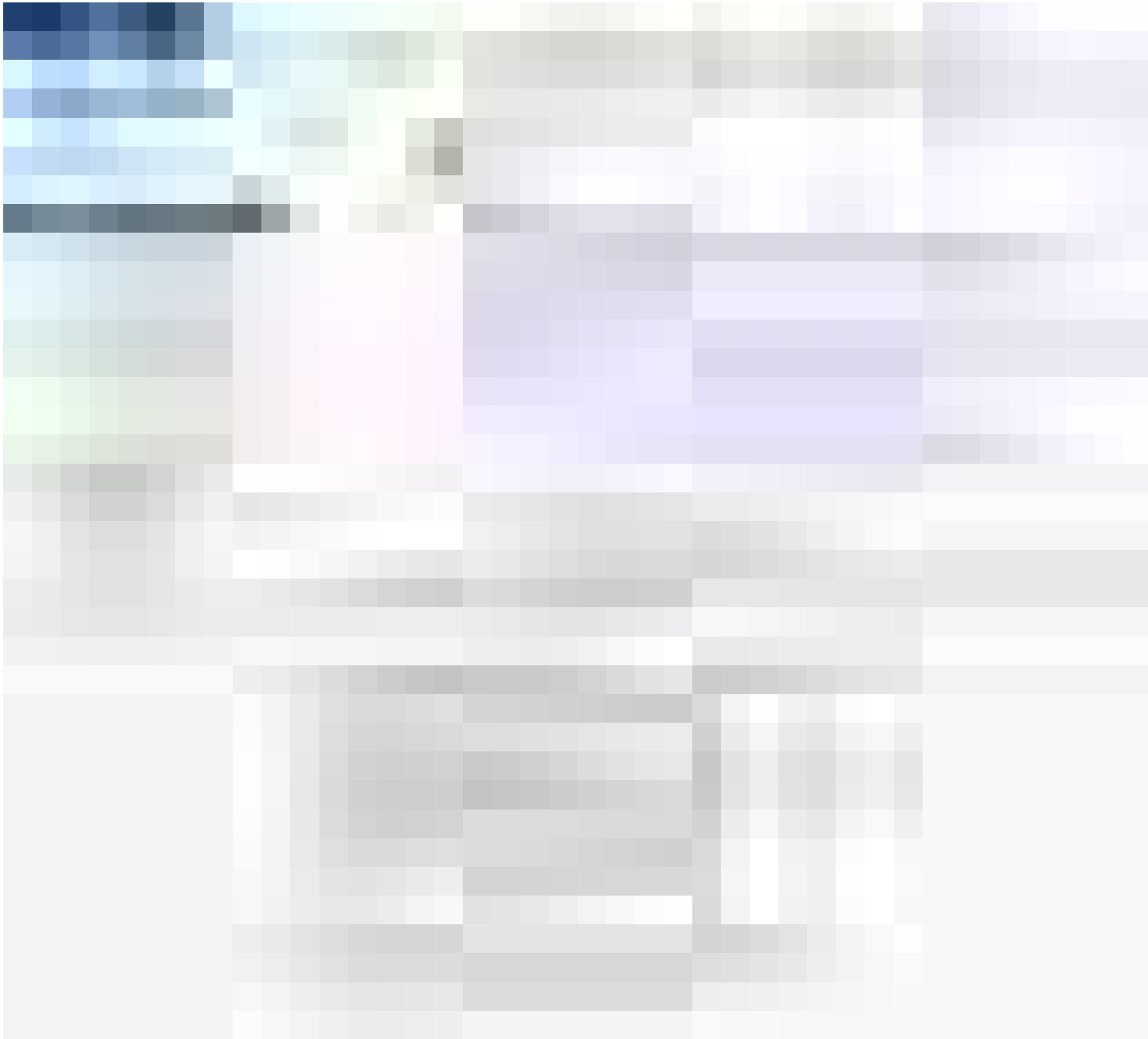
Note: This feature is only available with transpilable projects.



File Watch

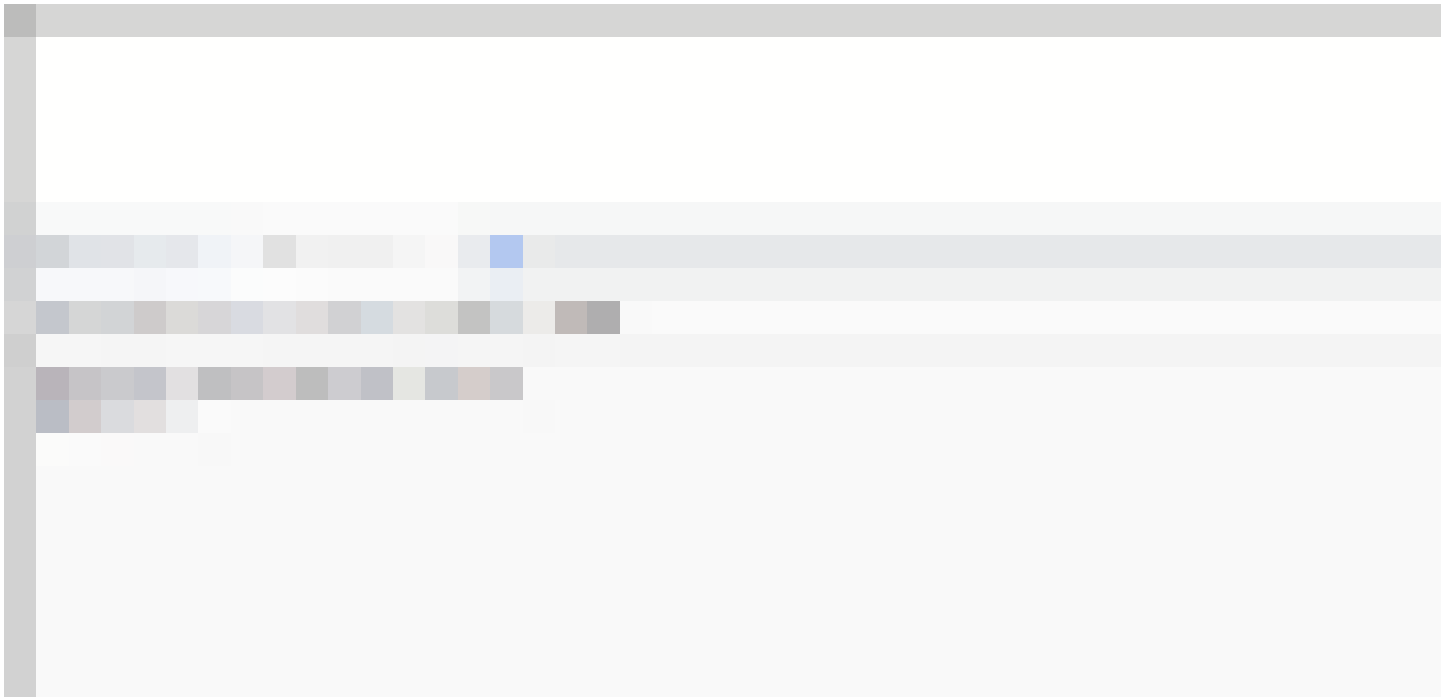
In the old IDE, whenever you made changes to project files such as adding a new HTML file, uploading new images and so on, the Previewer was not able to detect these changes automatically. You needed to manually refresh the Previewer. However, with the new IDE, such changes are detected and the Previewer will be updated automatically. This feature is called *File Watch*.

Here is a quick and simple example to demonstrate this feature:

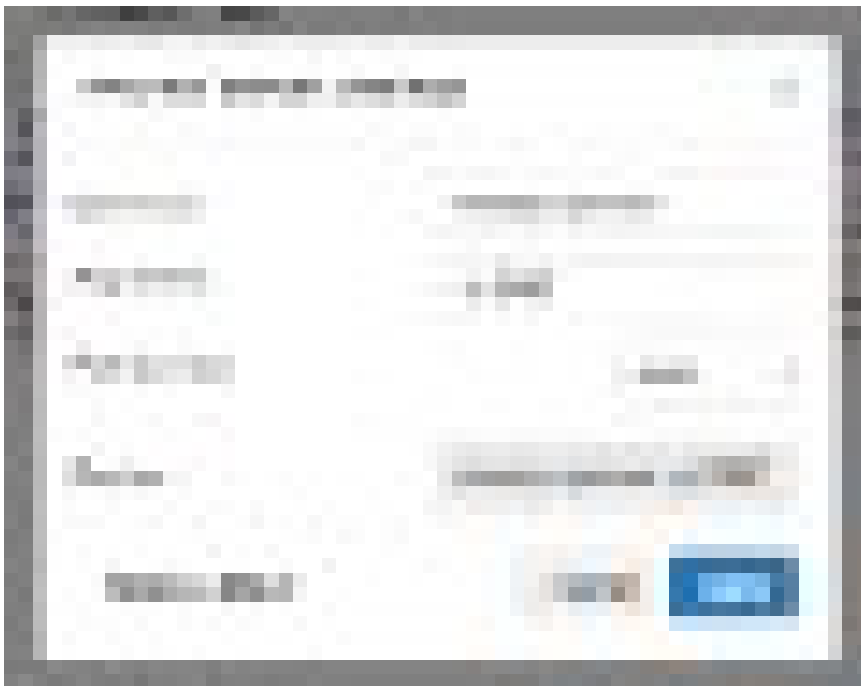


Server setting

Preview Log tab shows the output from the Preview server for each project. Previewer will connect to the Preview server once it accepts HTTP request. By default, `monaca preview` command and port `8080` are used in the Preview server. However, you can change to your own Preview server by clicking the gear icon.

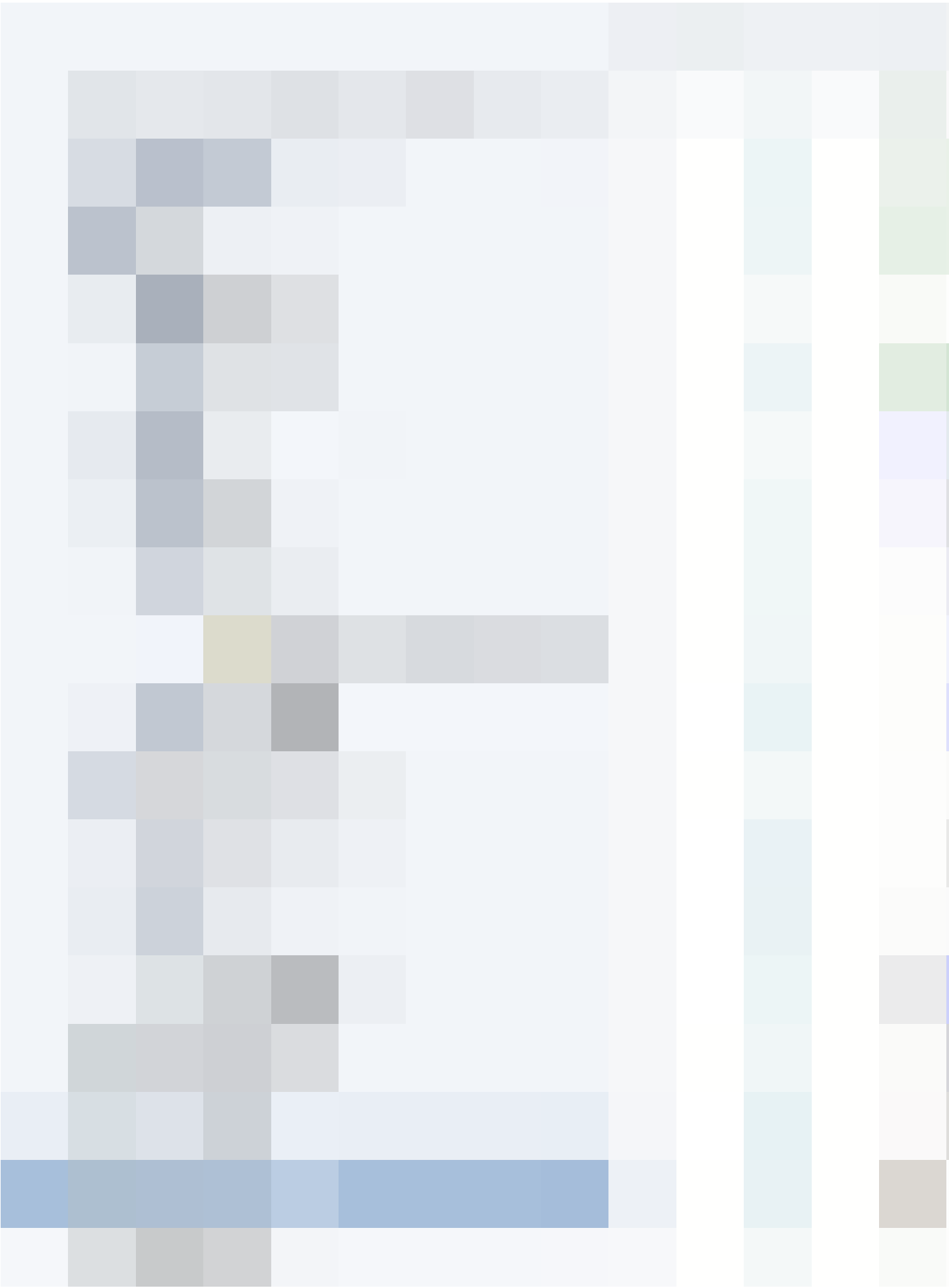


In the Settings dialog, serving command and port number can be configured.



2. Golden Layout

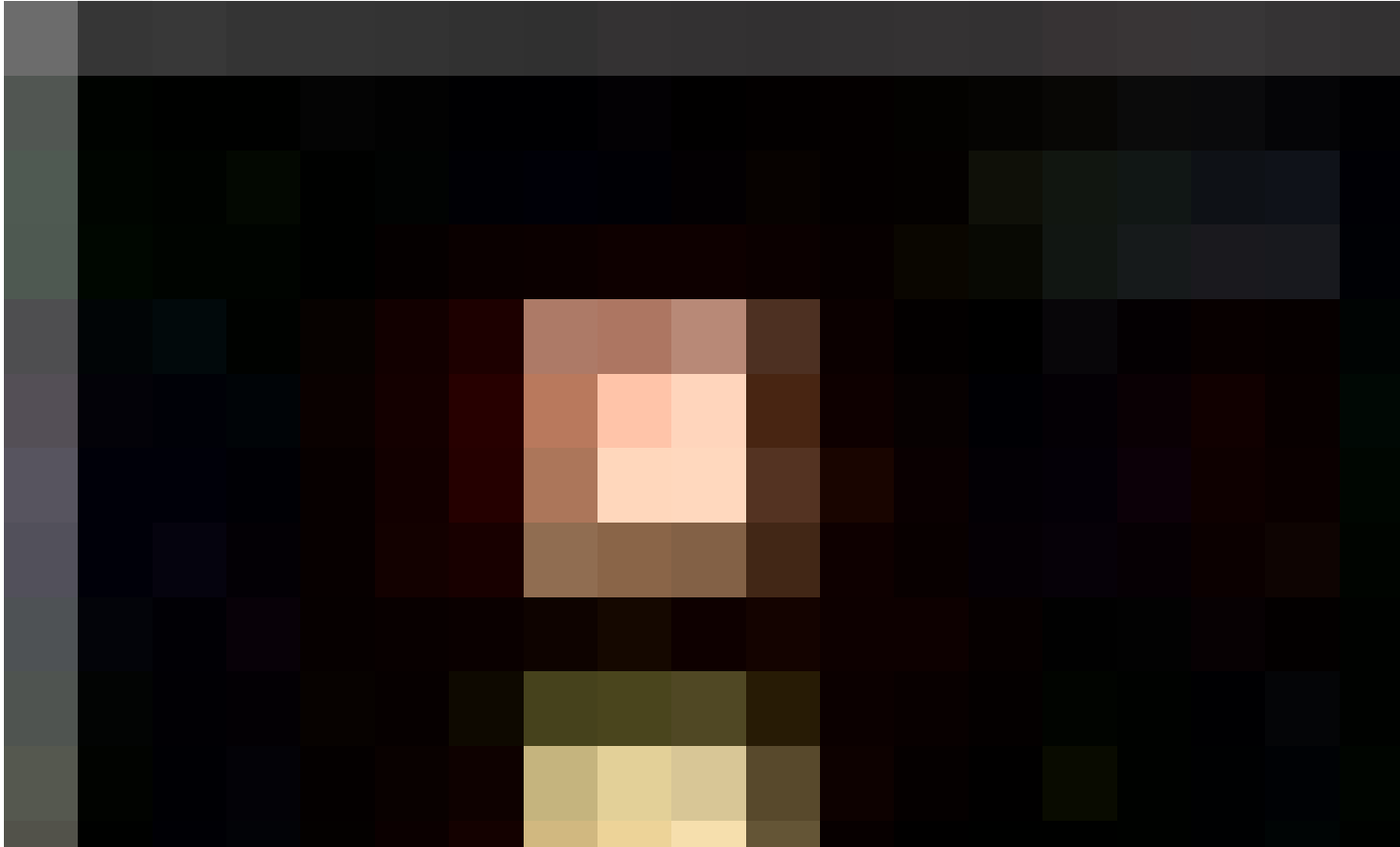
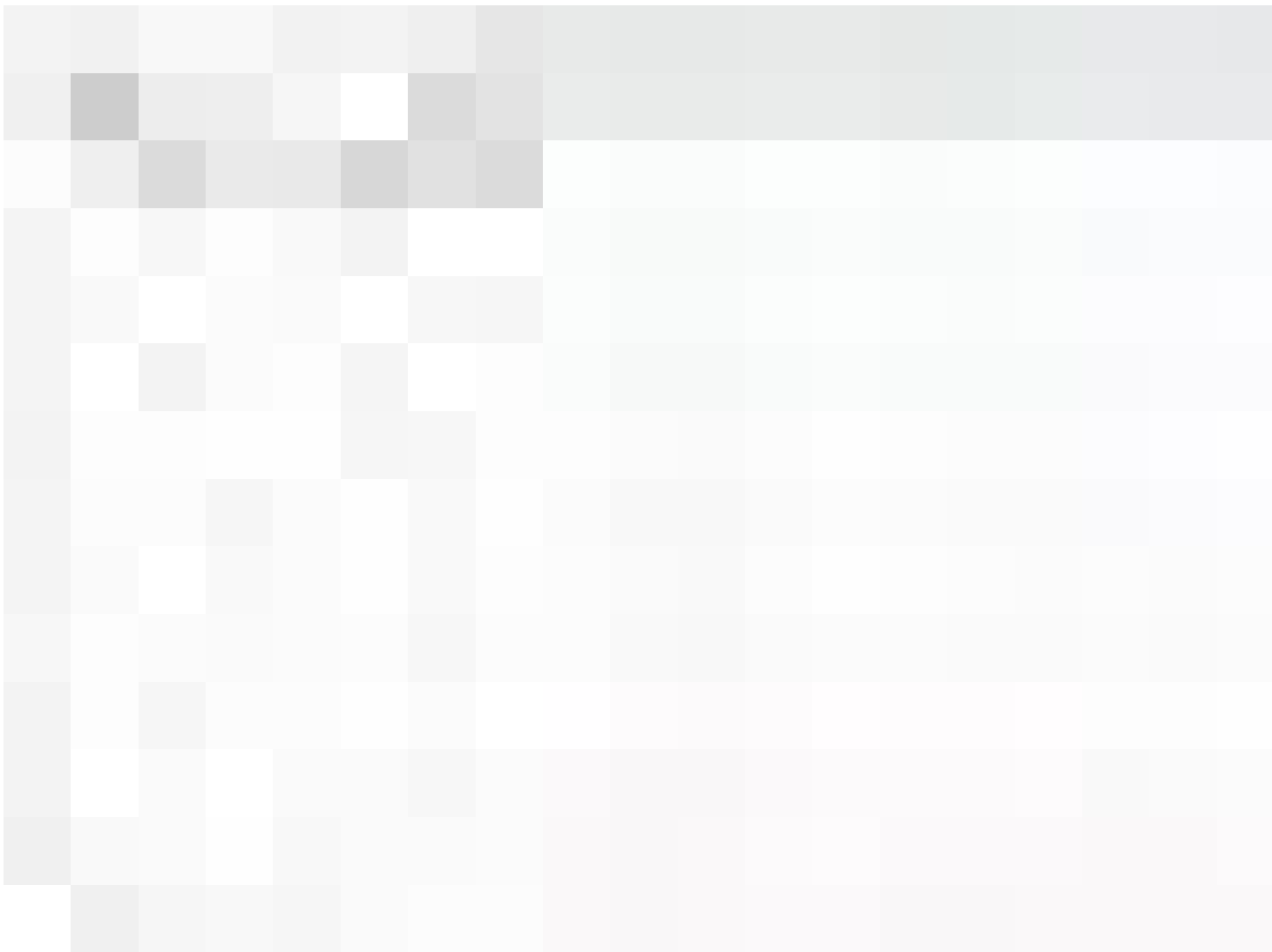
[Golden Layout](#) is a multi-window javascript layout framework manager for web apps, providing you with a flexible layout IDE. Unlike the old IDE, all panels can **be moved** wherever you want and **be maximized/minimized**; letting you adapt the interface to your likings.



3. Multiple Previewer

With this new feature, not only can we create however many **previewers** we want in the IDE, but also, thanks to the integrated terminal, we can support **transpiling projects previsualization** (see *Integrated Terminal* section).

With the concurrent multiple preview panels, you can now easily test your applications on various OS and device models at the same time in the IDE, reducing the testing time, as well the development time.



4. Monaco Editor

We have replaced the code editor from Ace editor to [Monaco](#) in the new IDE. The main reasons are:

- Auto-completion in many languages such as TypeScript, HTML, CSS or Onsen UI
- Basic syntax highlighting
- Better IntelliSense and validation

To sum up, the new IDE enables you to develop **transpilable projects** (e.g. VueJS, React, Angular2, etc) right in the Monaco Cloud IDE, without a need for Monaco CLI or Localkit. On top of that, the Integrated Terminal provides you with additional functionalities such as the **File Watch** and **Hot Reloading**. Furthermore, a new flexible layout, multiple previewers and efficient code editor will give you a productivity boost during your development.

We hope that the new Monaco Cloud IDE is making your life easier! Please let us know what you think. And don't forget to give us a clap if you like this post ;-)

Using Git with Integrated Terminal in Monaca Cloud IDE



[MPyK](#)

[Jun 21, 2018](#) · 6 min read





Hello, Git lover! The recent news about Microsoft acquiring GitHub might come as a surprise to some of us. Due to this acquisition, some people even consider to switch to a different Git services. Fortunately, at Monaca, we support both GitHub and GitSSH integration. We have just released the new look of Monaca Cloud IDE recently and it's packed with awesome features. Read our previous

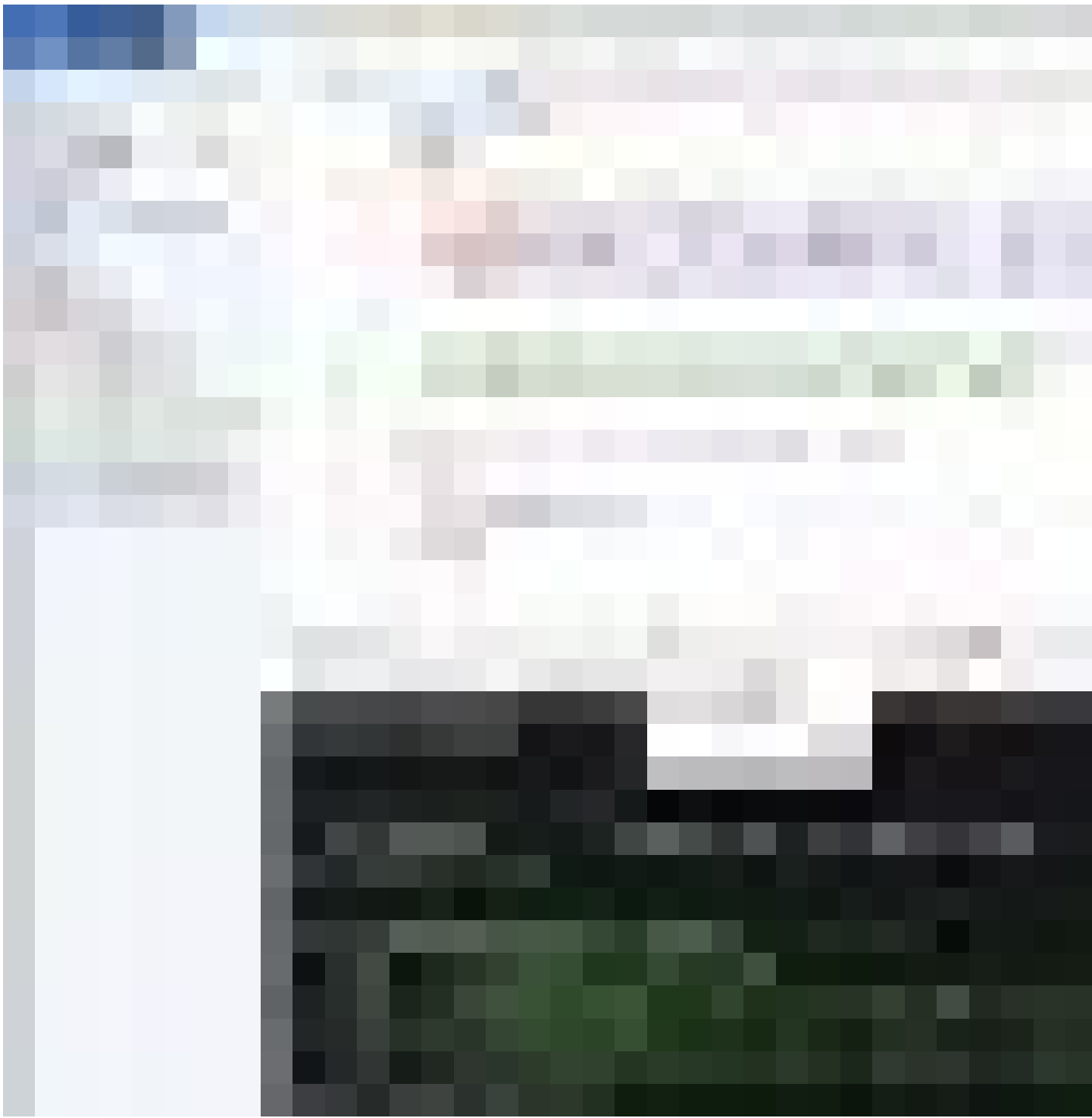
[blog](#) to get a grasp of it. And for those who haven't tried our awesome Monaca Cloud IDE yet, please give [it](#) a try. I'm sure you will love it.

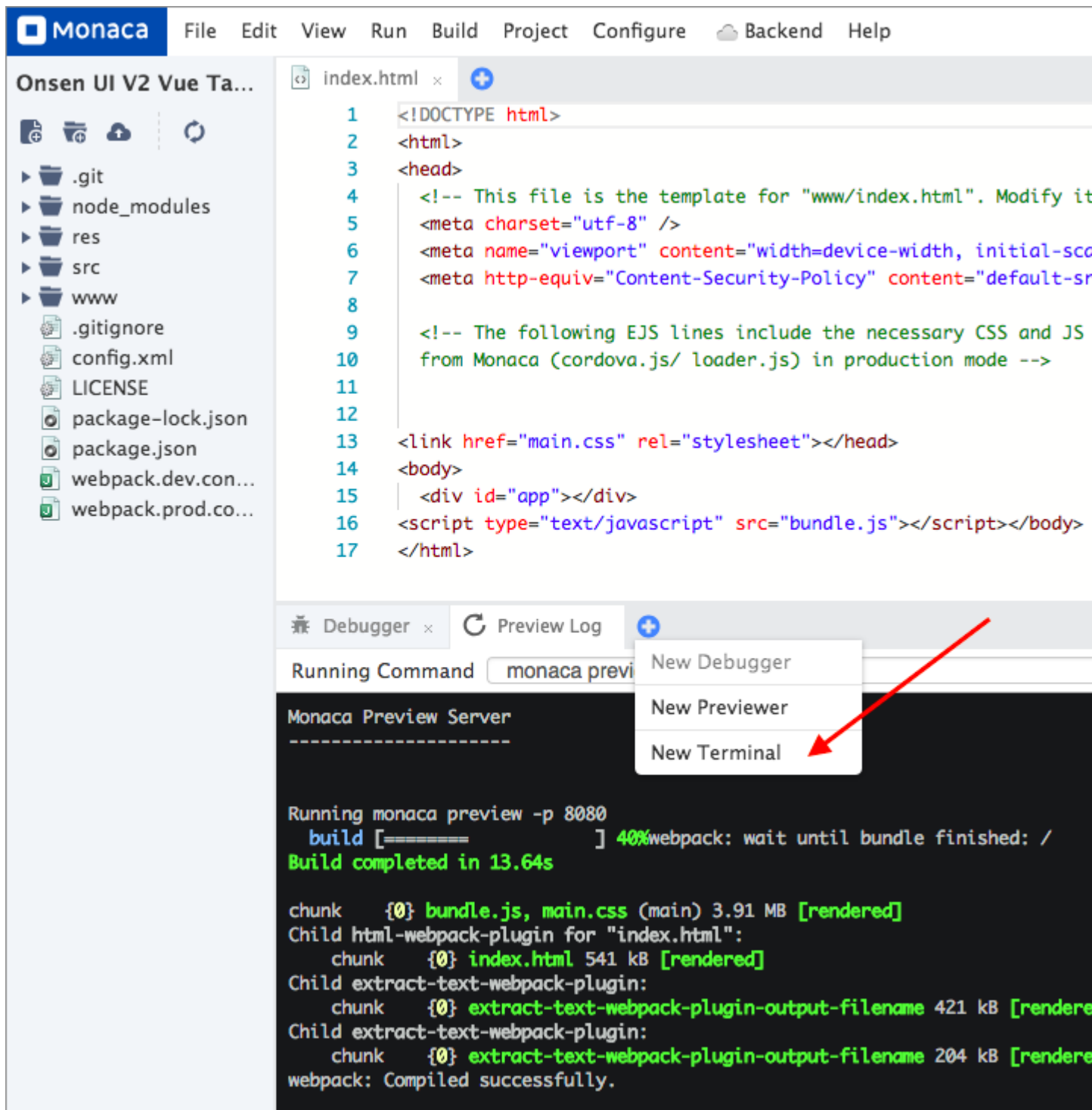
Prior to the launching of the terminal feature, Git integration is also available within Monaca Cloud IDE with user-friendly interface but with limited operations. There are two ways to integrate your Monaca account with a Git service — [Git SSH Integration](#) and [GitHub Integration](#). Thank to the terminal feature, we can now enjoy full support for Git commands and operations.

In this article, we will cover both Git SSH and GitHub integrations with the new Monaca Cloud IDE.

Launching Terminal

First and foremost, after you create a project in the new cloud IDE, let's open the terminal by clicking on the plus icon and choose **New Terminal**.





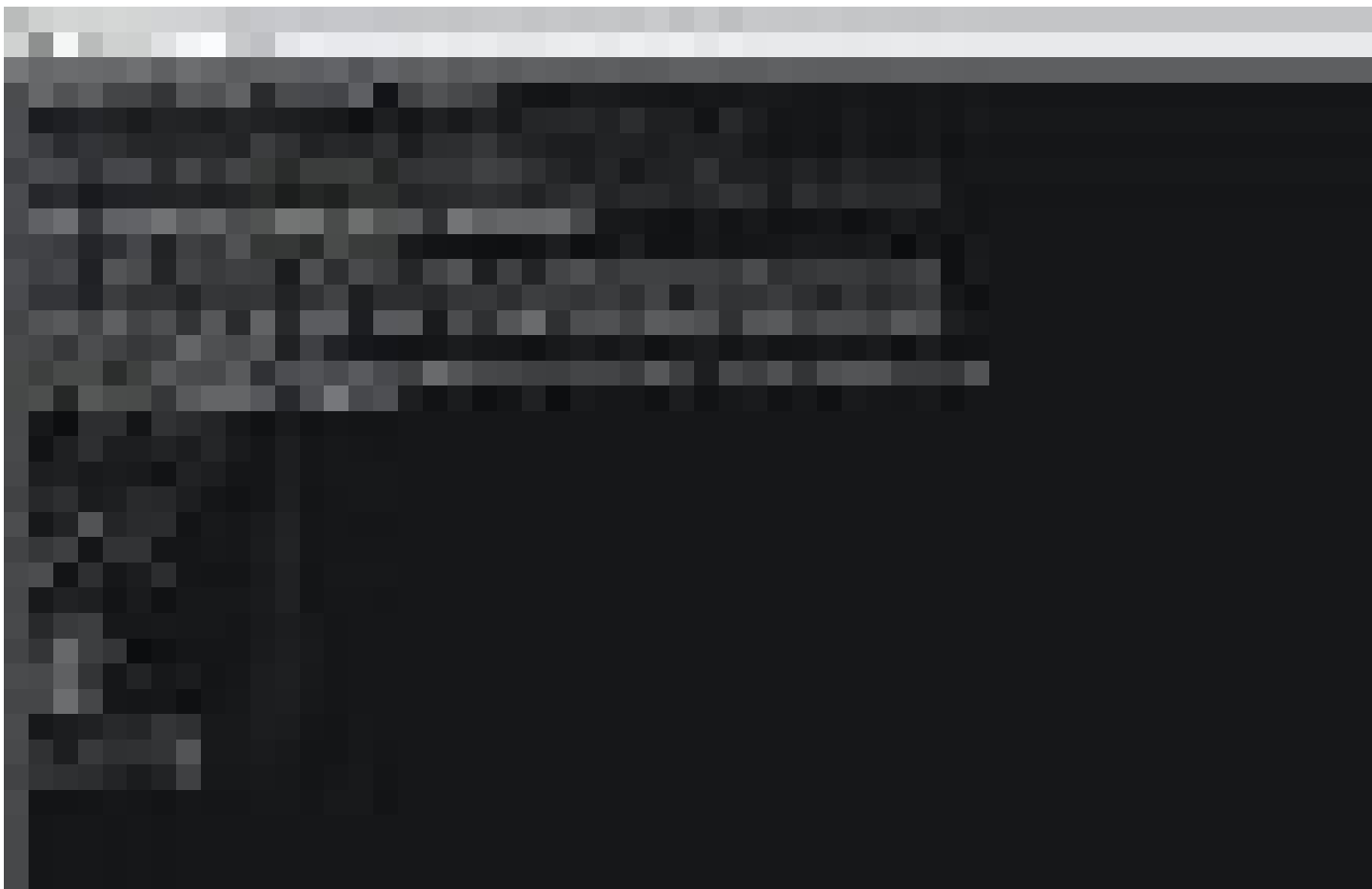
Here you can use most of the UNIX basic commands. For instance, if you type `cat /etc/debian_version` in the terminal console, the output would be `9.4` (at the time of writing). As you may also notice in the picture, `/project` is the default directory after you first open the terminal. This is the root directory of each project. Next, we will see how we can integrate Git services with Monaca.

Git SSH Integration

Step 1: Generating SSH Key

You can (re)generate your private and public key with `ssh-keygen` command. You can do it by running `ssh-keygen -t rsa -C 'youremail@address.com'` in the terminal console.

Your SSH Keys will be stored at `/home/terminaluser/.ssh/` or `~/.ssh/` directory, if you accept the default.



```

Terminal x +
Welcome to Monaca Terminal Console!
/project $ ssh-keygen -t rsa -C 'youremail@address.com'
Generating public/private rsa key pair.
Enter file in which to save the key (/home/terminaluser/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/terminaluser/.ssh/id_rsa.
Your public key has been saved in /home/terminaluser/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:1CHK3MIy0IIHpvpAr0a1Lz dq0j8h42Ez14PjarFnKmc youremail@address.com
The key's randomart image is:
+---[RSA 2048]-----+
|.+. . . |
|+ o..+ o o . |
|.o oo * o . |
|o o .o o |
|o..o. S |
|.&.=. |
|+o%.o+ |
|o*E=+ . |
|==Oo. |
+----[SHA256]-----+
/project $ █

```

Don't forget to replace `youremail@address.com` with your "real" email address.

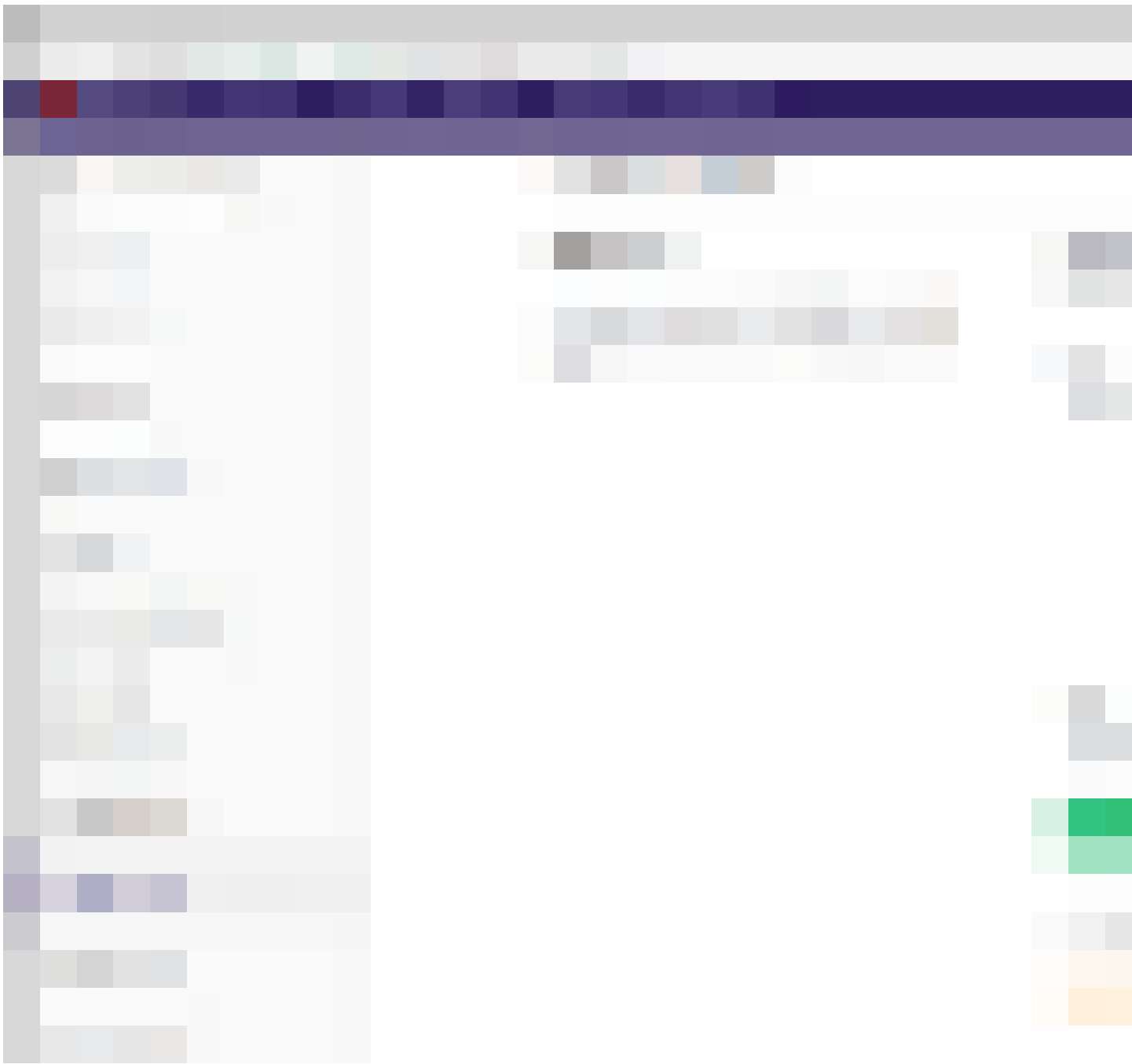
The output might be different in your environment.

Step 2: Adding the SSH Key to Git Service

The SSH key needs to be added to your Git service account for authentication. If you keep the default path, your public key would be available at `~/ .ssh/id_rsa.pub`. Copy and paste this key to the SSH key management page of your Git service account.

In this example, we will show how to add the [key with GitLab](#). For other Git services, please refer to their documentation.

After login to your Gitlab account, navigate to **User Settings** → **SSH Keys**, paste in your public key, enter title then click **Add key** button.



← → ↻ 🏠 Secure | <https://gitlab.com/profile/keys>

GitLab Projects ▾ Groups Activity Milestones Snippets

User Settings

- Profile
- Account
- Billing
- Applications
- Chat
- Access Tokens
- Emails
- Password
- Notifications
- SSH Keys**
- GPG Keys
- Preferences

User Settings > SSH Keys

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

Before you can add an SSH key, you must first add a public key to your profile.

Key

Paste your public key here

Title

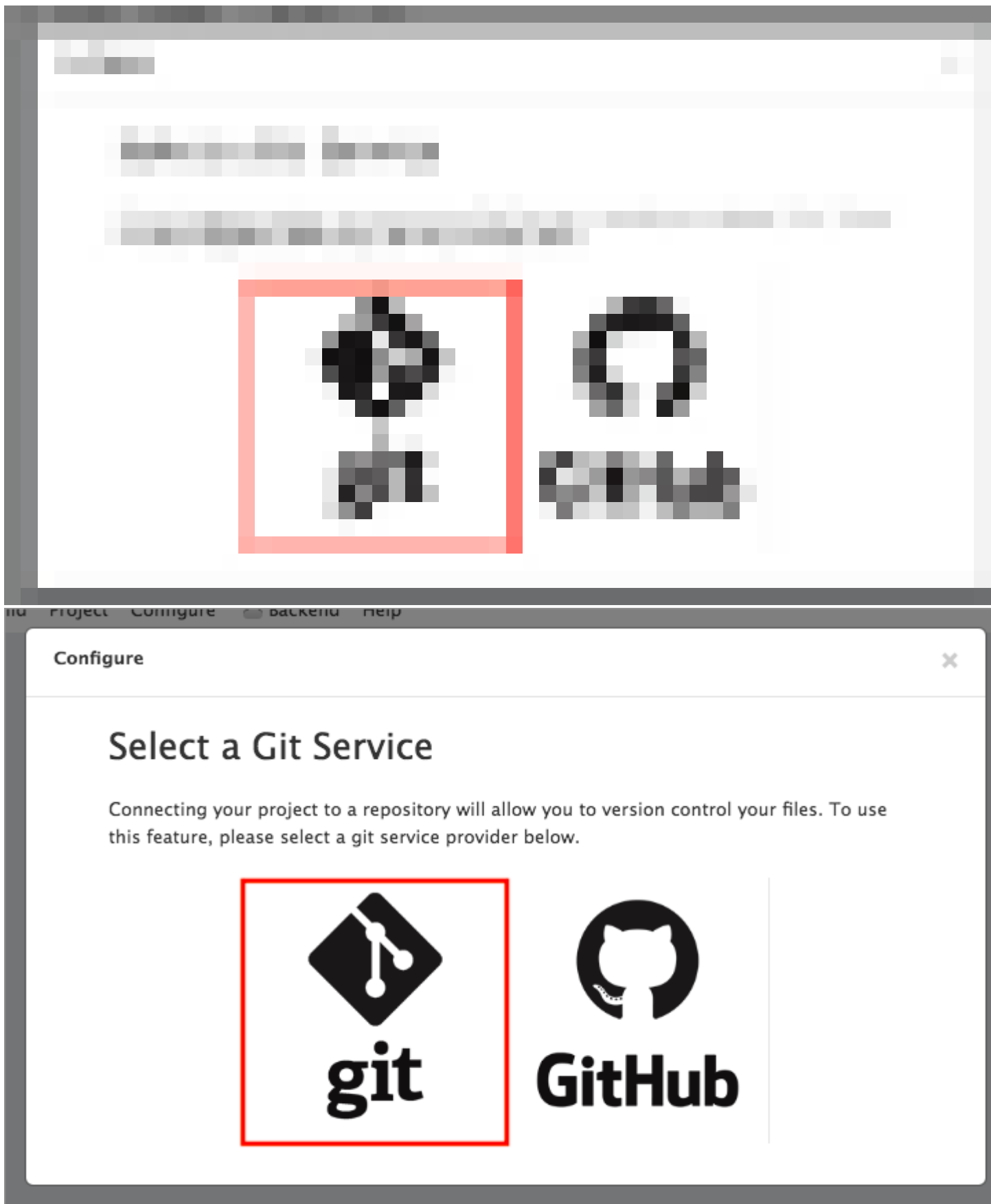
your public key title

Add key

Your SSH keys

Step 3: Connecting your project with the Git repository

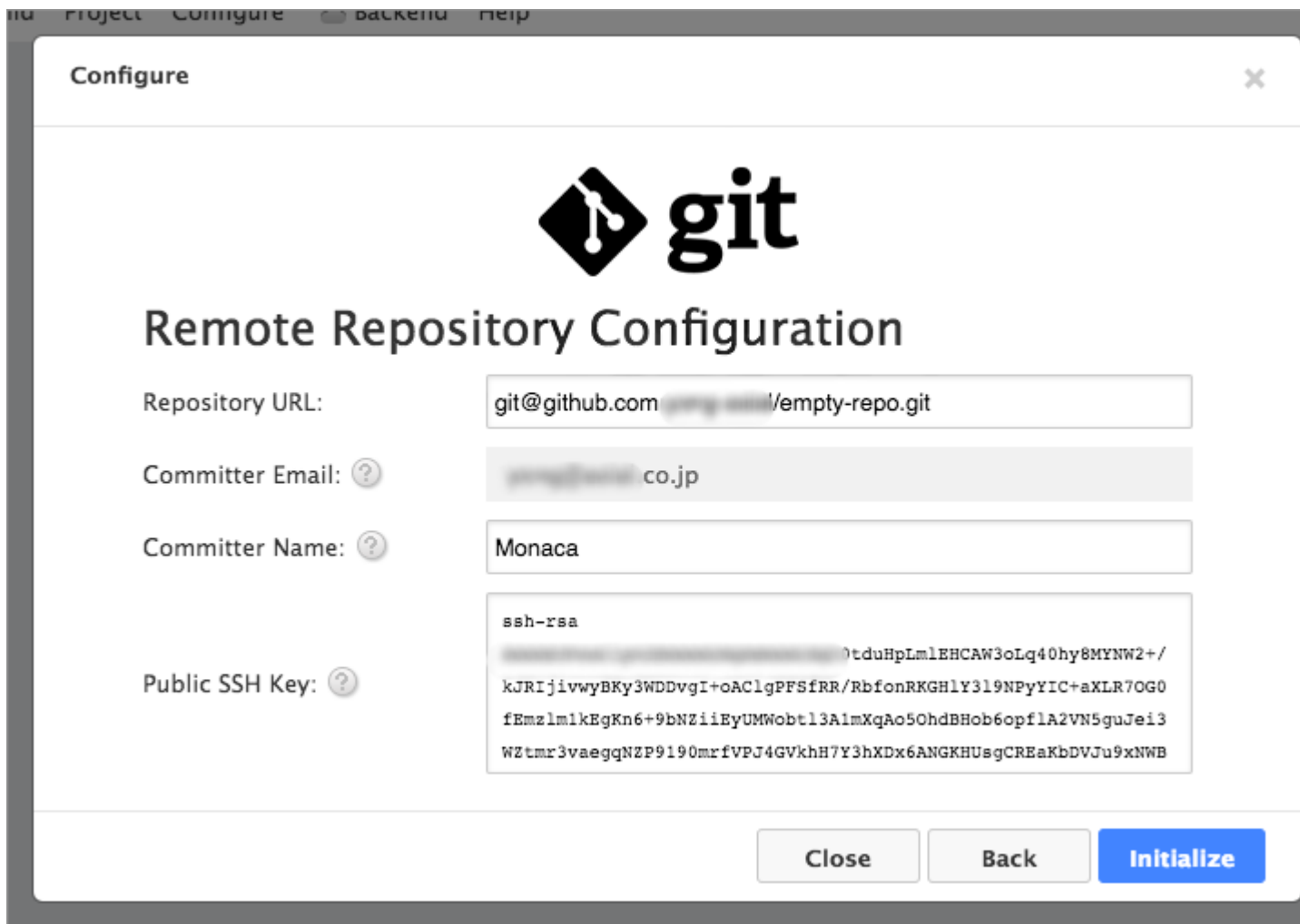
1. Open a project in Monaca Cloud IDE.
2. From the top menu, go to **Project** → **VCS Configure**. Then, choose Git option as shown below




3. Input your SSH Git repository URL and committer name.

Note: The repository must be empty and can not be changed after configured.





Configure

 **git**

Remote Repository Configuration

Repository URL:

Committer Email:

Committer Name:

Public SSH Key:

4. Next, click on **Initialize**. Your project is then being uploaded to the new repository `empty-repo.git` in this example. By default, Monaca will create the first commit as `Project's Initial Commit` in your master branch. You may check it by issuing `git log` in your terminal console.

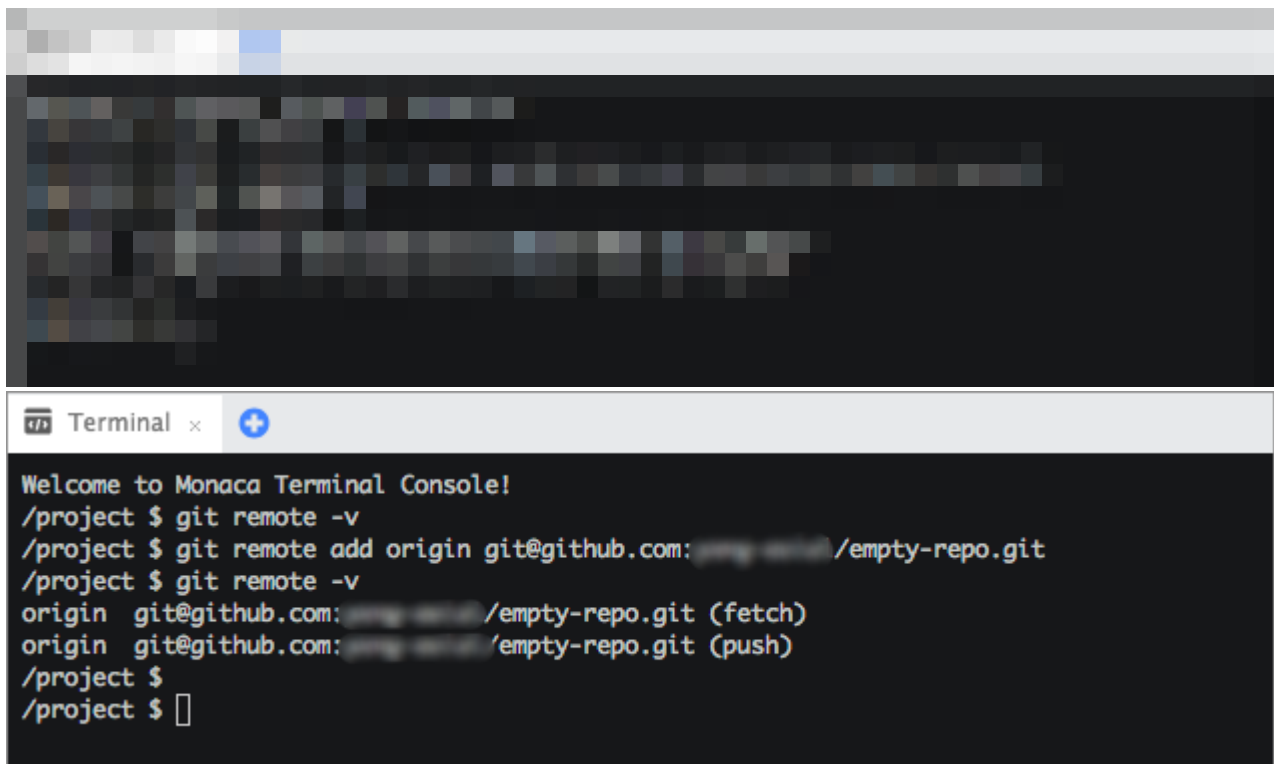
5. Phew... You're almost there. There is only one step left which is to config remote repository from the terminal. It's done via `git remote` command. Therefore, go ahead and add the remote origin by typing:

```
git remote add origin git@gitlab.com:UserName/empty-repo.git
```

Otherwise, if you already have a remote origin, you can overwrite it by typing:

```
git remote set-url origin git@gitlab.com:UserName/empty-repo.git
```

You can check whether you have registered the right remote repository by typing `git remote -v`.



```
Welcome to Monaca Terminal Console!
/project $ git remote -v
/project $ git remote add origin git@github.com:username/empty-repo.git
/project $ git remote -v
origin  git@github.com:username/empty-repo.git (fetch)
origin  git@github.com:username/empty-repo.git (push)
/project $
/project $
```

Now, you're ready!

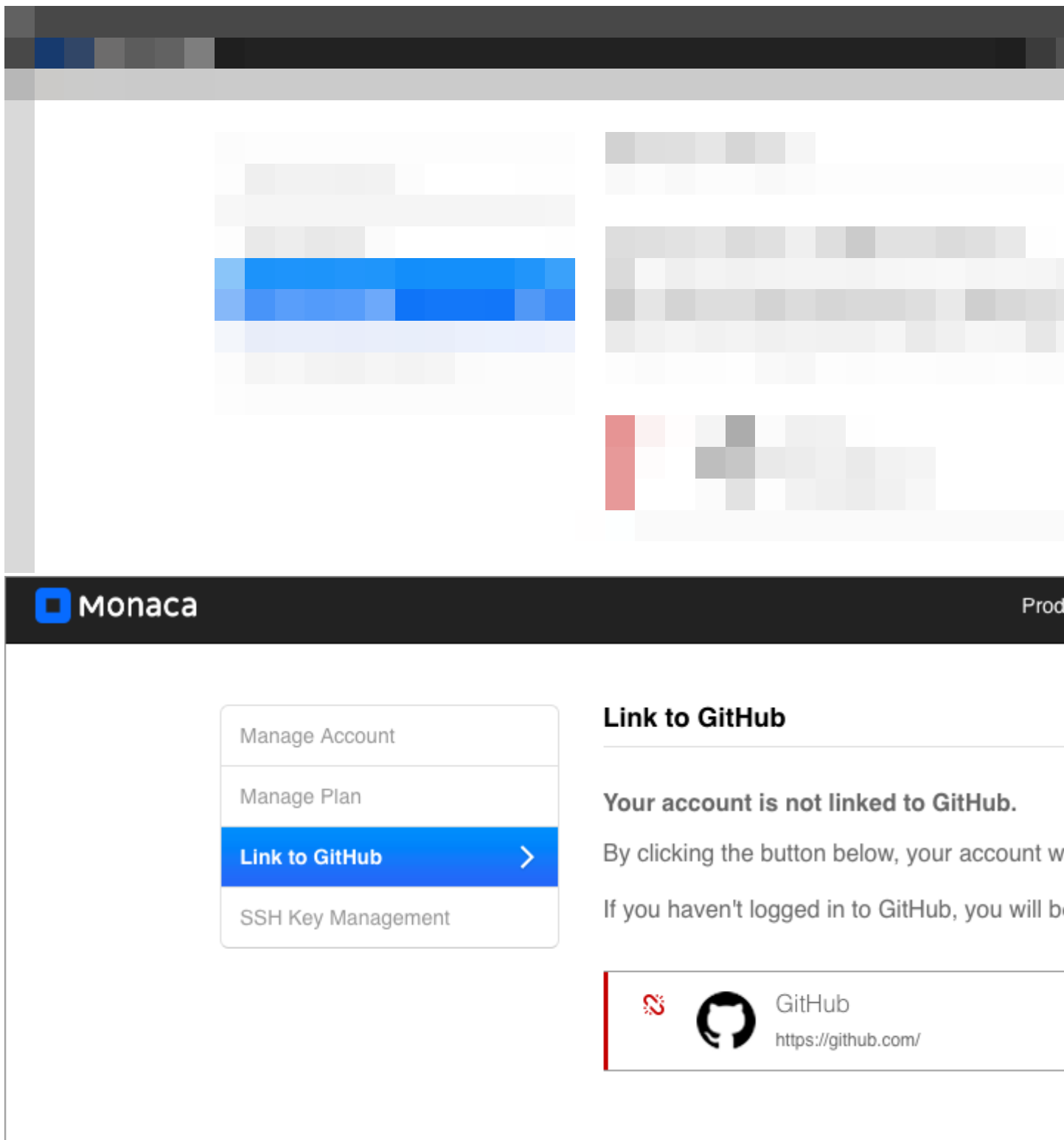
Try modifying the project and push your *second* commit!

GitHub Integration

This section is dedicated to those (including me) who are loyal to the Github regardless of the news. Come to think of it, it's not that bad, right?

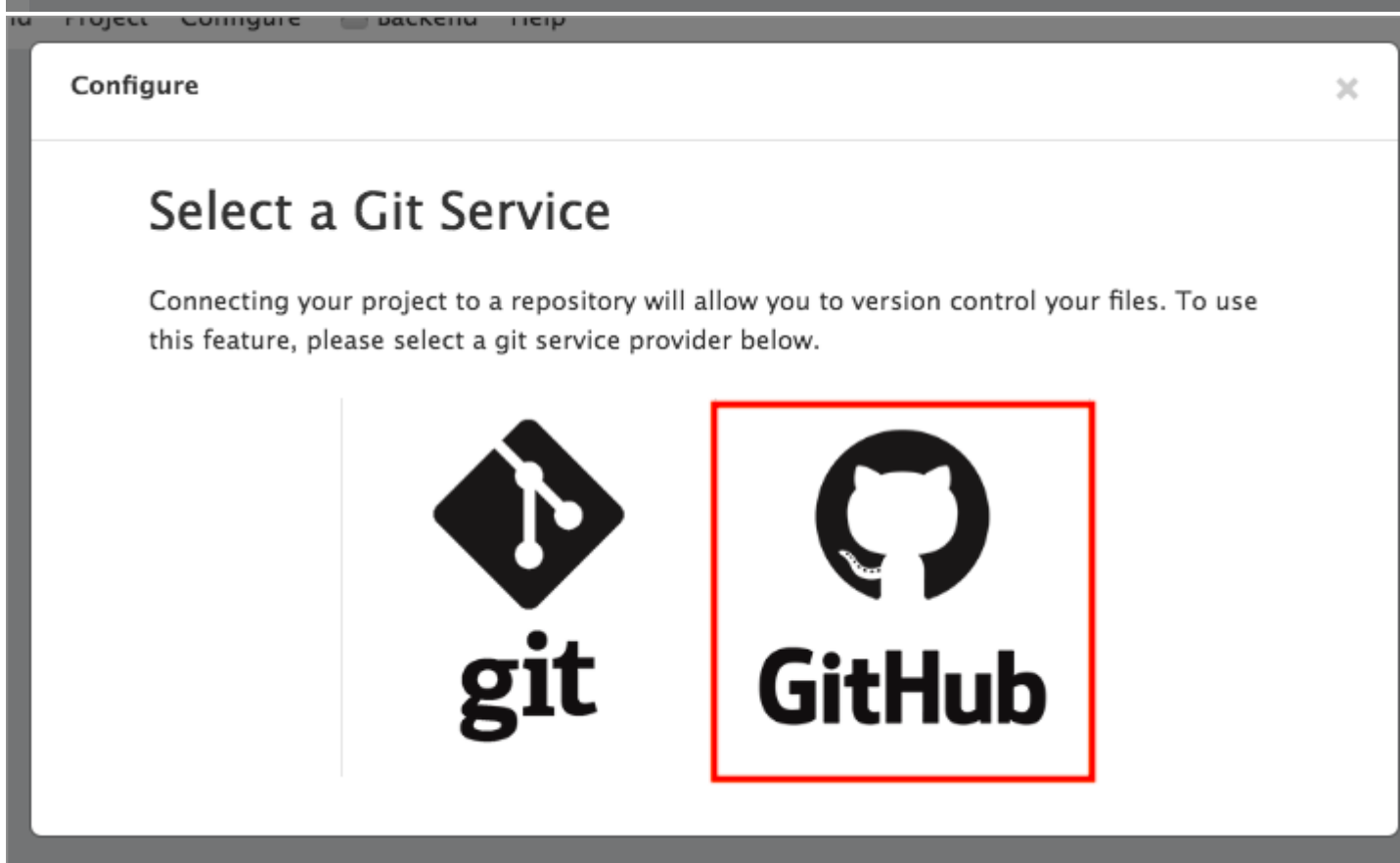
Step 1: Linking GitHub Account to Monaca

Navigate to [Link to GitHub](#) page and click on **Link** button. Then, follow the linking wizard. You may want to refer to this [guideline](#).



Step 2: Connecting the project with your GitHub Account

1. Open a project in Monaca Cloud IDE.
2. From the top menu, go to **Project** → **VCS Configure**. Then, choose GitHub option as shown below



3. Select your GitHub repository and input committer name.

Note: The repository must be empty and can not be changed after configured.



Likewise, you then need to add remote repository. In this example, I will add remote repository with the https option.

```
git remote add origin https://github.com/UserName/another-  
empty-repo.git
```

Since you are using https option, so you will need to enter username and password when you push the code.

2-factor Authentication

In addition, if you turn on 2-factor authentication, you will need to generate personal token first as follows:

1. Go to this [link](#) to generate your personal token.
2. Click on **Generate new token** , input **Description** and **authorizingScope** , then click **Generate Token** .



Settings / Developer settings

OAuth Apps

GitHub Apps

Personal access tokens

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Token description

Testing_GitHub_Integration

What's this token for?

Select scopes

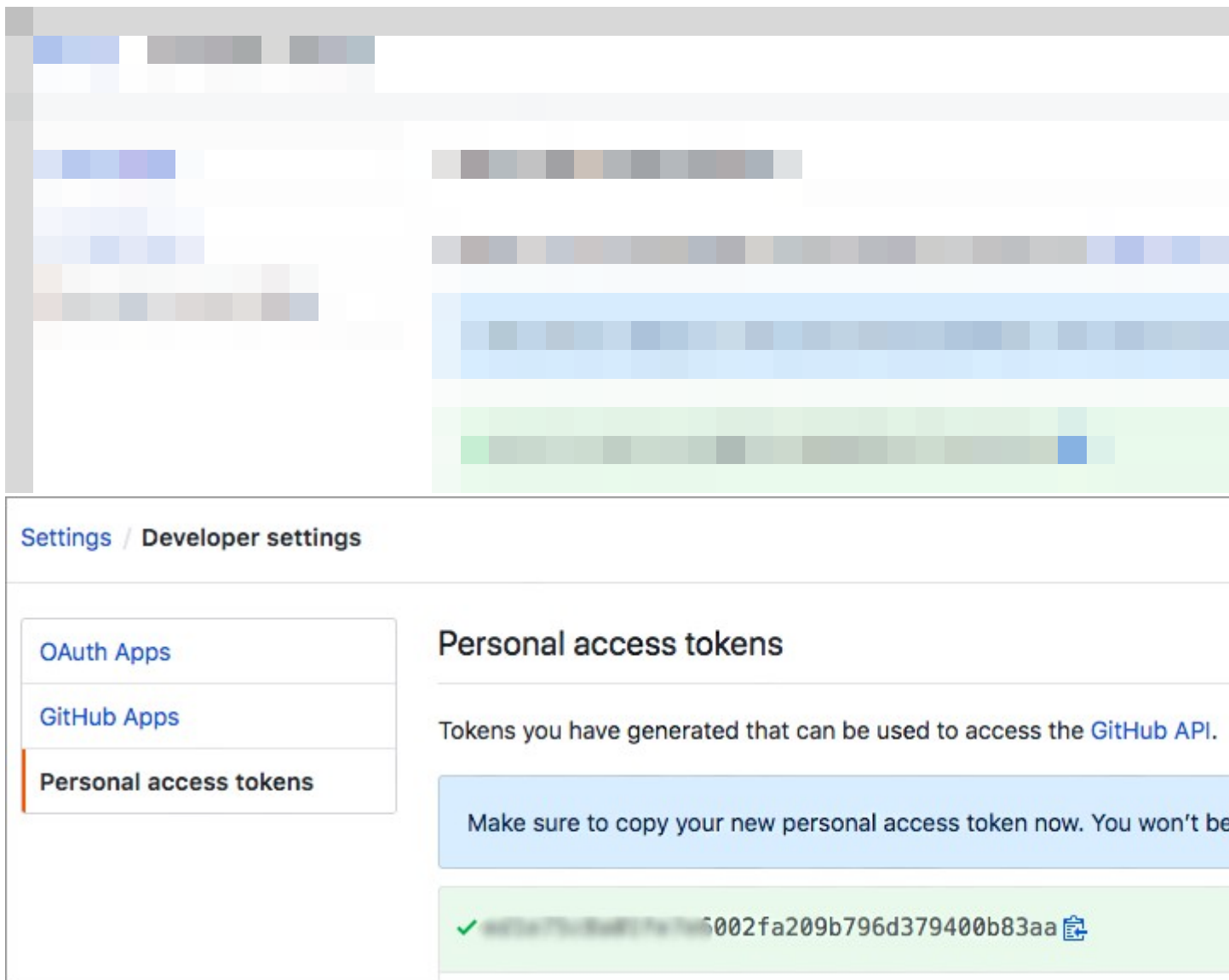
Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> admin:org	Full control of orgs and teams
<input checked="" type="checkbox"/> write:org	Read and write org and team membership
<input checked="" type="checkbox"/> read:org	Read org and team membership
<input checked="" type="checkbox"/> admin:public_key	Full control of user public keys
<input checked="" type="checkbox"/> write:public_key	Write user public keys
<input checked="" type="checkbox"/> read:public_key	Read user public keys
<input checked="" type="checkbox"/> admin:repo_hook	Full control of repository hooks
<input checked="" type="checkbox"/> write:repo_hook	Write repository hooks
<input checked="" type="checkbox"/> read:repo_hook	Read repository hooks
<input checked="" type="checkbox"/> admin:org_hook	Full control of organization hooks
<input checked="" type="checkbox"/> gist	Create gists
<input checked="" type="checkbox"/> notifications	Access notifications
<input checked="" type="checkbox"/> user	Update all user data
<input checked="" type="checkbox"/> read:user	Read all user profile data
<input checked="" type="checkbox"/> user:email	Access user email addresses (read-only)
<input checked="" type="checkbox"/> user:follow	Follow and unfollow users
<input checked="" type="checkbox"/> delete_repo	Delete repositories
<input checked="" type="checkbox"/> write:discussion	Read and write team discussions
<input checked="" type="checkbox"/> read:discussion	Read team discussions
<input checked="" type="checkbox"/> admin:gpg_key	Full control of user gpg keys (Developer Preview)
<input checked="" type="checkbox"/> write:gpg_key	Write user gpg keys
<input checked="" type="checkbox"/> read:gpg_key	Read user gpg keys

Generate token

Cancel

3. After clicking Generate Token button, GitHub will tell you the token . Make sure you copy it to somewhere safe.



Now, it's ready!

Terminal x +

```
Welcome to Monaca Terminal Console!
/project $ git remote -v
/project $ git remote add origin https://github.com/monaca/another-empty-repo.git
/project $ git remote -v
origin https://github.com/monaca/another-empty-repo.git (fetch)
origin https://github.com/monaca/another-empty-repo.git (push)
/project $ touch awesome_codes
/project $ git add .
/project $ git commit -m "awesome messages"
[master 200ddf4] awesome messages
3 files changed, 17 insertions(+)
create mode 100644 .monaca/.snapshot/.history.db
create mode 100644 .monaca/.snapshot/www/index.html.1529289363.96
create mode 100644 awesome_codes
/project $ git push origin master
Username for 'https://github.com': monaca
Password for 'https://monaca@github.com':
Counting objects: 7, done.
Delta compression using up to 24 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 867 bytes | 0 bytes/s, done.
Total 7 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/monaca/another-empty-repo.git
22e3aad..200ddf4 master -> master
/project $
```

Note that, you need to input your GitHub's username and password (or personal token if you turn on 2-factor authentication) when you push the code.

Conclusion

As a result of [terminal integration in Monaca Cloud IDE](#), we can do a lot now in the IDE. The ability to use all Git commands is just one of the awesome features we have been working on. Although this article does not touch on much detail of Git operations, please feel free to play around with it. Likewise, in order to make your application development even more efficient, try using the terminal functions by all means. If you have any questions or feedbacks, feel free to contact us anytime!

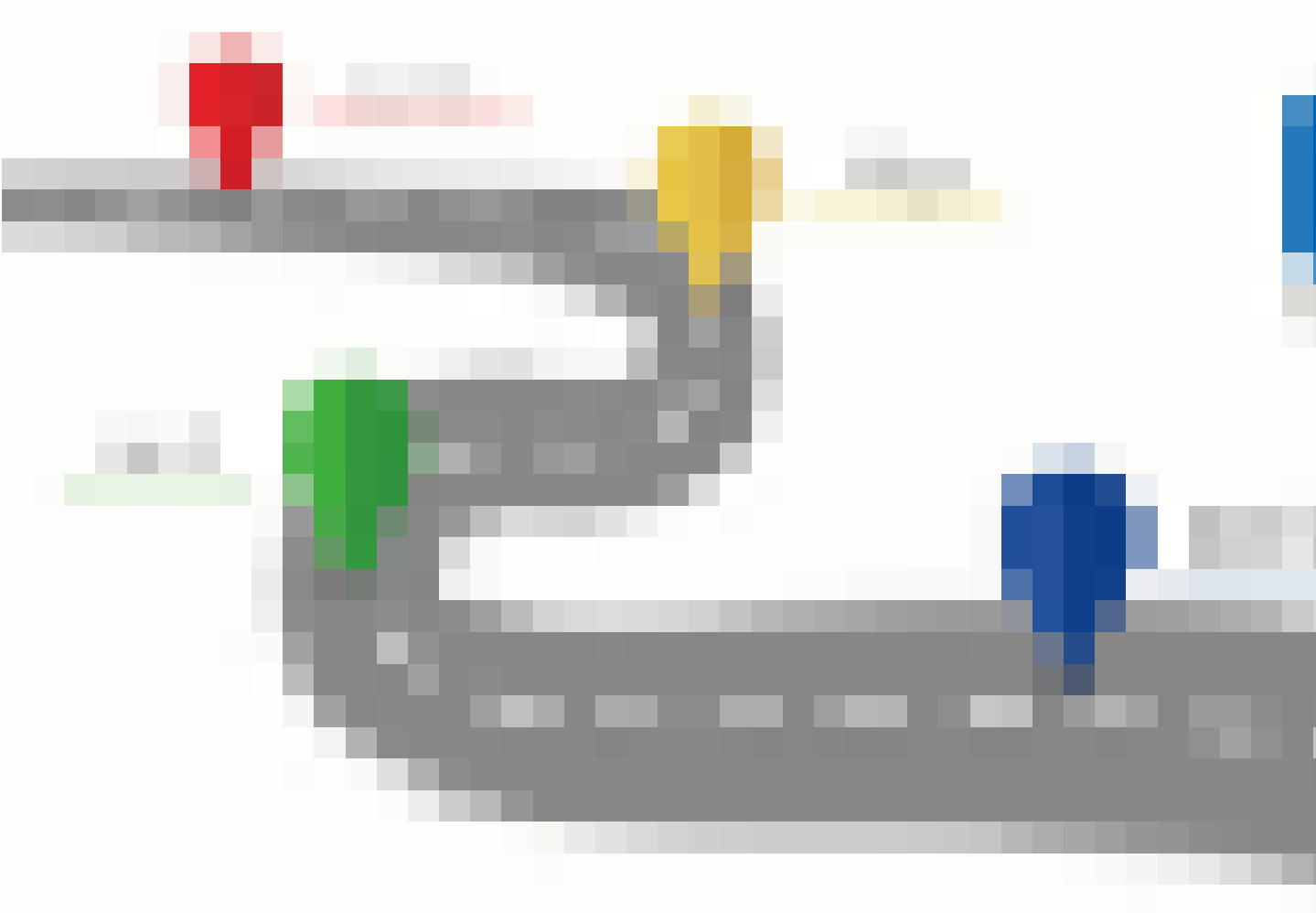
Thanks for reading!

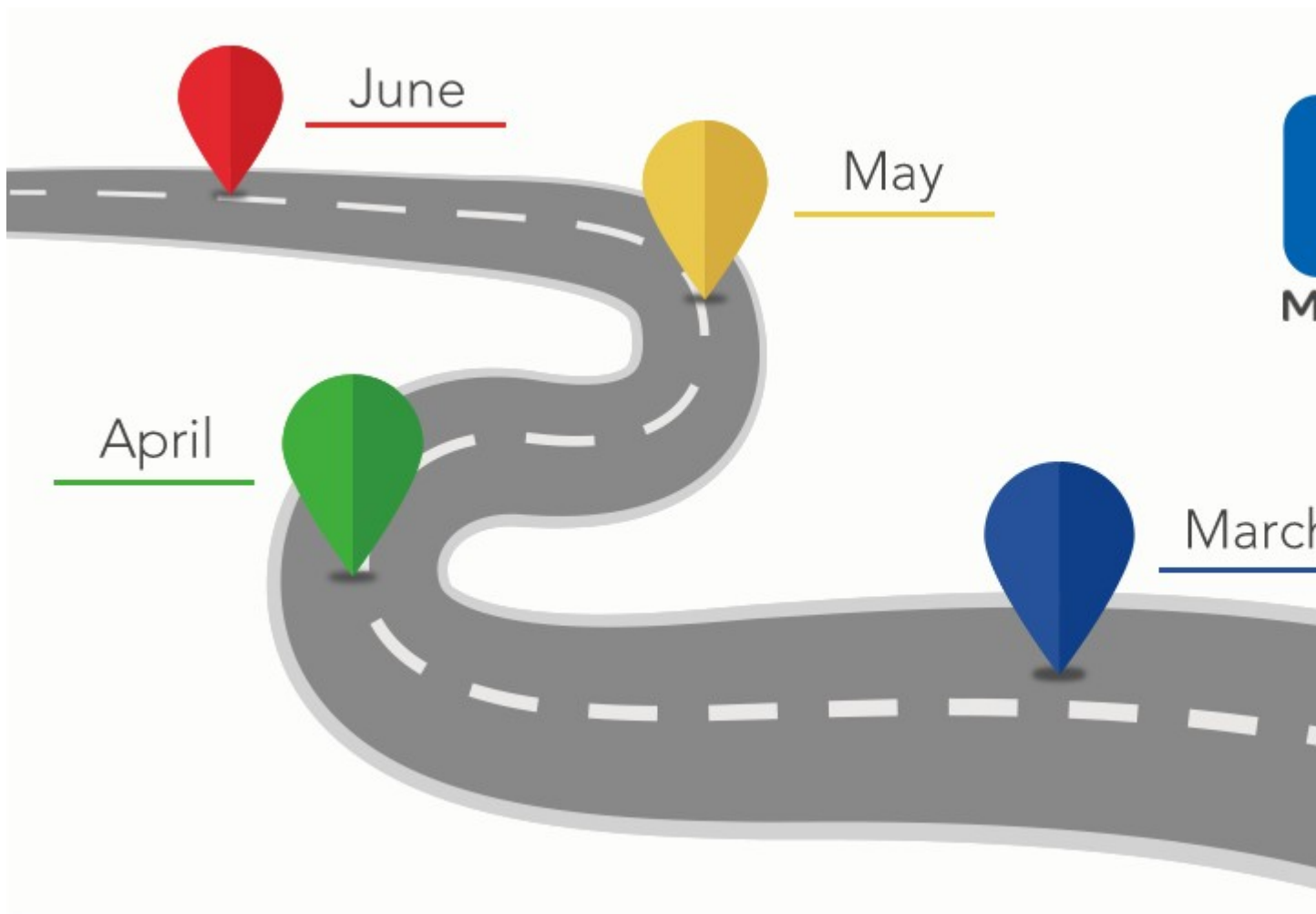
Monaca & Onsen UI: Next 1–2 Months



[Khemry Khourn](#)

[Mar 19, 2018](#) · 5 min read





We are starting a new series about short-term roadmaps for Monaca & Onsen UI to give you a more concrete insight of what we have been doing and what to expect in the upcoming months.

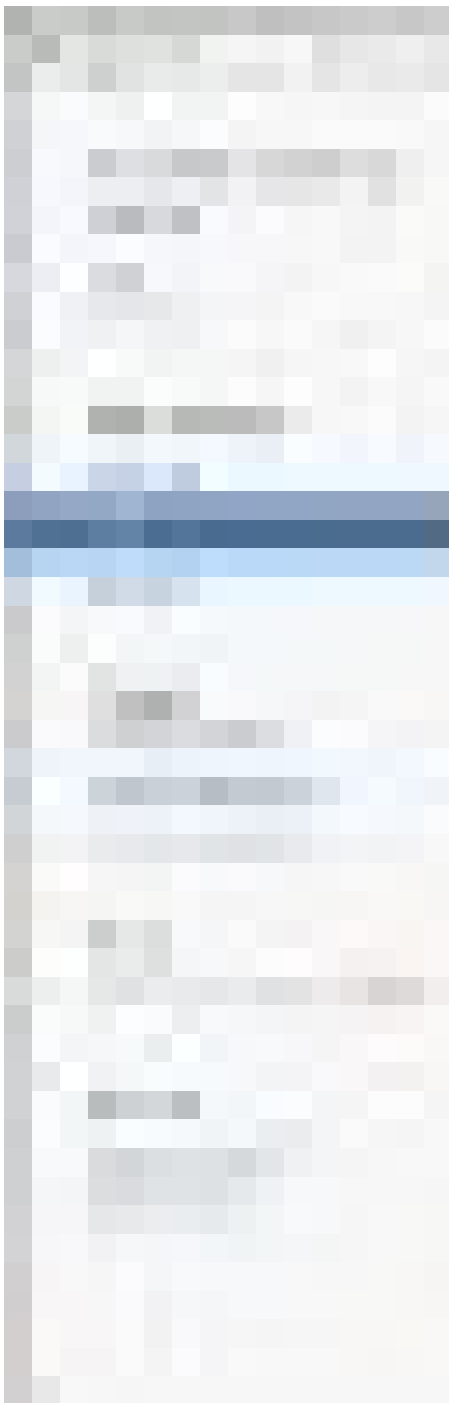
Let's start by recapping our activities in February, followed by the expected releases in the next 30–60 days.

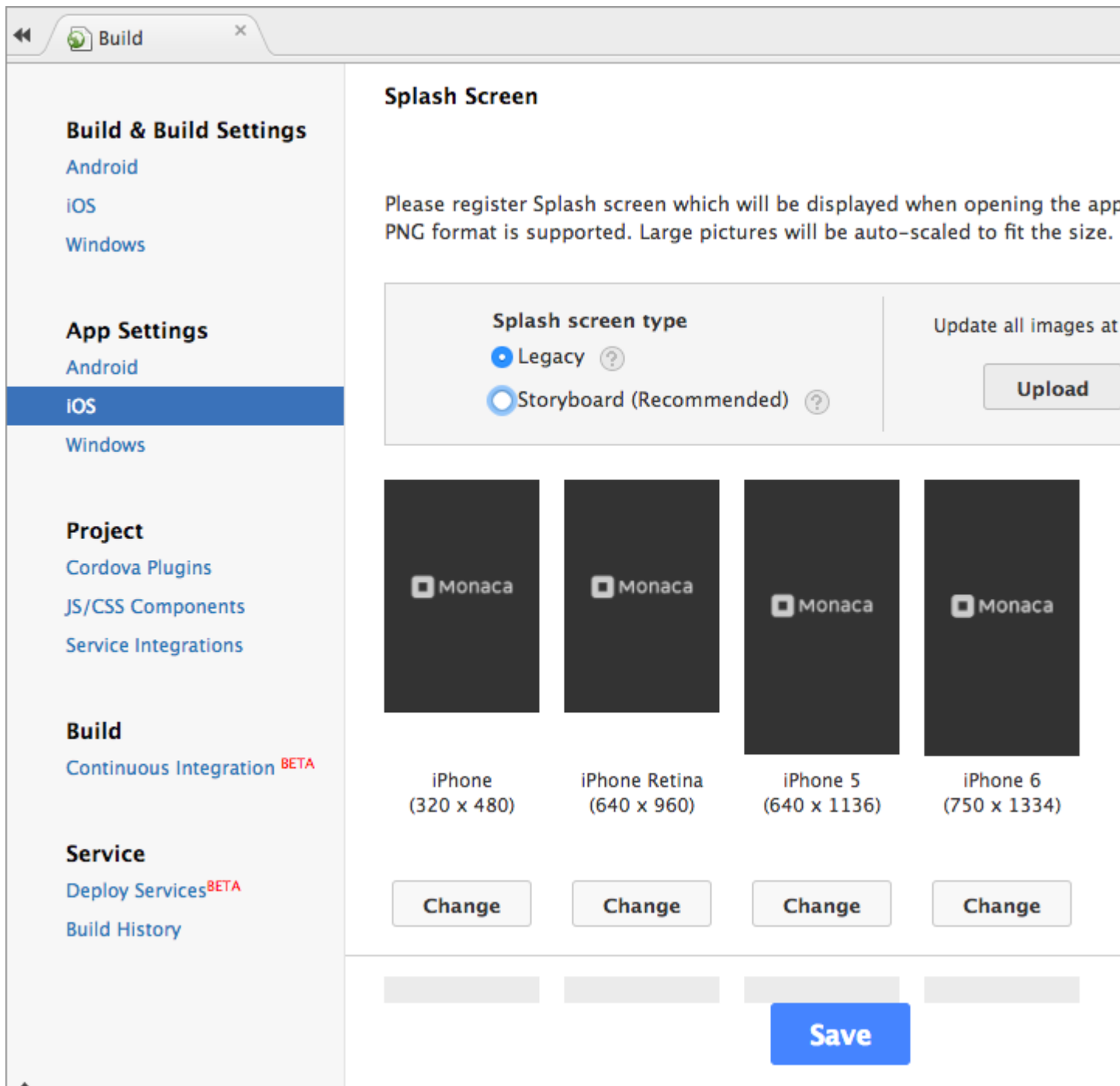
What's up in February

Monaca — New iOS Supported Features in Cloud IDE

With the recent release of iPhone X and iOS 11, various requirements have been introduced to the build and deploy process. One example is the App Store icon (1024 x 1024) which is required when deploying your app to the app market.

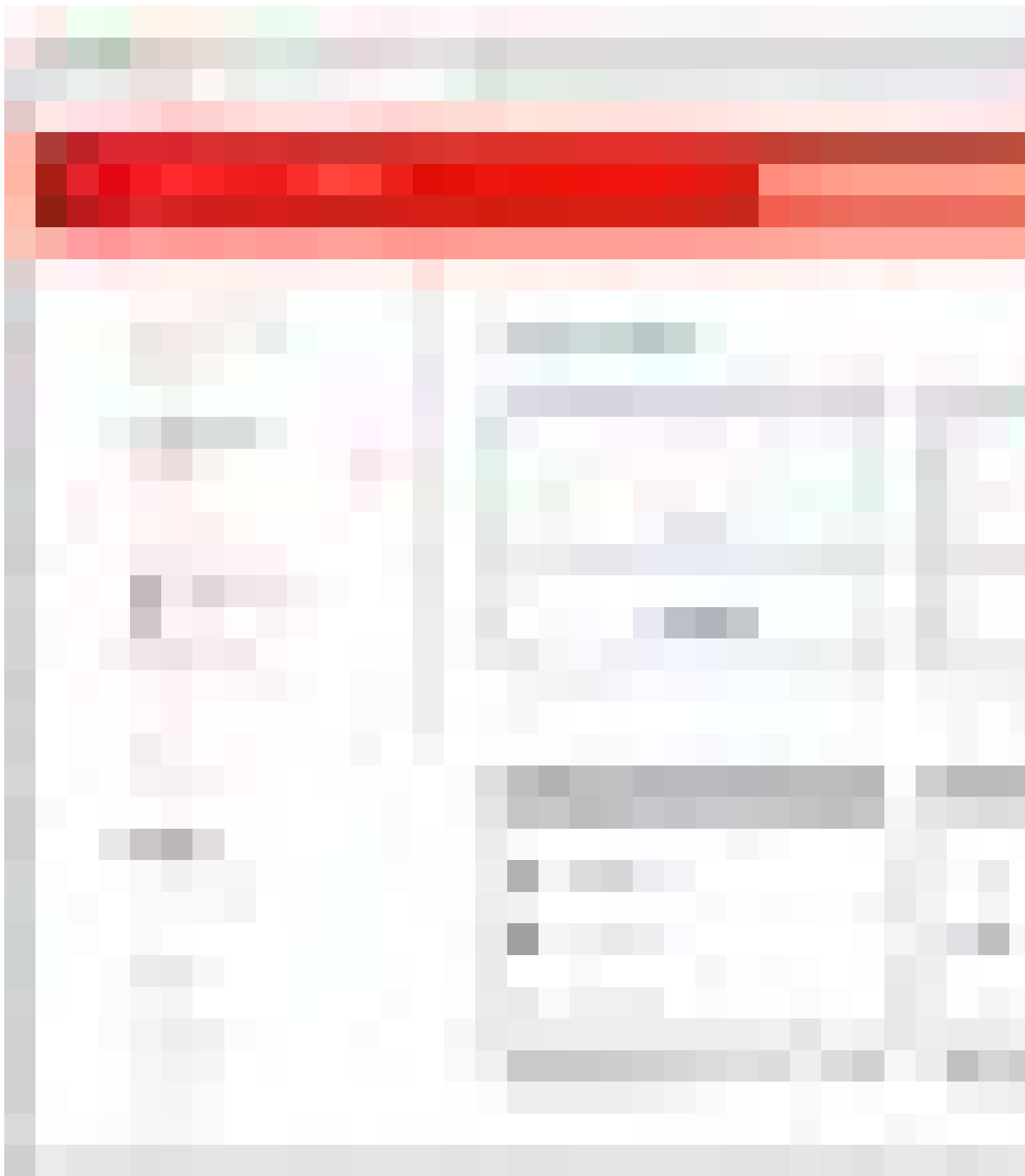
These new requirements have been integrated into the Monaca platform so that you can continue to configure, build, and deploy your apps with ease. Additionally, the long-awaited support for the Storyboard splash screen has been integrated!





Onsen UI — Improvement on UI/UX for Components & Support for Japanese Users

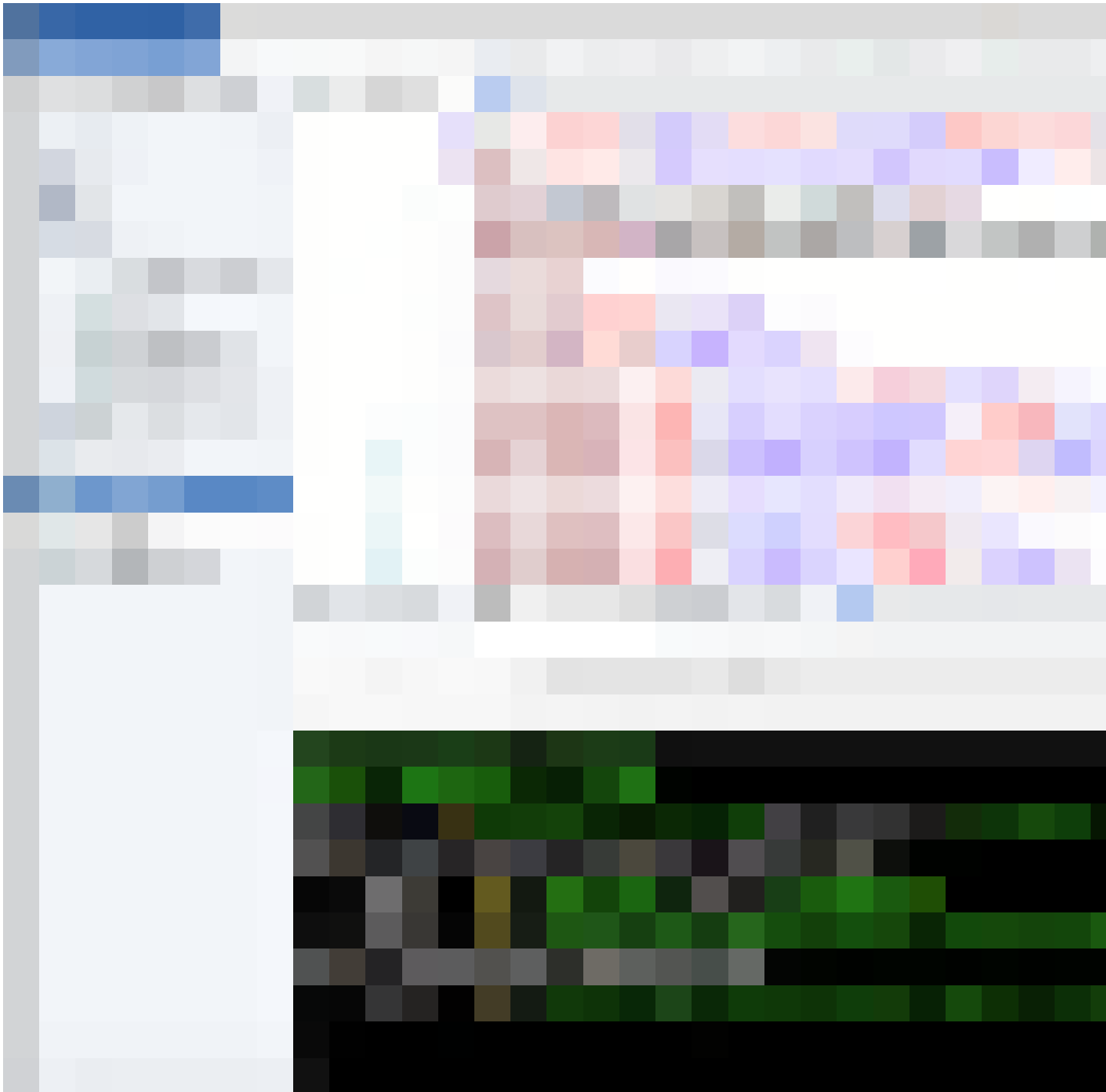
- The Onsen UI team continues to strive to bring the best user experience with every web component we provide. Even though this month there are no new major components, but various improvements were made to enrich the user interface and experience of our existing Navigator, Tabbar, Splitter, Switch, and other components.
- [Theme roller](#) for Japanese users was out! It's been a while since we released Onsen UI theme roller (English version); however, the Japanese version has just been released. Number of Japanese users for Onsen UI has been steadily growing. Therefore, this would be very helpful to them.



What's Next

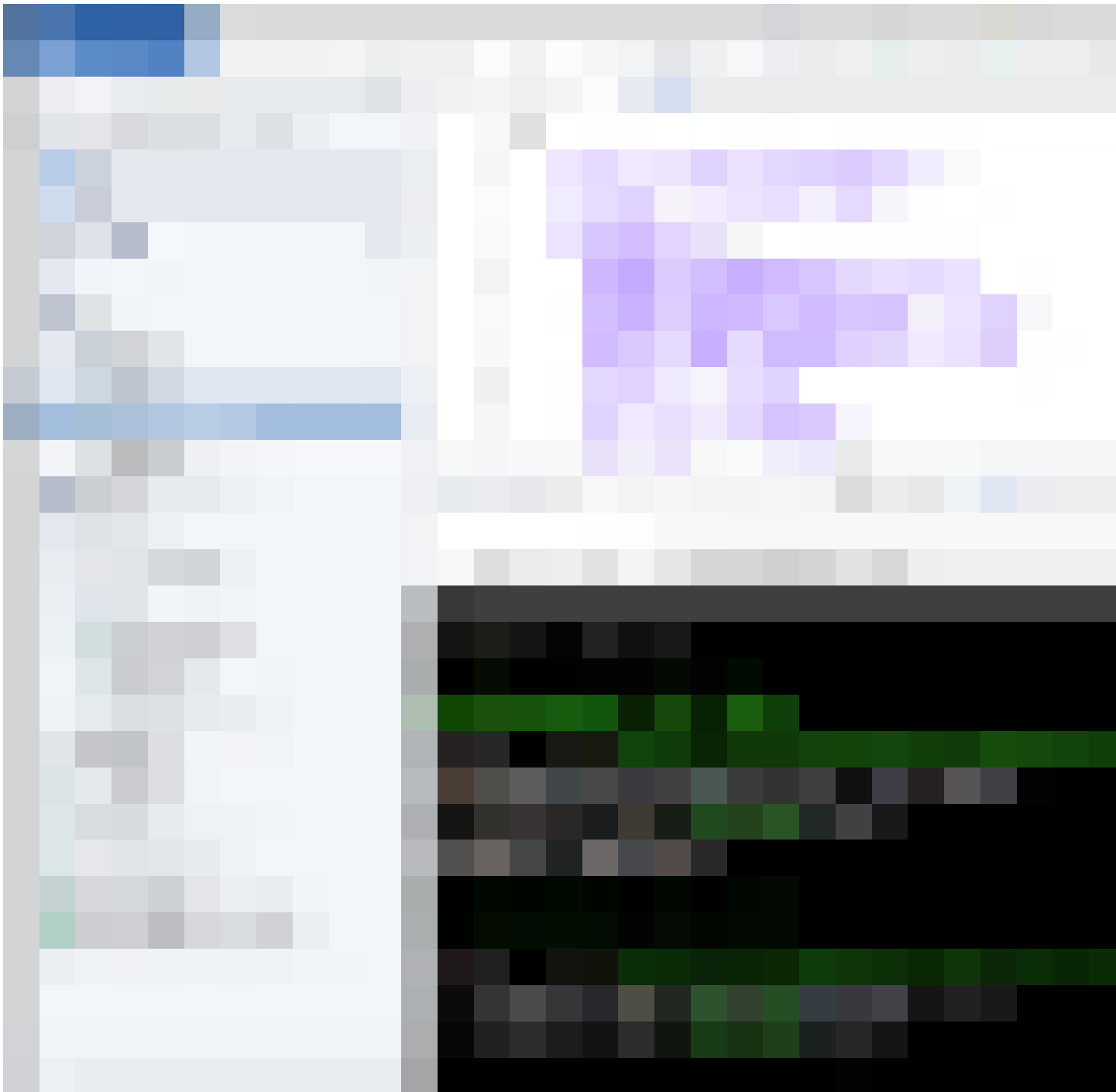
Monaca — A Sneak Peek into the New Cloud IDE

Improving user development experiences with Monaca is what we are striving for. Our developer team has been working hard for months in order to bring you the **BRAND NEW** Monaca Cloud IDE with a completely revamped look! We can't wait to release this new and better IDE, and are looking forward to seeing your feedbacks.



1. Transpiling Support in the Cloud IDE

This is one of the most important features in the new Cloud IDE. It expands your ability to create, manage and develop **transpilable project** such as React, VueJS, Angular, and more. Not only does it transpile but the previewer is also launched from the embedded terminal. This means projects with **hot-loading** support is also available in the Cloud IDE!



2. Better Developer Experience & More User Friendly Cloud IDE

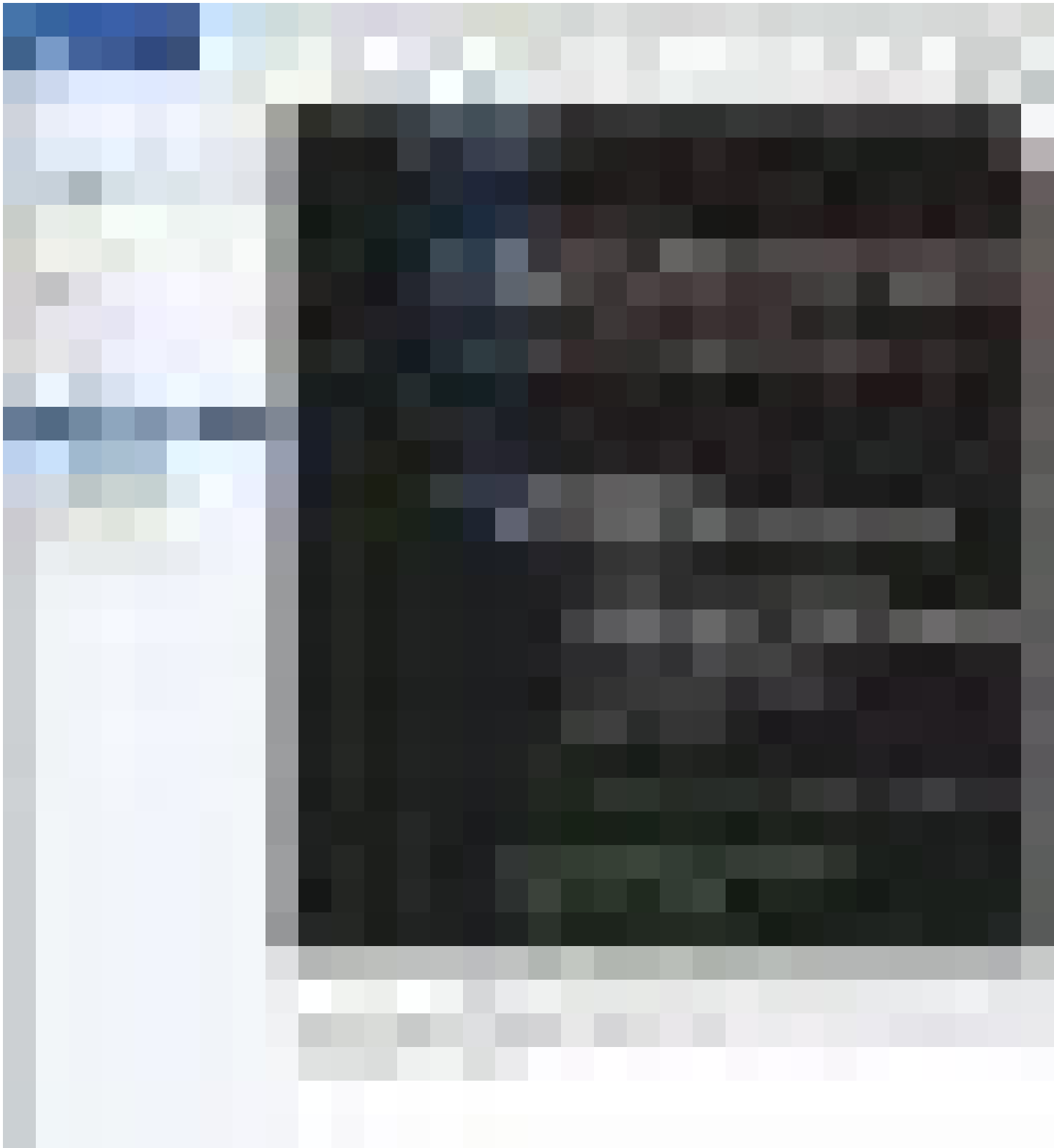
Taking into the account the importance of developer experiences, we have been adding and changing various features and user interfaces in the IDE.

One of those features is the new **flexible layout**. In the current IDE, all panels are fixed on a specific layout. Unlike the current IDE, all panels can be dragged and placed on any positions. Each panel can also be maximized/minimized with ease. As a result, you can arrange the layout as you

prefer. This new layout is made by [Golden layout](#) which a multi-screen layout manager for webapps.



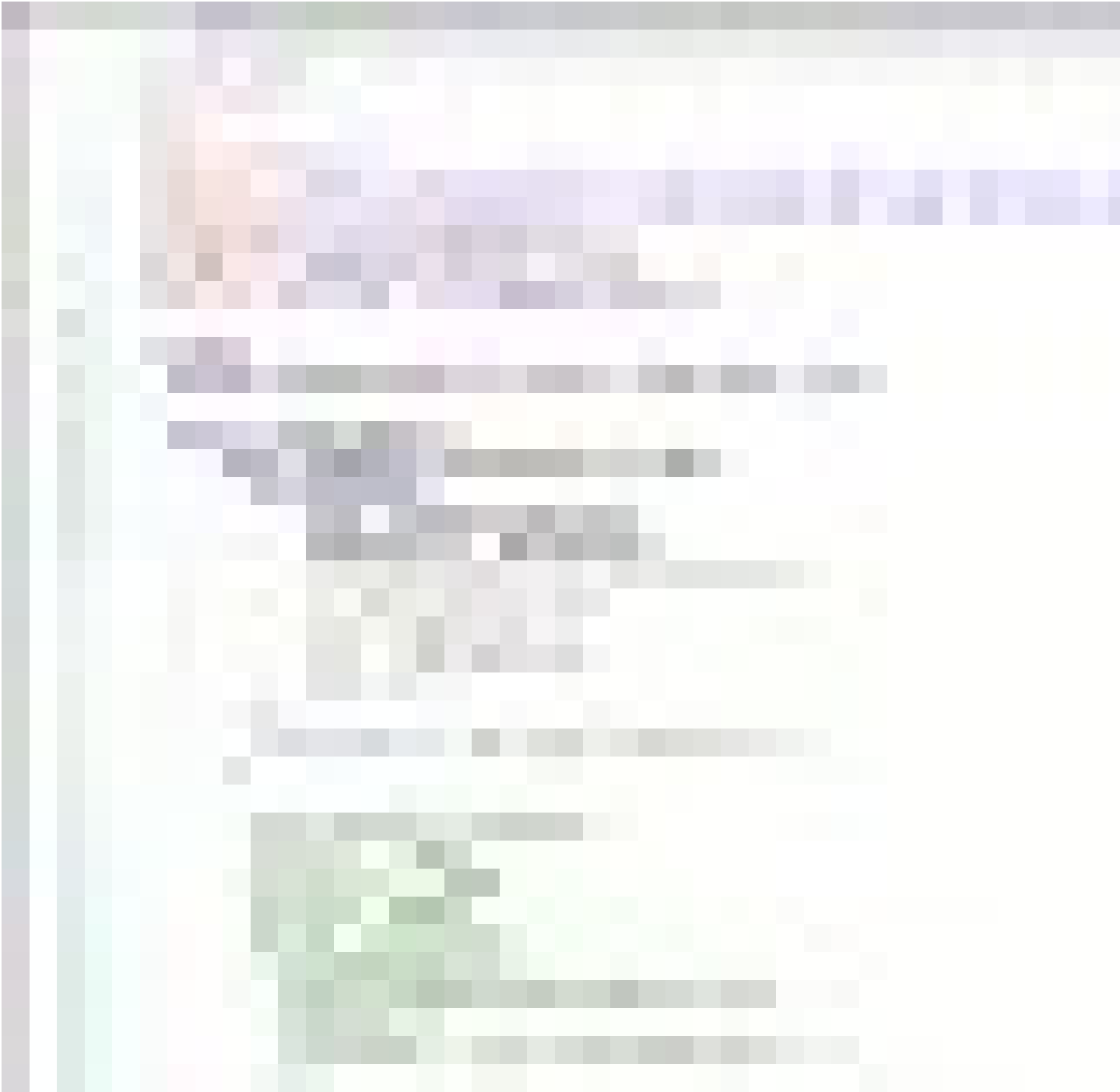
In addition to the flexible layout, the ability to preview your app in various devices' layout at the same time is also added. Rather than having only one Preview window at a time, the new IDE allows you to have **multiple Preview windows** at the same time.



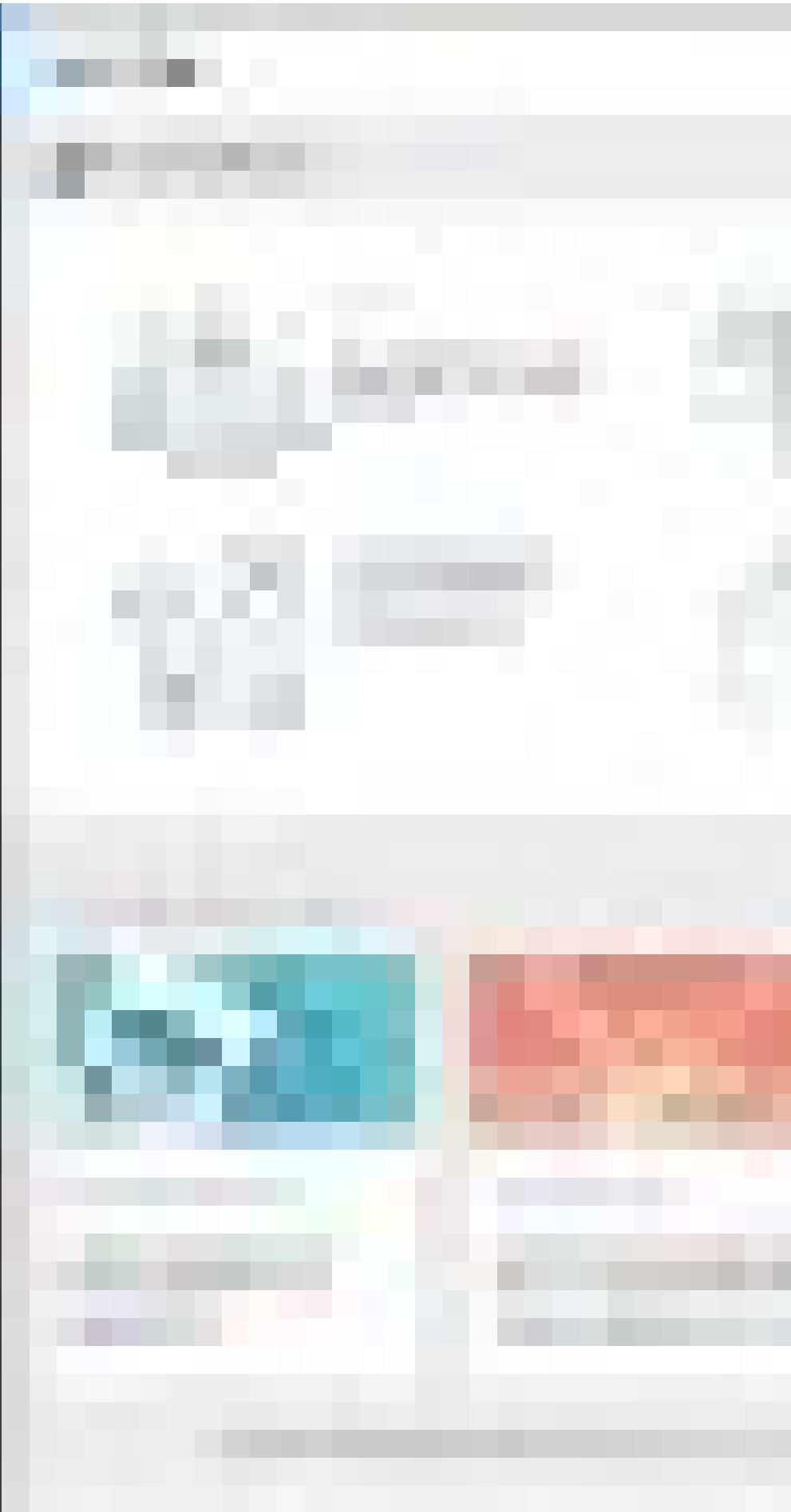
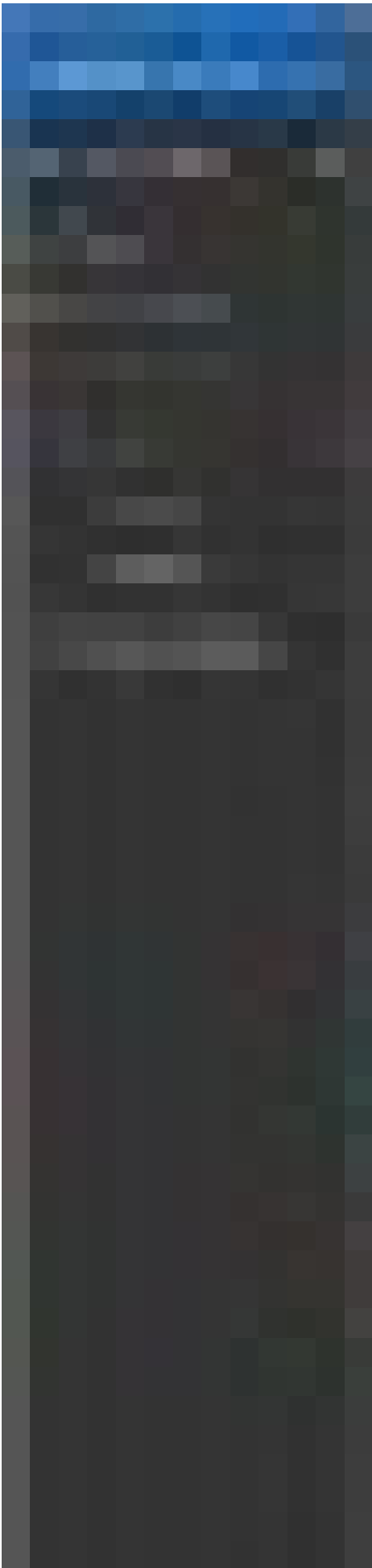
A majority of time for app development is spent on writing code. Therefore, using the right code editor would **speedup the development time and efficiency**. For this reason, we are replacing Ace editor (used in the current IDE) with [Monaco](#) which fits very well with Monaca for various reasons such as:

- A more robust auto-completion system that can be expanded with the use of a Language Server. Some languages we will provide right out of the box are TypeScript, HTML, JavaScript and Onsen UI. And more to come such as React and Vue.
- Basic syntax colorization

- Rich intelliSense & validation

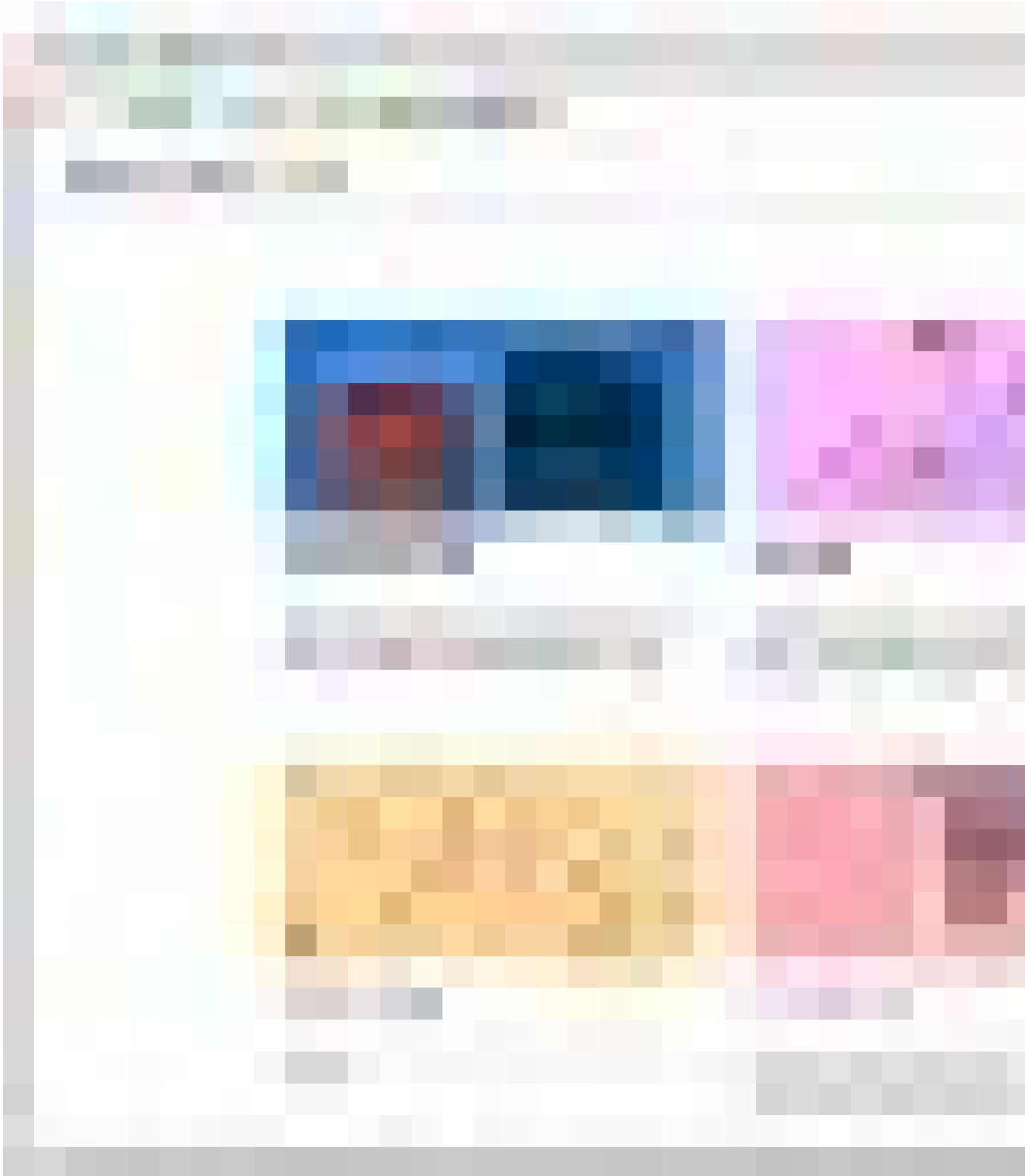


Last but not least, the new IDE will unveil an **entirely new look** for Monaca Backend! The new Monaca Backend will be on a separated window with a complete new look! All backend components such as User, Collection, Push notification and Mail template are much more easier to manage. The navigation between them are also becoming more user-friendly.



3. New & Improved Monaca Documentation

We've recently released our renewed [Monaca Documentation site](#). In the meantime, there are no major changes in the contents of the Monaca docs. However, we are planning to revamp various contents in order to make it easier for users such as adding more examples for API usages and more useful sample applications and demos.



Onsen UI — More Tutorials & Improved Documentation

Onsen UI documentation has gone through a major change in contents. More examples and detailed descriptions are being added to each Onsen UI component. Moreover, more useful tutorials are coming as well!

We are very excited about the upcoming releases in March! And we can not wait to hear what you think about them!

Cheers,

Monaca & Onsen UI Team

Unlimited Free Push Notifications with OneSignal and Cordova



[Khemry Khourn](#)

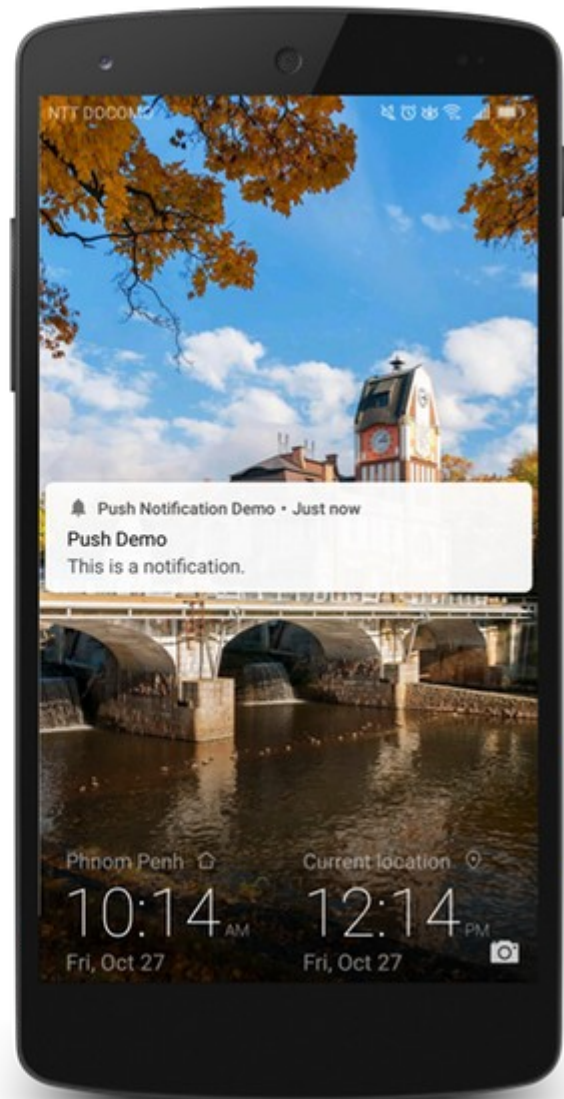
[Oct 30, 2017](#) · 7 min read

Push notifications are a communication channel between users and apps. It allows the apps to reach out to users with short messages in order to trigger some actions or interests from them. For this reason, push notifications are used by nearly every major apps for transaction and re-engagement.

There are many push notification services available out there. However, we are going to use [OneSignal](#) service in this article. You will learn how to add push notifications feature in a Cordova app using [OneSignal](#) service and [Monaca Cloud IDE](#). We will demonstrate how to send push notifications and create a handler when the user opens a received push notification.

Tested Environments: Android 7.0 & iOS 10.1





What is OneSignal?

OneSignal is a multi-platform push notification service and it is totally FREE to use.

OneSignal has become the most widely used push notification service for both web and mobile developers due to various reasons such as:

- **Free:** supports unlimited devices and notifications without any charges.
- **Easy to use:** setup push notification in OneSignal is simple and easy to follow.
- **Multi-platform support:** provides a single UI and API to deliver messages across various platforms such as iOS, Android, Amazon Fire, Windows Phone, Chrome Apps, and so on.
- **Multi-SDK support:** provides SDKs for nearly every major cross-platform mobile development environment, including Unity, PhoneGap, Cordova, React Native, Intel XDK and so on.

What is Monaca?

Monaca is a Cross-platform hybrid mobile app development platform which has various cloud services and tools such as Cloud IDE (browser-based IDE), Localkit and CLI.

Monaca is a very simple development tool for Cordova apps. Even for people with less experience in app development can start using Monaca right away. Monaca has a flexible development environment. You can either develop on the cloud or locally. Without any installation, you can build your app and include any Cordova plugins easily within a few clicks.

Monaca also comes with [Monaca Debugger](#) app which is used to test your app on the fly in real-time without building it every time.

Step 1: Getting Started with OneSignal

App Registration with OneSignal

In order to integrate OneSignal service with Monaca app, OneSignal's **App ID** is required to initialize with [OneSignal-Cordova-SDK](#) plugin which is used in the demo app later.

In order to get the OneSignal's **App ID**, please do as follows:

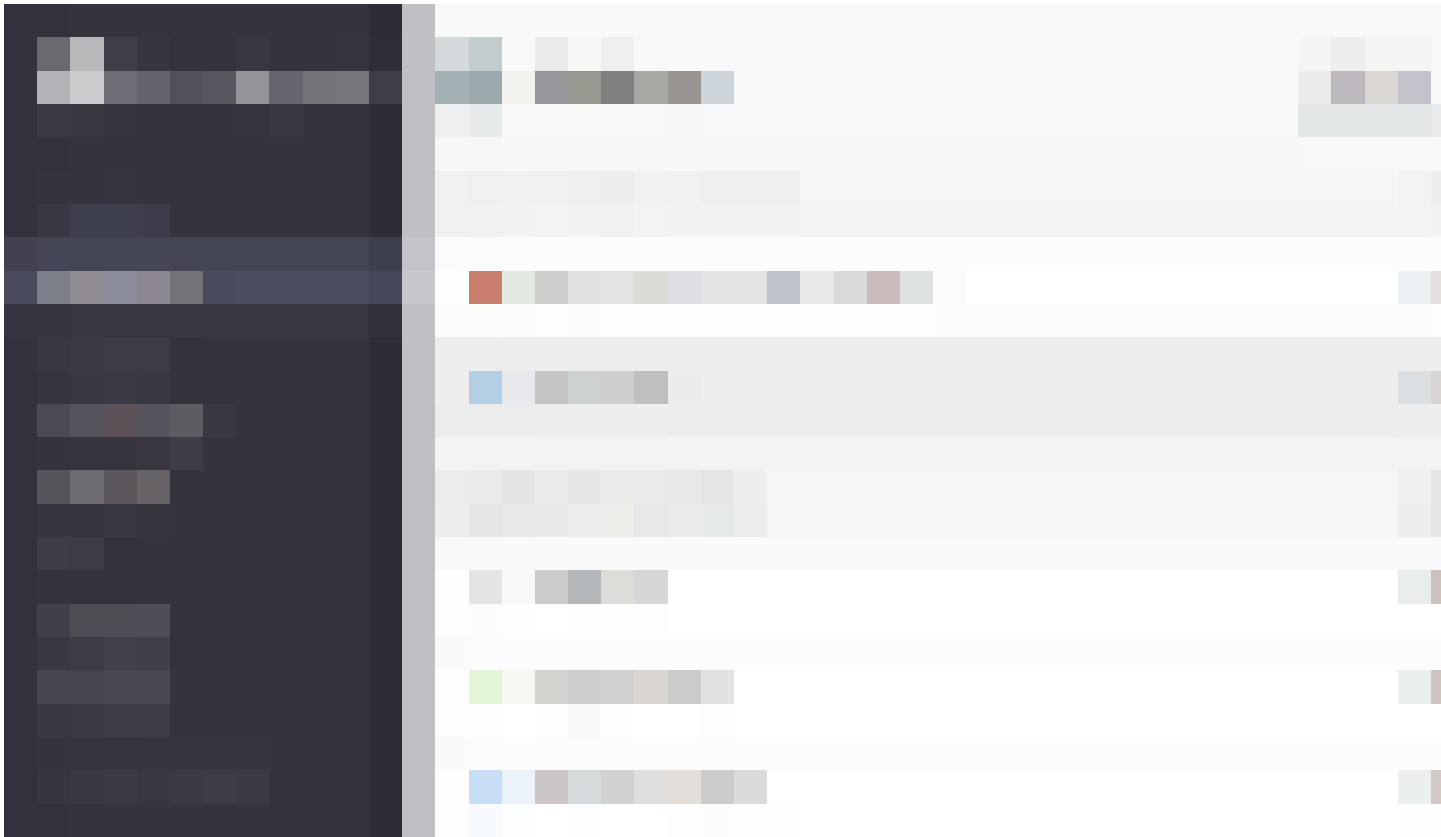
1. Register an account at [OneSignal](#).
2. Click on **Add a new app**. Fill in the name of the app and click **Create**.
3. Then, the platform configuration dialog will appear. Close the dialog for now. We will configure the platform later.
4. Go to **App Settings** and select **Keys & IDs**. You can find OneSignal's **App ID** here. We will use this in the application later.



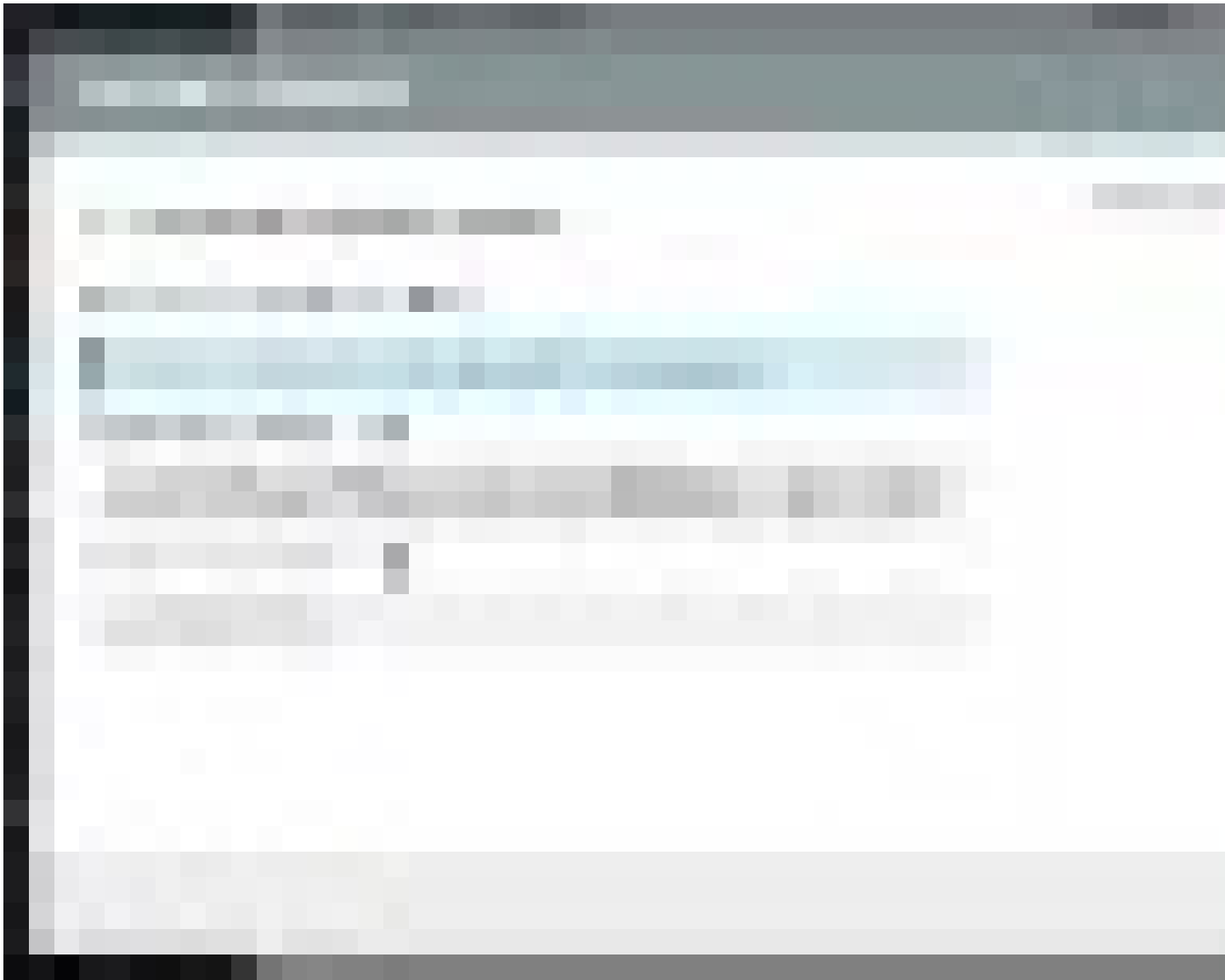
Push Notification Settings for Android

After successfully created the application in OneSignal, we are ready to configure the push notification settings for Android. In order to do so, please follow the instruction below:

1. Go to **App Settings** and click on **CONFIGURE** button of **Google Android**.



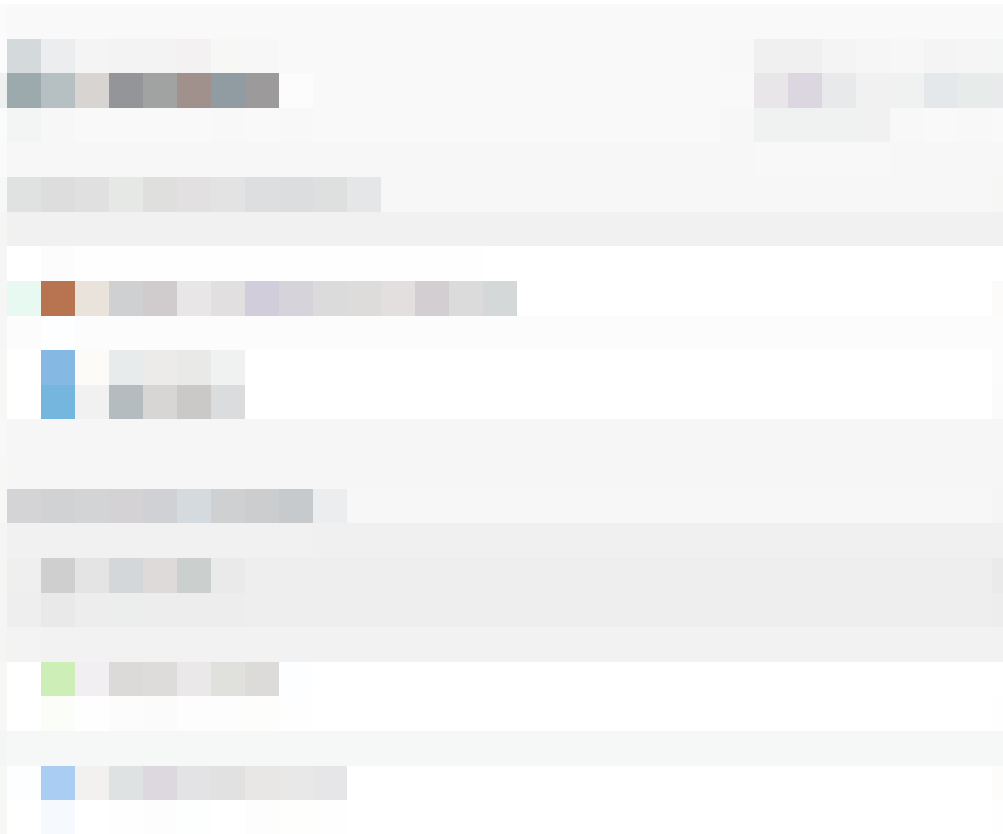
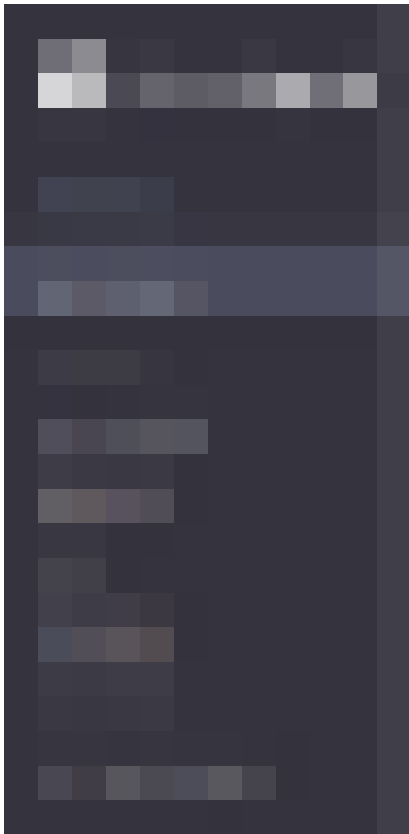
2. Fill in the **Google Server API Key & Google Project Number** and click **SAVE**. For more information on how to obtain these key and number, please refer to [Getting API Key from Firebase Console](#).



Push Notification Settings for iOS

In order to configure the push notification settings for iOS, please do as follows:

1. Go to **App Settings** and click on **CONFIGURE** button of **Apple iOS**.

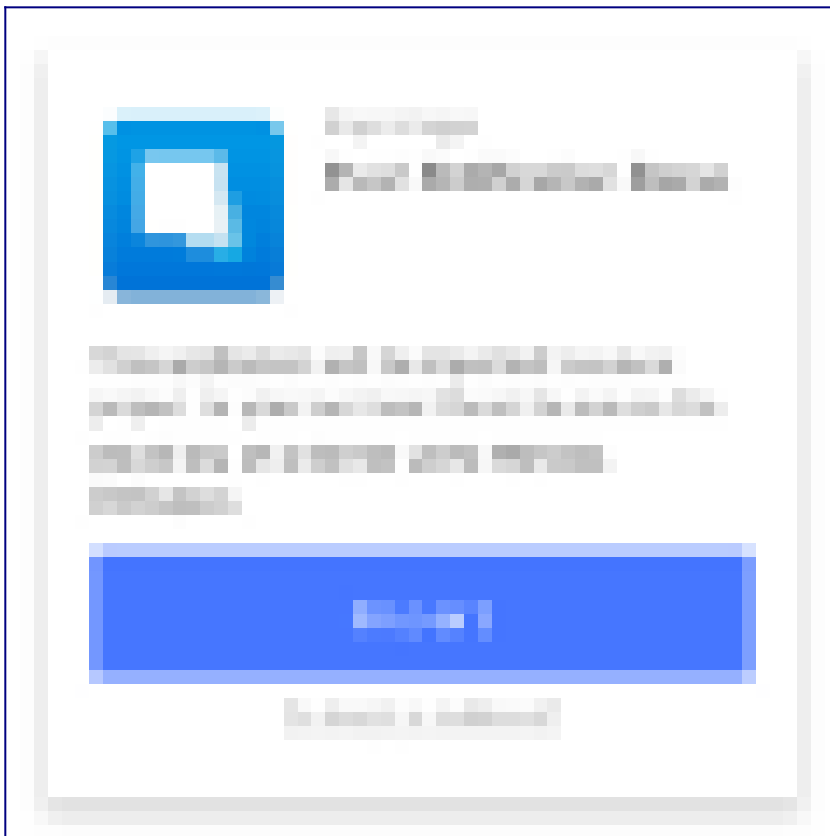


2. Upload the **Production Push Certificate (.p12 file)** and input its password (if available). Then, click **SAVE**. You may want to refer to [Generate an iOS Push Certificate](#).



Step 2: Importing Push Notification Demo Project into Monaca Cloud IDE

1. Click on the link below to directly import the **Push Notification Demo** project into your Monaca account. If you don't have a Monaca account, please register [here](#).

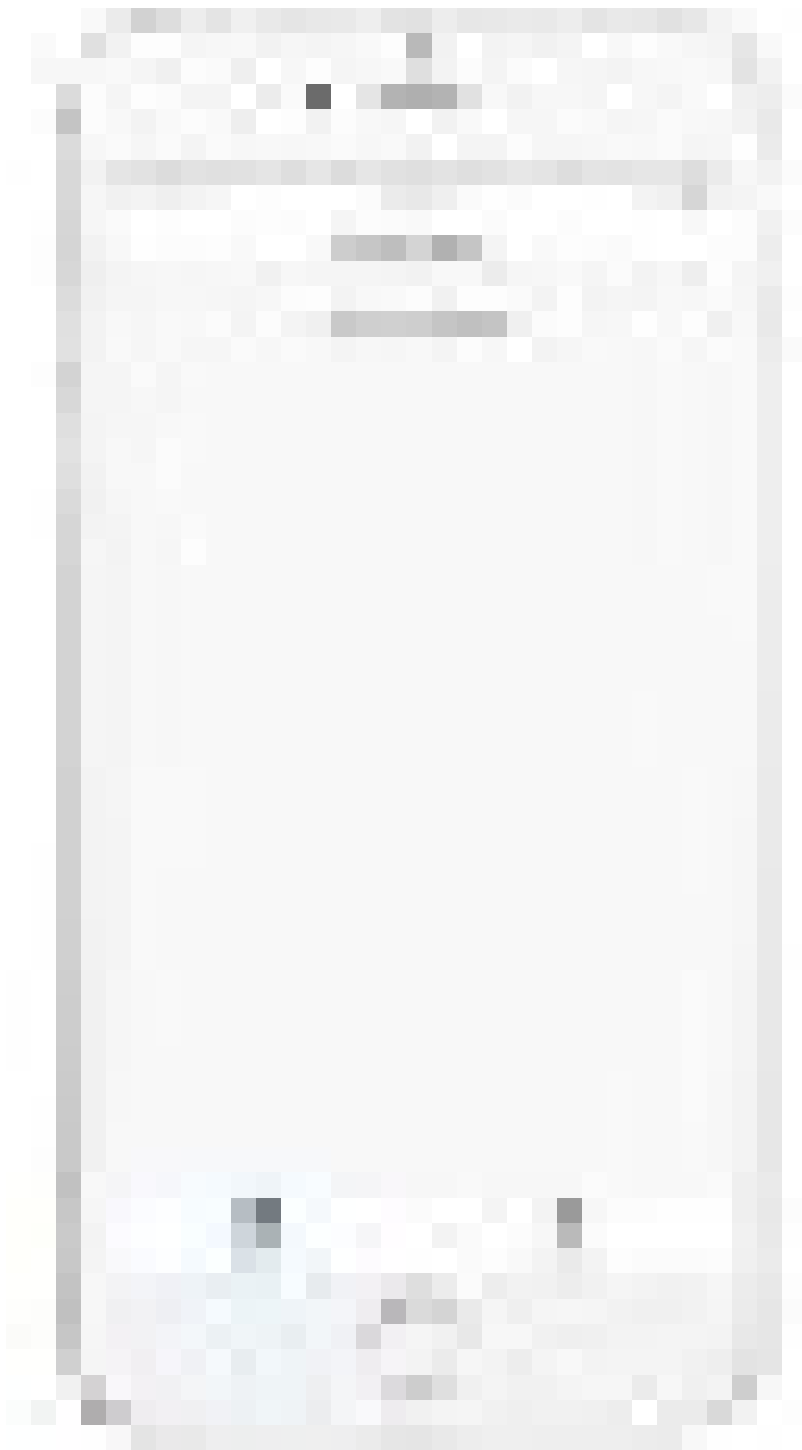


2. Open the project and replace your OneSignal's **App ID** in the following snippet within `index.html` file and save the change.

Step 3: Building & Installing the App

In order to test the application, build the application and install it on your device:

1. From Monaca Cloud IDE, build the project for iOS or Android. You may want to refer to:
 - [Build Monaca Apps for iOS](#)
 - [Build Monaca Apps for Android](#)
2. Install it on your device.
3. Run the app. By default, the **Page 1** tab is the default tab.



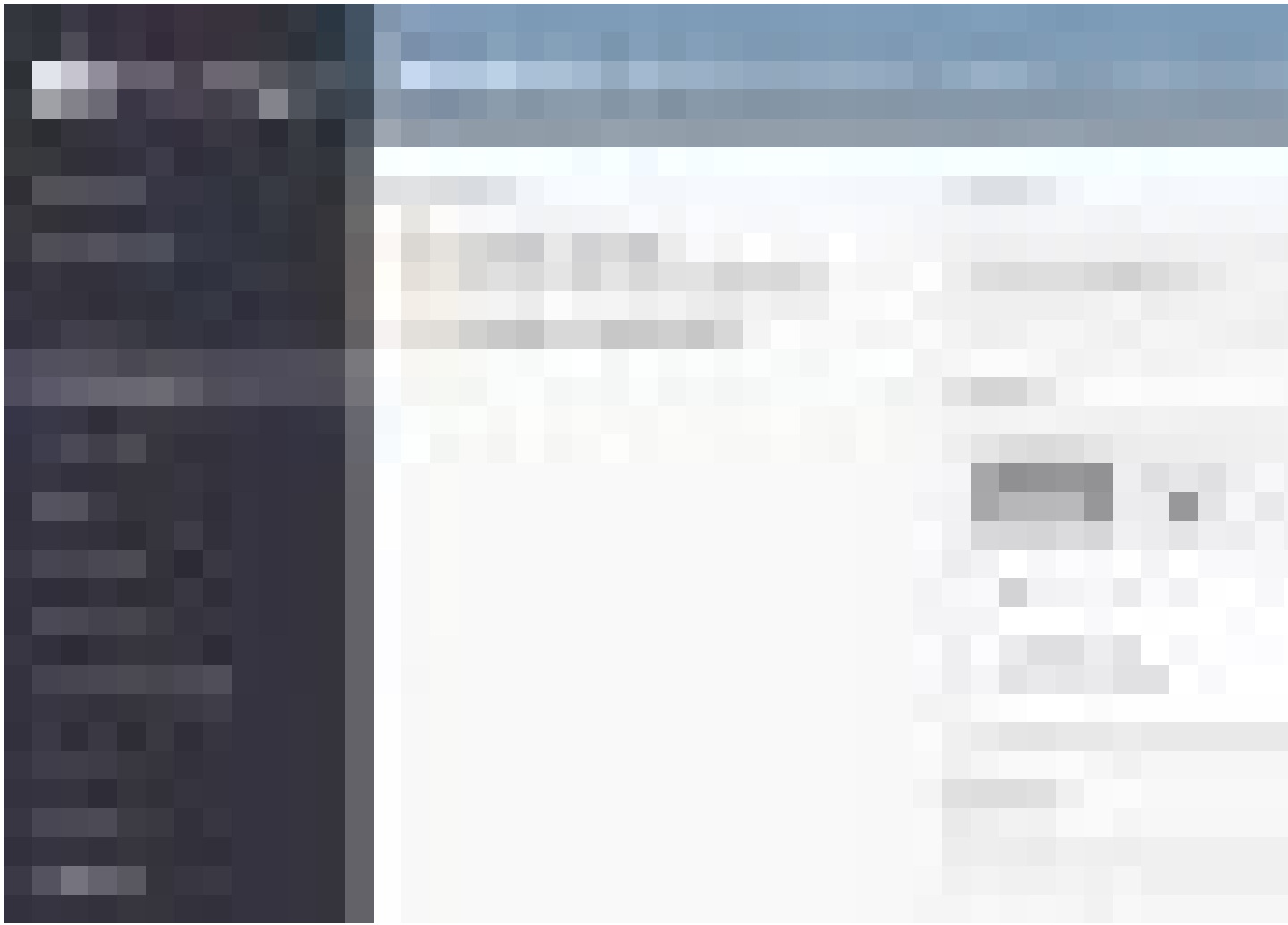
9. If you click on **Page 2** tab, you will see the following content. By default, the data from push notification is set to **NONE**. It will change if the received push notification has any data.



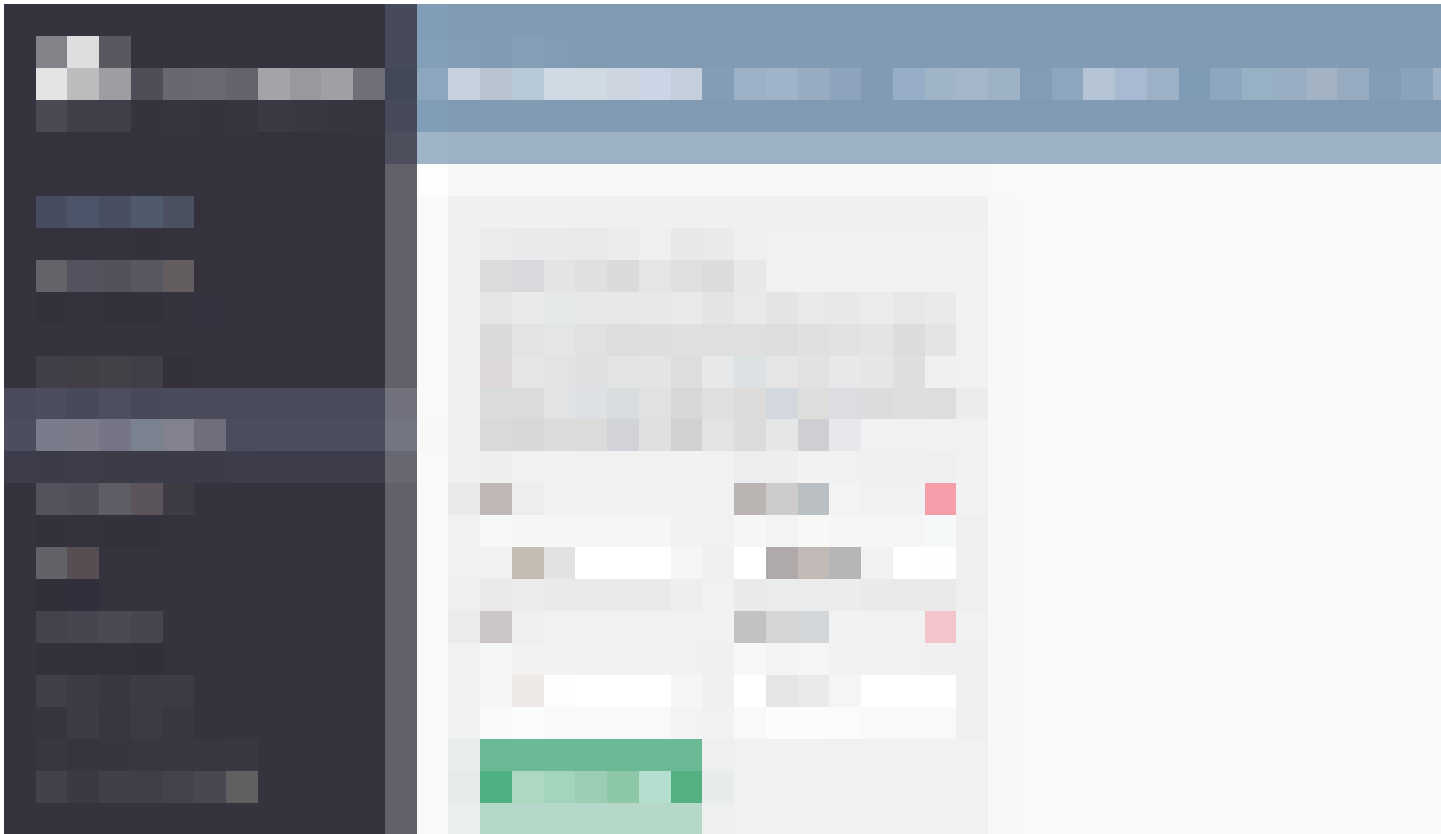
Step 4: Sending Notifications from OneSignal

Until this step, we assume that you already have configured the push notification with OneSignal and installed the Push Notification Demo app on your device. Therefore, we are ready to start sending the notification from OneSignal:

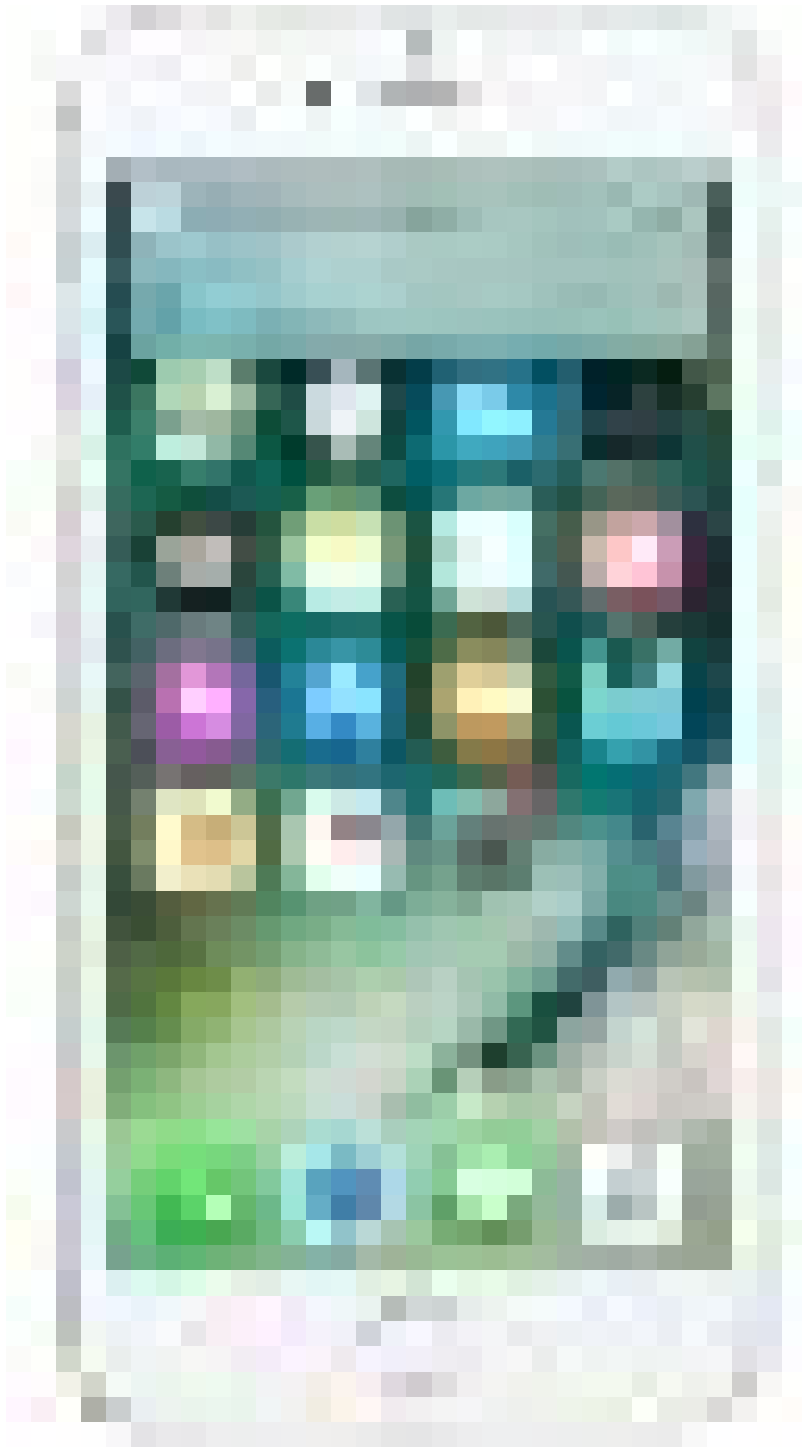
1. From OneSignal Dashboard, go to **New Message**.



2. Select **Send to Everyone** option and click **NEXT**.
3. Fill in **Title**, **Subtitle & Message**. Then, click **NEXT**.
4. Scroll down to **ADDITIONAL DATA** section and add two custom key values pairs as shown in the screen below. Then, click **NEXT**.



5. Schedule the time to send the push notification. For now, just keep the default setting and click **CONFIRM**.
6. Review the configuration for the message. Then, click **SEND MESSAGE**.
7. It may take several seconds for the device to receive the push notification. In iOS, the push notification looks like this:

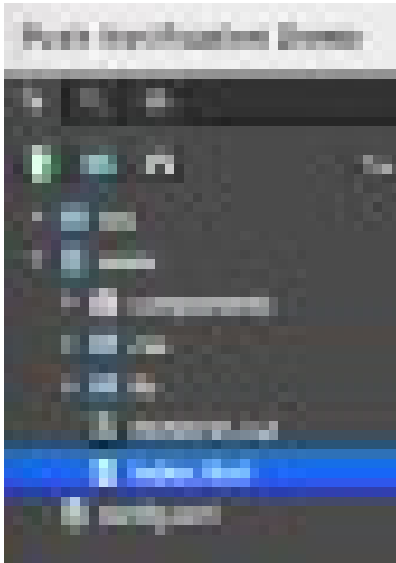


8. Once the push notification arrives, click on it. This should open the app and automatically open **Page 2** tab and the data from the push notification is shown as well:



NOTE: For a complete guide on how to configure a message in OneSignal, please refer to [Sending Notifications](#).

Step 5: Understanding the Application



This is a very simple single-page app using [Onsen UI v2](#) and Angular 1. The app has two tabs: **Page 1** & **Page 2** tabs created by a tabbar component using `<ons-tabbar>`. **Page 1** & **Page 2** tabs point to `page1.html` and `page2.html`, respectively. Each page is created by using `<template>`.

The app has one simple controller called `AppController`. Within the controller, the `data` variable (used in Page 2 representing the data from a push notification) is initialized to be `NONE`. If the push notification is sent with any data, this variable will be updated by using a function called `notificationOpenedCallback`. This function is triggered when a user opens a notification. Within this function, the `data` variable is assigned to the JSON data found in the received push notification. After that, change the app to **Page 2** tab using `setActiveTab()` function.

`startInit` and `endInit` starts and ends the initialization of the OneSignal plugin, respectively. In between the start and end of the plugin, there are various handlers/methods you can use such as `handleNotificationReceived`, `handleNotificationOpened`, `inFocusDisplaying`, `iOSSettings`, and so on. For more information, please refer to [Cordova SDK for OneSignal](#).

Conclusion

As push notification has become such an important feature for mobile apps to interact with users, we are demonstrating how to integrate and use OneSignal, most popular and free multi-platform push notification service, with Monaca (Cordova/hybrid) apps.

We hope that you find this article useful in order to add a push notification feature to your own apps. If you have any questions or suggestions, please do not hesitate to comment here. **Thank you very much!**

Develop a Monaca Project with Visual Studio Code



[Onsen UI & Monaca Team](#)

[Aug 4, 2016](#) · 4 min read





[Visual Studio Code](#) is an open source code editor developed by Microsoft and based on [Electron](#), a framework allows creating desktop GUI applications using the Node.js.

Visual Studio Code comes with tons of features like debugging, Git integration, syntax highlighting, IntelliSense and more. It is also highly customizable. On top of that, Microsoft recently announced [Cordova Tools](#) for Visual Studio Code. With Cordova Tools extension, you can build, debug and preview Cordova hybrid mobile apps on Visual Studio Code.

With that in mind, in this blog post, we will quickly explain how to create a Monaca hybrid application using Microsoft's Visual Studio Code with Cordova Tools.

Prerequisites

First of all, make sure you have Node.js installed on your device, then install Cordova using the following command:

```
npm install -g cordova
```

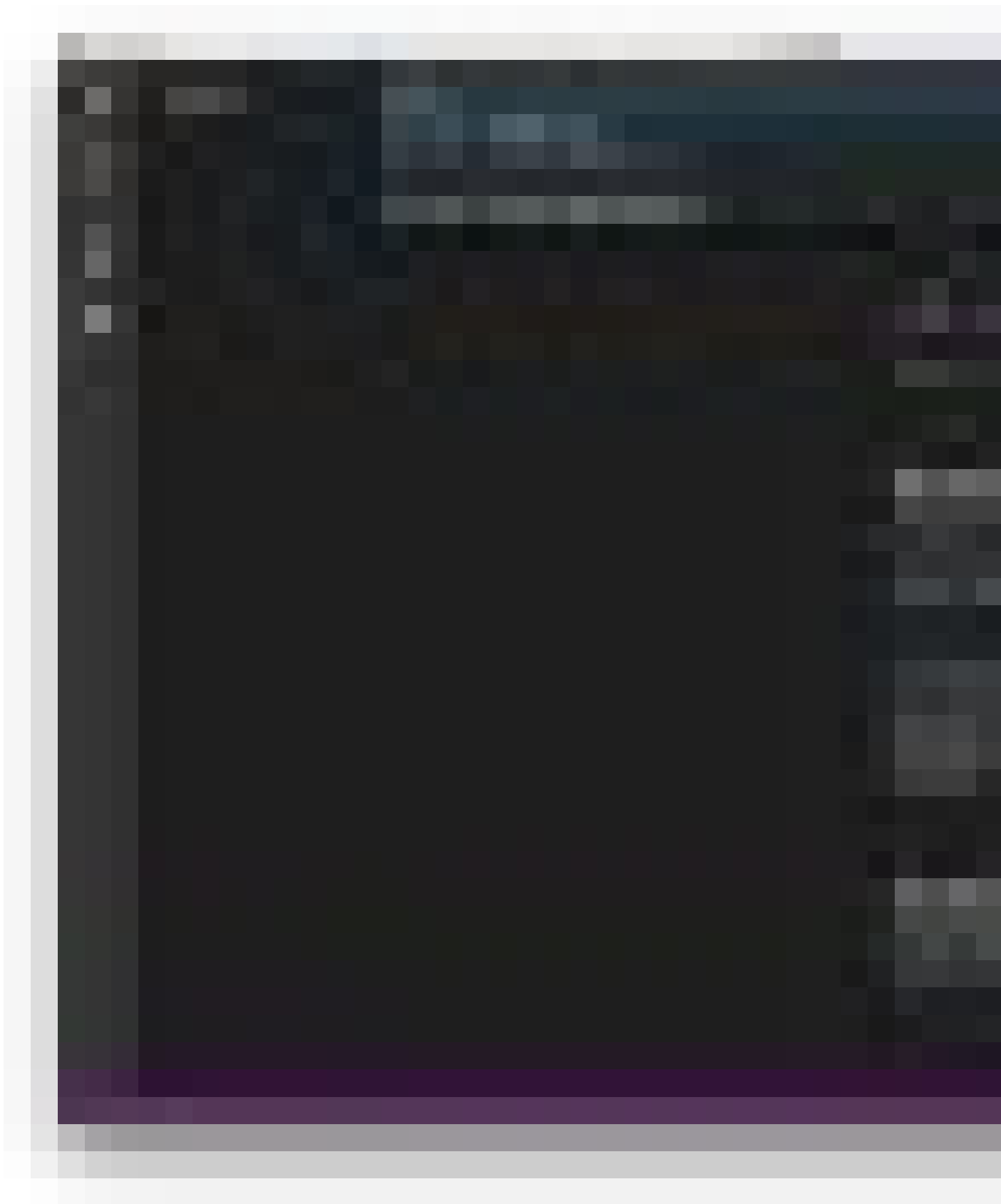
If you are targeting iOS devices, install the libraries using Homebrew. Note that iOS debugging is not supported in Windows.

```
brew install ideviceinstaller ios-webkit-debug-proxy
```

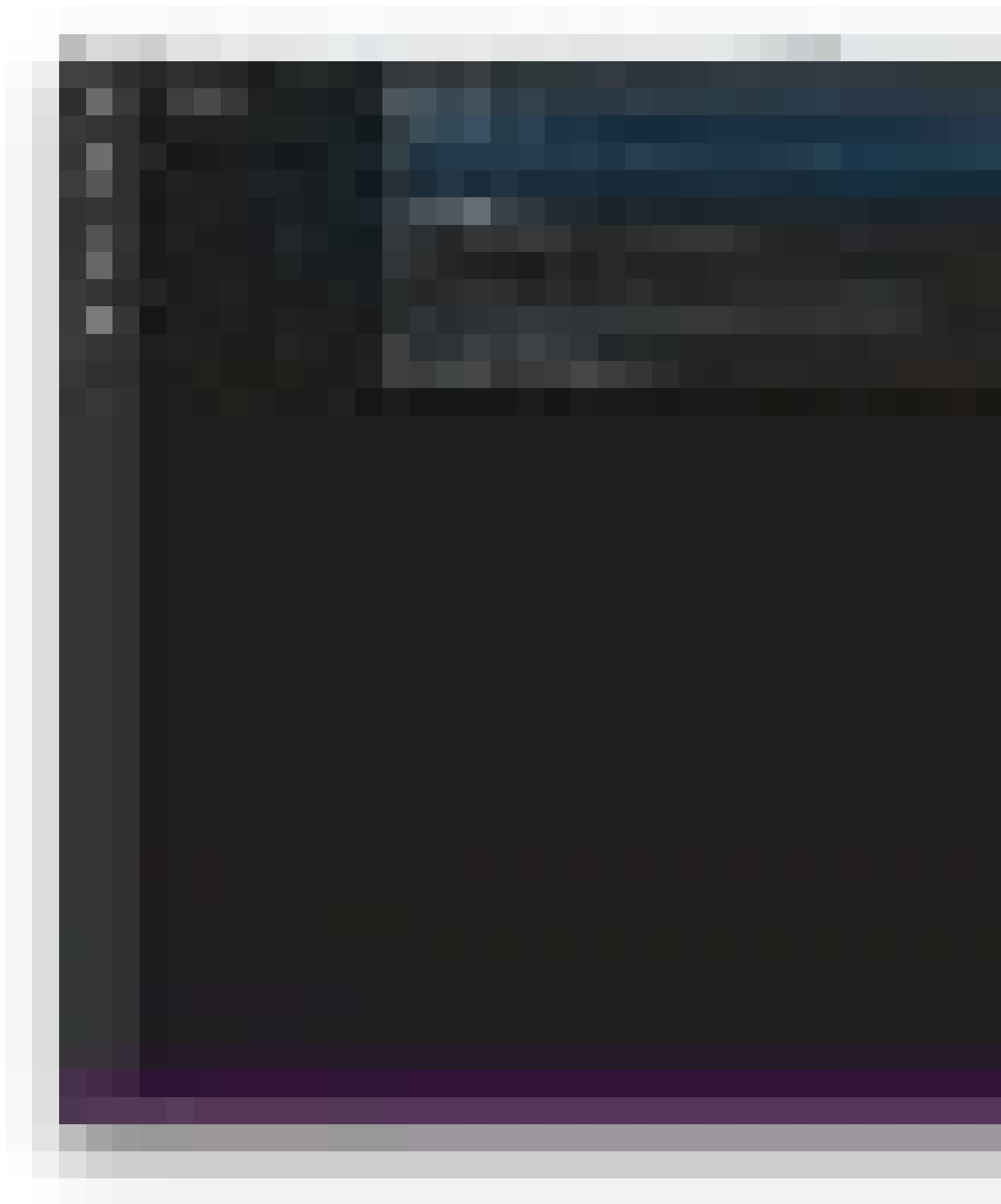
Installation

If you don't have Visual Studio Code, download it from [here](#). Once installed, start it up and install Cordova Tools extension. In order to do that, press CTRL+SHIFT+P (CMD+SHIFT+P for Mac OS

X) and the Command Palette will open. At this point, simply type `Install Extension` and hit enter.



Find Cordova Tools by simply typing `cordova`. Your command line should look something like this:



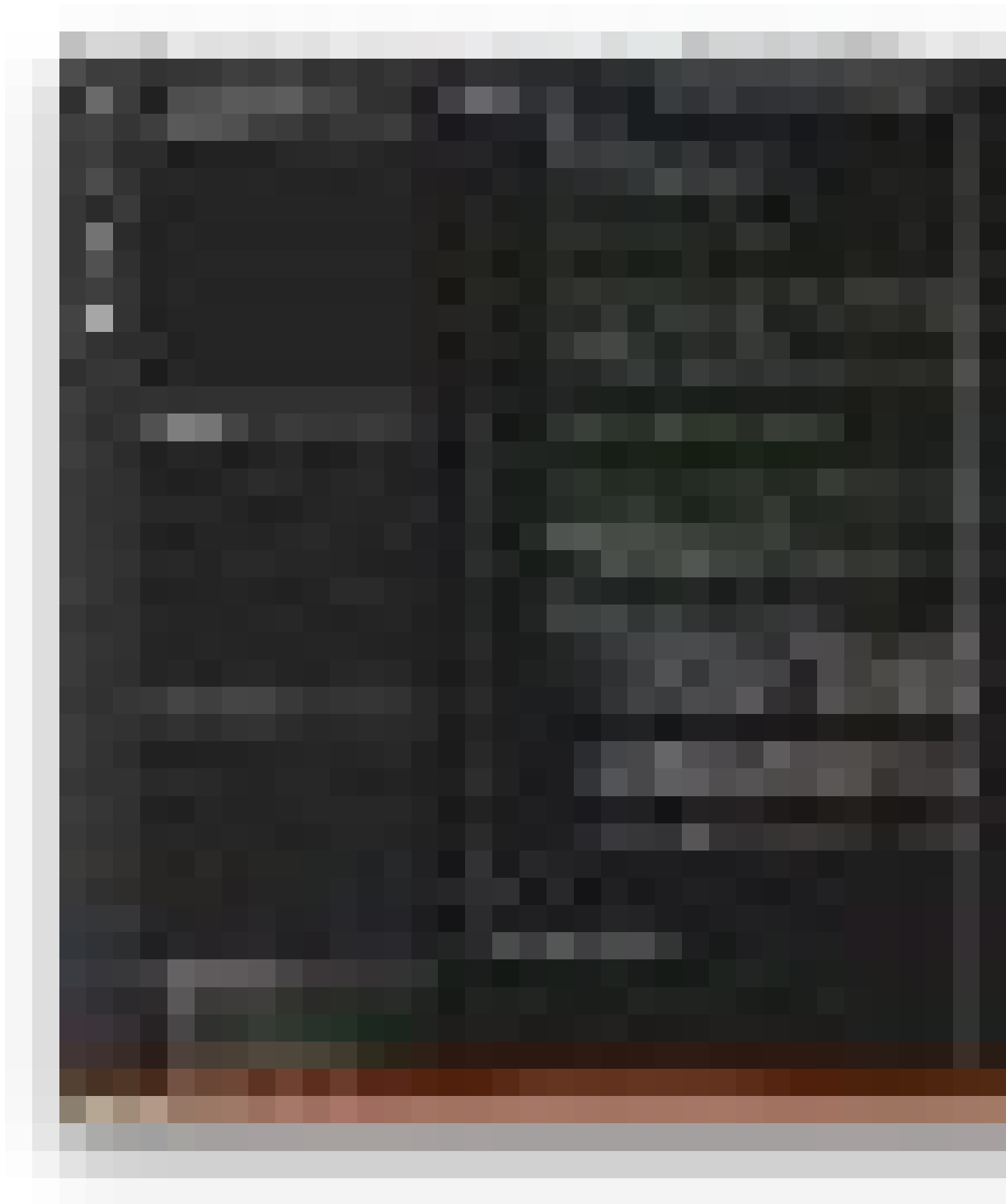
Once you hit enter, Visual Studio Code will immediately install the extension.

Project Creation

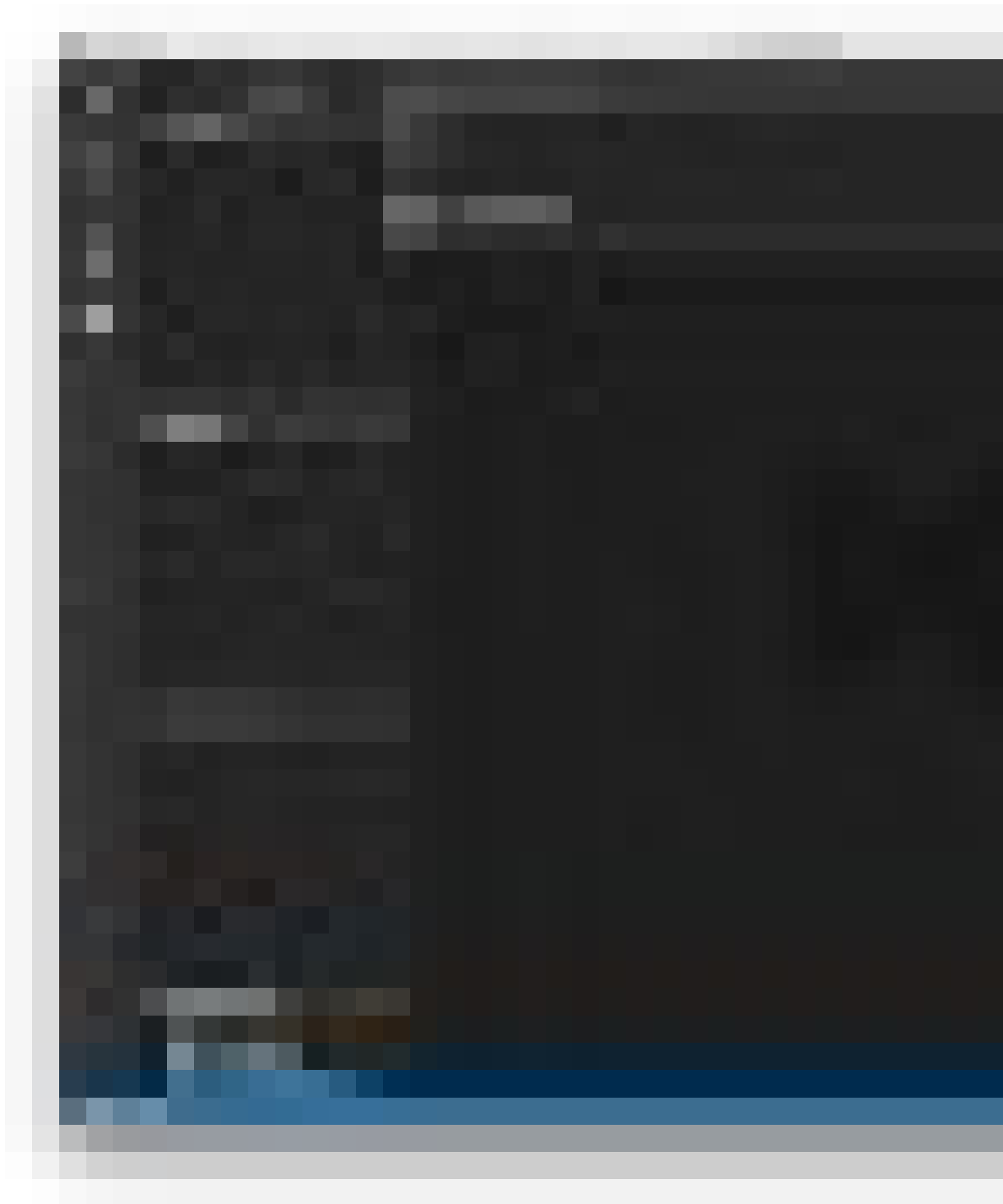
In this tutorial, we will be using [Monaca LocalKit](#). First of all, create a new Monaca project based on the desired template, then copy the previously created Cordova project's `platforms/ios` (or `platforms/android`) directory into the Monaca application's `platforms` directory.

Open the Project in Visual Studio Code

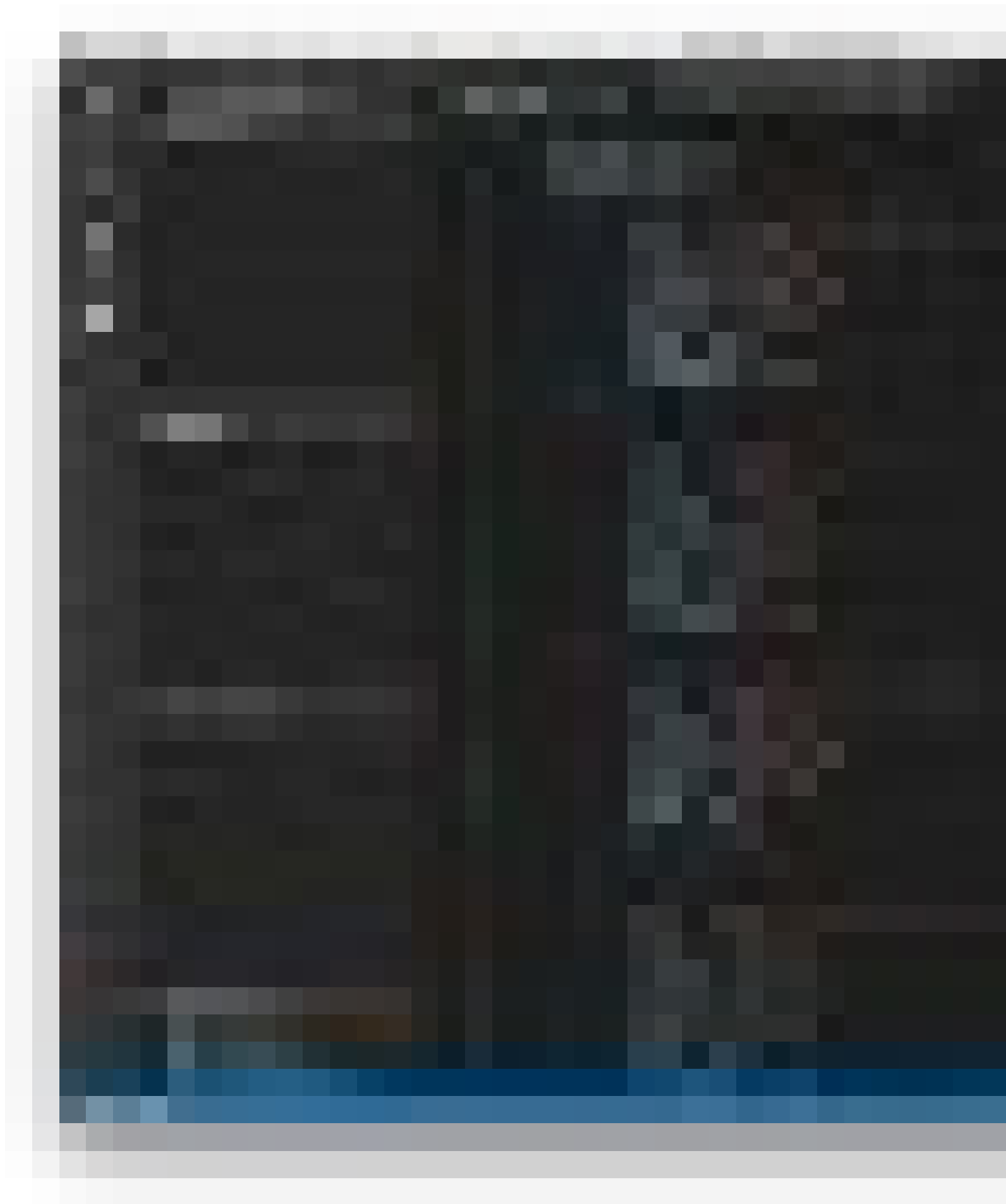
Open the Monaca hybrid mobile app project in Visual Studio Code, you should be able to see your directory tree on the left side of the window. You also have the ability to set breakpoints in JavaScript files by clicking in the left margin.



Click on the debug icon in the sidebar or **CMD+SHIFT+D**, then click on the gear icon next to the debug button in the upper left side and select Cordova.



We can now run our hybrid mobile apps on an iOS/Android simulator, emulator, or actual device.



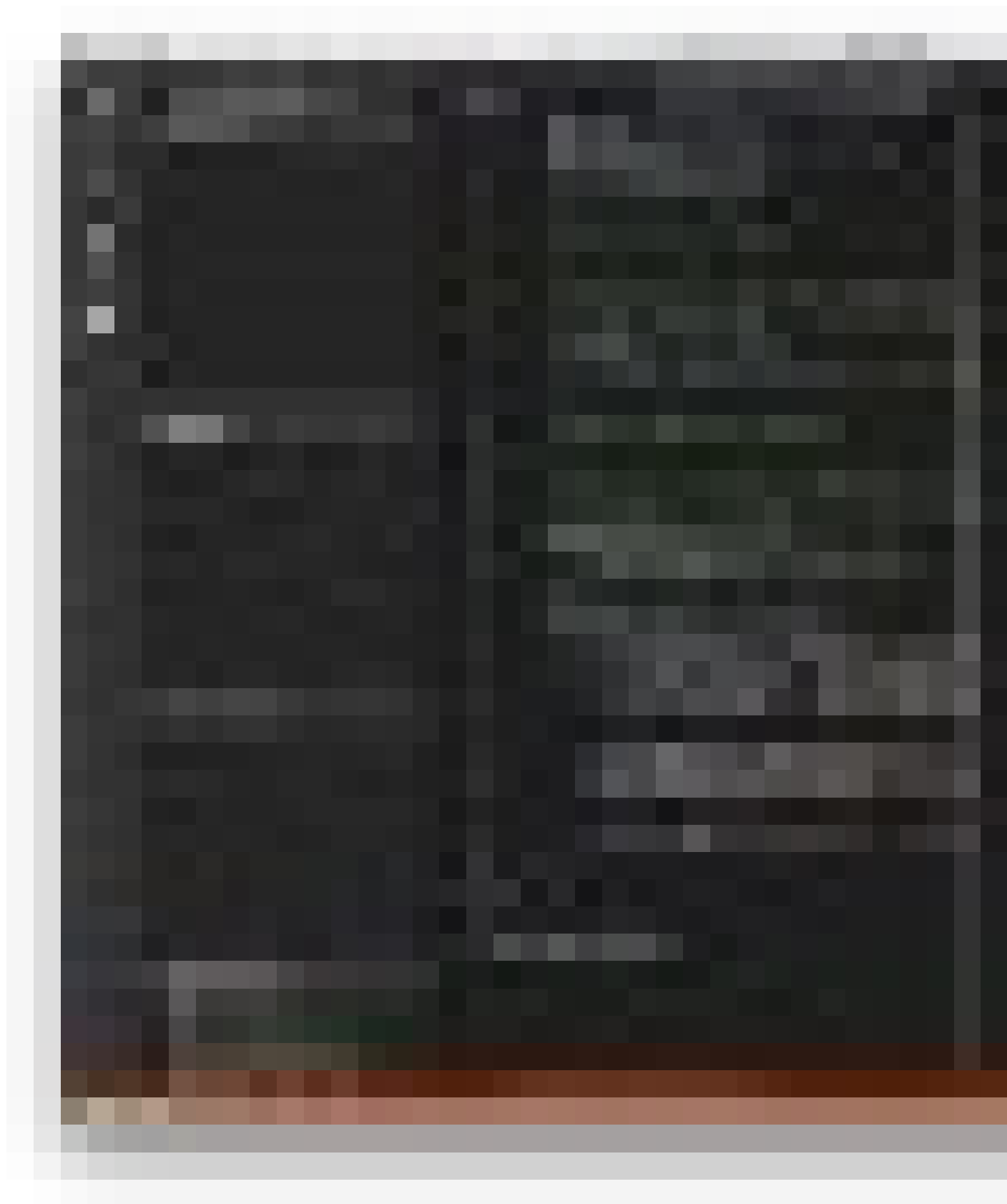
You can also run some JavaScript directly in Visual Studio Code's console. Here is an example of what happens when we run the following command:

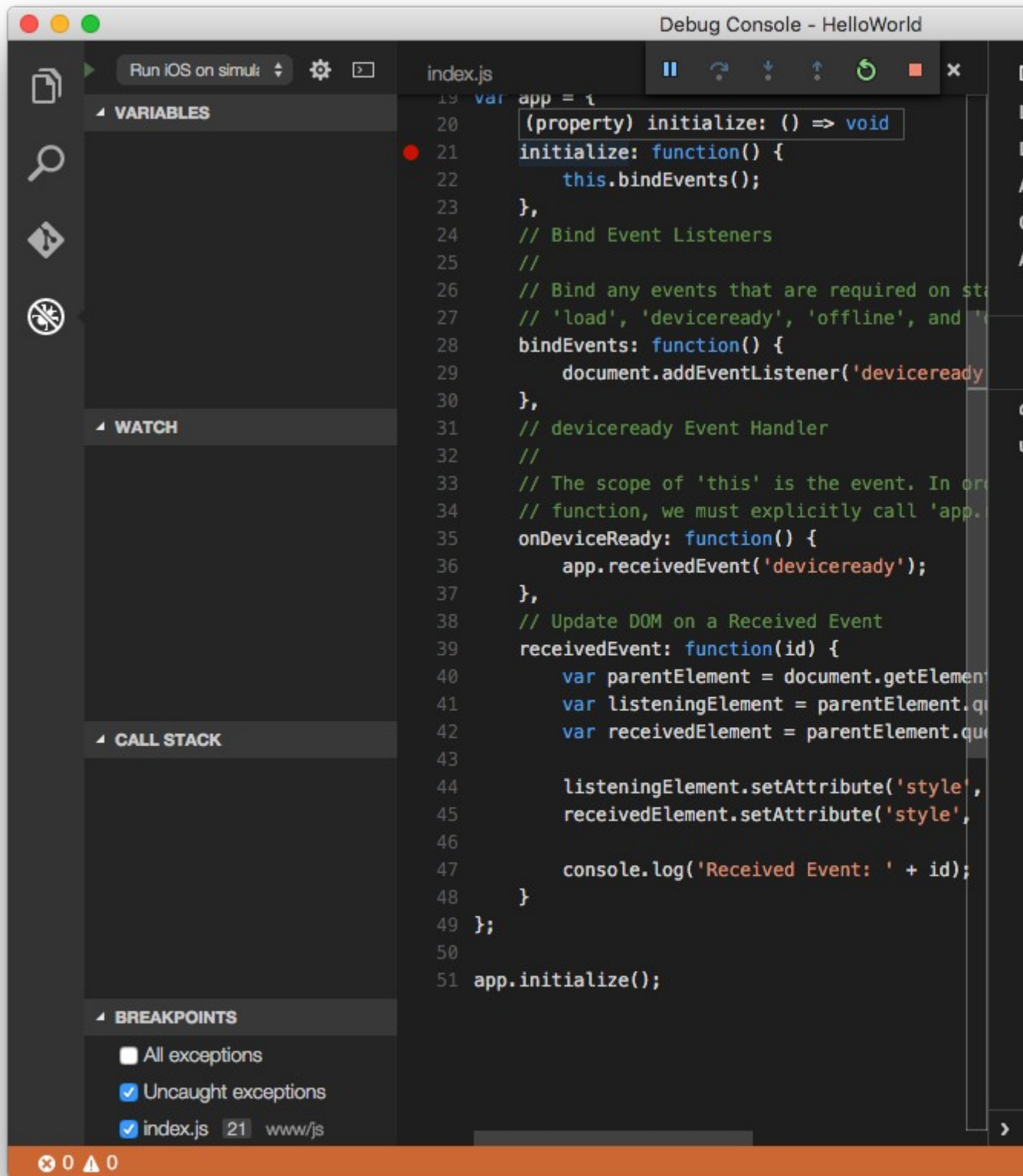
```
alert(true);
```



Features of Visual Studio Code

Today, we have quickly gone through how to create Monaca hybrid mobile apps using Visual Studio Code. As previously mentioned, Visual Studio Code provides varieties of features out of the box, just keep in mind that a stable version has just been released so you still may encounter some minor bugs. However, Built-in support for Git and debugging tools are great. Also, IntelliSense, snippets, code refactoring, and Extensions will surely increase your productivity. It can be definitely considered a powerful and reliable tool to create Monaca and Cordova applications.





Developing hybrid mobile applications with Onsen UI



[Fran Dios](#)

[Feb 23, 2015](#) · 12 min read

Hybrid mobile applications have become trendy in the last years. Writing native applications could be hard for casual developers, those who don't know too much about programming and just want to make a simple app (or not so simple) as quickly as possible. It requires learning different programming languages (and different ways to interact with each device) for every platform where you want to deploy your application, although it certainly has some advantages: best possible performance, full access to the device hardware, etc.

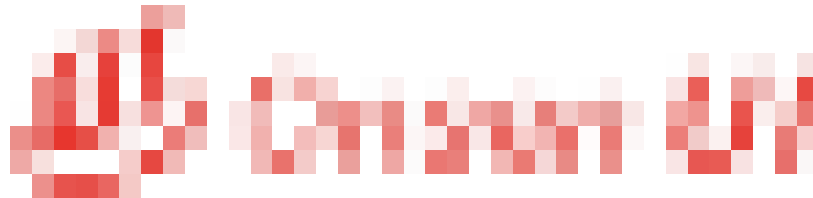


On the other hand, hybrid applications relies in a set of technologies that won't change regardless the platform, the same as in web applications: **HTML, CSS and Javascript**. And since this set of technologies is so common on the Internet we will find unlimited resources and tutorials about how to implement this or that, anything we need. We could even use any common tool or framework for web development also for hybrid apps, such as jQuery or AngularJS, thus making the development process much simpler and faster. Although hybrid apps are usually not as efficient as native ones, the truth is that current smartphones are powerful enough to run most of the applications we could imagine without noticing any difference.

In this post we are going to develop an example hybrid application (Memo app) to show how to use [Onsen UI](#), a framework to build quick and fancy User Interfaces. I am going to use [Monaca IDE](#) to develop this application (you can create a [fully functional free account here](#)) since it works pretty well with Onsen UI and [its debugger](#) is just awesome. You can check and fix your app from the IDE itself or from your phone (mobile debugger is available in [Play Store](#) and [App Store](#) for free) in a

few seconds. Plus, for those who are not advanced mobile developers, Monaca will ease most of the process of importing plugins, debugging and building the application. If you prefer to use something different it's totally fine, just remember to manually include the libraries that we are going to need.

Developing with Onsen UI and AngularJS



Onsen UI is a HTML5 framework to create modern and usable User Interfaces, so that we don't need to fight too much with the appearance of our application. It keeps the UI development simple while we can focus in the functionality of the application itself. Onsen UI is meant to be used along with [AngularJS](#), but it is also possible to use [jQuery](#) or any other framework. Here we will use AngularJS but will only describe the tricky parts in order to focus on Onsen UI.

The application we will make is a Memo App where we can store a list of tasks, classify them by category, modify and delete. To do this we are going to use the following Onsen UI components:

- `ons-navigator`
- `ons-toolbar`
- `ons-sliding-menu`
- `ons-list`
- `ons-carousel`
- `ons-template`
- `ons-popover`

You can learn how these components work together, so you can, for example, include an sliding-menu only inside one specific page and not in the others, or make every element of a list into a carousel. [Here you have the description of all Onsen UI components](#), as well as some [quick styles](#) that you may want to take a look at.



We will use Onsen UI 1.2.1 in this example. First, [if you are using Monaca](#), create a Onsen UI minimum template project in the Dashboard and open it. AngularJS is already included in the default libraries, so there is no need to do anything else. If you don't use Monaca just follow [these instructions to install Cordova and Onsen UI](#) and include the necessary libraries.

Before starting, let's explain how do we use AngularJS in this application. In AngularJS we bind every page or main element of our application to a [controller](#) that will store the necessary variables and functions for the scope related to the bound page. For example, in order to list our current tasks in the main page we will need to store an array of tasks in the scope related to the main page, i.e. the main page controller. We can also have functions in this controller to modify our array of tasks, such as delete or complete functions. However, sometimes it would be better to split this functionality into different controllers because, for example, we address the involved functionality in different pages but still need to access to the same data (for instance, adding a new task).

Therefore, we will need a way to allow communication between our controllers. In the current approach we use a [service](#) to manage all the storage of the task itself and common functionality, so that our controllers will call this service any time they need to access or modify the task list. Every task and task list are objects defined in *models/Memo.js*, and each task object contains the needed information for the task itself, including name, category, description, creation date and whether it is completed or not. This is just an example and is not meant to be the best approach, so you can change the model as you want. The model is not related to Onsen UI so we won't speak too much about it here, although you can have a look to the code on Github.

Enough details so far, let's get down to work.

Coding a Memo Application

All the code of this application is [available to download on Github](https://frandiox.github.io/OnsenUI-Memo-App/). It is recommended to check it out if you have any doubt while reading this post. You may also want to play a little bit with the application to understand how it works before getting hands dirty:

<https://frandiox.github.io/OnsenUI-Memo-App/>

All of the style and design is easily customizable!

Approach and code

First element and welcoming page

In Onsen UI we should have **only one main element** that will decide the [pattern](#) of our application in order to manage several pages. We could use a sliding menu to change between pages, a tab bar and so on. We will use the most common one, the navigation pattern, what will allow us to have a parent-child relationship between pages (we have a stack of pages and we can navigate from one to another). Therefore, we write our ons-navigator as the main element in our `index.html`:

We declare the name of our navigator with `var="myNavigator"`, so we can refer to it later on as a global variable. Inside the navigator we will display a welcoming page (with `ons-page`) where we show the title of the app with an `ons-toolbar`.

Tip #1 ~ Notice that, even if we specify `class="center"` for the title, the real style behavior is determined by the OS where the application is run. On iOS it will be displayed in the center, but on Android it's left-aligned.. To fix the title in the center we have specify `fixed-style` in the `ons-toolbar`

We could display a welcoming image as well, but let's show a simple icon using `ons-icon` instead. Onsen UI allow to display icons from [Ionicons](#) and [Fontawesome](#) just like part of the font, so it will be displayed in any device. Our last element in this welcoming page will be a start button that will take us to the actual application. We use an `ons-button` for that and choose one of the possible [modifiers](#) to change its appearance. When the user clicks on this button, the navigator will change the current page to the next one. We use the method `resetToPage(...)` (instead of `pushPage(...)`) just to remove all the previous pages from the stack and start from zero in the page we specify (`slidingmenu.html`). With this we will prevent to return to the welcoming page by pressing any *"back"* button.

Menu

Once we have our first element and our first page, let's make the next pages that will be displayed after the welcoming page. The next one will be the main page where we will see all the tasks that our application stores. However, since we want to have a sliding menu in this page, we need to declare this element as the parent of the main page, what means that we will call this sliding menu first. To do so, let's create a `slidingmenu.html` page containing our actual sliding menu. Onsen UI allows you to create a new HTML page inside the `index.html` itself by using

templates, what is quite handy. It works exactly in the same way but, if our pages are small, we can have them all in the same place.

Our `ons-sliding-menu` will take the content of the menu itself from `menu-page="menu.html"`, and this sliding menu will be displayed over `main-page="memo.html"`. We set `swipeable="false"` to avoid a conflict of sliding with our future *carousel*, but you can change this in anyway you need.

Let's see now the content of our `menu.html`:

Static `ons-list-item`:

Dynamic `ons-list-item` (with AngularJS' `ng-repeat`):

Once again we use an `ons-template`, so this page will be located in `index.html`. Inside the `ons-toolbar` we just set the title of the menu and include a button to close it. This menu will allow to choose a certain view of our tasks based on task's category, and this category list will be accessed from `ng-controller="categoryController"`. There are 3 static views over the tasks: show all of them, show only the completed tasks and show the ones without a specified category. Of course, this is just an example, you can change them or even delete them without problem.

After these static views we define a repeatable `ons-list-item` (by using AngularJS' `ng-repeat` in the last `ons-item-list` in this code) that will iterate over our category list and display every category in the menu. Since we also want to display the number of tasks that each category has, we need to have that information in our `categoryController` as well. For instance, we have a counter variable called `countAll` that stores the total number of tasks in the memo app. Since this variable is stored in a service (`memoService`) and can be modified by several controllers (for example, when a task is created or deleted), we need to set a listener to this variable in `categoryController`, so that AngularJS will update its value automatically to the view everytime it is modified:

Changes in the view will be held in `categoryController`'s method `setViewRefresh(...)`.

Main page

So far so good, let's look at the content of the main page, `memo.html`, where the actual task list will be displayed:

Lists of carousel tasks:

For every task, level 0 of the carousel with 3 different actions (buttons):

For every task, level 1 of the carousel with the basic information:

Since this is a big and important page, it is preferable to set it in **its own file** instead of using `ons-template` inside `index.html`. In general, we will only include there relatively small things like the *sliding menu* and the *popover*. Either way, once again we use an `ons-toolbar` to set the title

and a link to a new page (`additems.html`, explained later), but also a button to open the *sliding menu*.

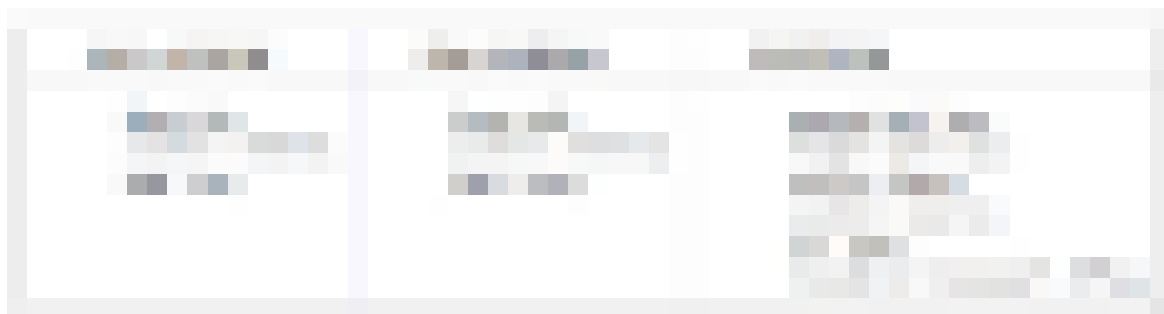
Tip #2 ~ Here we have a little trick to make the *sliding menu* swipeable only when it's open and not when it's closed, so it doesn't conflict with the *carousels*. We prepare two events to change the behavior of the sliding menu like this:

Besides, we will show a message in case there is no task displayed with the current view by using `ng-hide`. And finally, the bulk of this page, the dynamic `ons-list` where each item is an `ons-carousel`. We use `ng-repeat` to iterate over the task list that `memoController` provides, and each of these items will be defined as a `ons-carousel` of two levels where the level 1 will be a summary of the task and the level 0 (to go from 1 to 0 we have to swipe to the right) will contain buttons to perform actions over the selected item.

Tip #3 ~ The buttons in the level 0 of a carousel are displayed vertically and horizontally centered no matter how many there are.



This is a CSS trick:



We can delete it by just triggering a function in our `memoController` that removes this item from the task list, modify it in a similar way, or mark it as completed. For the complete action, in addition to trigger the corresponding function behind the scenes (to check a task as completed in our objects), we want to make it fancy and **restore the corresponding carousel** to its level 1 (the task summary) and show an icon to know that it is now completed:

Tip #4 ~ In order to *restore a carousel* to its original position we need to dynamically set a distinct name for every carousel in our list, so we can refer to it later on and change its level.

`var="{{'carousel.id' + $index}}"` does exactly the trick, naming every new item as *carousel.idX*, where *X* is the item's index in the `ons-list`. Notice that if we name these items in a static way, i.e. `var="myCarousel"`, every item in `ng-repeat` will overwrite the previous one, so in the end only the last one will be

accessible by that name. In order to call the function `setActiveCarouselItemIndex(1)` over the clicked item we need to use bracket notation instead of dot notation: `carousel['id'+$index].setActiveCarouselItemIndex(1);`

Show details & Modify task

Let's address now to the show-details/modify task action. We have a new page for it, `itemdetails.html`, with its own controller:

This new page is called from the main page by `myNavigator.pushPage(...)`, what basically adds a new page to the stack in our `ons-navigator`. Later on we can come back to the previous page in the stack by using `myNavigator.popPage()` or an `ons-back-button`.

In this page we display all the content of a specified task inside HTML inputs, so in addition to see the content we can modify it. It is necessary to bind every input to an `ng-model`, so we can access to the new values from the controller. For instance, the name of a task is bound to `ng-model="item_name"`, so we can access it with `$scope.item_name` from `detailsController`.

Add new task and popOver



Finally, the page to add a new task to the list is called `addItem.html`, and is the simplest one. Exactly like the previous page, we bind the HTML inputs to `ng-model`'s, and in our `addItemController` we just add the new task to the task list in `memoService` after checking that all the inputs are valid. In the case that, for example, the name input is empty, we want to alert the user about it and avoid creating a new task. To do that we create an `ons-popover` in our `index.html`:

Quite simple. And now we need to display it from our controllers (`addItemController` and `detailsController`, when saving a modification):

This last line will show the popover right on the HTML element `'#item-name'`, we just need to control when to display it (only when the input is empty).

Final notes

Since this application is just an example of Onsen UI and AngularJS it does not support persistent data with *window.localStorage*, *Web SQL*, *IndexedDB*, or anything else, though it is possible to implement it. Instead, we add some example data when the app starts, so we can test it easily.

Conclusion

We have reviewed so far how to build a not-so-trivial hybrid mobile application with a quite fancy User Interface using Onsen UI as well as several tips that could become handy in some situations. AngularJS is not compulsory but recommended since it works very well with OnsenUI and makes things a lot easier. As mentioned before, [all of this code is on Github](#), so you can change it and try any new feature that comes to your mind.

And that's pretty much it, hybrid mobile applications are certainly worth a try, especially because tools like Onsen UI keep everything simple. If any doubt comes up, don't hesitate to ask anything in the comments or on Stack Overflow under the tag [onsen-ui](#). More tutorials will be coming soon!

LokiJS: in-memory NoSQL datastore for Cordova apps



[Fran Dios](#)

[Jul 8, 2015](#) · 2 min read



[LokiJS](#) is a lightweight JavaScript in-memory database made by [Techfort](#) that provides a document-oriented alternative to the already well known SQLite. For many applications a NoSQL approach could be more preferable than relational data when working with complex object stores. LokiJS gives us the chance to do this in a very efficient way.

LokiJS is intended to be used as an in-memory database with the possibility of persisting the data. It is compatible with Node.js (filesystem) and browsers, which means that it can be also used in our hybrid mobile Cordova apps (*localStorage* and *indexedDB*).

A block of code is worth a thousand words:

Furthermore, LokiJS supports field indexing for faster document access and provides a built-in *DynamicView* class that also enables to use indexes on data subsets for even faster performance. *DynamicViews* work as queries that are recomputed in the background when data changes, so once a view is created you never have to worry about recalling or refreshing data, it is always updated and available.

It is also available for AngularJS through the [lokijs](#) module:

Its low footprint and high performance, along with a bunch of other features, make it worth giving a try! You can follow its [Github repo here](#) and learn more about it [here](#). We hope you find this tool useful for your future hybrid apps!

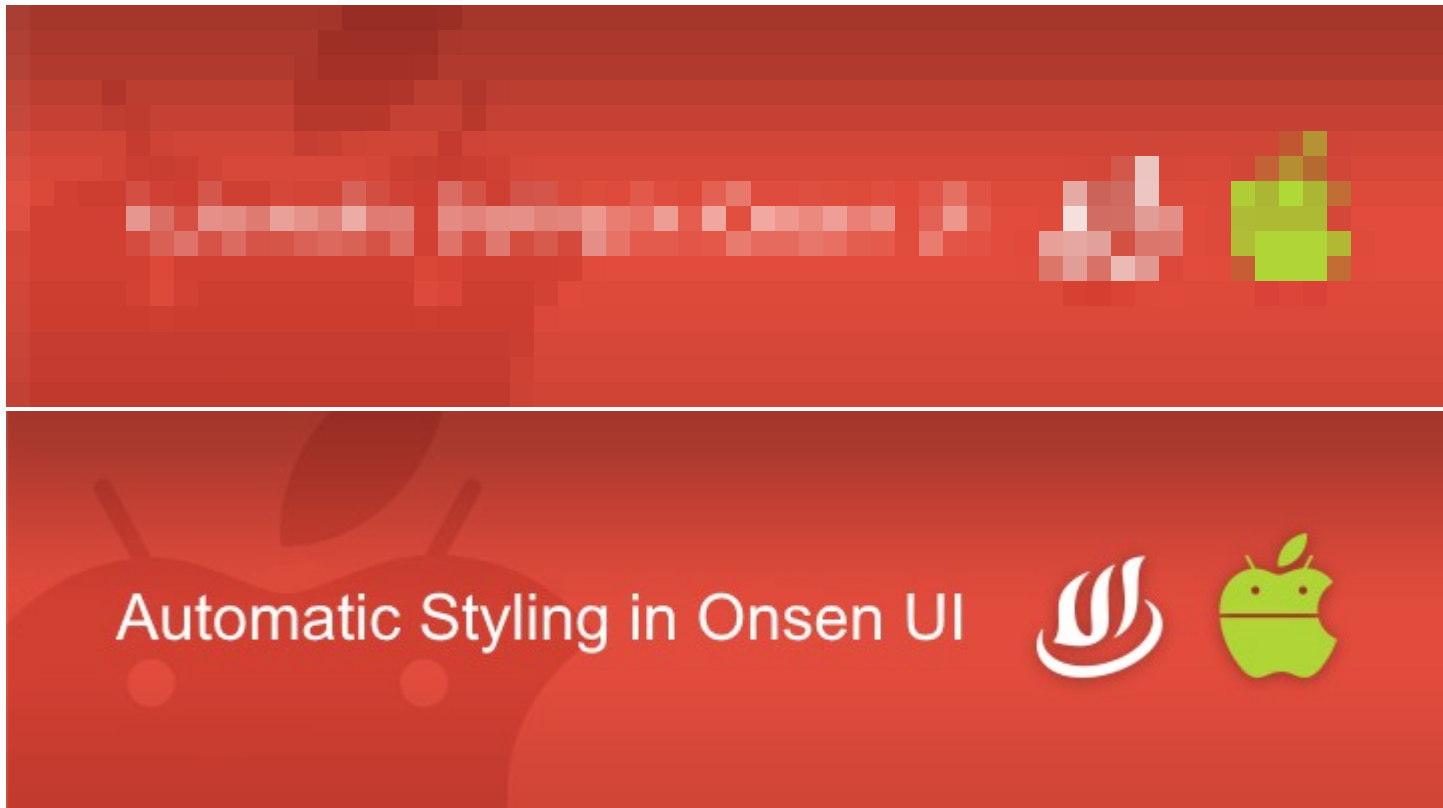
Originally published at [onsen.io](#) on July 8, 2015.

Style your app automatically with Onsen UI



[Fran Dios](#)

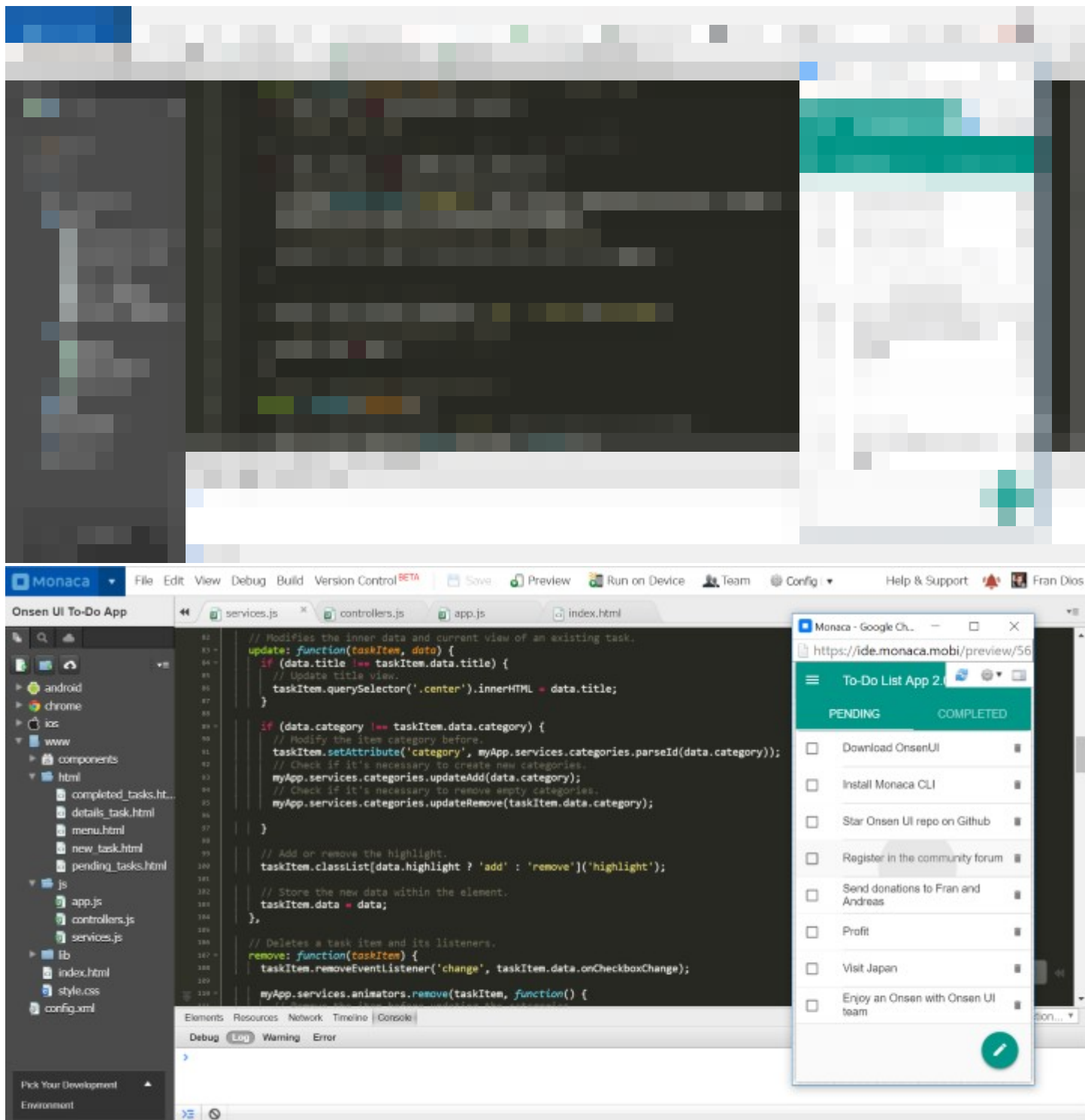
[Mar 7, 2016](#) · 11 min read



As we mentioned in a [previous post](#), we just released a new killer feature called *automatic styling* for [Onsen UI 2.0](#). This feature allows you to style your app for both iOS and Android automatically with little to no effort.

In this post we will explain how to make a sample To-Do app step by step using as many Onsen UI 2.0 features as we can: implementation in Vanilla JavaScript (framework agnostic), automatic styling, notification promises and combination of Navigator + Tabbar + Splitter (new sliding menu) in the same app.

We will develop and test the app with [Monaca](#) since we can easily display the app as iOS or Android in the browser by just changing the device type in the preview settings. And most importantly, we can see (and debug) the final result in any real device with Monaca Debugger in no time.



For local development, you can always use the device emulator included in Google Chrome's DevTools and switch between iPhone and Nexus.

To-Do List app

We will make a simple To-Do List app. This app is perfect to show what we can do with the new beta in Vanilla JavaScript. First, you can play a bit with the app in the following iframes. You can also open [this app link](#) in your device to see the automatic styling in action.

Onsen UI Memo App 2.0

[Edit description](#)

frandiox.github.io

Onsen UI Memo App 2.0

[Edit description](#)

frandiox.github.io

As usual, the [code is available on Github](#). It's full of explanatory comments so it should be easy to follow!

As mentioned above, this sample app is implemented in Vanilla JavaScript without any other library or framework, so it could be a good reference when you start your own app. There are many ways to make a To-Do List and here we just show one of them. Perhaps not the best, perhaps not the worst, but we believe it could be instructive. Please choose your own style when you start your projects.

Anyhow, let's get started with the tutorial.

App structure: HTML

We will put all the navigation components (`ons-navigator`, `ons-splitter`, `ons-tabbar`) in `index.html` using `ons-template` so we can easily manipulate the flow in the same file. The pages which this components load will be located in `html` directory as separate files.

What we want

First issue we need to think about is how we want to structure the app to fit our needs. Here, we want to have:

A list of "pending tasks" and a list of "completed tasks" separated in a tabbar.

A list of categories that we can use to filter the tasks. We want this to be located in a menu to put it out of the normal view.

A page where we can add new tasks. In this page we shouldn't be able to see either category list nor task lists.

A page where we can modify existing tasks and see their details. Same as the previous one, we don't want to see categories or other tasks here.

What we use

Then, let's choose the order of the components wisely so afterwards we don't need to do extra job to make it work:

- Tabbar

Starting from the lowest level navigation component, we want an `ons-tabbar` to separate our pending list page from our completed list page in two tabs.

- Splitter (menu)

We need a menu common to these two lists. Therefore, `ons-splitter` component should be the parent of the previous `ons-tabbar`. Otherwise we would need to create two splitters, one for each list page.

- Navigator

Now that we have the basic functionality covered with the previous two components, we want to hide all of this and change the view to a “new task” or “modify task” page. We can achieve this by using an `ons-navigator` as the parent component of the other two. This navigator will change all its content when pushing a new page, hence hiding the menu and the tabs when we create/modify a task.

```
<ons-navigator id="myNavigator" page="splitter.html">
</ons-navigator>
```

Notice that this navigator is not wrapped inside `ons-template` since it's the main component of the app.

App structure: JS

In this sample app we have structured our JS code in a specific way. We don't want to mix our markup with our logic so we avoid to include `onclick` or `onchange` attributes in our elements. We will define some initialization function (controller) for every page and set up its functionality from there. This is called *[unobtrusive JavaScript](#)*. Also, we will store all our logic in a `window.myApp` object instead of using global variables for everything we need to store. This way we don't pollute the global scope (except for `myApp` object itself). As a side note, we are using `querySelector` all along the app just to be homogeneous instead of using `getElementById`, `getElementsByClass`, etc. Thus, our app will be organised in the following way:

- `app.js`

Contains the basic app setup: call the corresponding initialization controllers when the pages are ready and fills the task list with initial data. This is the important part:

We listen for page `init` events and trigger the corresponding page initialization function that we store in `myApp.controllers`.

- `controllers.js`

As already mentioned, here we store all the initialization functions. For example, let's have a look to “New Task Page Controller”:

As you can see there, we just add some functionality to the “Save” button in New Task page. We have included `component` attribute just to be able to find these elements later on and add their functionality.

Specifically, here we create a new task item on click using the `tasks` service. In case the title input is left blank, we show an alert and abort the item creation.

- `services.js`

Here comes the dense part. These are the functions that do the hard work in our app. We have organised them in “services” although behind the scenes this is just a JavaScript object with sub-objects:

For instance, let’s see how we can create a new task item when `myApp.services.tasks.create(...)` is called. This is the first part:

Here we just create a new `ons-list-item` with the given `data` object that contains title, category name, description, etc. If we could just use [ES2015 template literals](#) for this task it would be much easier, wouldn’t it?

Since in Onsen UI 2.0 every component inherits from `HTMLElement` you could create them using `document.createElement('ons-list-item')`, for example. This would create a `<ons-list-item></ons-list-item>` node without any custom attribute and it would be compiled as is. However, in general we need to create elements directly with attributes and content so the component compilation actually consider this information. To do this we can create an empty `div` element (`var template` in our case) and insert our custom component inside with the [innerHTML property](#). After this you just need to extract the content of this `div` element (`template.firstChild` for us).

Moreover, in this app we are storing the data of each task within the `ons-list-item` itself rather than in internal arrays (`taskItem.data = data;`). This is a very simple and handy approach possible in Vanilla JavaScript or jQuery, but in case you use a framework like AngularJS or React.js this could be troublesome. When these frameworks copy/move elements the inner properties could disappear, so it is advisable to store the data in real arrays and let the framework manage the rest.

In this list item template we set a `category` attribute that we will use later on to filter tasks depending on their categories. This way, we will be able to find the ones that don’t match our selected category with a simple query and set `item.display = 'none'` to those we want to hide. In case you don’t know what `class="left"` and the other classes mean, please read this [previous post](#) to understand how they work. We will go into details about the `ons-icon` that appears in this template in the following section. For now, let’s have a look at the next part of this service:

This adds some functionality to complete or undo the task. We are adding a listener for checkbox’s `change` event to handle this. Whenever the checkbox changes, the list item will notice and act consequently. We could just use `onClick` as well but for the sake of showing more possibilities we chose this other approach. As you can see in this code, our `completion` functionality basically changes the item from “pending” list to “completed” list. This makes the app super simple, without the need of moving internal data in arrays. Once again, if you use complex frameworks you should leave to them the DOM manipulation. But this is just Vanilla :)

As for the `animators` service, don’t worry too much, it simply adds custom CSS classes with keyframes and timeouts and finally run the callback we pass as parameter when the animation

finishes. Notice that these animations are not part of Onsen UI, we just made them for this sample app so we won't stop here (you can check the Github repo instead).

Remember that, since we added an event listener to the list item, we will need to remove it when we delete the element to prevent memory leaks. That's why we saved the handler in a named property `taskItem.data.onCheckboxChange`, so we can do `taskItem.removeEventListener('change', taskItem.data.onCheckboxChange)` at any time. This is what `myApp.services.tasks.remove(taskItem);` exactly does, and we want it to happen when we click on the right part of the list item (where the trash icon is located).

Also, we want to push a new page when we click on the body of the task so we can see its information and modify it. In order to do this, we just take our navigator and perform a `pushPage` passing a reference to the item itself rather than an index. This simplifies the app avoiding the need for internal arrays, as already mentioned. That's the reason why we stored `taskItem.data` within the element before. Again, in Angular or other framework you'd need to pass an index instead.

And finally, we append this item to the pending tasks list:

If the second argument of [`insertBefore`](#) is `null`, the item is appended to the end of the list. This way we insert urgent tasks at the top and non urgent tasks at the bottom.

Automatic Styling

So far we have seen how to structure the app in an organised way, separating the view from the logic. Let's see now how to make a good use of the automatic styling tools.

In general, the styling will be automatic by default unless you call `ons.disableAutoStyling()` at the beginning of your app or specify `disable-auto-styling` attribute in those elements you want to style manually. This means that most of your app will get Material Design styles automatically when running on Android. However, there will be parts in the app where this is not possible. For example, in order to save some new information in iOS we may have a large button at the bottom or a toolbar button at the top, while in Android we'd possibly have a floating action button instead. We thought it would be better to leave this decisions to the developer rather than assuming them by ourselves. However, we released a bunch of tools to make this task quite easy. Let's have a look at different places of this app where we use these tools.

- Tabbar position

We defined our tabbar element in `index.html`. We want it to be displayed at the bottom for iOS and at the top for Android. So far we could only specify `position="top"` or `position="bottom"`, but from now on we can also use `position="auto"`:

```
<ons-tabbar id="myTabbar" position="auto">
  ...
</ons-tabbar>
```


This will display the tabbar on top or bottom depending on the platform. The default behavior if you don't specify any position is still `position="bottom"`, so make sure you specify `position="auto"` if you want this behavior.

- `ons-if`

There will be parts of the layout that only make sense on iOS and others only on Android. That's why we created `ons-if` component. This element accepts an attribute `platform` with `ios`, `android`, `windows` and `other` as possible values. Apart from that, it also accepts `orientation` attribute so you can display elements when the device is in `landscape` or `portrait` mode. For example, in order to use different button elements to push pages in `index.html`, we do the following:

Notice that both of them have the same `component="button/new-task"` attribute since they will do exactly the same. However, only one of them will be present at the same time since the other will be removed by `ons-if`.

Another example of `ons-if` can be seen in the details page:

These inputs are automatically filled with the task data. In Material Design it's possible to see the floating label explaining what's the content of the input but iOS does not display such information. Therefore, we want to display a static label on the left of the `ons-list-item` but we want this left part to be visible only on non Android platforms. To accomplish this we just use `ons-if` as the left part of the items with `class="left"`. This way, Android won't have a left part on its list items right as we want.

- `ons.platform`

`ons.platform` contains a bunch of methods to check the current platform. Two of them are `isAndroid()` and `isIOS()`, which return a boolean value. For more details check out the [reference page](#). As seen in the previous section, we can use this utilities to make decisions in JavaScript like choosing an icon or another for the template:

```
... + '<ons-icon ... icon="' + (ons.platform.isAndroid() ? 'md-delete' : 'ion-  
ios-trash-outline') + '></ons-icon>' + ...
```

- Animations

The default animations for `ons-navigator` now changes depending on the platform. The default pushing page animation for iOS remains the same (slide) but for Android we have now some kind of fade + lift animation. Also the basic lift animation slightly differs in Android since it creates a black mask behind the pushed page. If you want to fix an animation for every platform you need to specify `{ animation: ... }` as options in the navigator methods.

In this app we also change tabbar animation to be “slide” only on Android. To do this we set the `animation` attribute from the initialization controller:

```
document.querySelector('#myTabbar').setAttribute('animation',  
ons.platform.isAndroid() ? 'slide' : 'none');
```


Promises for async methods

Every asynchronous method now returns a promise that resolves to the element that the method is pushing, popping, etc. In this app we use `ons.notification` to easily create dialogs and handle the user input with the new promises:

Conclusion

The automatic styling feature truly makes possible to write a hybrid app and run it on both iOS and Android (and perhaps Windows Phone at some point, why not). It will likely make your app development process much smoother. And that's what we want here at Onsen UI, make your hard development easy, so we hope you like this feature and enjoy using it. In case you find any issue with this, please report it in the [community forum](#) (questions) or in our [Github repo](#) (bugs). And of course, leave comments if you have anything to add :)

If you liked what you've seen here, please don't forget to [leave us a star on GitHub](#). Happy coding!

Migrating from Other Platforms

Monaca is built on top of Cordova. Therefore, any Cordova-like projects can be imported into Monaca. This page provides guides on how to migrate your existing projects from various platforms into Monaca. In addition to the migration procedure, each guide assists you to quickly get started with Monaca from when you start importing the existing project to building an app from it.

Welcome to Monaca and enjoy the ride!

- [Key Points](#)
- [Migrating from Ionic](#)
- [Migrating from Vue](#)
- [Migrating from Angular](#)
- [Migrating from React](#)
- [Migrating from Telerik Platform](#)
- [Migrating from PhoneGap Platform](#)