Rocket.Chat Developer

Developer Guides

- Contributors instant start on Gitpod — /guides/developer/contributors-instant-start-on-gitpod
- Quick Start on Linux — /guides/developer/quick-start
- Mobile apps — /guides/developer/mobile-apps
- Developing on Windows 10 — /guides/developer/developing-on-windows-10

Contributors instant start on Gitpod

If you are contributing to Rocket.Chat and will likely be working less than 50 hours a month on it, the quickest way to get started is via Gitpod.

Rocket.Chat Everywhere Development Environment on Gitpod

Gitpod runs a shared environment SaaS for developers working on open source projects. Please be respectful of other developers' need and support Gitpod on their commercial side if you are able to.

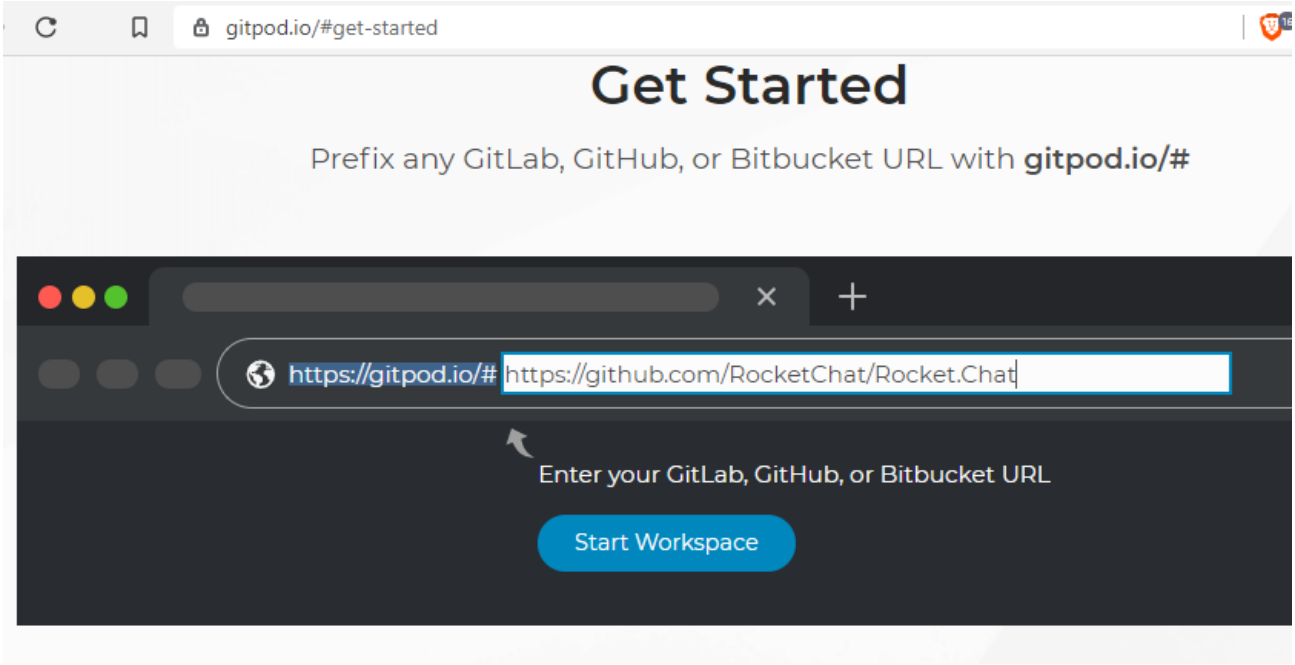
Gitpod hosts the entire development environment and you will be able to contribute to Rocket.Chat wherever and whenever you have access to a browser; even from Internet Cafes and Chromebooks.

Step by step

Go to https://gitpod.io and enter the Rocket.Chat Github project URL

https://github.com/RocketChat/Rocket.Chat

(you can also supply your Github fork of Rocket.Chat here)



Start your workspace, linking it to your Github account if necessary.

A workspace will be created and loaded with familiar Visual Studio Code environment.

Next, start a terminal in your workspace. And install meteor.

```
gitpod /workspace/Rocket.Chat $ curl https://install.meteor.com/ | sh
```

There will be warning messages, you do not have and do not need sudo (root access) on Gitpod. Next, add the newly installed meteor to your path.

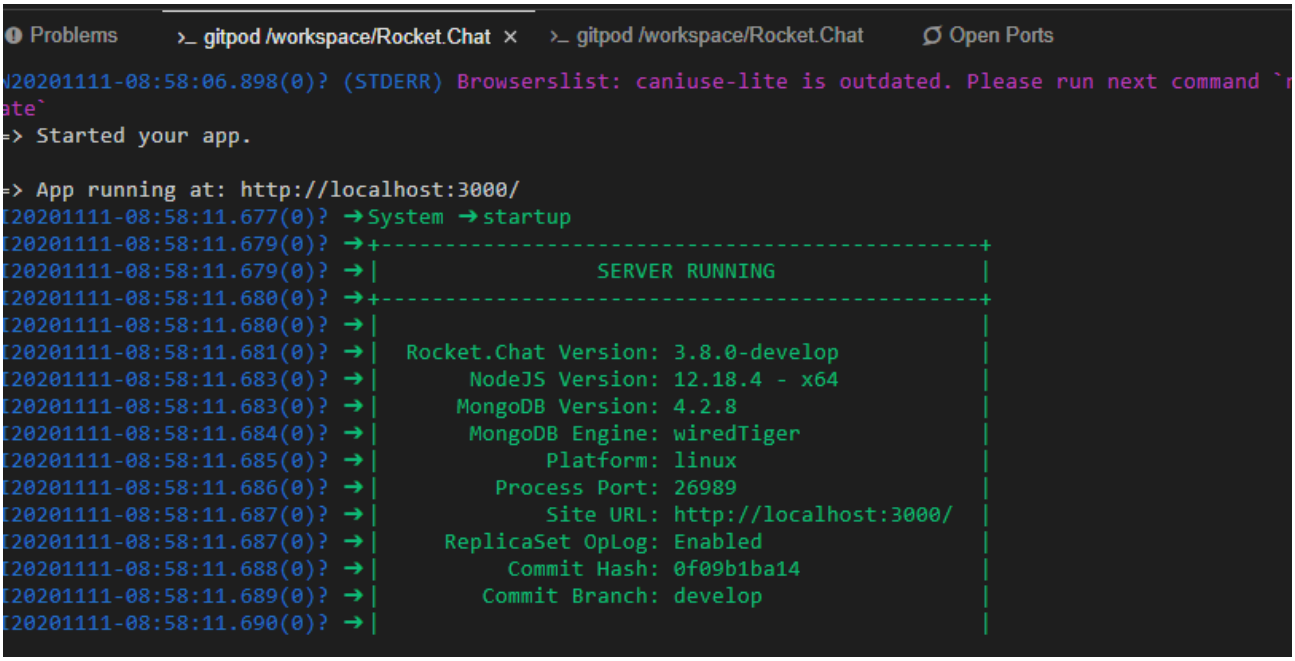
```
gitpod /workspace/Rocket.Chat $ PATH=$PATH:/home/gitpod/.meteor
```

Finally, install the node dependencies and start your server.

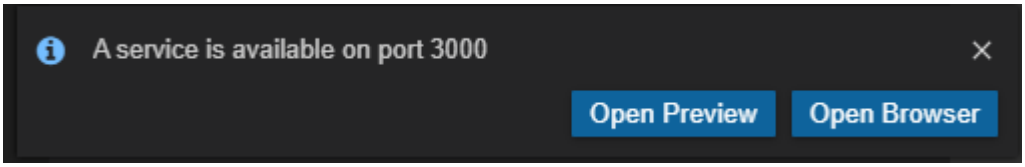
```
gitpod /workspace/Rocket.Chat $ meteor npm i
```

```
gitpod /workspace/Rocket.Chat $ meteor npm start
```

After a few minutes, your development environment should be up and running.



Once the server starts, you will see a popup window indicating that your server is running on port 3000. Click the **Open Browser** button to view and interact with your Rocket.Chat server instance.



You can modify the code in Visual Studio Code and see the changes immediately on the server instance thanks to hot code reload.

Start contributing to Rocket.Chat!

Quick Start on Linux

You can run Rocket.Chat for development on a Linux machine or VM. The following instruction has been tested on a new Ubuntu 18.04 LTS installation. Try to find and use a *NEW Ubuntu server installation* with no other un-necessary software installed, not a "Desktop" or "Client" installation.

DO NOT use a system where you already have nodeJS installed to avoid problems.

During the build, memory usage will be nearly 8G, this is the minimum level of RAM recommended for development workstations. (If you are not doing any development, and just deploying a Rocket.Chat server - the RAM required can be as low as 1G.)

IMPORTANT: Note that there is no need to install mongo, nodejs, or npm on the base operating system. If you have any of these already installed; start over, or use another CLEAN system.

Development should be performed under a regular user account (not `root`) on Linux. There should be no need to run `sudo` at all. Running `sudo`, even once, during the installation – may mess up file permissions in irreversible manner.

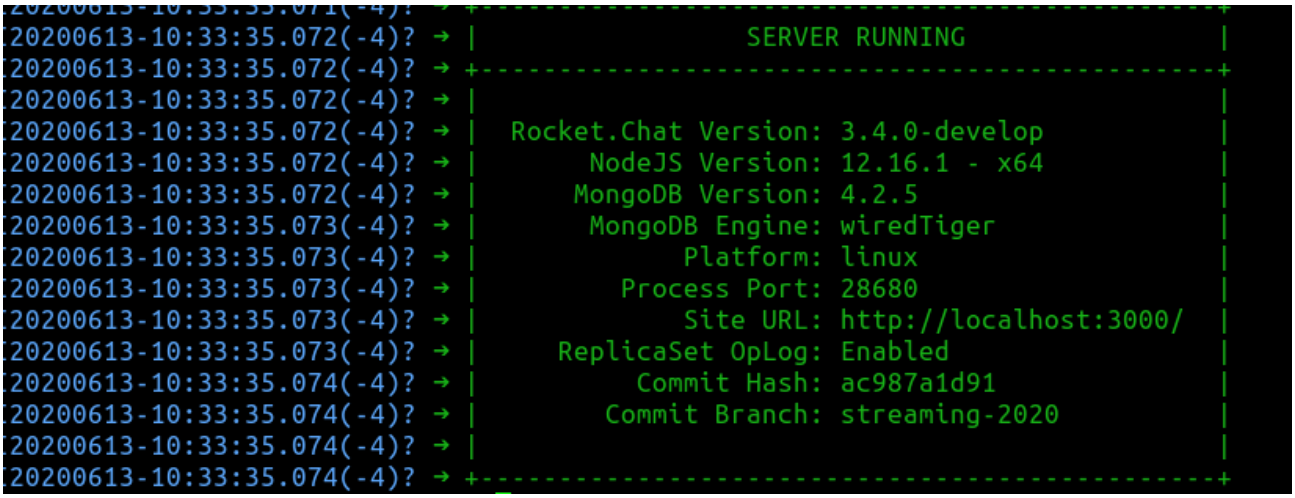
You may notice build WARNINGs related to *peer dependencies* or other transitive dependencies. They are typically safe to ignore unless you are coding the feature or modules that require them.

1. Install tools required
   `sudo apt install g++ build-essential git curl python-minimal`
2. Install meteor
   `curl https://install.meteor.com/ | sh`
   (Under some circumstances, you may need to install a specific (older) release of Meteor instead of the latest, always check the `.meteor/release` file of the Github code repository to determine if you need to do this before you install meteor)
   There is no need to install `node` or `npm`, as meteor already includes them. Verify by:
   `meteor node -v`
   `meteor npm -v`
3. Get rocket.chat code

`git clone https://github.com/RocketChat/Rocket.Chat.git`
(you may want to fork the code on Github first, and then clone your fork)

4. Install modules
   `cd Rocket.Chat`
   `meteor npm install`
5. Start building (the first build can *take 10 or more minutes*, and you may see various warnings or minor errors -- please be patient; subsequent dev builds after the first will be 5 minutes or less)
   `meteor npm start`

When the server is ready, you will see a box with "Server Running" title:



This means that a Rocket.Chat server is running from your computer. To access the server, navigate to

`http://localhost:3000`

Other references:

- Git
- Meteor

Editing Rocket.Chat Files

Editing files is relatively simple. After you run `git clone`, the files from the repository are saved on your computer. You can go to the cloned repository folder and edit or add files to

Rocket.Chat. When you make changes to Rocket.Chat the server will automatically rebuild.

Sometimes changes can shut down the server, if that happens just run `meteor npm start` again.

The Rocket.Chat code base is very large. You may need to increase this system parameter on your operating system for the files-change watcher to operate efficiently.

On Windows

Using Windows to develop is not recommended at this time, instead is better to create a Linux virtual machine and follow the steps mentioned above, but if for some reason you really need to build Rocket.Chat on Windows you can find a community supported guide over ~~here~~.

We look forward to the official release of Windows Subsystem for Linux 2 (WSL 2) when Rocket.Chat development on Linux may finally become viable.

See Also

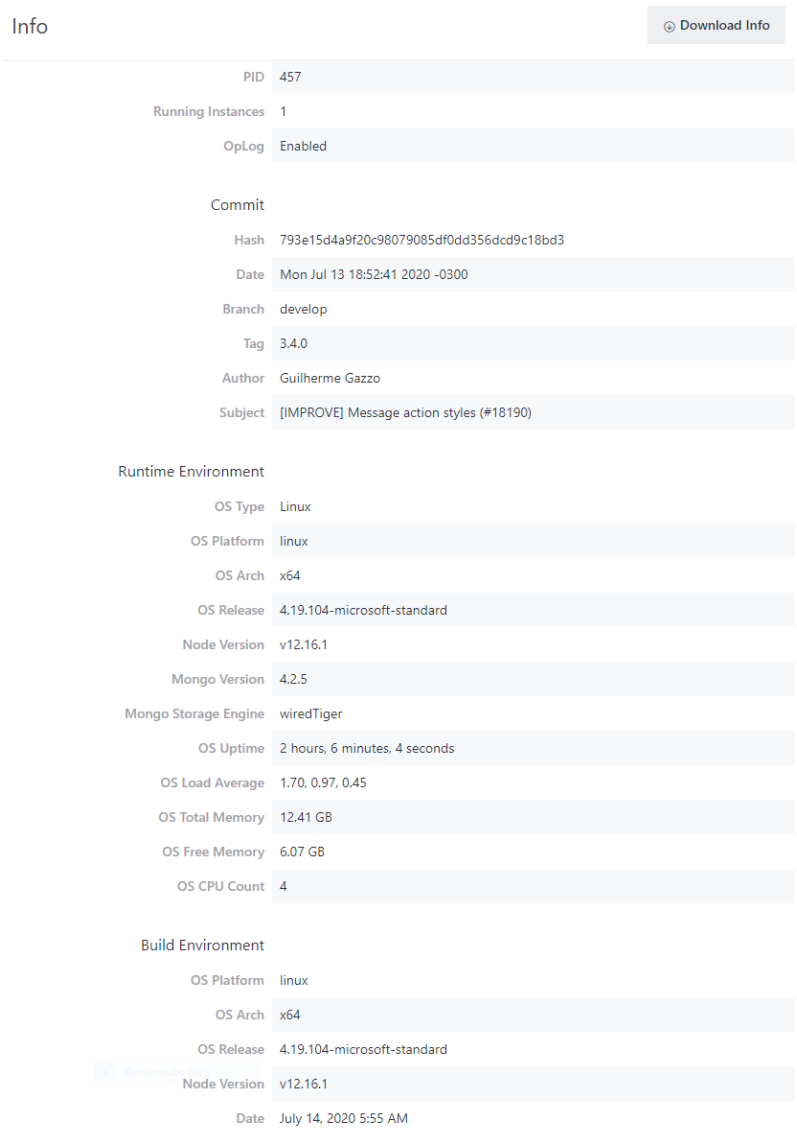
- Supporting SSL for Mobile Apps
- Development Troubleshooting
- ~~Deployment Methods~~

Developing on Windows 10

Microsoft finally released Windows Subsystem for Linux 2 (WSL 2) in June of 2020. Before this time, development of large and complex NodeJS based servers/full-stack applications such as Rocket.Chat on Windows is close to impossible.

WSL 2 is a complete architectural overhaul of Linux on Windows, installing a full genuine Linux kernel (built by Microsoft) alongside the classic Windows kernel. The Linux kernel and Windows kernel can now share system resources, such as memory and CPU, at a granularity not previously possible. It also includes major performance optimization on cross-subsystems file sharing, boot, and other developer-relevant areas.

You must be using **Windows 10, version 2004 or later** to take advantage of WSL 2, and to setup Rocket.Chat development.



Before you start

The following are prerequisites for developing Rocket.Chat on Windows 10:

1. Make sure you have **Windows 10, version 2004** or later
2. Install and configure **WSL 2** by following Microsoft documentation, making sure to select **Ubuntu 20.04 LTS** distribution as your choice of Linux
3. Download and install the latest Linux Kernel Updates

Machine requirement

Building Rocket.Chat code requires a minimum of 8 GB of RAM memory on the Linux subsystem. (If you are not doing any development, and just deploying a Rocket.Chat server - the RAM required can be as low as 1GB.) On version 2004, about 4 GB of RAM appears to be reserved for the Windows subsystem. You will need a Windows machine with the following minimal requirement to develop Rocket.Chat:

- 12 GB of RAM memory (16+ GB highly recommended)
- 4 or more cores on CPU (at least 3 GHz boosted, 4.2 GHz or higher recommended)
- 80 GB of available fast SSD storage (PCIe 4.0 NVMe SSD recommended)

Setting up development environment

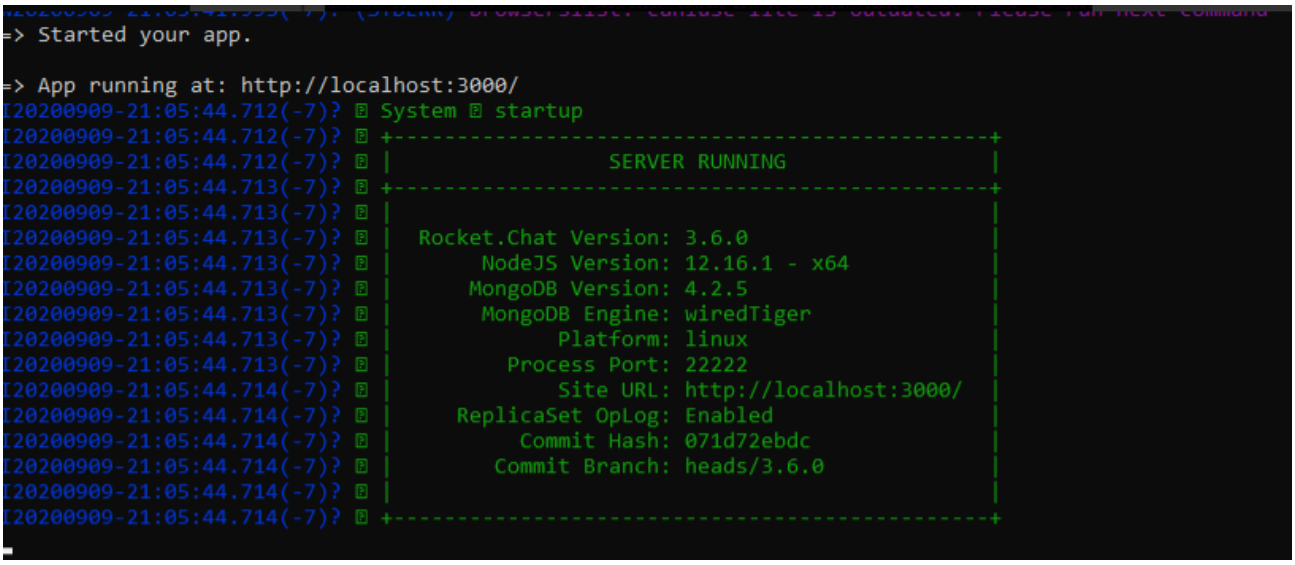
> **IMPORTANT**: Note that there is no need to install mongo, nodejs, or npm separately
>
> **NOTE:** Development should be performed under a regular user account, not Administrator.
>
> **NOTE:** During build, you may notice *WARNING* related to *peer dependencies* or other transitive dependencies. They are typically safe to ignore unless you are coding the feature or modules that require them.

1. Open a **WSL 2 shell** (not Powershell). Update Linux
   `sudo apt-get update sudo apt-get dist-upgrade`
2. Install tools required
   `sudo apt-get install build-essential git curl python2-minimal`
3. Install meteor
   `curl https://install.meteor.com/ | sh`
   (Under some circumstances, you may need to install a specific (older) release of Meteor instead of the latest, always check the `.meteor/release` file of the Github code repository to determine if you need to do this before you install meteor)
   There is no need to install `node` or `npm`, as meteor already includes them. Verify by:
   `meteor node -v`

`meteor npm -v`

4. Make sure you are on the WSL 2 filesystem, `pwd` should return `/home/yourusername`
   Get rocket.chat code:
   `git clone https://github.com/RocketChat/Rocket.Chat.git`
   (you may want to fork the code on Github first, and then clone your fork)
5. Install modules
   `cd Rocket.Chat`
   `meteor npm install`
6. Start building (the first build can *take 10 or more minutes*, and you may see various warnings or minor errors – please be patient; subsequent dev builds after the first will be 5 minutes or less)
   `meteor npm start`

When the server is ready, you will see a box with "Server Running" title:



This means that a Rocket.Chat server is running from your computer. To access the server, use Chrome, Brave, or Firefox browser, and navigate to

`http://localhost:3000`

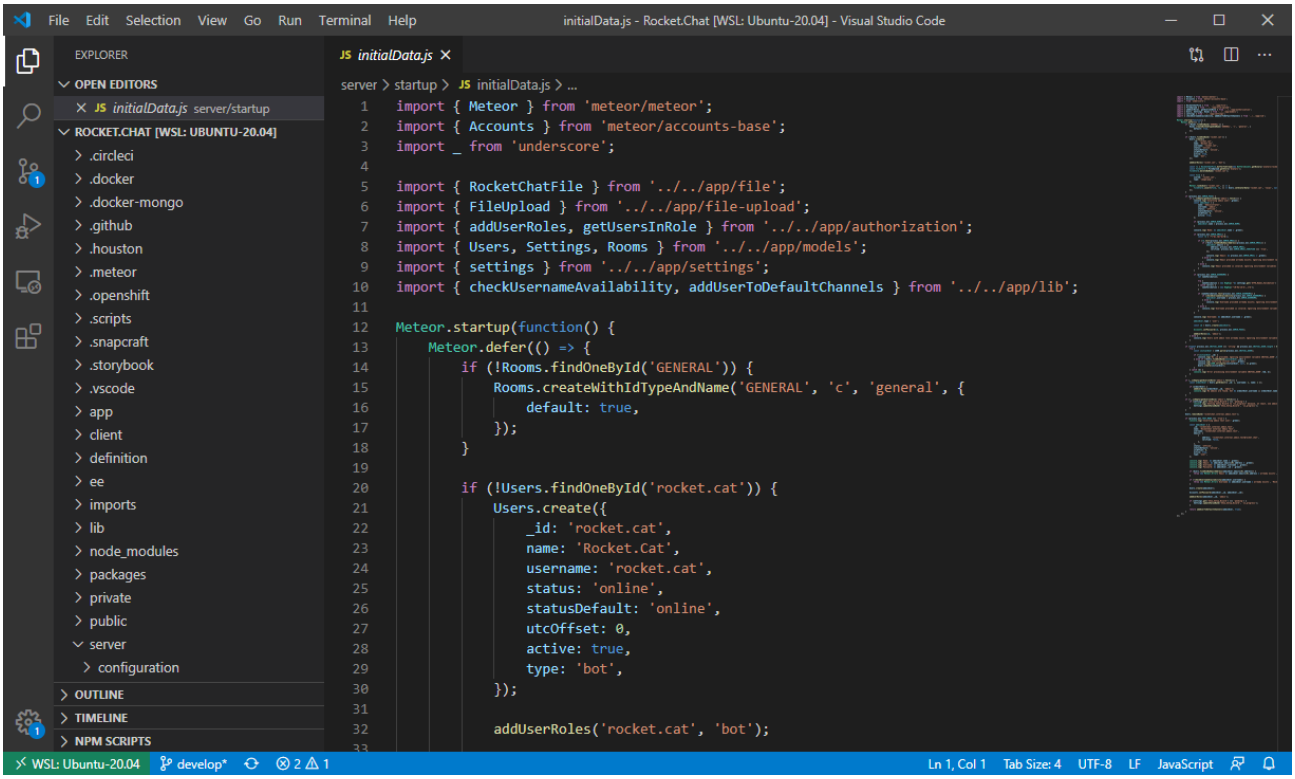
Other references:

- Git
- Meteor

Editing Rocket.Chat Files

On Windows 10, the best coding environment to use is Visual Studio Code. Install Visual Studio Code from Windows. Make sure you also install the VS Code extension named Remote - WSL Optionally install the Windows Terminal extension

Editing files is relatively simple. After you run `git clone`, the files from the repository are saved on your computer. You can go to the cloned repository folder and edit or add files to Rocket.Chat. From a WSL shell, you can start Visual Studio for Code using command `code .` .



When you make changes to Rocket.Chat the server will automatically rebuild.

Sometimes changes can shut down the server, if that happens just run `meteor npm start` again.

Development Workflow

1. Start working on an issue you're assigned to. If you're not assigned to any issue, find the issue with the highest priority you can work on, by relevant label and assign it to yourself. Priority is given by milestones. You should always check issues in the current milestone, then short-term, middle-term and long-term in that order.
2. You are responsible for the issue that you're assigned to. If you are not able to do it or you believe you won't make it to the selected milestone, talk to your team, lead or manager to have it reassigned or postponed.
3. Once your code has been deployed in the release-candidate environment, please verify that your changes work as intended. We have seen issues where bugs did not appear in development but showed in production (e.g. due to merge issues).

Choosing something to work on

Start working on things with the highest priority in the current milestone. The priority of items are defined under labels in the repository, but you are able to sort by priority.

After sorting by priority, choose something that you're able to tackle and falls under your responsibility. Take a good look on our labels and see which ones apply for your position.

To filter very precisely, you could filter all issues for:

- Milestone: Whichever is the current version
- Assignee: Unassigned
- Label: Your label of choice. For instance `feat: api`, `feat: integration / plugin`, `feat: webrtc`, `subj: ui/ux`, `subj: security`
- Sort by priority

If you're in doubt about what to work on, ask your team, lead or manager. They will be able to tell you.

Creating a Pull Request

Anytime you start to work on something different, make sure you create or switch to a branch specific to the feature you're working on. You can choose to create a Pull Request anytime during your development phase, just make sure you add the label `stat: in progress` while the PR is not ready for merge (and remember to remove the label when it is). When naming your Pull Request, please start the name with one the following tags for identifying changes:

- [NEW] for new features (eg.: `[NEW] WhiteBoard integration`)
- [FIX] for bug fixes. You should include the issue number(s) in parenthesis, whenever possible. (eg.: `[FIX] OTR timeout problems (#629, #2535)`)
- [BREAK] for giving proper attention to changes that will break previous versions of Rocket.Chat (eg.: `[BREAK] Change notification setting type from boolean to string`)

Getting your pull request reviewed, approved, and merged

There are a few rules to get your pull request accepted:

1. All checks have passed
2. Travis CI runs automatically when you push your pull request. If Travis fails, take a look at the reasons for failing. If it fails for no apparent reason, try running it again.
3. You must sign the Contributor License Agreement (CLA)
4. At least one team member must approve the Pull Request. If you don't know who to ask for an approval, let your team, lead or manager know you need one, and someone will be assigned as reviewer.

Finishing a release branch

When the state of the `release-candidate` branch is ready to become a real release, some actions need to be carried out. First, the release branch is merged into `master` (since every

commit on `master` is a new release by definition). Next, that commit on `master` must be tagged for easy future reference to this historical version. Finally, the changes made on the `release-candidate` branch need to be merged back into `develop`, so that future releases also contain these bug fixes.

The release process is as follows:

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff release-candidate
Merge made by recursive.
(Summary of changes)
$ git tag -a [version number]
```

To keep the changes made in the release branch, we need to merge those back into develop, though. In Git:

```
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff release-candidate
Merge made by recursive.
(Summary of changes)
```

To create the new `release-candidate` based on `develop`:

```
$ git checkout release-candidate
Switched to branch 'release-candidate'
$ git merge --no-ff develop
Merge made by recursive.
(Summary of changes)
```

Every team member should strive to thoroughly test the `release-candidate` branch, especially regarding issues they've worked on. Any bug-fixes applied to `release-candidate` should immediately be cherry-picked into `develop`.

You may read all about our branching model on A successful Git branching model, by Vincent Driessen

---

Hotfixes

Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version.

Hotfix branches are created from the `master` branch:

```
$ git checkout -b hotfix-[current-version].[sub-version] master
Switched to a new branch "hotfix-X.X.X"
```

Then, fix the bug and commit the fix. Later, merge the bugfix into `master` and also into `release-candidate` (and `develop` if the bugfix cannot wait for the release branch to be finished).

```
$ git commit -m "Fixed severe production problem"
[hotfix-X.X.X abc12d3] Fixed severe production problem
5 files changed, 32 insertions(+), 17 deletions(-)

$ git checkout master
Switched to branch 'master'
$ git merge --no-ff hotfix-X.X.X
Merge made by recursive.
(Summary of changes)
$ git tag -a [current-version].[sub-version]

$ git checkout release-candidate
Switched to branch 'release-candidate'
$ git merge --no-ff hotfix-X.X.X
Merge made by recursive.
(Summary of changes)
```

Error reporting

We use GitHub Issues as our error reporting tool. When creating a new issue, always state the version you're working on (write the version number, not `latest`). If you don't know the version you're working on, access `/api/info` on your instance. Include as much detailed information as you can, such as animated gifs and/or screenshots of error, code samples and reproduction steps.

Labelling issues

To allow for asynchronous issue handling, we use milestones and labels. Leads and product managers handle most of the scheduling into milestones. Labelling is a task for everyone.

Most issues will have labels for at least one of the following:

- Feat (`feat: accessibility` , `feat: embed` , `feat: livechat` , etc.)
- Subject (`subj: mobile` , `subj: security` , `subj: ui/ux` , etc.)
- Type (`type: bug` , `type: new feature` , `type: discussion` , etc.)

All labels are listed on the labels page.

Labels for community contributors

Issues that are beneficial to our users, 'nice to haves', that we currently do not have the capacity for or want to give the priority to, are labeled as `contrib:` , so the community can make a contribution. We categorize them into `easy` , `intermediate` and `experts needed` to help contributors pick an issue to work on when joining the project, based on how difficult is the work we believe the issue demands.

Code Styleguide

CSS Styleguide

Comments

- Prefer line comments (`//`) to block comments.
- Prefer comments on their own line. Avoid end-of-line comments.
- Write detailed comments for code that isn't self-documenting:
  - Uses of z-index
  - Compatibility or browser-specific hacks

Formatting

- Use tabs for indentation
- Use dashes over camelCasing in class names.
- Prefer to not use ID selectors
- When using multiple selectors in a rule declaration, give each selector its own line.
- Put a space before the opening brace `{` in rule declarations
- In properties, put a space after, but not before, the `:` character.
- Put closing braces `}` of rule declarations on a new line
- Put blank lines between rule declarations, but not in nesting selectors.
- Put nesting after all properties.
- Try to not use `!important` .
- Lowercase, always :)

Bad

```
blockquote {
    .clearfix;
    margin: .5em 0;
    &:first-child {
        margin-top: 0;
    }
    &:last-child {
        margin-bottom: 0;
    }
    padding-left: 10px;
    position: relative;
    &:before {
        content: ' ';
        width: 4px;
        position: absolute;
        border-radius: 2px;
        left: 0;
        top: -1px;
        bottom: -1px;
    }
}
.login-terms {
    font-size: smaller;
    width: 520px;
    padding: 10px;
    max-width: 100%;
    margin: auto;
    a {
        font-weight: bold !important;
        text-decoration: underline;
    }
}
```

Good

```
blockquote {
    .clearfix;
    margin: .5em 0;
    padding-left: 10px;
    position: relative;

    &:first-child {
        margin-top: 0;
    }

```

```
    &:last-child {
        margin-bottom: 0;
    }

    &::before {
        content: ' ';
        width: 4px;
        position: absolute;
        border-radius: 2px;
        left: 0;
        top: -1px;
        bottom: -1px;
    }
}

.login-terms {
    font-size: smaller;
    width: 520px;
    padding: 10px;
    max-width: 100%;
    margin: auto;

    a {
        font-weight: bold;
        text-decoration: underline;
    }
}
```

Selectors

Create an empty line before selectors

Bad

```
.class {
    border: 0;
}
.second-class: {
    text-align: right;
}
```

Good

```
.class {
    border: 0;
}

.second-class: {
    text-align: right;
}
```

Use a single space after selectors

Bad

```
.class{
    border: 0;
}
```

Good

```
.class {
    border: 0;
}
```

Add an empty line before nested selectors

Bad

```
.class {
    border: 0;
    &.another-class {
        color: #ffffff;
    }
}
```

Good

```
.class {
    border: 0;

    &.another-class {
        color: #ffffff;
    }
}
```

Use an empty line before non-nested selectors

Bad

```
.class {
    border: 0;

    &.another-class {
        color: #ffffff;
    }
}
.logo {
    display: block;
}
```

Good

```
.class {
    border: 0;

    &.another-class {
        color: #ffffff;
    }
}

.logo {
    display: block;
```

```
}
```

Don’t add whitespaces inside of the brackets within attribute selectors

Bad

```
.input[ type="text" ] {
    border: 0;
}
```

Good

```
.input[type="text"] {
    border: 0;
}
```

Don’t add a whitespace between operators within attribute selectors

Bad

```
.input[type = "text" ] {
    border: 0;
}
```

Good

```
.input[type="text"] {
    border: 0;
}
```

Always use a single space between the combinators of selectors

Bad

```
.class>.button {
    font-size: 1rem;
}
```

Good

```
.class > .button {
    font-size: 1rem;
}
```

Add a newline after the commas of selector lists

Bad

```
.class,  .another, .another-class {
    padding: 1.5rem;
}
```

Good

```
.class,
.another,
.another-class {
    padding: 1.5rem;
}
```

Don’t add a whitespace inside of the parentheses within pseudo-class

selectors

Bad

```
.class:not( .another ) {
    margin: 5px;
}
```

Good

```
.class:not(.another) {
    margin: 5px;
}
```

Double colon for applicable pseudo-elements

Bad

```
.class:before {
    border-width: 2px;
}

.class:first-child {
    color: #000000;
}
```

Good

```
.class::before {
    border-width: 2px;
}

.class:first-child {
    color: #000000;
```

```
}
```

Properties

Don’t add empty line in first/last property or in between properties

Bad

```
.room-list {

    border: 0;
    padding-left: 0;

    color: #ddfc32;

}
```

Good

```
.room-list {
    border: 0;
    padding-left: 0;
    color: #ddfc32;
}
```

Don't add properties in single-line. Use one property per line

Bad

```
.sumbit {color: #ffffff; background-color: #000000;}
```

Good

```
.sumbit {
    color: #ffffff;
    background-color: #000000;
}
```

No empty blocks

Bad

```
.button {}
```

Good

```
.button {
    float: left;
}
```

Longhand properties must be combined into one shorthand property

Bad

```
.class {
    padding-left: 12px;
    padding-top: 5px;
    padding-bottom: 8px;
}
```

Good

```
```

```
.class {
    padding: 5px auto 8px 12px;
}
```

Don’t use shorthand properties that override related longhand properties

Example

```
.class {
    border-color: #ffffff;
    border: 1px solid #000000;
}
```

Always add a trailing semicolon in the end of a declaration

Bad

```
.another-class {
    color: #ffffff;
    padding: 2px
}
```

Good

```
.another-class {
    color: #ffffff;
    padding: 2px;
}
```

Add a newline after the colon of declarations

Bad

```
.another-class {
    box-shadow: 0 0 0 1px #5b9dd9, 0 0 2px 1px rgba(30, 140, 190, 0.8);
}
```

Good

```
.another-class {
    box-shadow:
        0 0 0 1px #5b9dd9,
        0 0 2px 1px rgba(30, 140, 190, 0.8);
}
```

Don’t duplicate properties within declaration blocks

Bad

```
.another-class {
    display: block;
    margin-top: 2rem;
    display: inline-block;
}
```

Good

```
.another-class {
    margin-top: 2rem;
    display: inline-block;
}
```

Don’t duplicate selectors along the file

Bad

```
.some-class {
    display: block;
}
.another-class {
    position: absolute;
}
.some-class {
    margin-top: 2rem;
    display: inline-block;
}
```

Good

```
.some-class {
    display: block;
    margin-top: 2rem;
    display: inline-block;
}
```

Add a whitespace before bang (!) declaration

Bad

```
.class {
    margin-left: 12px!important;
}
```

Good

```
.class {
    margin-left: 12px !important;
}
```

Use spaces around calc operators

Bad

```
.class {
    width: calc(~"200px-1rem");
}
```

Good

```
.class {
    width: calc(~"200px - 1rem");
}
```

Remove units for zero lengths

Bad

```
.button {
    padding: 10px 0px 0px 2px;
}
```

Good

```
.button {
    padding: 10px 0 0 2px;
}
```

Use a leading zero for fractional numbers css than 1

Bad

```
.message-form {
    margin-right: .5rem;
}
```

Good

```
.message-form {
    margin-right: 0.5rem;
}
```

Don’t add trailing zeros in numbers

Bad

```
.flex-tab {
    padding-bottom: 1.500rem;
}
```

Good

```
.flex-tab {
    padding-bottom: 1.5rem;
}
```

Don’t add newlines in strings

Bad

```
.nav-link::before {
    content: "sample
        text";
```

```
}
```

Good

```
.nav-link::before {
    content: "sample text";
}
```

Colors

Lowercase, always

Long notation

Bad

```
.another-class {
    background-color: #FFF;
}
```

Good

```
.another-class {
    background-color: #ffffff;
}
```

Internationalization

When developing in Rocket.Chat, you have the ability to add strings to our translations files in wich later will be translated to other languages by our community translators.

Tools Used

For selecting strings for the correct language we use the TAP:i18n meteor package.

For managing contributions from the translators community we use lingohub. If you are interested in contributing to Rocket.Chat translations please see ~~Translating~~

Adding Strings to the Translation Files

To have a string translated you will firstly need a `key` that would be the identifier of the string you want translated, for example the string `This room is read only` should have the key as `room_is_read_only`. Please have in mind when naming a key that spaces should be replaced with underscores (`_`) and it should be named in english, as is the language selected for Rocket.Chat's code. Finally you will have the `key/value` pair will look like this: `"room_is_read_only": "This room is read only"`

You can also specify `placeholders`, that will allow you to change information on the string via parameters when calling the `i18n` method. A parameter will be surrounded by two double underscores (`__ __`), and it will look like this, `"Conversation_closed": "Conversation closed: __comment__."` where `__comment__` can be replaced by any string provided in the parameters.

After that you will need to add this key to its respective `i18n.json` file under the i18n folder.

When your pull request is merged, our contributors at `LingoHub` will be notified of the new string and start translating it.

Using Translated Strings on the code

Now that you have added your strings into the translation files, is time to use them in the code!

When writing on a `.js` file you can use the global method `TAPi18n.__()`. Now depending where you are calling this method from it can have different parameters.If you call it from the back-end of Rocket.Chat you will have to specify a language (in which you can normally grab from an user's object). The method will look like this, `TAPi18n.__('YOUR_KEY_HERE', {}, user.language)`. You can also pass parameters on the second argument to replace placeholders on the translated strings. Now if you are calling from the front end you can simply use `TAPi18n.__('YOUR_KEY_HERE', {})` that will translate the selected string to the user's current selected language.

If the selected key is not present in the respective `.i18n.json` file it will default it to english, if no key is found it will display the key inserted in the method as a string.

Now if you are in a `.html` file, you can simply surround the string with `{{ "{{_ " }}}}` for example `{{ "{{_ YOUR_KEY_HERE" }}}}`. It will work the same as the method mentioned above, only with the convenience of adding it directly to the `.html` file.

Sometimes you can find some methods that requires an object with a `i18nLabel` or `i18nDescription`. In these cases you only need to insert the key of the string, the method will do the rest.

Mobile apps

→ Whitelabeling mobile apps /guides/developer/mobile-apps/whitelabeling-mobile-apps

→ Supporting SSL for development on Rocket.Chat /guides/developer/mobile-apps/supporting-ssl

→ Analytics & Data Usage /guides/developer/mobile-apps/analytics

Whitelabeling mobile apps

In this guide we will cover how to rebrand Rocket.Chat Mobile Apps to suit your styling.

Here we will show you how to customize:

- The App Icons
- Splash Screens
- App Name
- Colors

Important

- This document is updated after every release, so we can guarantee it's stable for production
  - `develop` branch might be different from this
- Keep in mind that you will need an **intermediate** knowledge of Android/iOS development and basic Javascript knowledge
- Our repo contains targets/build flavors to build both our Experimental and Official apps
  - Both apps are equal, but released at different pace on the stores
  - If you see an Experimental folder, don't be scared of breaking anything. It's just a folder containing the assets for the non-official app

---

Repo

- Make sure you have both iOS and Google developer accounts and the respective development environments working
  - You can follow this guide: https://reactnative.dev/docs/getting-started
- Clone https://github.com/RocketChat/Rocket.Chat.ReactNative
- Checkout `single-server` branch (git clone -b single-server https://github.com/RocketChat/Rocket.Chat.ReactNative)

General

- Create an account on https://www.bugsnag.com/
- Set `BUGSNAG_API_KEY` on config.js
- Set `server`, `appGroup` and `appStoreId` on app.json
  - `appGroup` must be the same App Group created for the iOS app
- Change app colors on colors.js

Firebase

Creating a new project on Google Cloud Platform

- Visit https://console.cloud.google.com/home/dashboard
- By the text Google Cloud Platform there is a dropdown, open and then “Create project”
- In the dialog give an project name then “Create”
- Wait the creating process, you can follow in notifications by your avatar
- You will be redirected to the project page after creation

Creating new Firebase project

- Visit https://console.firebase.google.com/
- Click on “Add Project”
- Enter the project name you created on previous step
- Follow the wizard until Firebase project is created
- We’re going to create the apps later on the tutorial

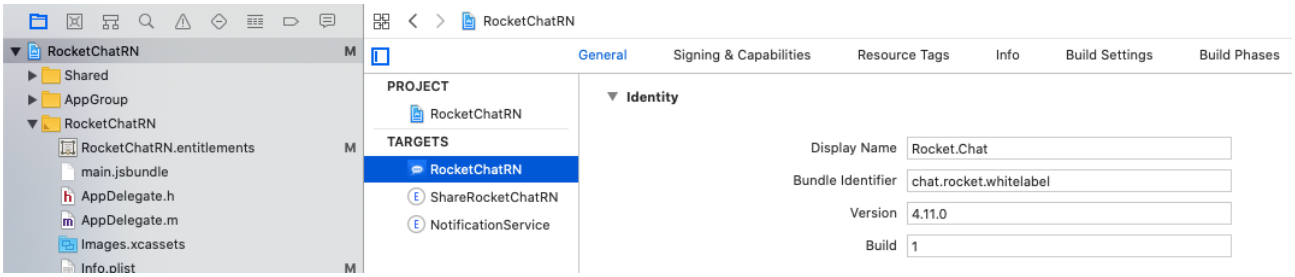
---

iOS

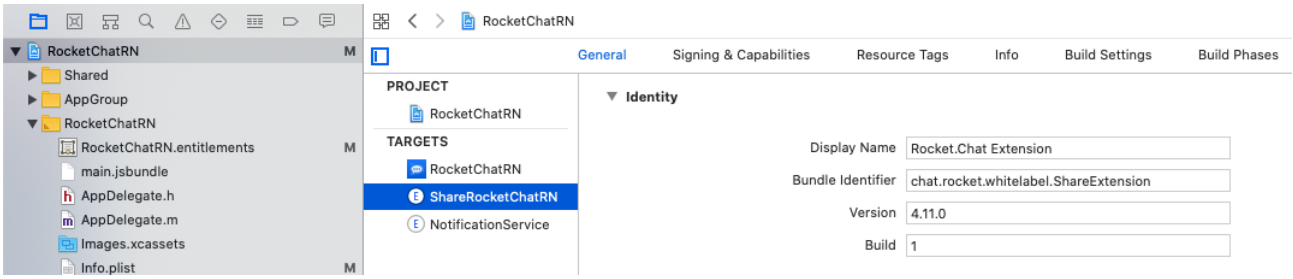
General setup

- Open RocketChatRN.xcworkspace on Xcode (11.7 or newer)

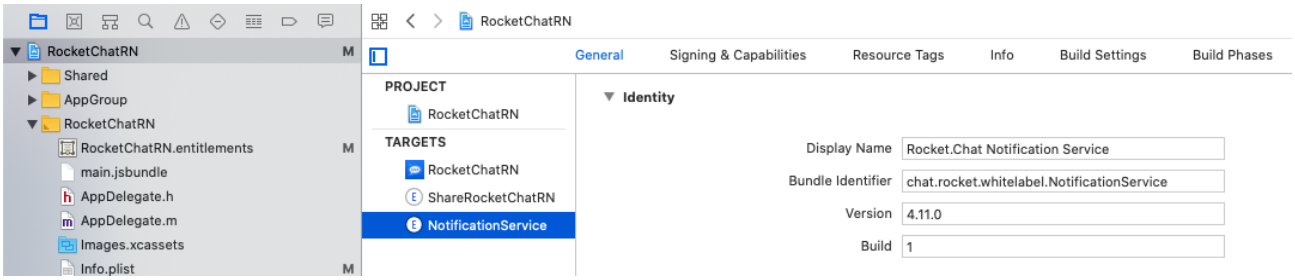
- On General tab, select “RocketChatRN” and change Display Name, Bundle Identifier, Version and Build
    - Note: as explained on Important section, we have two targets and we're going to cover the default one on this doc, which is the Experimental app.



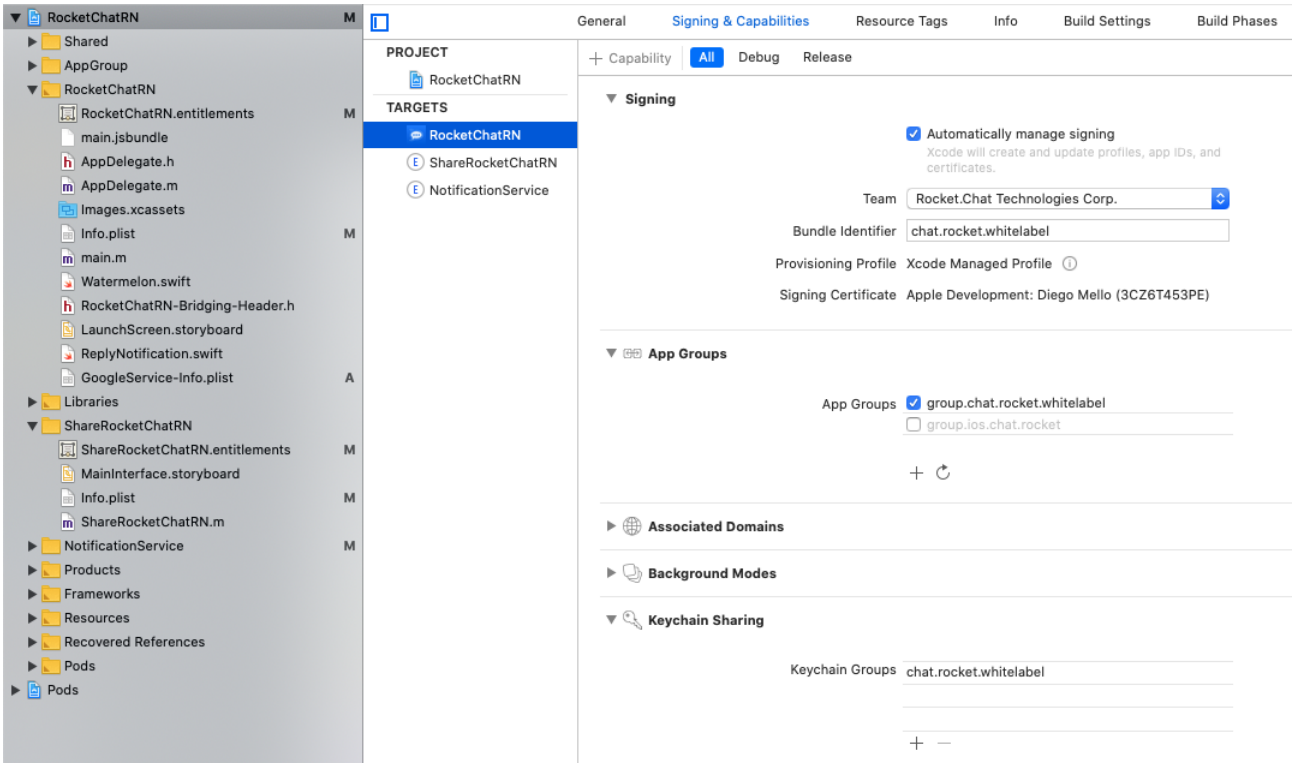
- Select “ShareRocketChatRN” and change the same properties
    - `Display Name` and `Bundle Identifier` are different from the previous target
    - `Version` and `Build` must be the same on all targets



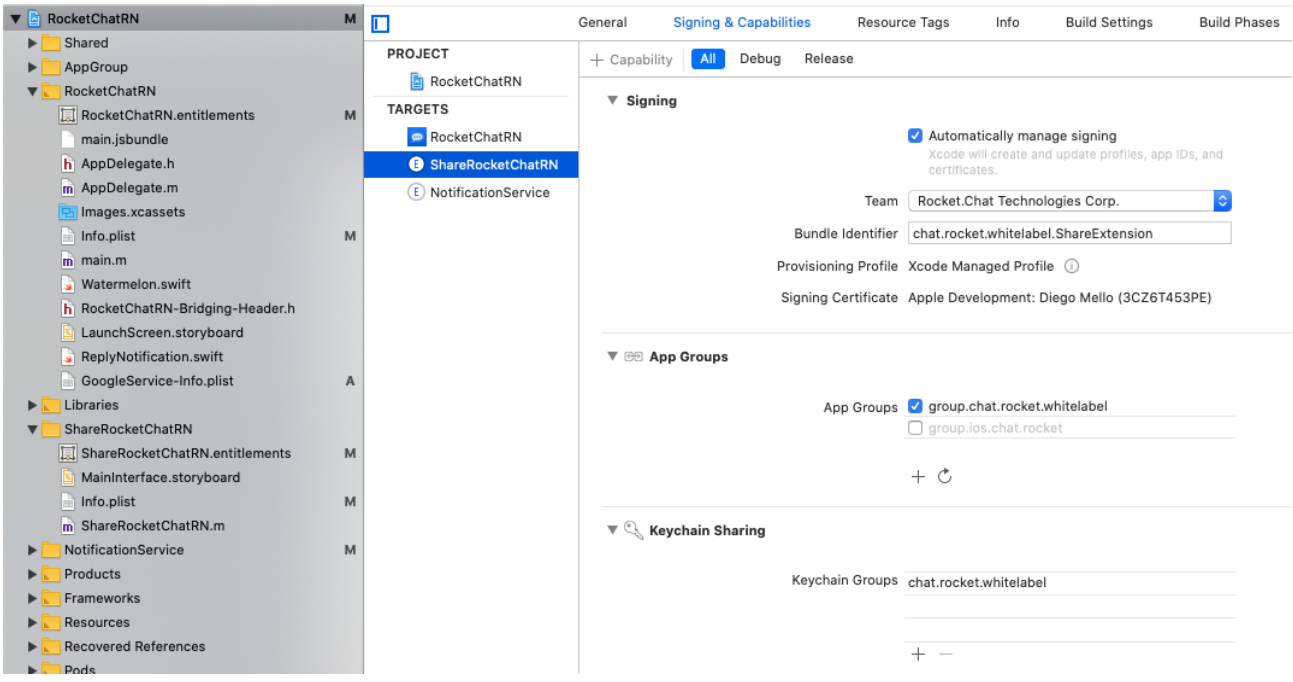
- Select “NotificationService” and change the same properties
    - `Display Name` and `Bundle Identifier` are different from the previous target
    - `Version` and `Build` must be the same on all targets



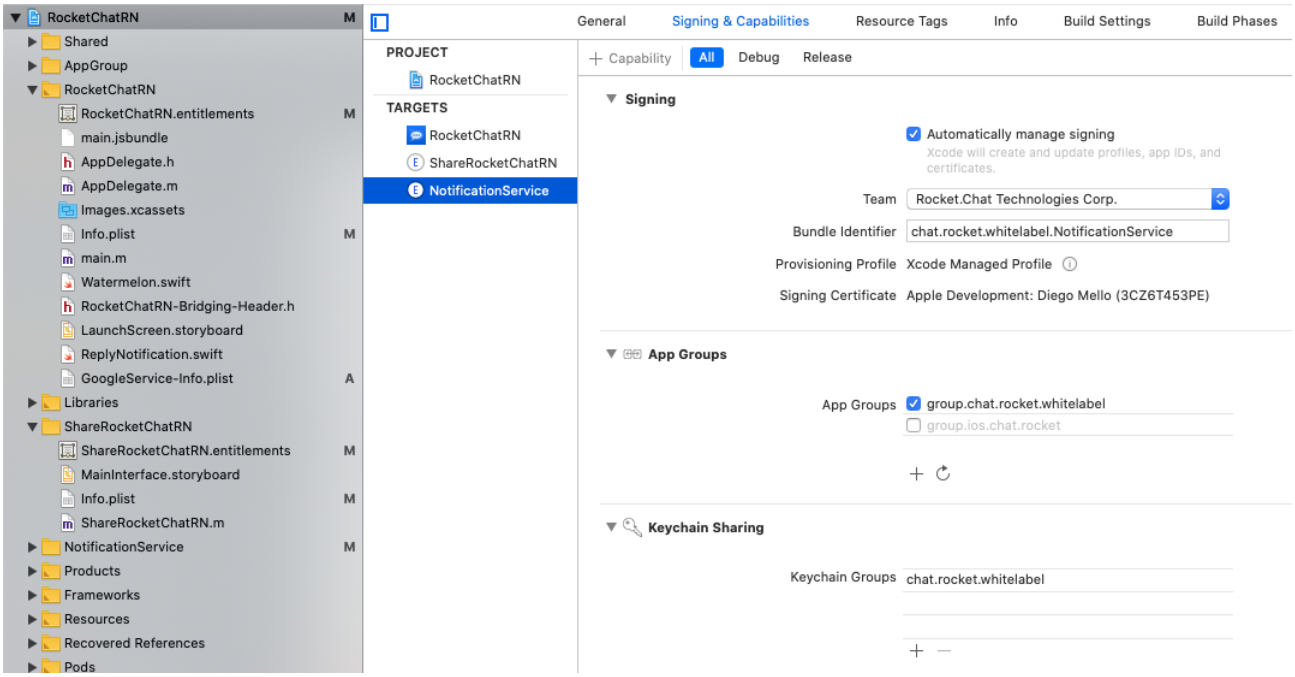
- On Signing and Capabilities, check “Automatically manage signing”, select your app group and add a keychain group



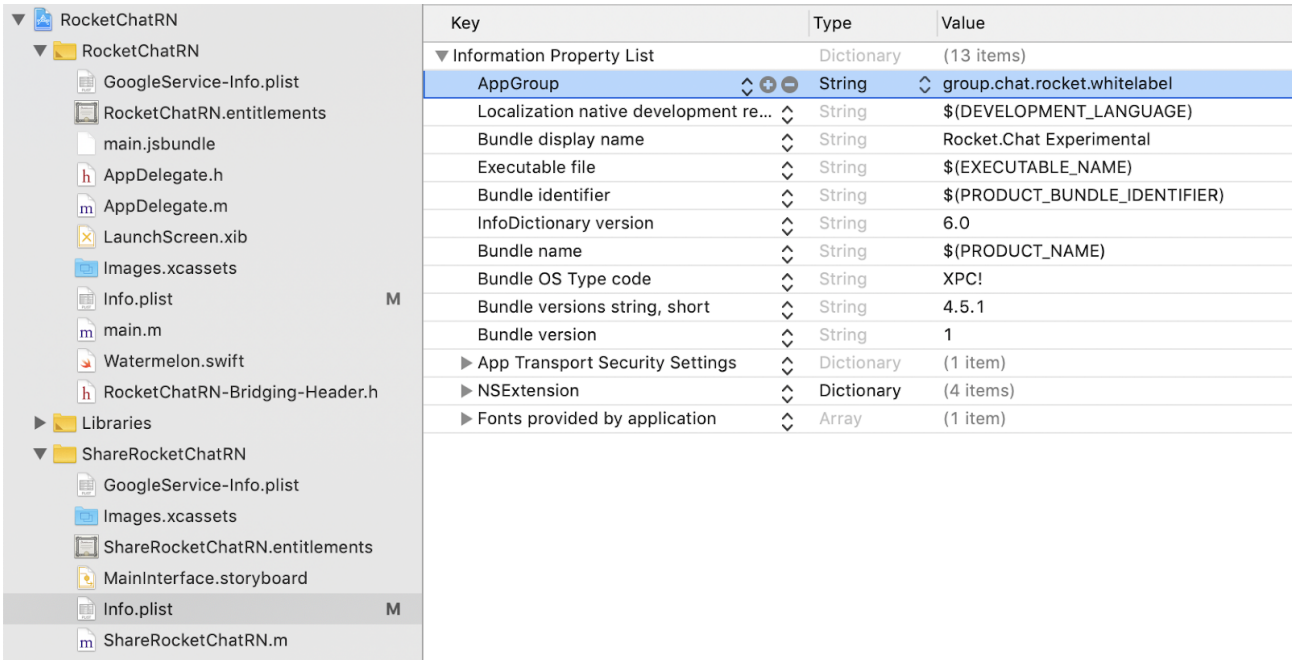
- Select “ShareRocketChatRN”, check “Automatically manage signing”, select your app group and add the same keychain group

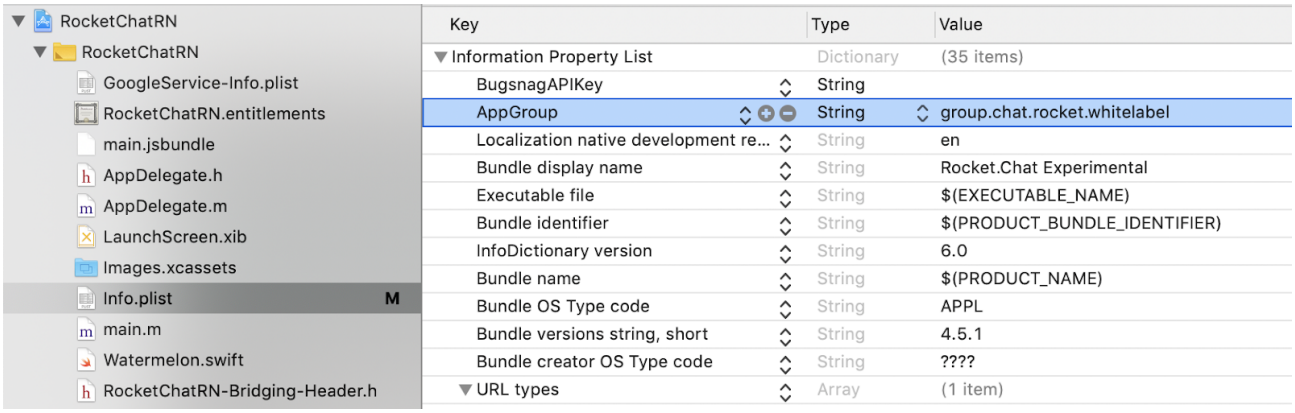


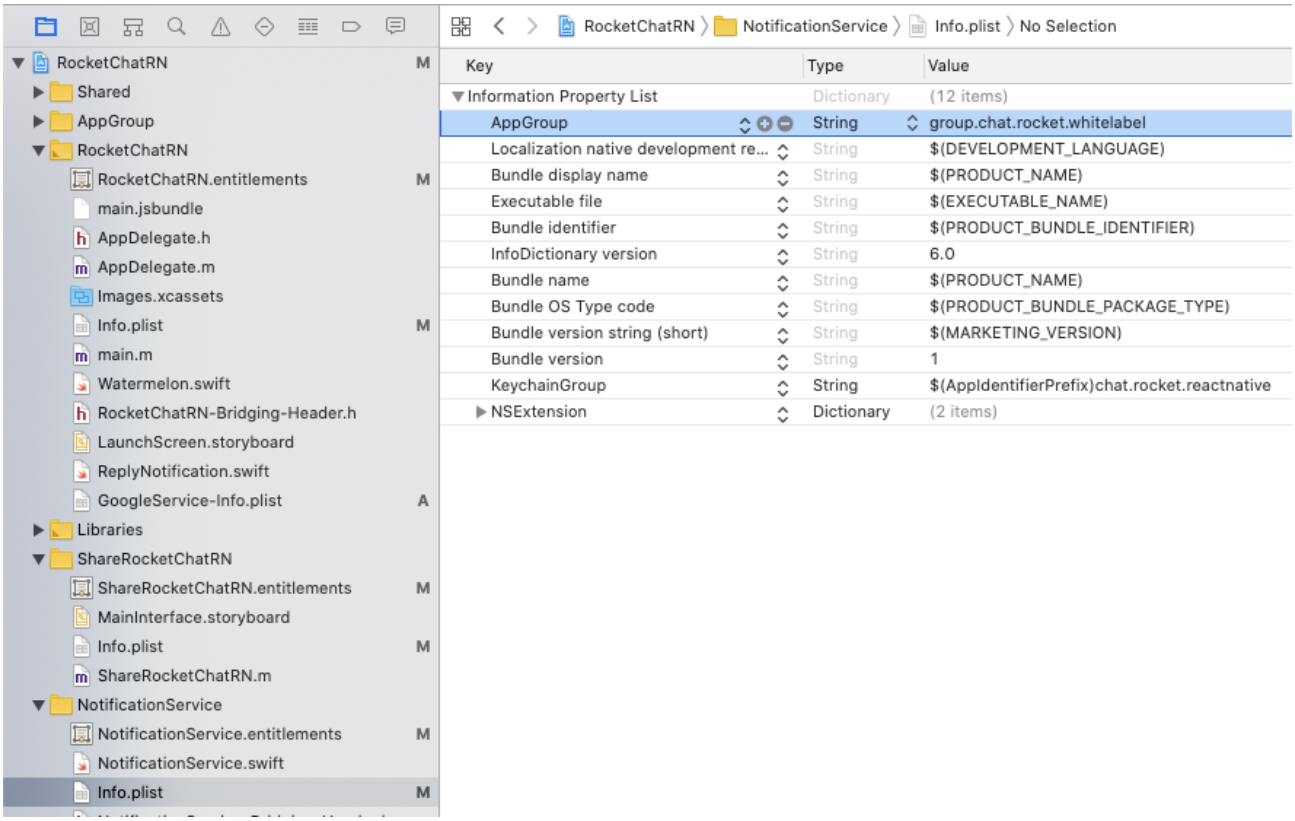
- Select “NotificationService”, check “Automatically manage signing”, select your app group and add the same keychain group



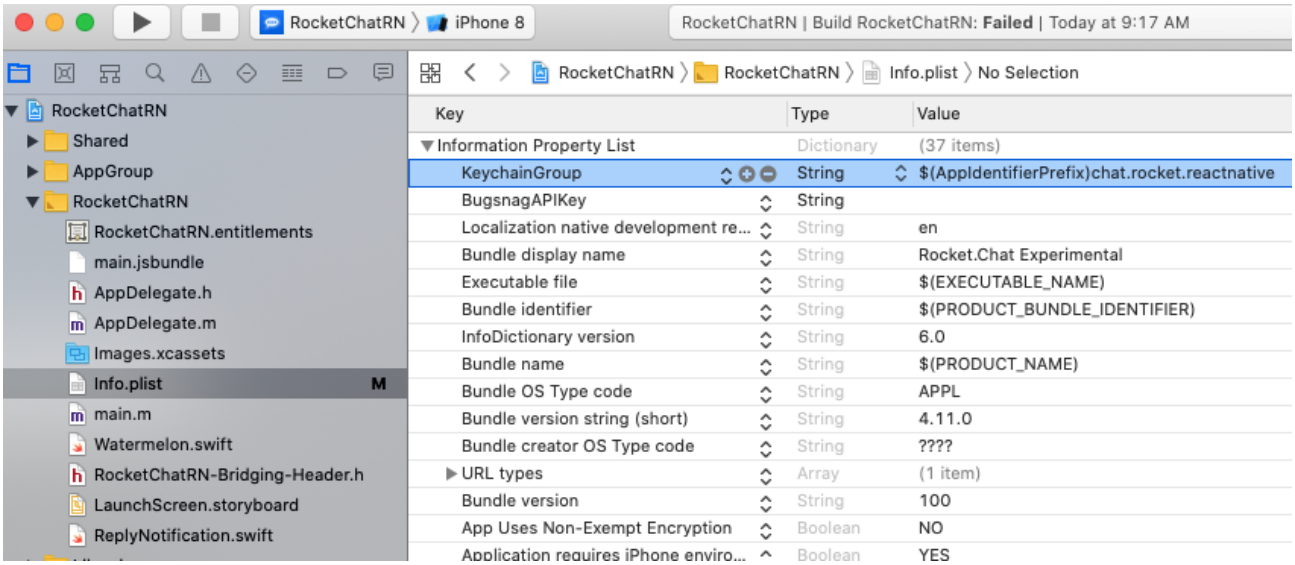
- Set the same app group on `RocketChatRN/Info.plist`, `ShareRocketChatRN/Info.plist` and `NotificationService/Info.plist`

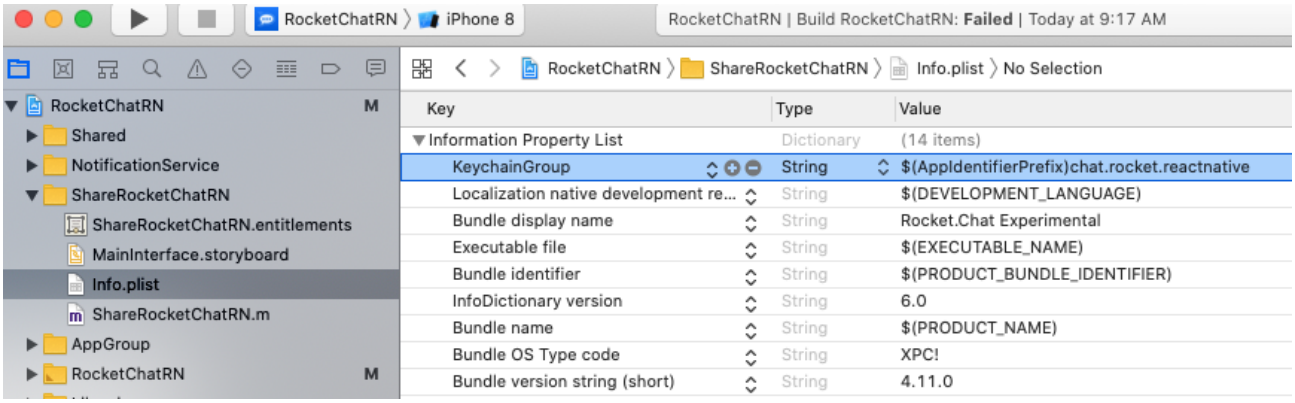


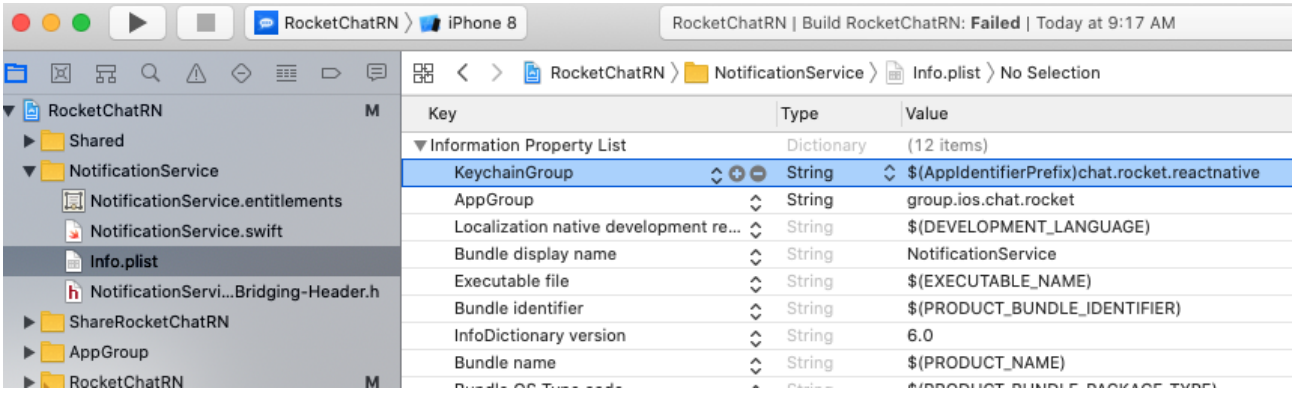




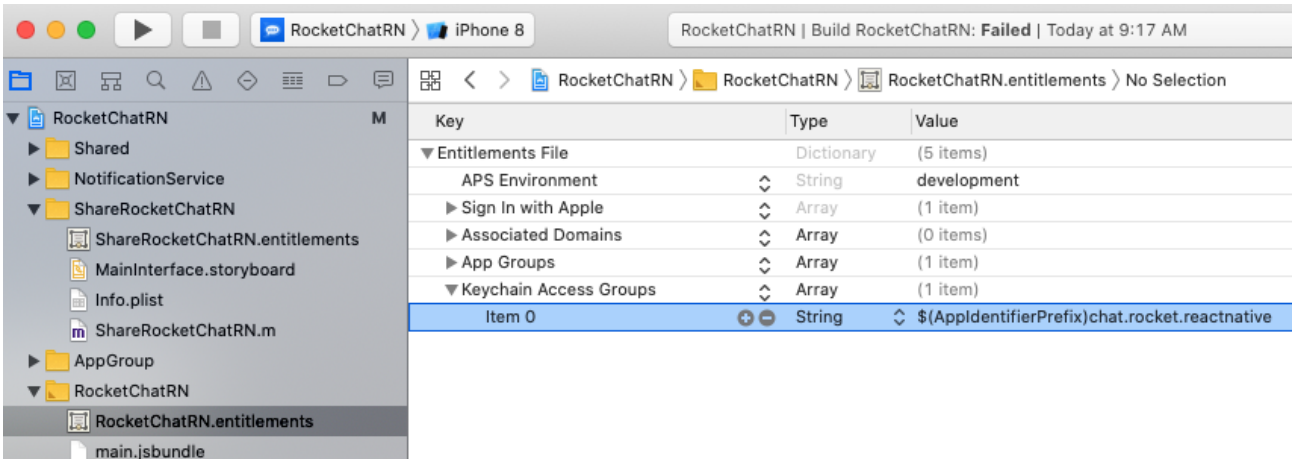
- Set the same keychain group on `RocketChatRN/Info.plist`, `ShareRocketChatRN/Info.plist` and `NotificationService/Info.plist`

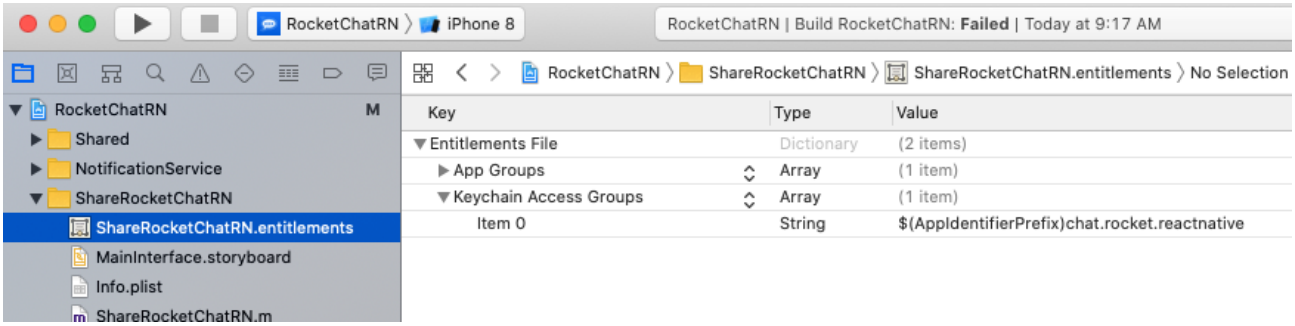


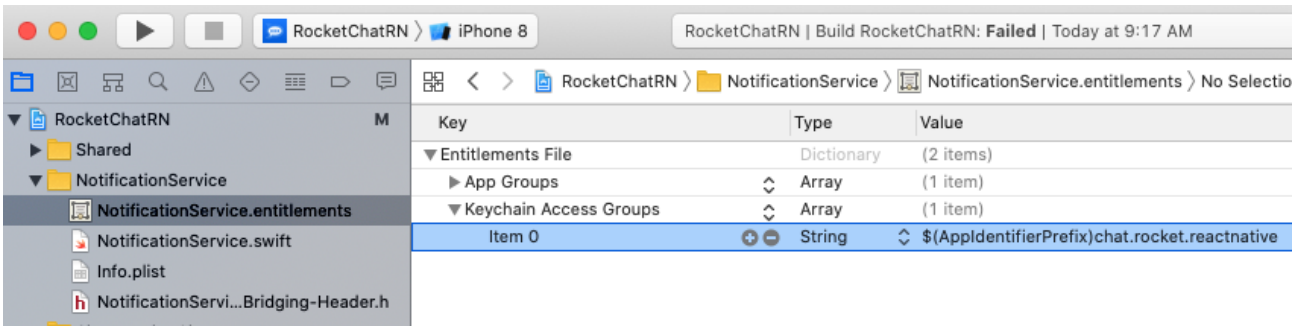




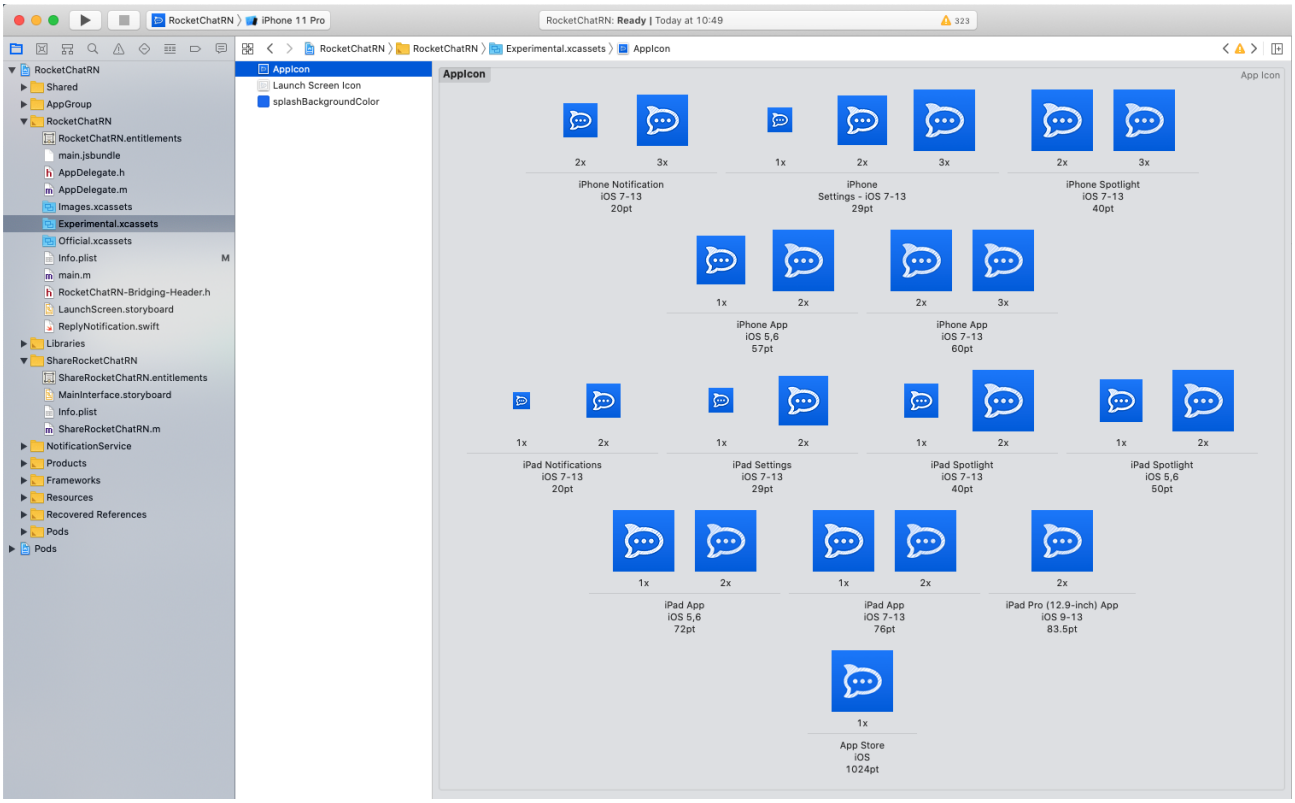
- It needs to be the same on all entitlements



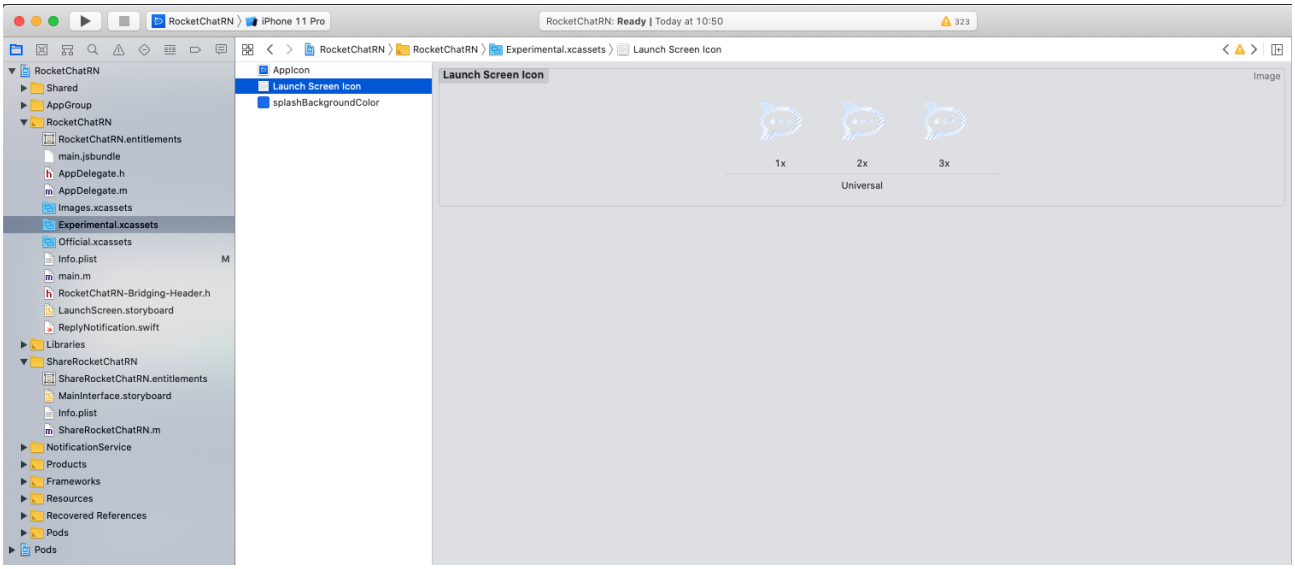




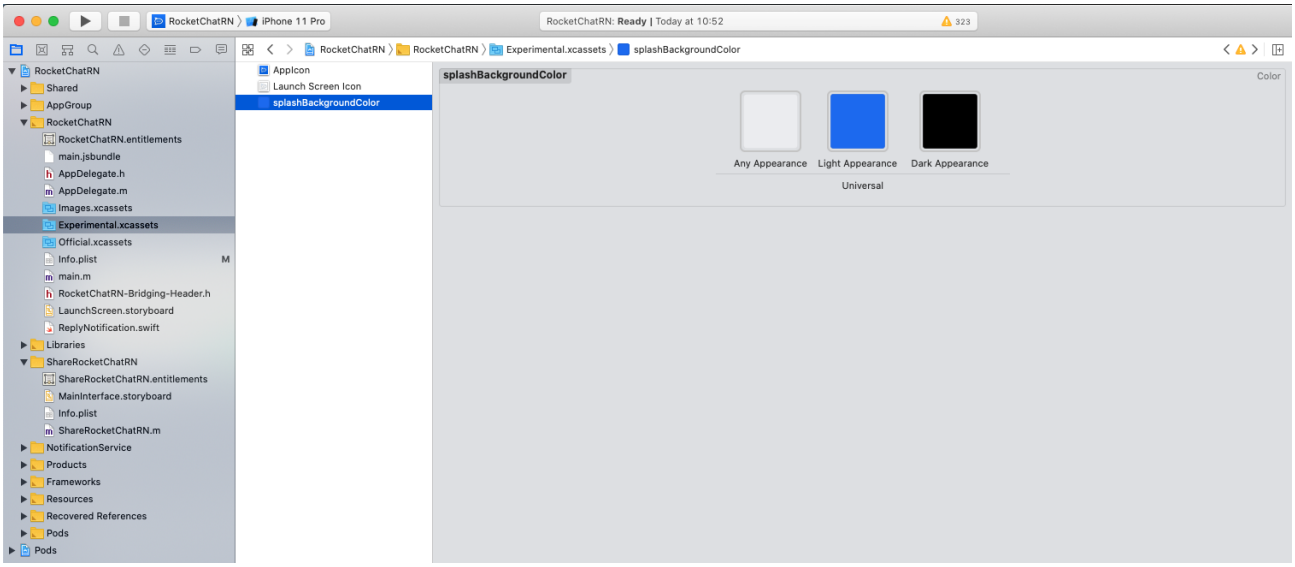
- Change the app icon on `Experimental.xcassets/App Icon`



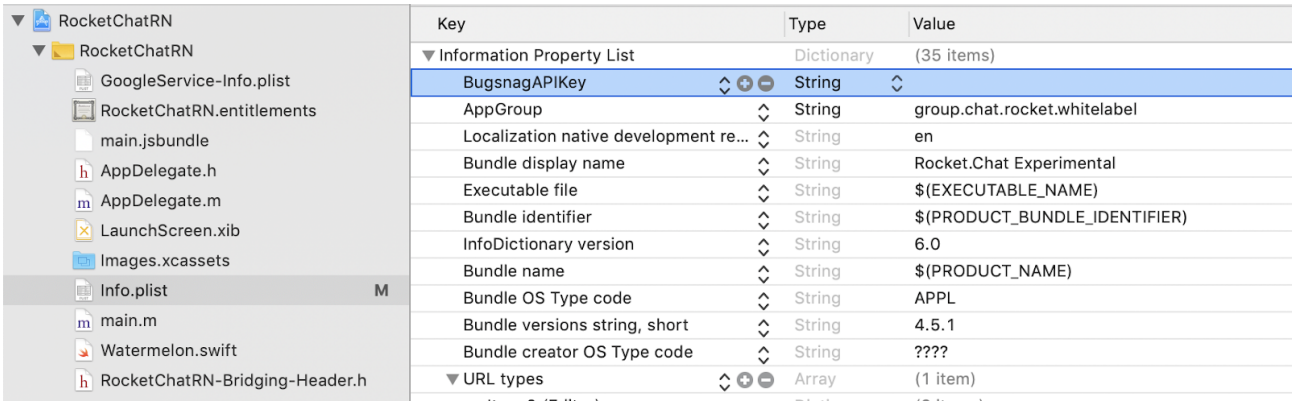
- Change the app splash screen on `Experimental.xcassets/Launch Screen Icon`



- Change the splash background colors on `Experimental.xcassets/splashBackgroundColor`

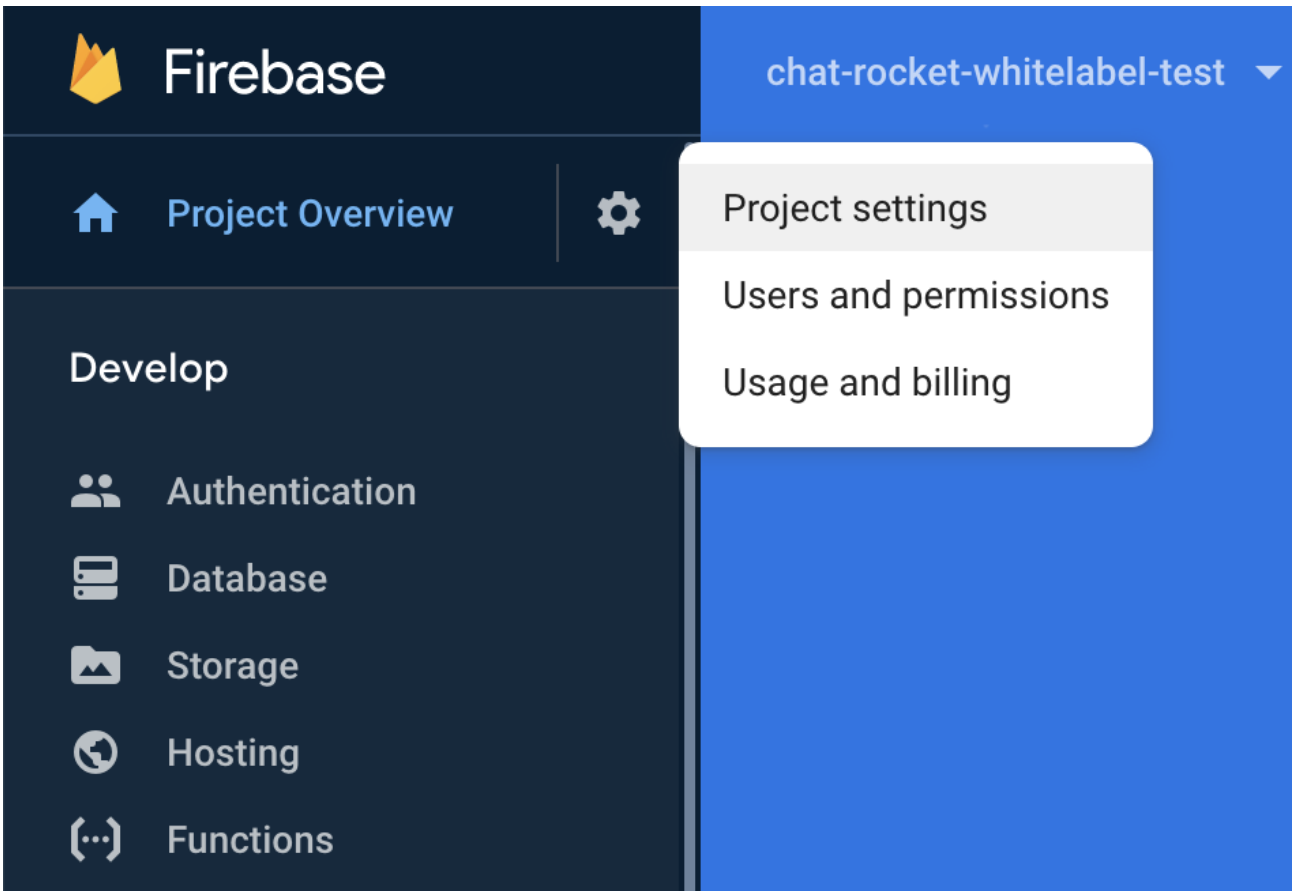


- Set your Bugsnag API key on `RocketChatRN/Info.plist`

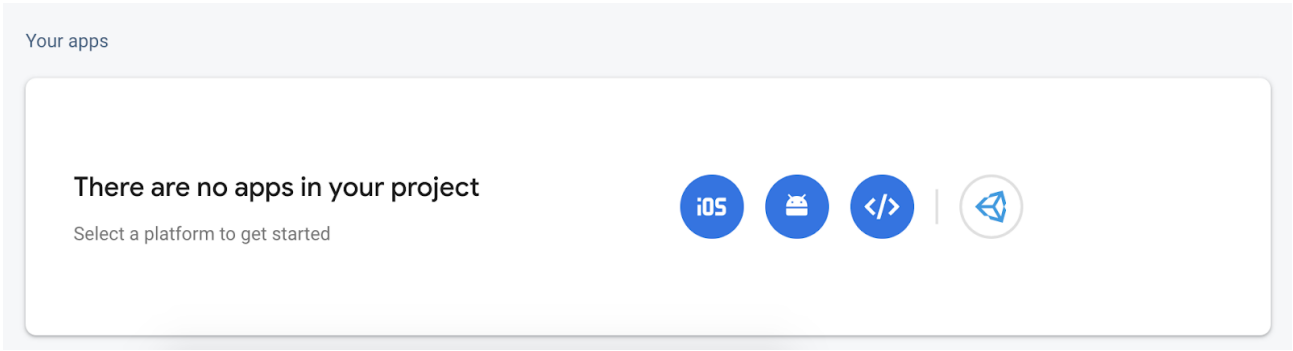


Generating iOS app on Firebase

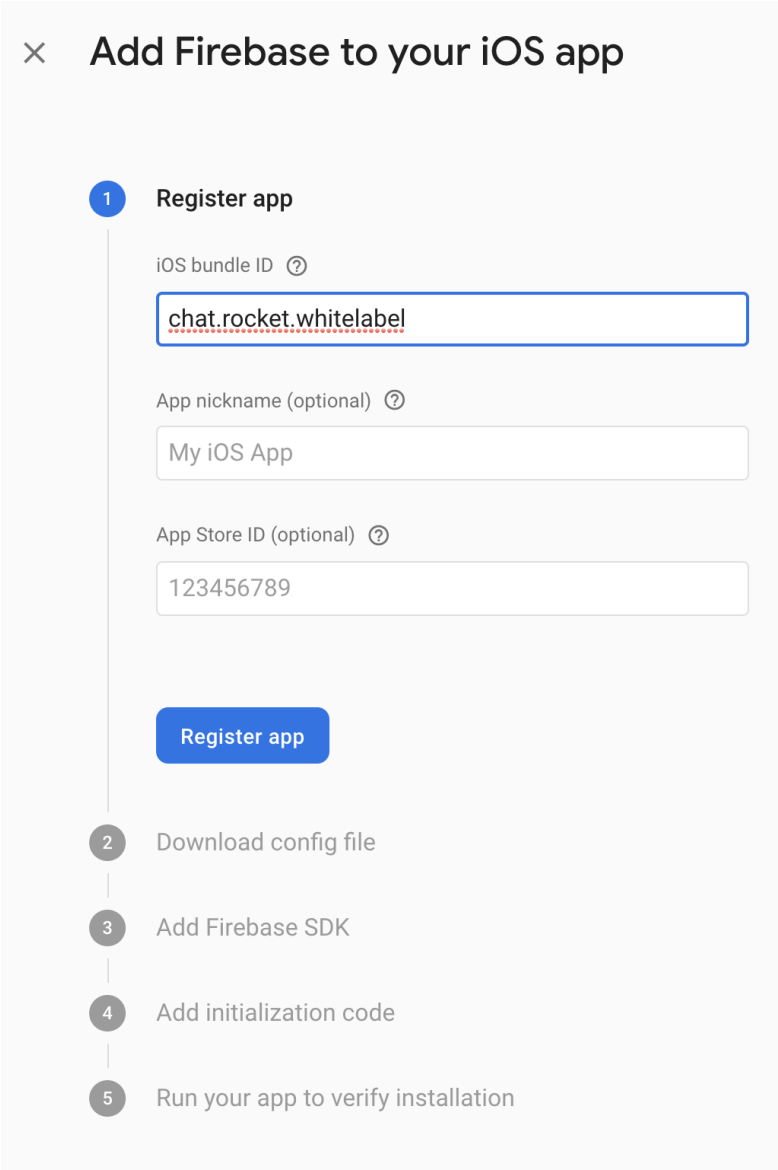
- Visit the project overview on https://console.firebase.google.com
- Click on the gear icon and then “Project settings”



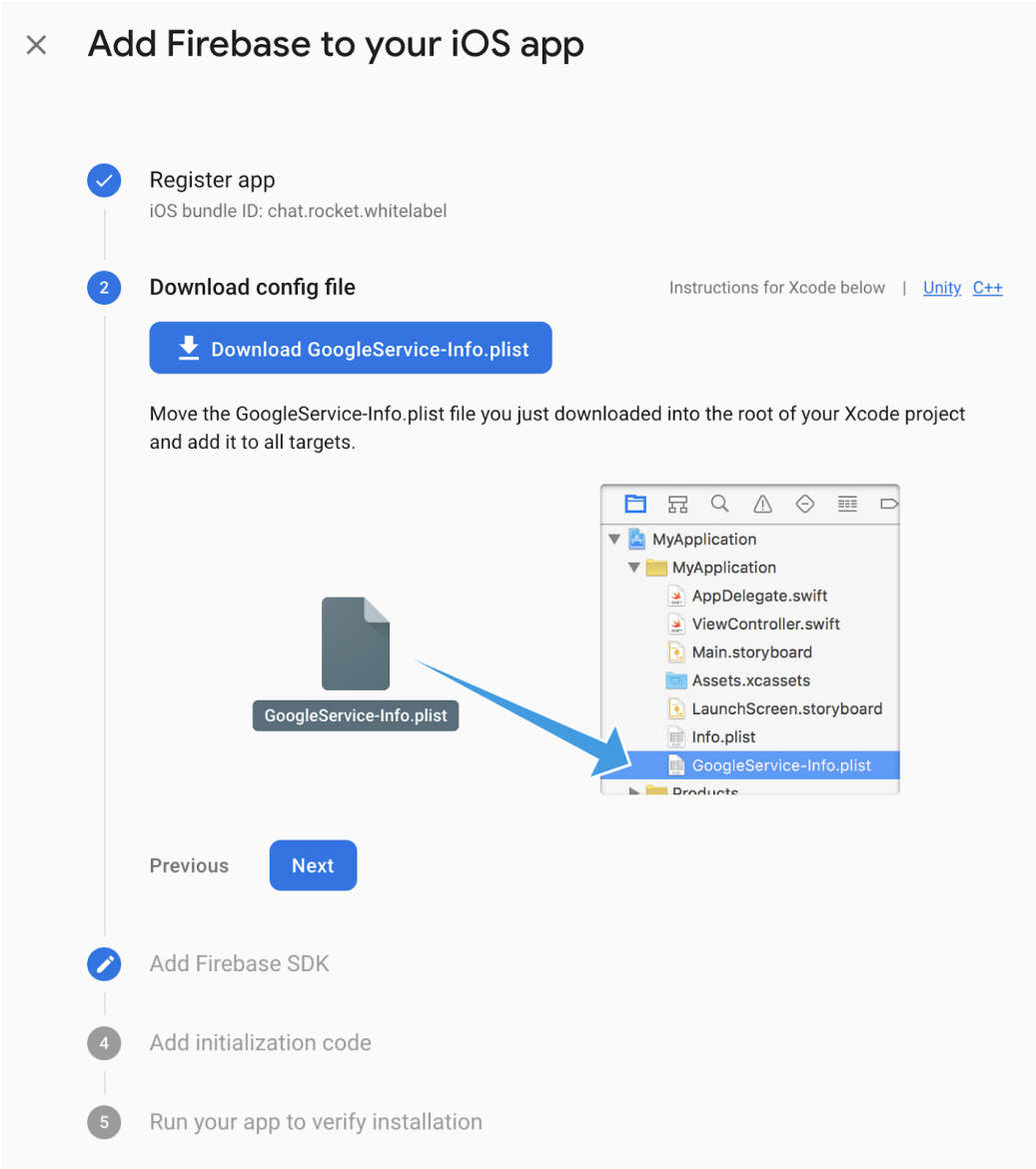
- On “General” tab, click on “iOS” button under “Your apps” section



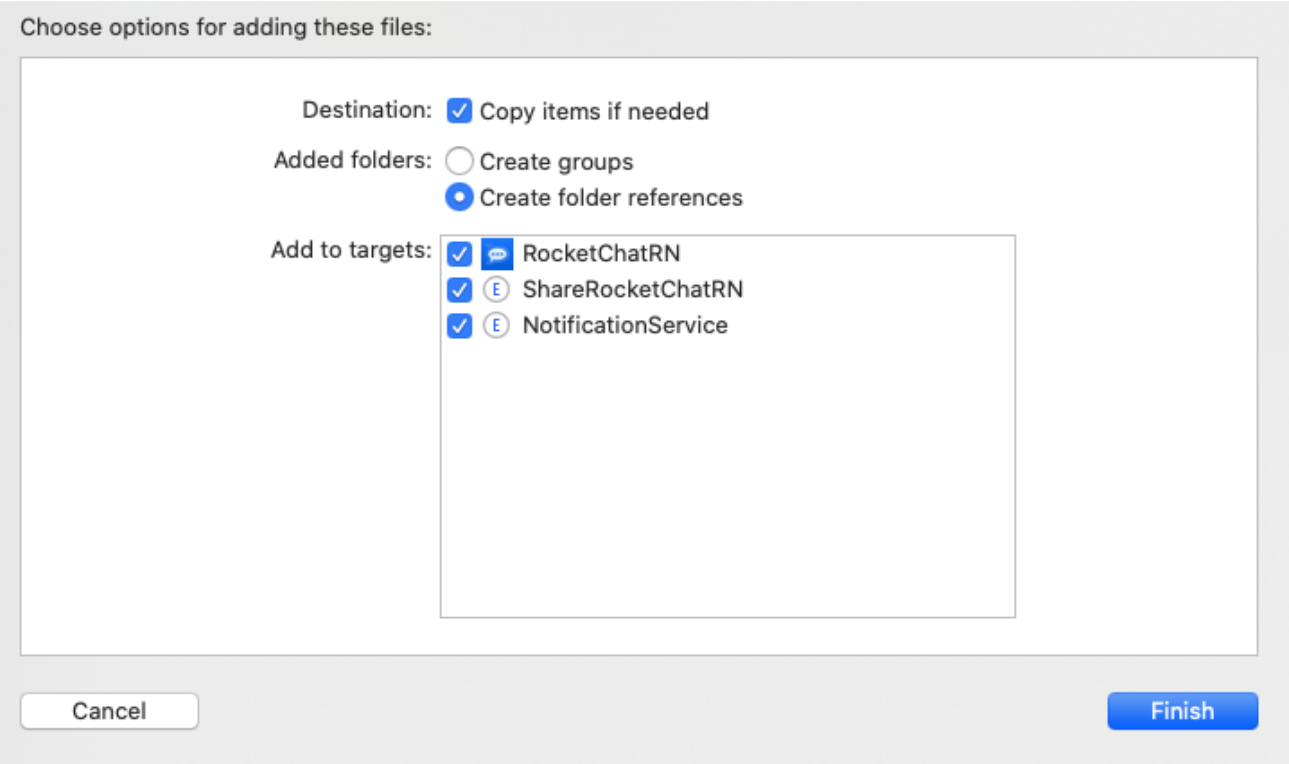
- Enter your bundle ID and then “Register app”



- Download config file and move it as instructed



- Add it to all targets



Running the app

- Execute the following on project terminal
  - `yarn`
  - `npx pod-install`
  - `yarn ios`

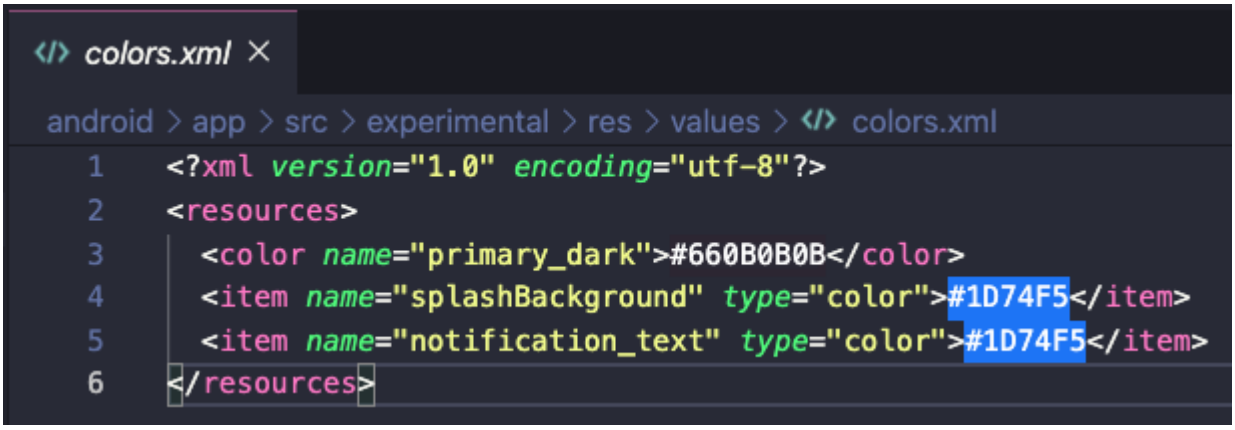
Android

General setup

- Similarly to iOS, we have build flavours to generate our Official, Experimental and F-Droid versions of the app
  - `experimental` and `official` folders contain app icons and splash screens
  - `play` and `foss` folders contain necessary code to run the app with or without Google Play services, respectively
    - `foss` build doesn't contain push notifications implemented
  - `main` folder contains core implementations

    - `debug` folder contains code to run the app in debug mode
    - This doc is going to focus on building the Experimental app, so we're going to use `experimental`, `play` `debug`, and `main` folders

- Set `APPLICATION_ID`, `VERSIONCODE` and `BugsnagAPIKey` on `./android/gradle.properties`
- Generate a new image asset for `ic_notification` and target `main`
- Generate a new image asset for `ic_launcher` and target `experimental`
- Change splash screen background and notification text color on `./android/app/src/experimental/res/values/colors.xml`



- Change splash screen logo on `./android/app/src/experimental/res/drawable-xxhdpi/splash.png`
- Change app name and share extension name on `./android/app/src/main/res/values/strings.xml`

strings.xml

android > app > src > experimental > res > values > strings.xml

```xml
<resources>
    <string name="app_name">Rocket.Chat Experimental</string>
    <string name="share_extension_name">Rocket.Chat Experimental</string>
</resources>
```

Generate upload key

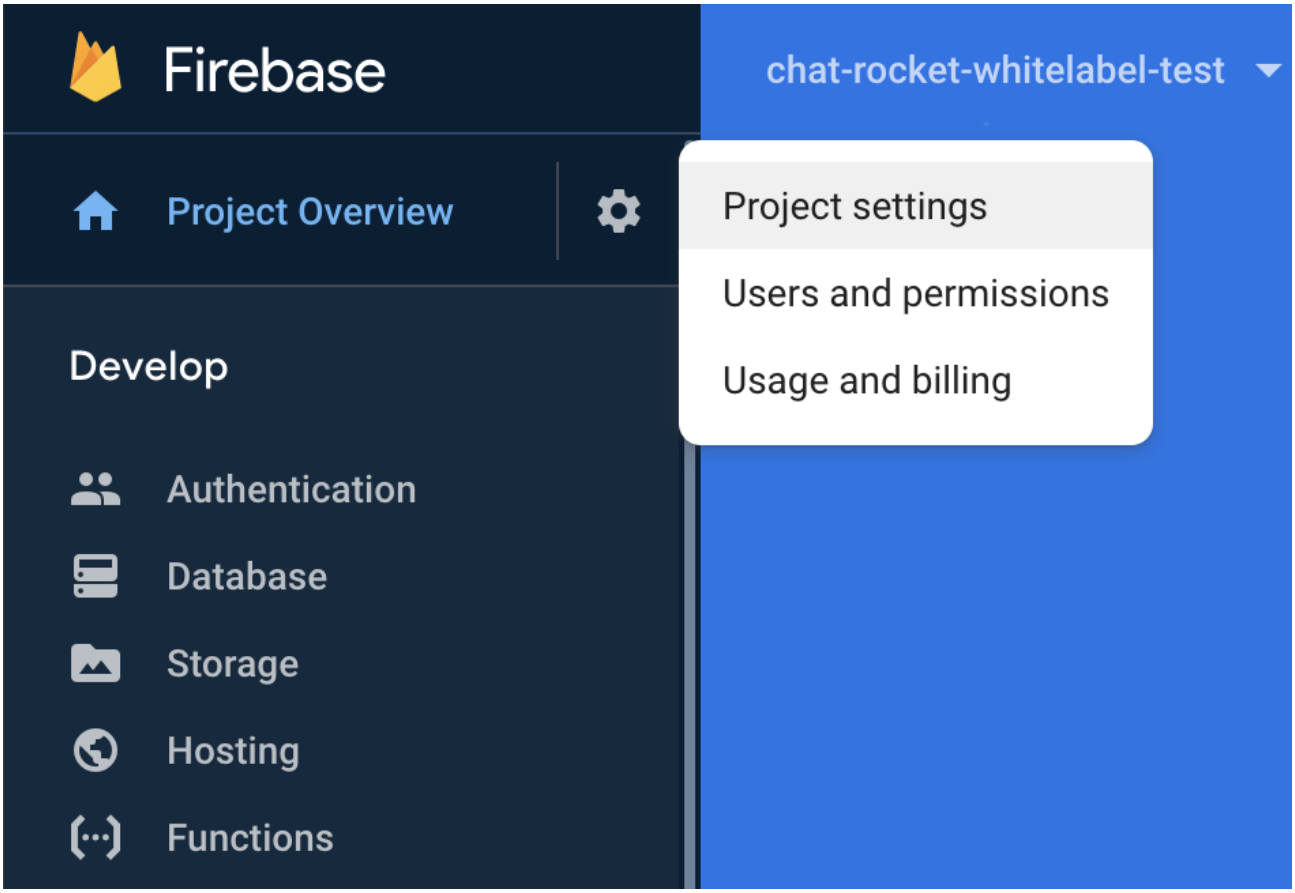
- This step will generate the keystore that is going to verify your app on Google Play
  - You can use this guide as reference: https://reactnative.dev/docs/signed-apk-android#generating-an-upload-key
- Execute the following on terminal
  - `cd android/app`
  - `keytool -genkeypair -v -keystore my-upload-key.keystore -alias my-key-alias -keyalg RSA -keysize 2048 -validity 10000`
- Credentials will be prompted

```
➜ keytool -genkeypair -v -keystore my-upload-key.keystore -alias my-key-alias -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:
What is your first and last name?
  [Unknown]:
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
        for: CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Enter key password for <my-key-alias>
        (RETURN if same as keystore password):
Re-enter new password:
[Storing my-upload-key.keystore]

Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard format using "keytool -importkeystore -src
keystore my-upload-key.keystore -destkeystore my-upload-key.keystore -deststoretype pkcs12".
```

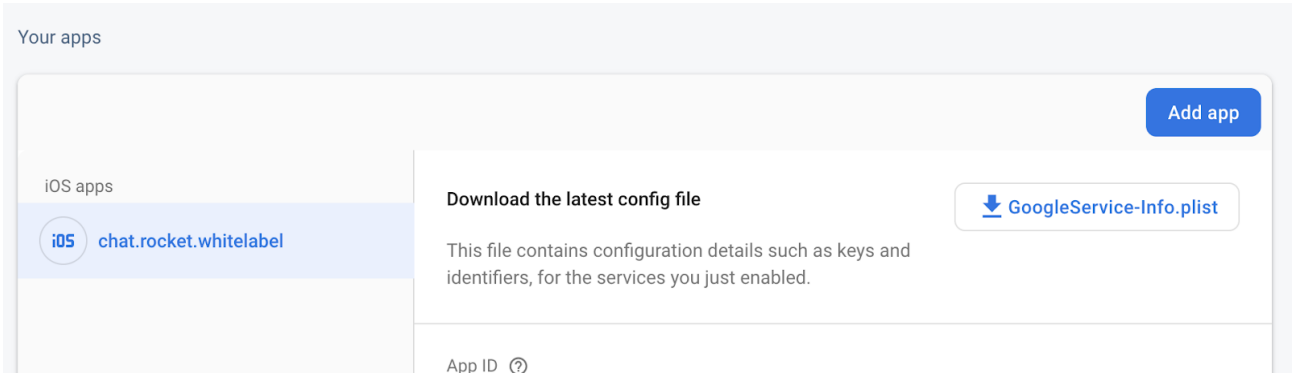
- Set `KEYSTORE_PASSWORD` and `KEY_PASSWORD` on ./android/gradle.properties with the passwords you were prompted

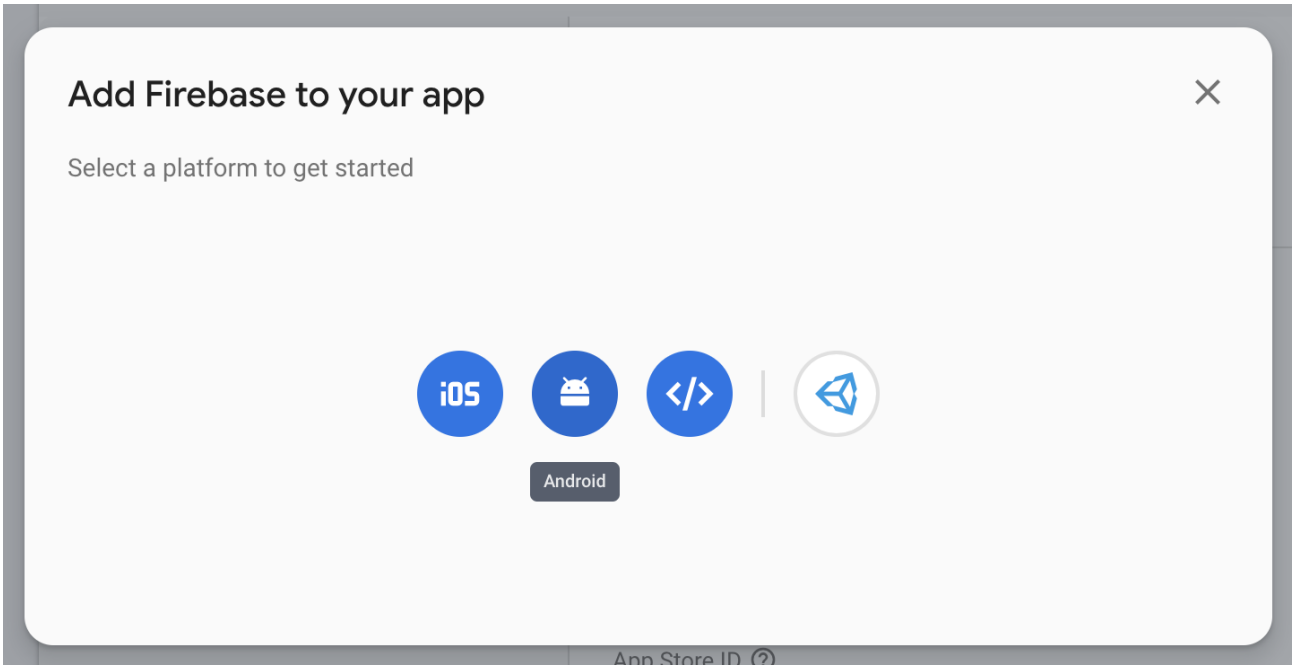
Generating Android app on Firebase

- Visit the project overview on https://console.firebase.google.com
- Click on the gear icon and then “Project settings”

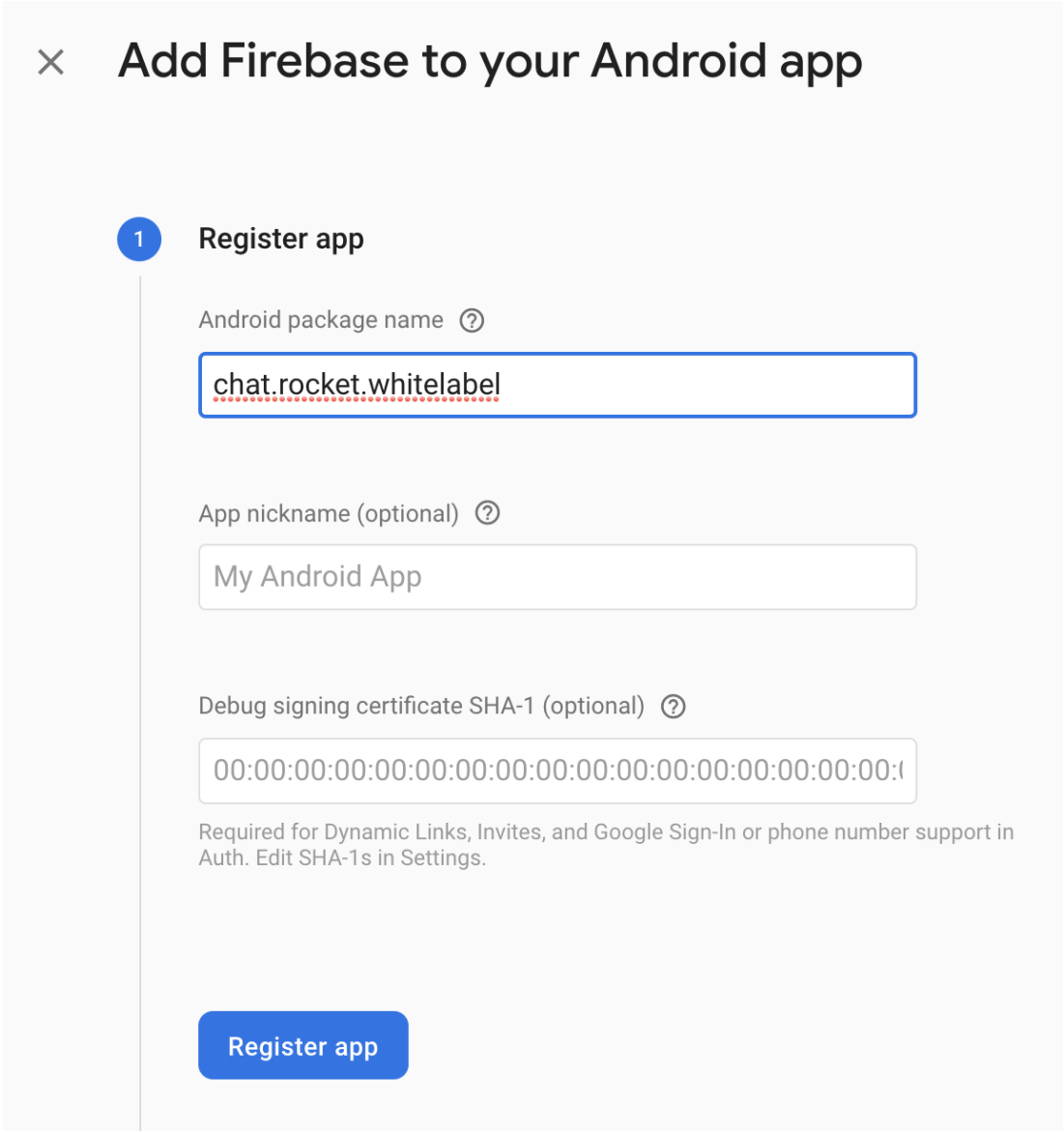


- On “General” tab, click on “Add app” button under “Your apps” section and then “Android”





- Enter your bundle ID and then “Register app”



- Download config file and move it as instructed

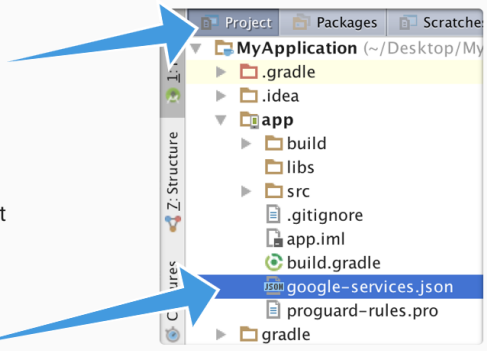
Add Firebase to your Android app

1. **Register app**
   Android package name: chat.rocket.whitelabel

2. **Download config file**
   Instructions for Android Studio below | Unity C++

   Download google-services.json

   Switch to the **Project** view in Android Studio to see your project root directory.

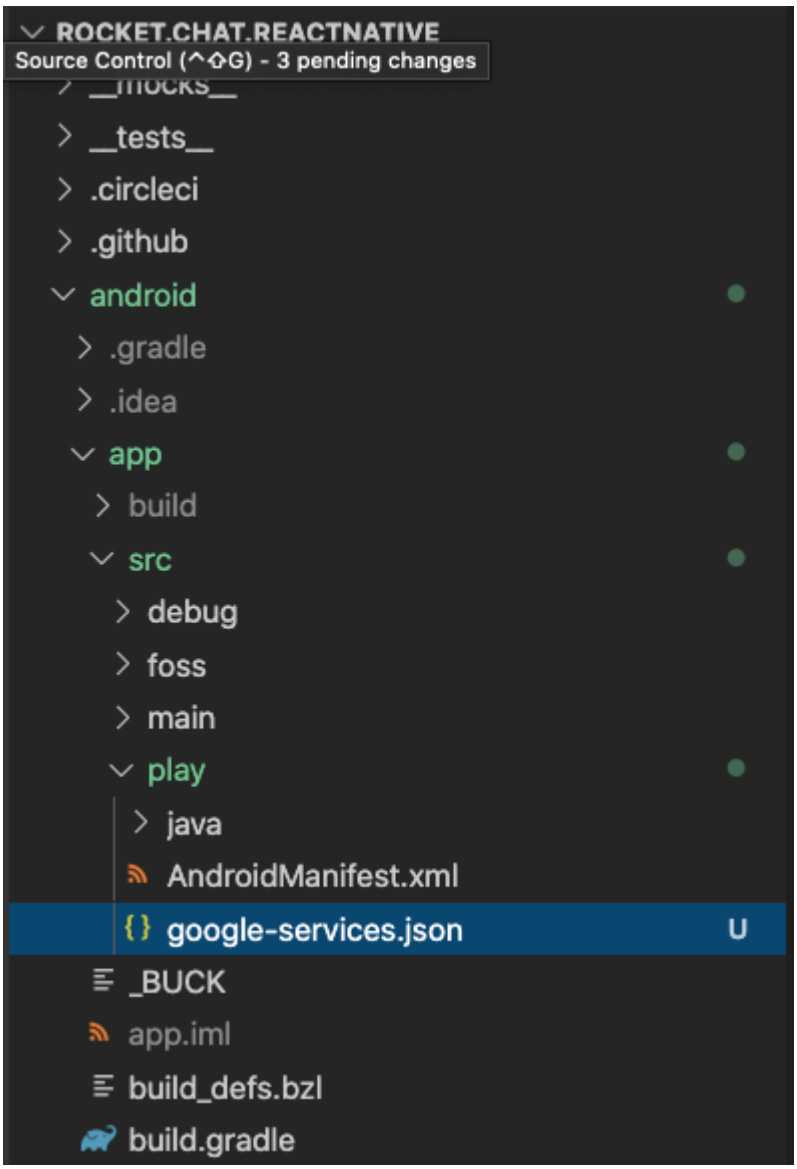
   Move the google-services.json file you just downloaded into your Android app module root directory.

   Previous Next

3. Add Firebase SDK

4. Run your app to verify installation



Running the app

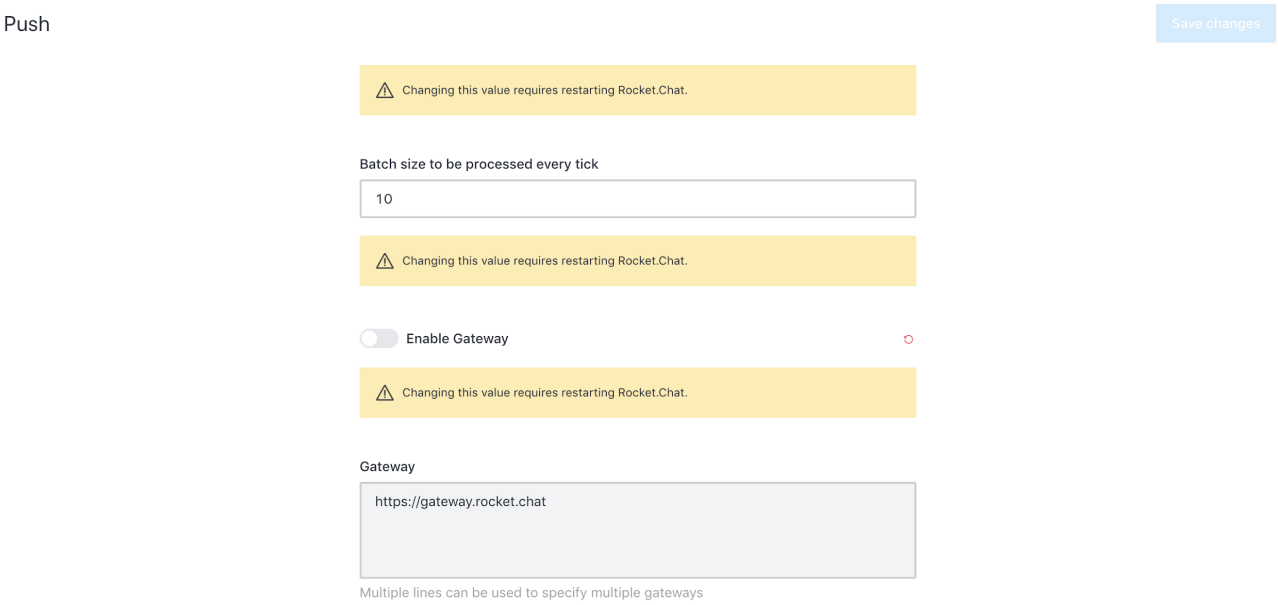
- Execute the following on project terminal
  - `yarn`
  - `yarn android-whitelabel <YOURAPPID>`
  - For example, the app created on this document would use `yarn android-whitelabel chat.rocket.whitelabel`
- Note: this script uses `experimentalPlayDebug` build flavor. When you build your app on release mode, use `experimentalPlayRelease`
  - Refer to https://developer.android.com/studio/build/build-variants for more info about how it works

---

Push notification

Configuring gateway

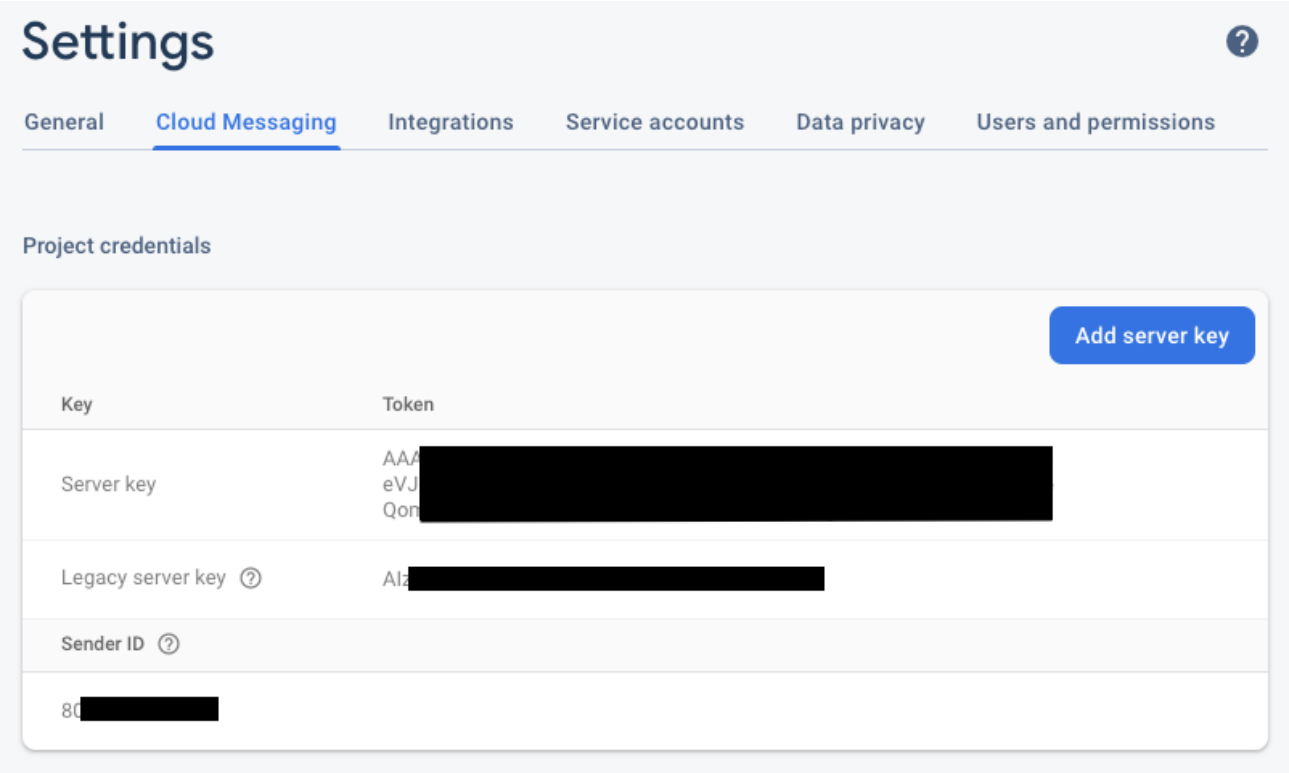
- Go to your Rocket.Chat admin page > Push
- Disable “Gateway” and press “Save changes”



- Also disable “Production”, if you’re trying in debug mode
- Expand “Credentials and Keys” section

Configuring Android

- Go to Cloud Messaging on Firebase settings
- Copy “Server Key” token from Firebase into “GCM API Key”
- Copy “Sender ID” into “GCM Project Number”



Configuring iOS

- Make sure you've done "Creating Push Notifications certificates" first
- In your terminal, go to the folder which contains your push files (CSR, .cer, .p12).

Generating PEM files (Development)

- Execute
  - `openssl x509 -in aps_development.cer -inform der -out DevPushCert.pem`
  - `openssl pkcs12 -nocerts -out DevPushKey.pem -in yourP12File.p12`
- You **must** set a password for your PEM file

Generating PEM files (Production)

- Execute
  - `openssl x509 -in aps.cer -inform der -out PushCert.pem`
  - `openssl pkcs12 -nocerts -out PushKey.pem -in yourP12File.p12`
- You **must** set a password for your PEM file

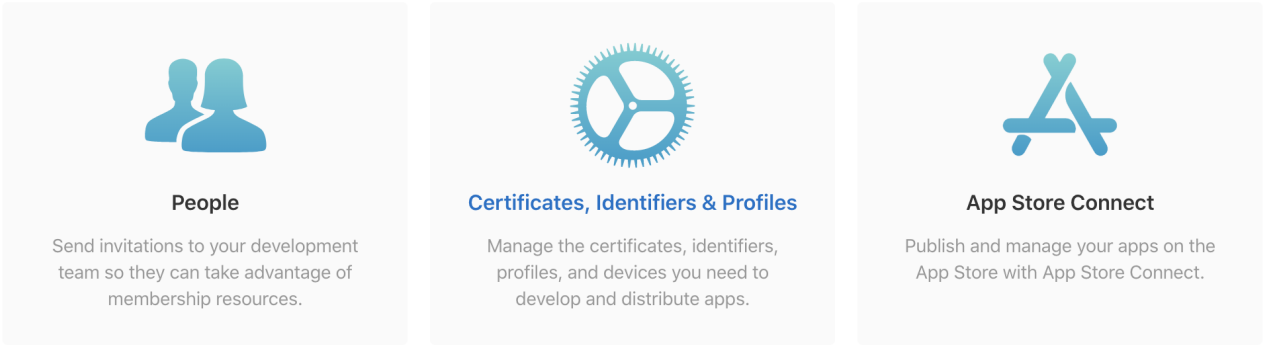
Copying PEM files to Rocket.Chat

- Copy the contents of your development PEM files and password into APN Dev Key, APN Dev Cert and APN Dev Passphrase
- Copy the contents of your production PEM files and password into APN Key, APN Cert and APN Passphrase
- You can use `cat` on terminal to get the content of your PEM files
    - `cat PushKey.pem`
- Save and restart your server
- Log into the server as the same user on your mobile device and close it (it won't receive push notification, if it's open)
- Open Push settings on admin from desktop and click "Send a test push to my user"

Developer Apple

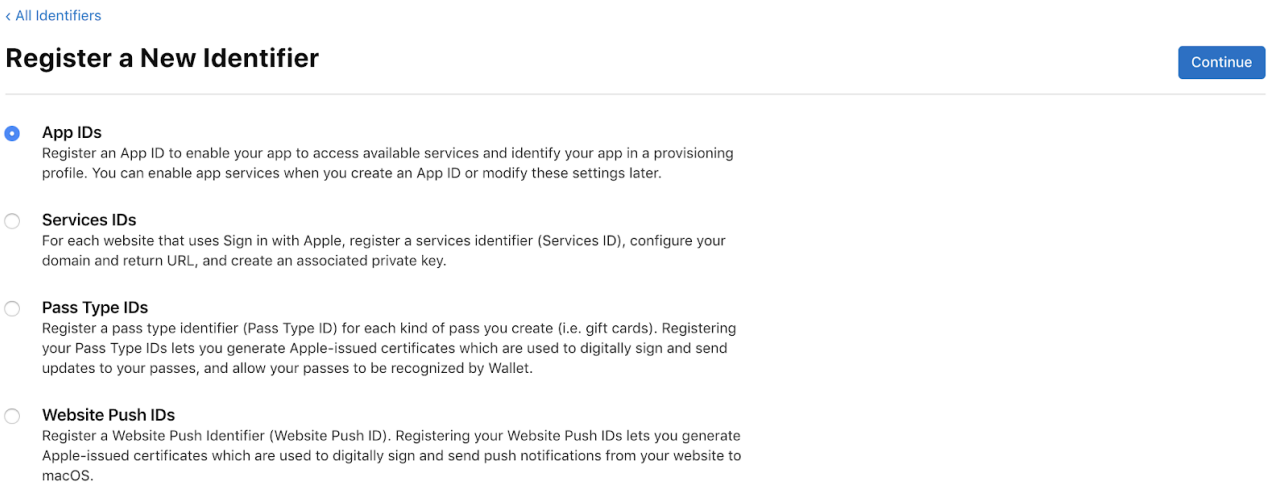
Login to Apple Developer

- Visit https://developer.apple.com/account
- Enter your credentials
- Click on Certificates, Identifiers & Profiles

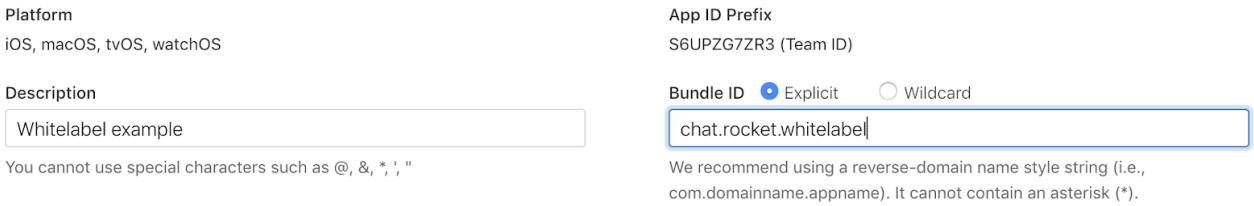


Creating an App Identifier

- Visit https://developer.apple.com/account/resources/identifiers/list
- Click to add Identifier
- Select App IDs and Continue



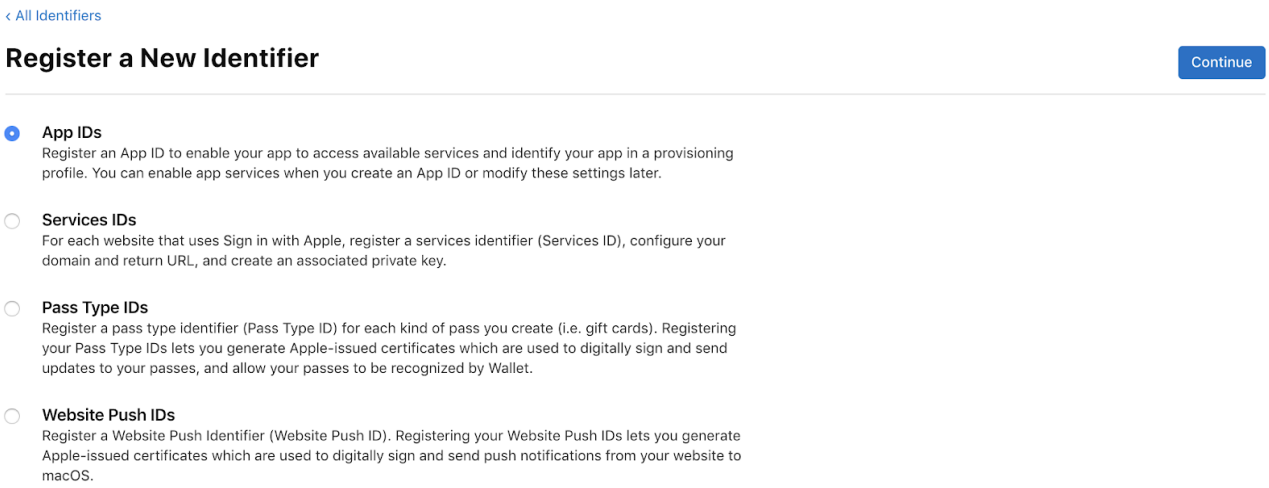
- Add description and Bundle ID



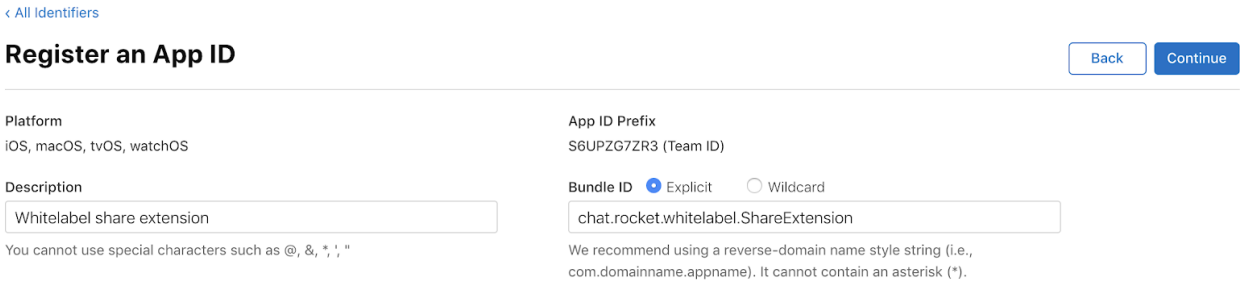
- On Capabilities, select App Groups and Push notifications
- Click “Continue” and then “Register”

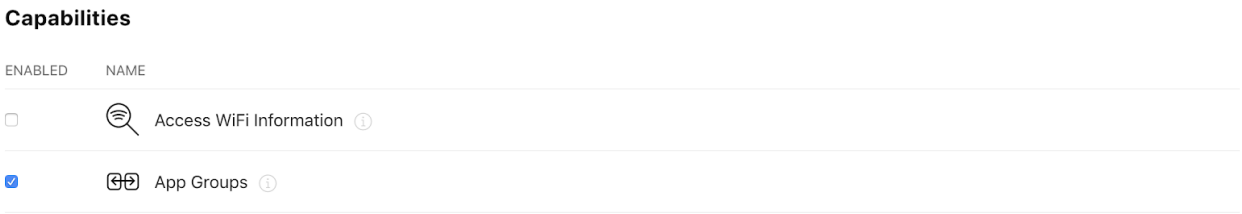
Creating an App Identifier for our Share Extension

- Share Extension is a version of the app that opens when you share data from another app to Rocket.Chat. For example, share a photo from the gallery.
- Visit https://developer.apple.com/account/resources/identifiers/list
- Click to add Identifier
- Select App IDs and Continue



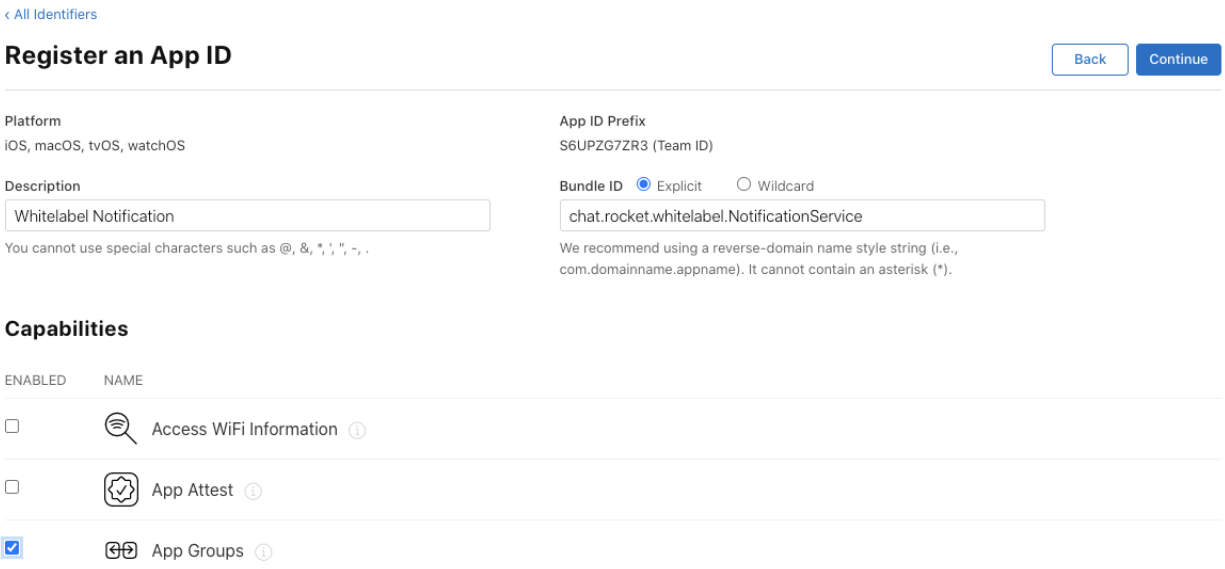
- Add description and Bundle ID





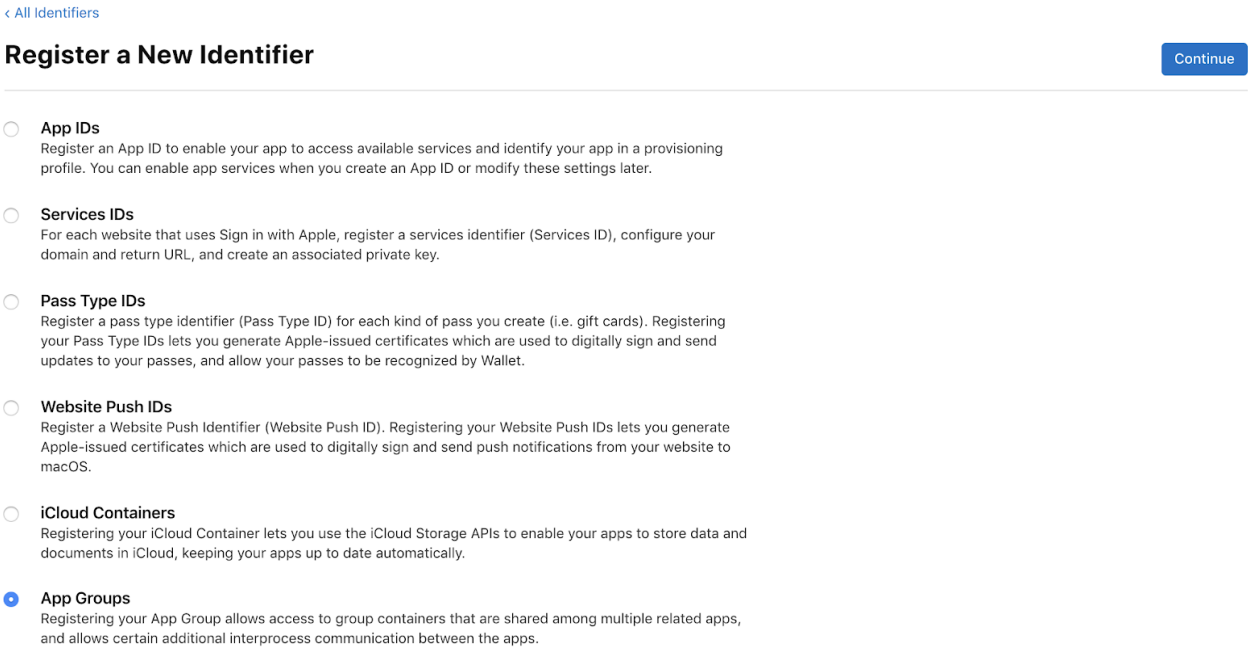
- This time, select only App Groups under Capabilities
- Click “Continue” and then “Register”

Create an App Identifier for our Notification Service

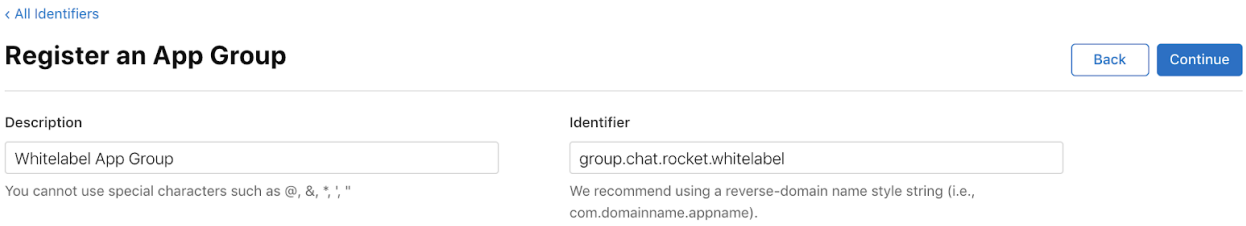


Creating an App Group

- Visit https://developer.apple.com/account/resources/identifiers/list
- Click to add Identifier
- Select App Groups and Continue



- Enter a description and an Identifier



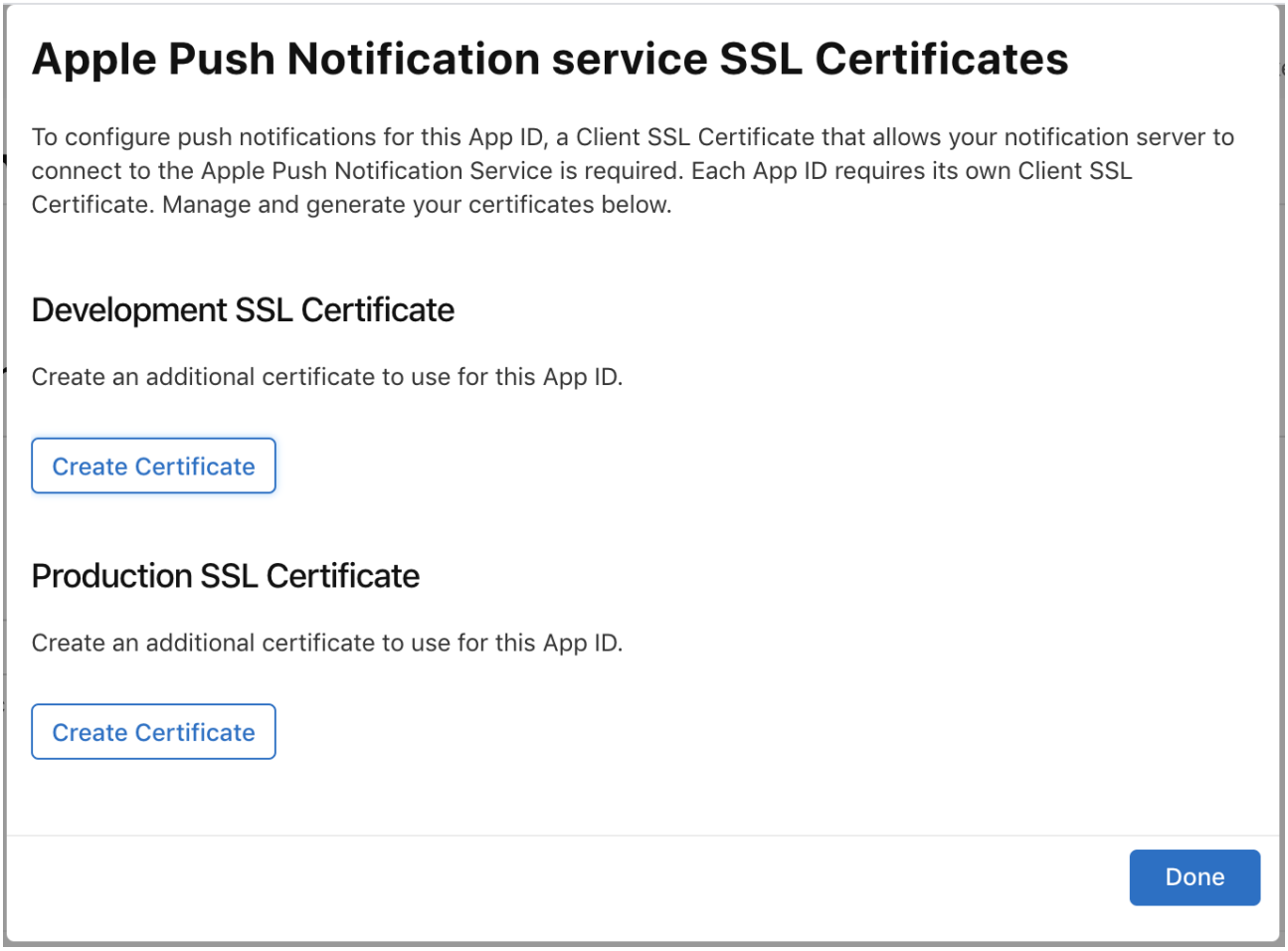
- Click “Continue” and then “Register”

Applying App Group

- Visit https://developer.apple.com/account/resources/identifiers/list
- Click on the first identifier you created
- On “App Groups”, click “Configure”
- Select the App Group you created and click “Continue”
- Click “Save”
- Repeat these steps for the second identifier you created for the Share Extension and NotificationService

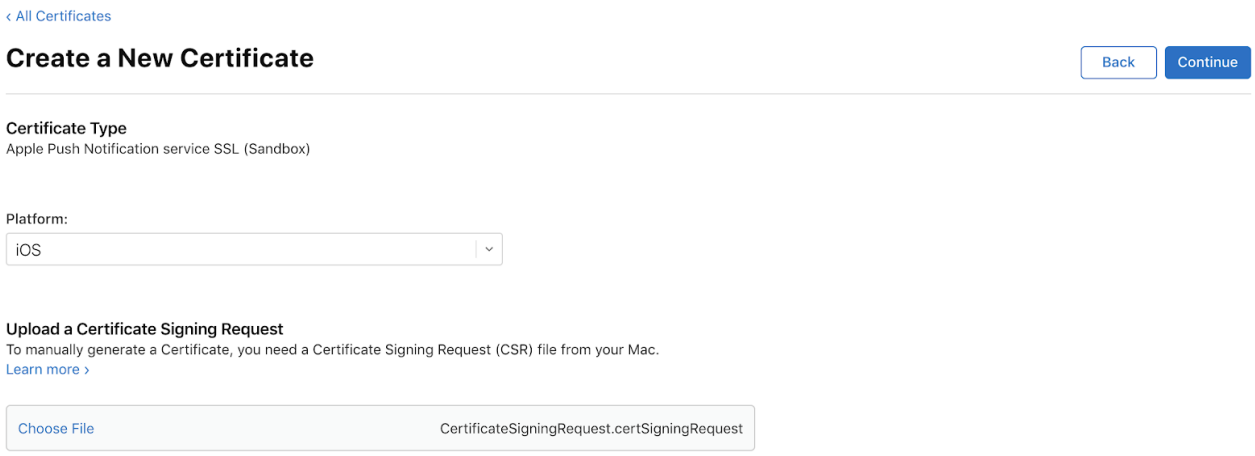
Creating Push Notifications certificates

- Visit https://developer.apple.com/account/resources/identifiers/list
- Click on the first identifier you created
- On “Push Notifications”, click “Configure”

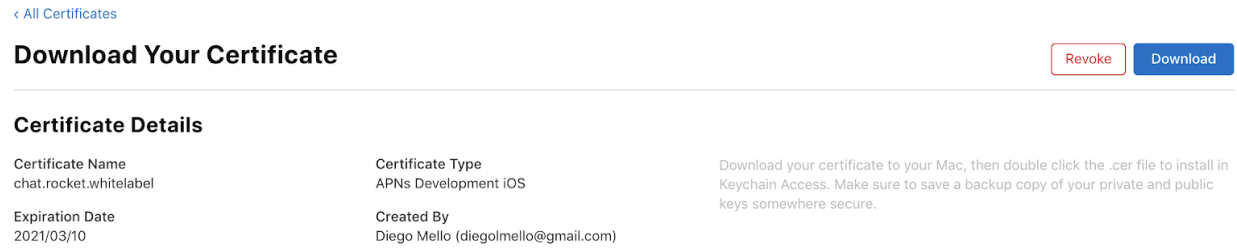


Development SSL Certificate

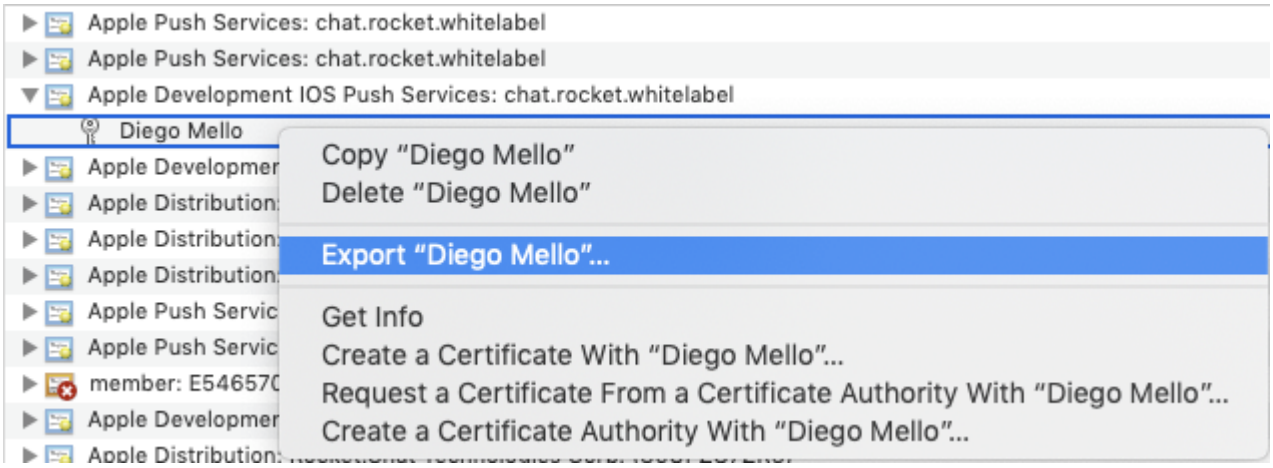
- On “Development SSL Certificate”, click “Create Certificate”
- Follow Apple’s tutorial to generate a Certificate Signing Request: https://help.apple.com/developer-account/#/devbfa00fef7
- Select the certificate you created an click “Continue”



- Download the certificate and install it on your machine (follow instructions on the screen)



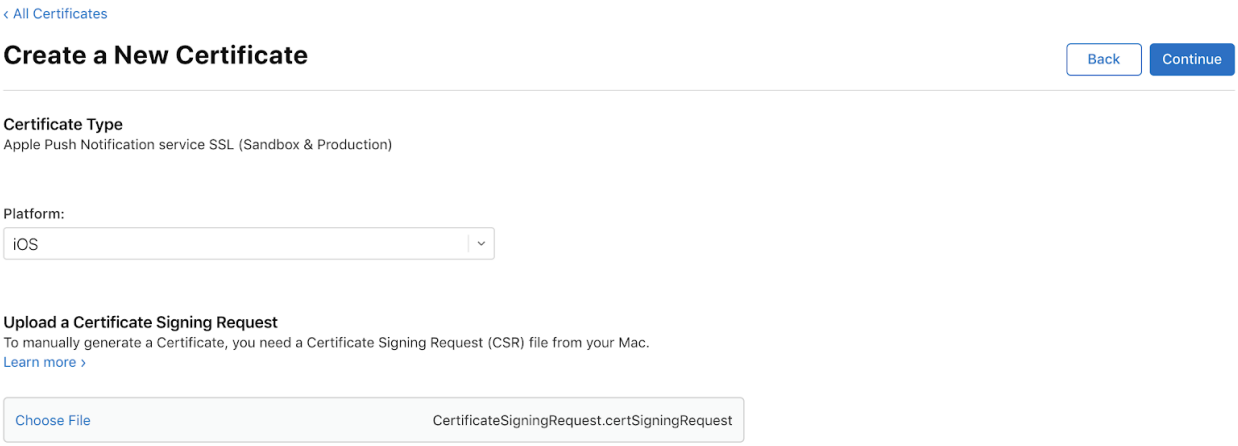
- After installing it, “Keychain Access” should have opened automatically on your Mac
- Export the certificate to generate a .p12 file



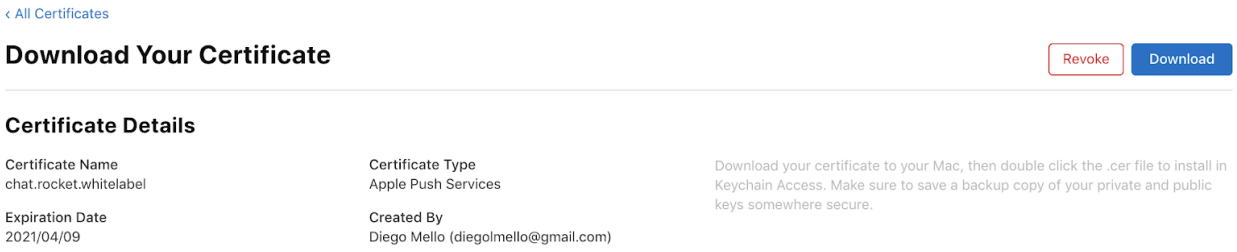
- For simplicity, save it in the same folder of your CSR and .cer. You’ll need it later.

Production SSL Certificate

- On “Production SSL Certificate”, click “Create Certificate”
- Follow Apple’s tutorial to generate a Certificate Signing Request: https://help.apple.com/developer-account/#/devbfa00fef7
- Select the certificate you created an click “Continue”



- Download the certificate and install it on your machine (follow instructions on the screen)



- After installing it, “Keychain Access” should have opened automatically on your Mac
- Export the certificate to generate a .p12 file

- For simplicity, save it in the same folder of your CSR and .cer. You’ll need it later.

Supporting SSL for development on Rocket.Chat

If you are working with mobile apps, it is required that your server supports SSL.

Rocket.Chat is a "middle-tier application server", by itself it does not handle SSL. However, Rocket.Chat works well with several industrial grade, battle-tested, reverse proxy servers that you can configure to handle SSL.

You should find yourself in one of the two situations:

- The Rocket.Chat server is publicly accessible on the internet.
- The Rocket.Chat server is not accessible on the internet.

This doc has been broken down into two separate sections, walking you through either of the camps you might find yourself in.

The Rocket.Chat server is publicly accessible on the internet

If your server is publicly accessible, it is recommended that you use a service like Let's Encrypt to obtain your SSL certificates. A detailed guide for configuring your choice of SSL Reverse proxy servers is provided here: Configuring SSL Reverse Proxy

If you are using Ubuntu, this can be configured automatically with the help of Snaps. A guide for which is provided here: Installing Rocket.Chat on Ubuntu with Snaps

---

The Rocket.Chat server is not accessible on the internet

If your server is not accessible on the internet, you will need to provide self signed certificates to configure SSL on the server.

In this doc, we will be creating a self signed root certificate and using it to generate our SSL certificates. The steps written below have been adapted from Self Signed Certificate with Custom Root CA

Step 1: Create Root CA

Create Root Key

Attention: This is the key used to sign the certificate requests, anyone holding this can sign certificates on your behalf. So keep it in a safe place!

```
openssl genrsa -des3 -out Rocket.Chat-root.key 4096
```

If you want a non password protected key just remove the `-des3` option

Create and self sign the Root Certificate

```
openssl req -x509 -new -nodes -key Rocket.Chat-root.key -sha256 -days 1024 -
```

Here we used our root key to create the root certificate that needs to be distributed in all the computers that have to trust us.

NOTE: It is **not recommended** that you distribute this root certificate in production. A breach of the above-generated key will open every device that trusts your root certificate to ***potential security threats***.

Step 2: Create an SSL certificate

Create the certificate key

```
openssl genrsa -out mydomain.com.key 2048
```

Here, mydomain.com should be replaced with your IP address (Bonjour local domains work as well!)

Create the certificate signing request

Important: Please mind that while creating the certificate signing request is important to specify the `Common Name` providing the IP address or URL for the service, otherwise the certificate cannot be verified.

```
openssl req -new -key mydomain.com.key -out mydomain.com.csr
```

Generate the SSL certificate

Here, we are using the `mydomain.com` CSR along with the `Rocket.Chat-root` CA.

```
openssl x509 -req -in mydomain.com.csr -CA Rocket.Chat-root.crt -CAkey Rocke
```

Step 3: Configuring SSL for Rocket.Chat

The `mydomain.com.crt` and `mydomain.com.key` files generated above will be used as the certificate and the private key to configure SSL.

A detailed guide for configuring your choice of SSL Reverse proxy servers is provided here: Configuring SSL Reverse Proxy

Step 4: Trusting Certificate Authority

All the devices that need to communicate with the server during development, need to trust the root certificate we generated in `Step 1` (`Rocket.Chat-root.crt`)

- For Apple devices follow the instructions here: HTTPS and Test Servers
- For Android devices follow the link Add & remove certificates, and scroll down to "Work with CA certificates (trusted credentials)".

Installation instructions for other operating systems can be easily found online.

On successful installation of the root certificate, the device should be able to access Rocket.Chat over SSL.

Troubleshooting

If your device is not able to connect over SSL, please make sure that the URL has `https://` explicitly typed out before it.

Analytics & Data Usage

Crashlytics and Bugsnag

Both iOS and Android applications are using Crashlytics and Bugsnag to send crash information. Both platforms collect crash information from mobile apps, transmit and store them securely on their servers for developer diagnosis. This transmitted data contains no users, channels, or groups information; it does not contain any message content. The collected content contains only anonymous application and system state information during the crash that can be helfpul for diagnosis.

These reports can be disabled on both iOS and Android if the user goes to the Settings screen and disable crash reports.

Firebase Analytics

We use Firebase Analytics report events of usage in the app for some actions, such as sending messages, reacting, changing the theme of the app, etc. This is also anonymous information and does not contain any private information from the user. This is only being used with the purpose of understanding what features are being used more and how they're being used.

These reports can be disabled on both iOS and Android if the user goes to the Settings screen and disable crash reports.

UI and Theming

> NB: Theming for Rocket.Chat is an incomplete feature. You can follow discussion on the approach under issue #277

- How to create custom themes
- How to use UI color scheme
- How to use UI components

UI Components

This article stub needs descriptive content and example usage for each component declared in Rocket.Chat UI to improve ease of development and prevent furthering multiple components being contributed for the same purpose with conflicting styles and usage. Can you help?

Example content:

Side Nav

Describe usage for template development contributions

Provide example usage and html structure / class names of nested elements

- Content
  - Admin Link
  - Input Line
  - Button
- Account Box
  - Status
  - Account Link
- Rooms List
  - Header
  - Item
  - Item List

Custom Themes

Theming for Rocket.Chat is an incomplete feature and we encourage developers to contribute to this issue. We'd love to hear from anyone working on themes in the #skins-and-theming group channel.

To customize the Rocket.Chat UI you can either modify the `rocketchat-theme` or `rocketchat-ui` packages directly, but if you're keeping in sync with active development it would be easier to avoid conflicts by creating your own theme package.

Creating a Theme

You can add theme customisations to Rocket.Chat by just creating a Meteor package with your code, then adding it to the packages file.

Private themes would need to be maintained on your own fork of Rocket.Chat, but public themes could be published as a Meteor package outside the Rocket.Chat repo.

The minimum contents for a theme package would be a `package.js` file containing the description, e.g:

```
Package.describe({
    name: 'author:mytheme',
    version: '0.0.1',
    summary: 'My theme customisations.',
    git: 'https://github.com/author/my-rocket.chat-theme'
});
```

Then include dependent packages and your custom theme files. e.g:

```
Package.onUse(function(api) {
    api.versionsFrom('1.2');
    api.use([
```

```
        'templating',
        'rocketchat:lib',
        'rocketchat:theme'
    ]);
    api.use('templating', 'client');
```

Adding Stylesheets

The `rocketchat-theme` package has methods for including Less asset files in the build. Less files (and the a server.coffee or .js file to load them) must first be included in the `package.js` manifest (within the `Package.onUse` function), e.g.:

```
    api.addAssets([
        'assets/theme.less'
    ], 'server');
    api.addFiles([
        'server.coffee'
    ], 'server');
```

Then in `server.coffee` ...

```
RocketChat.theme.addPackageAsset -> Assets.getText 'assets/theme.less'
```

That will read in any styles and variables from your custom less file and compile it with the rest of the css.

Adding and Modifying Templates

A suggested approach for including custom templates and helpers is to use the `'aldeed:template-extension'` package (include it in your main packages file). In your

package manifest, declare use of the `template-extension` package, then add your template files to Meteor `api.addFiles([myfiles], 'client')`.

Here's an example replacing the unauthorized page template:

In `package.js`

```
    api.addFiles(['views/notAuthorized.html', 'client.coffee'], 'client');
```

In `views/notAuthorized.html`

```
<template name="myNotAuthorized">
    <h2>My Custom Not Authorized Page</h2>
</template>
```

In `client.coffee`

```
Template.myNotAuthorized.replaces 'notAuthorized'
```

See the docs for that package for more info on inheriting and overwriting templates and helpers.

UI Colors

> NB: The UI is in active development and component refactoring will change (and hopefully improve) the use of color. The immediate goal is to consolidate use of color and styles, before moving forward to a consistent style guide for Rocket.Chat.

Most* use of color in Rocket.Chat can be customised by changing color settings under Administration > Layout > Colors. We encourage developers to use the defined variables in their contributions (instead of hard-coding colors), to allow site owners to change the color scheme with consistent results.

See the theme variables file for current available color settings. All color settings are available to Less files as variables, as long as the Less files were compiled using the `addPackageAsset` method of `rocketchat-theme`.

Color Scheme

The Rocket.Chat color scheme consists of three groups of color settings, Alpha, Major and Minor Colors.

Some further variations of these colors are created in Less and not exposed to settings.

The naming of color settings/variables is not related to any specific component, the names reflect the visual hierarchy that will (hopefully) make it obvious and easy to carry forward consistent color usage in newly contributed components and theme development.

Alpha Colors

Semi-transparent black or white, used in components to shade/tint the background color, e.g. to indicate a selected or disabled state. The use of alpha colors allows site owners to easily change color scheme without defining every variation for every state of a component.

- transparent-dark
- transparent-darker

- transparent-light
- transparent-lighter

Alpha Colors Example

Alpha colors example colors

Major Colors

The primary palette of the app. Contributions and modifications to components should make use of these colors.

- content-background-color **#FFFFFF**
- primary-background-color **#04436A**
- primary-font-color **#444444**
- primary-action-color **#1d74f5**
- secondary-background-color **#F4F4F4**
- secondary-font-color **#A0A0A0**
- secondary-action-color **#DDDDDD**
- component-color **#f2f3f5**
- success-color **#4dff4d**
- pending-color **#FCB316**
- error-color **#BC2031**
- selection-color **#02ACEC**
- attention-color **#9C27B0**

Minor Colors

A set of minor colors for specific use cases will inherit from the major colors by default but can be used by admins who want more granular control over the color scheme.

- tertiary-background-color *defaults to component-color*
- tertiary-font-color *defaults to transparent-light*
- link-font-color *defaults to primary-action-color*
- info-font-color *defaults to secondary-font-color*
- custom-scrollbar-color *defaults to transparent-dark*
- status-online *defaults to success-color*
- status-away *defaults to pending-color*
- status-busy *defaults to error-color*
- status-offline *defaults to transparent-darker*

Dark UI

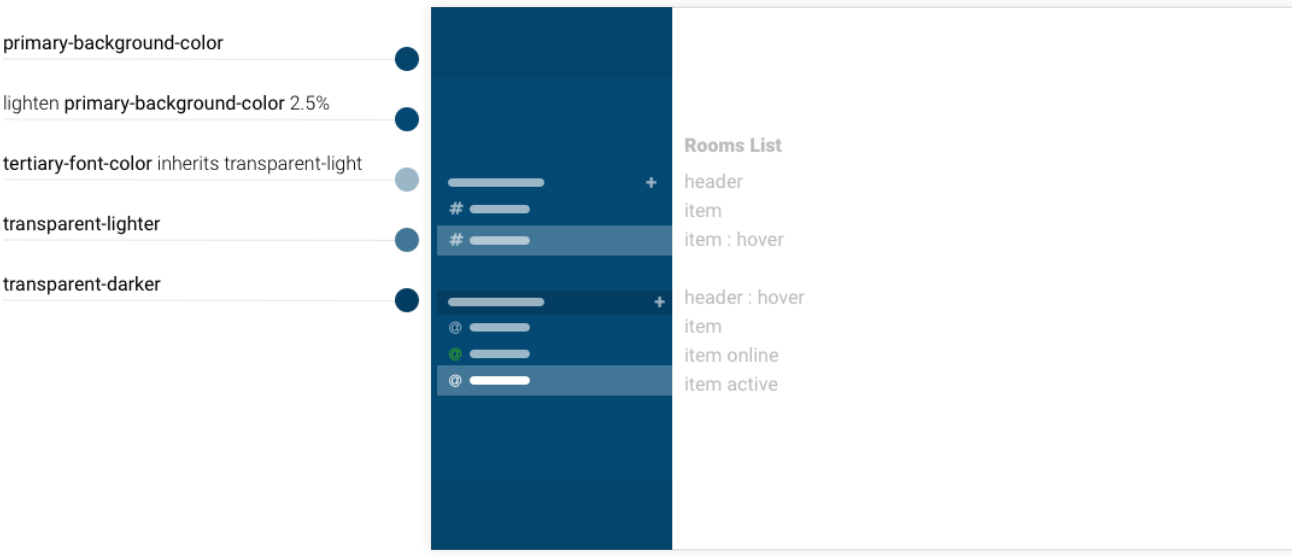
The computed colors allow owners to choose a **dark UI** with appropriate contrast. e.g. in a light UI, the disabled state might darken an element, but on a dark UI it should be lightened. The mixins achieve this by mixing the color with a contrast of the background color instead of using darken/lighten. See this example of form input states that dynamically contrast to both dark and light backgrounds.

Developers are encouraged to use mixins and computed colors in contributions instead of hard-coding variations, to maintain consistent balance and contrasts of colors regardless how the scheme settings may be changed by owners.

Default Colors

These examples show the implementation of the default color scheme with the main components of the Rocket.Chat UI.

Side Nav

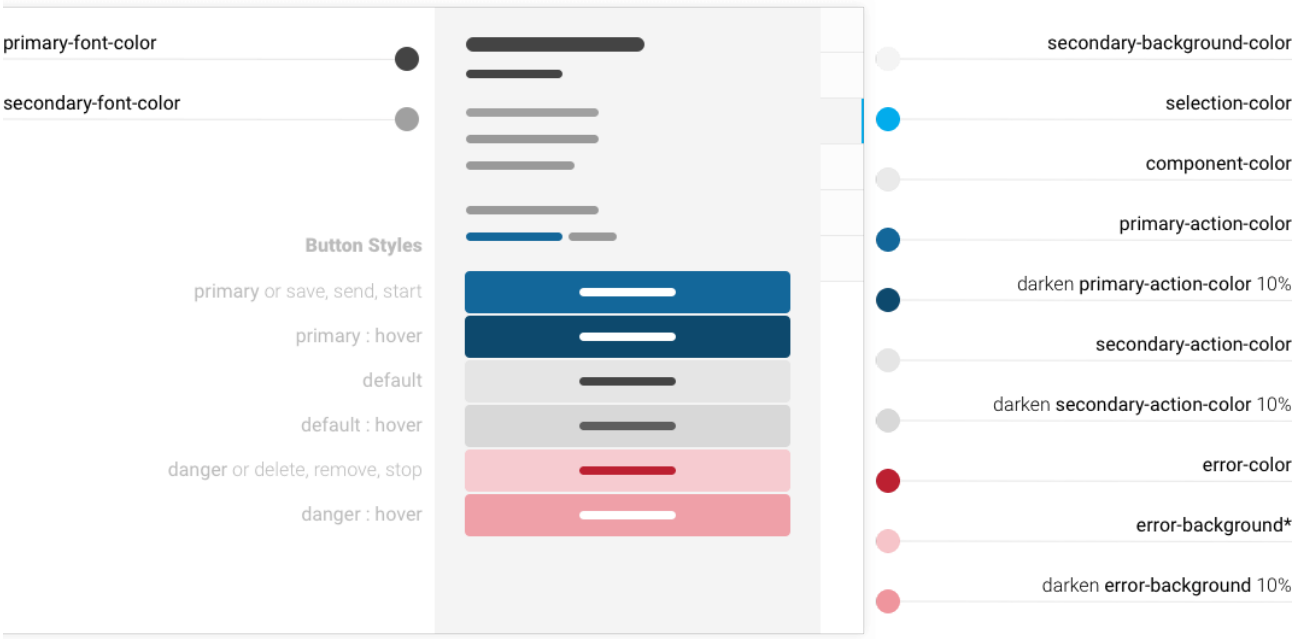


Side nav example colors

Account Box

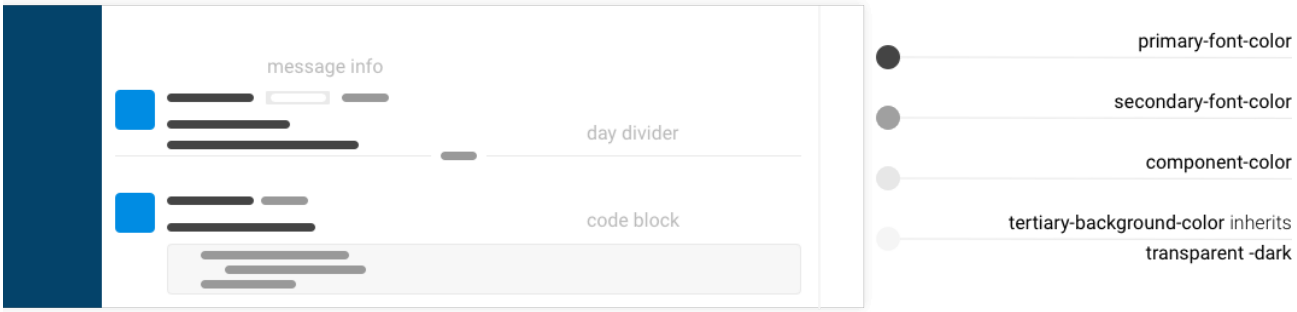
Account Box example colors

Flex Nav



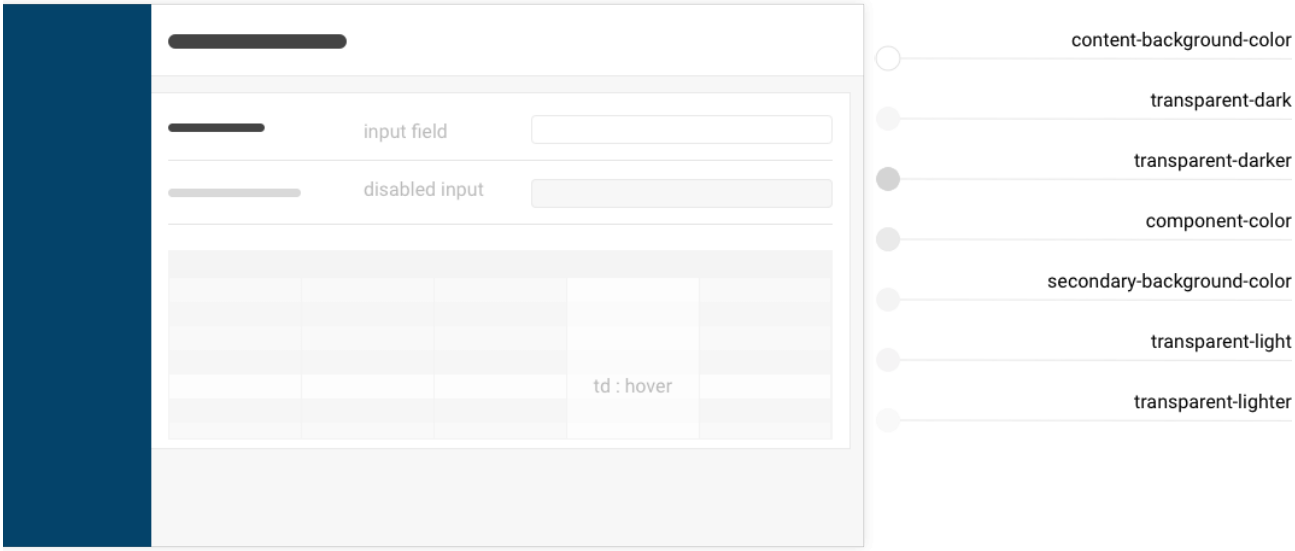
Flex nav example colors

Message Box



Message box example colors

Settings Page



Settings page example colors

Deeplink

General information

Our mobile and desktop clients have implemented deeplinking that will allow them to handle links meant for Rocket.Chat.

This is done using a special url that the applications are registered to handle.

There are two prefixes available:

- `https://go.rocket.chat`
- `rocketchat://`

We call `https://go.rocket.chat` links go links. Links here will work across web, mobile and desktop clients.

The alternative will only be handled by mobile and desktop clients.

In all examples below `https://go.rocket.chat/` is interchangeable with `rocketchat://`

Authentication

These links can be used for adding a server to your client in one click. This makes it easier for deployment. You then have a universal link that will add to the clients.

You can also include credentials to make it add the server and auto log them in.

Authentication links start with: `https://go.rocket.chat/auth`

Params:

- host: The host of the server;
- token (optional): The token of the user to be authenticated;

- userId (optional): The id of the user to be authenticated;

Examples:

Link directly to server:

```
https://go.rocket.chat/auth?host=open.rocket.chat
```

Link directly to server and authenticate:

```
https://go.rocket.chat/auth?host=foo.bar.com&token=123abc&userId=1234abcd
```

Channel / Group / DM

You can also link directly to a room.

These links start with: `https://go.rocket.chat/room`

Params:

- host: The host of the server;
- rid: The rid of the room to be opened;
- path (optional): The path URL to be opened on the web;

Example:

```
https://go.rocket.chat/room?host=open.rocket.chat&rid=GENERAL&path=channel/g
```

Testing

Requirements

- Google Chrome Browser

Getting Started

Start Meteor

Run meteor with the command below:

```
TEST_MODE=true meteor
```

To run the tests, the server **must** be started with the environment variable `TEST_MODE=true`. This will set all animations' duration to 0 and create an extra admin user with the login values:

```
_id: "rocketchat.internal.admin.test"
name: "RocketChat Internal Admin Test"
username: "rocketchat.internal.admin.test"
emails: "rocketchat.internal.admin.test@rocket.chat"
password: "rocketchat.internal.admin.test"
```

Run Tests

On another terminal window, run the test with the command bellow:

```
meteor npm run test
```

Deprecation

The methods and endpoints of the **Realtime API** and **Rest API**, respectively, that have been depreciated are listed below.

Realtime API

Method	Release deprecated	Release removed
`cleanChannelHistory`	0.64.0	0.67.0

REST API

Endpoint	Release deprecated	Release removed
`/user.roles`	0.63.0	0.66.0
`/channels.cleanHistory`	0.64.0	0.67.0
`/permissions`	0.66.0	0.69.0
`/permissions.list`	0.73.0	1.11.0
`/emoji.custom`	1.0.0	1.12.0
`/v1/info`	1.0.0	1.12.0

Embedded Layout

If you are embedding Rocket.Chat in your site, you can change Rocket.Chat's layout by adding `?layout=embedded` after your server URL.

Example https://open.rocket.chat/channel/general?layout=embedded

Adding this to the URL will change the layout of Rocket.Chat to a more "simplistic" view, hiding the left side bar with the channel lists and account management buttons. Adittionaly, you can enable the top navbar in the embedded layout by going to "Administration -> Layout -> Interface -> Show top navbar in embedded layout".

Setting a URL to embedded mode will "lock" that view in a single channel (since the user will not have access to the channel lists).

Embedded layout works wonders when using with Iframe integration since you can already login the user using iframe and only show a desired channel embedded to a page.

You can test this feature by going to https://open.rocket.chat entering a channel and adding `?layout=embedded` to the URL.

Note: If you want to stop users from accessing other channels, embedded layout alone will not work, since the user could change the URL of the embedded view via browser tools. For that you should edit the user permissions so they can't see other channels

Iframe Integration

Use your own login page to login users in Rocket.Chat

If you want to authenticate users using your own login page in place of the Rocket.Chat's login page via the iframe integration, go to `Administration > Accounts > Iframe` and enable it:

- How to use iframe integration for authentication

Use Rocket.Chat in your site/app inside an iframe

If you need listen to events, go to `Administration > General > Iframe Integration` and enable send (events) or receive (commands), depending on your needs:

- How to use iframe integration events
- How to use iframe integration commands
- Using embedded layout mode

What is iframe auth?

With `iframe` auth you can use your own authentication page/API to log in users on Rocket.Chat.

When enabled Rocket.Chat first do an `XMLHttpRequest` to the `iFrame API` URL trying to see if the user is already logged in at the third party website. If that doesn't succeed then Rocket.Chat will present the `Iframe URL` within an `iframe`, so the user logs in on the third party website which means he is authenticated on Rocket.Chat as well.

We have developed an example app written in NodeJS in order to help you understanding this authentication flow. Please take a look at iFrame Auth Example

---

Configuring

API URL and API Method

Configure how Rocket.Chat will call the third party system to either login or to verify if the user is already logged in, by setting `API URL` and `API Method` fields.

`API URL` refers to endpoint on the third-party system that will check if the user is already logged in to that system. The `API Method` is used to select the submission method Rocket.Chat will use to submit information to the `API URL` (for instance using `POST`).

If the user has already logged into the third-party system, the `API URL` should communicate to Rocket.Chat and return a JSON object containing either a `token` or `loginToken` property, otherwise (if the user is not already logged in) the `API URL` should return an empty body with status `401`.

The choice of which property `API URL` will return depends on how the third-party system decides to interface back with Rocket.Chat, as described in one of the two ways below:

Using Rocket.Chat API

If you have the user's password stored (or it is the same between your third party system and Rocket.Chat), you can use Rocket.Chat's REST APIs to log in the user, this way you will get an `authToken` back from Rocket.Chat that should be returned as `loginToken` by your endpoint.

At this point, if the user does not have a Rocket.Chat account yet, you can either use Rocket.Chat API to create an user using a admin account or register him.

After you log the user in, you should return a payload like the following:

```
{
  "loginToken": "already-saved-or-returned-login-token"
}
```

Managing MongoDB directly

In the case you have access to Rocket.Chat's database, you can connect there directly and manage the user record by yourself. This might be useful if you have MongoDB on your stack already and don't want to learn Rocket.Chat's API.

To do so the endpoint should connect on Rocket.Chat's MongoDB database and make sure the `generated-token` is saved on `users` collection on the corresponding user record. The `generated-token` should be saved on the field path `services.iframe.token` . This is how the user record should look like:

```
{
  "_id": "MZiFvWAf96876875u",
  "createdAt": new Date(1432252673528),
  "services": {
    "iframe": {
      "token": "generated-token"
    }
  },
  "emails": [
    {
      "address": "useremail@gmail.com",
      "verified": true
    }
```

```
  ],
  "name": "John Doe",
  "username": "john.doe",
  "active": true,
  "statusDefault": "online",
  "roles": [
    "user"
  ],
  "type": "user"
}
```

On this case, the response should be:

```
{
  "token": "generated-token"
}
```

IFrame URL

The URL of the page you want to show as the login page of your Rocket.Chat instance (this page can be created in any programming language and/or web framework).

The login page will then communicate back to Rocket.Chat using `postMessage` API.

After user logs in, you have to authenticate him on Rocket.Chat side, pretty much the same as you did before on `API URL` endpoint, but now you should return a JavaScript code that will be rendered within the `iframe`, depending how you logged in the user:

- If have used Rocket.Chat's APIs to log in the user or already have user's token saved in your end, return:

```
<script>
window.parent.postMessage({
  event: 'login-with-token',
  loginToken: 'your-token'
}, 'http://your.rocket.chat.url');
</script>
```

- If you have saved user's token connecting directly to Rocket.Chat's database on the user's field `services.iframe.token`:

```
<script>
window.parent.postMessage({
  event: 'try-iframe-login'
}, 'http://your.rocket.chat.url');
</script>
```

Using OAuth configured on Rocket.Chat's end

If you have OAuth services configured on Rocket.Chat, you can trigger them from within your login page as well.

To implement this authentication, after triggering the OAuth authentication you will receive a `postMessage` back from Rocket.Chat with user's credentials response from OAuth service. You need to manage the user creation/authentication on Rocket.Chat's database by yourself, the same as described earlier.

Facebook

```
window.parent.postMessage({
  event: 'call-facebook-login',
  permissions: ['email']
}, 'http://your.rocket.chat.url');
```

The reply will be a postMessage back to your page with:

```
{
  event: 'facebook-login-success',
  response: {
    // authResponse: Object
    // accessToken: "a7s6d8a76s8d7..."
    // expiresIn: "5172793"
```

```
    // secret: "..."
    // session_key: true
    // sig: "..."
    // userID: "675676576"
    // status: "connected"
  }
}
```

Or an error

```
{
  event: 'facebook-login-error',
  error: error,
  response: response
}
```

Google

```
window.parent.postMessage({
  event: 'call-google-login',
  //  scopes:
  //  webClientId:
}, 'http://your.rocket.chat.url');
```

The reply will be a postMessage back to your page with:

```
{
  event: 'google-login-success',
  response: {
    // "email": "rodrigoknascimento@gmail.com",
    // "userId": "1082039180239",
    // "displayName": "Rodrigo Nascimento",
    // "gender": "male",
    // "imageUrl": "https://lh5.googleusercontent.com/-shUpniJA480/AAAAAAA
    // "givenName": "Rodrigo",
    // "familyName": "Nascimento",
    // "ageRangeMin": 21,
    // "oauthToken": "123198273kajhsdh1892h"
```

```
  }
}
```

Or an error

```
{
  event: 'google-login-error',
  error: error
}
```

Twitter

```
window.parent.postMessage({
  event: 'call-twitter-login'
}, 'http://your.rocket.chat.url');
```

The reply will be a postMessage back to your page with:

```
{
  event: 'twitter-login-success',
  response: {
    // "userName": "orodrigok",
    // "userId": 293123,
    // "secret": "asdua09sud",
    // "token": "2jh3k1j2h3"
  }
}
```

Or an error

```
{
  event: 'twitter-login-error',
  error: error
}
```

How to login in Rocket.Chat with default account system while in development

When you activate the IFrame auth you will not be able to access Rocket.Chat's default login page, however if still need/want to use your Rocket.Chat's credentials to log in, you can do that by opening the browser's Developer Console and executing the following code:

```
Meteor.loginWithPassword('username-or-email', 'your-password');
```

Testing the iFrame Authentication

This article will describe how to test the iFrame authentication, using a test tool developed by Rocket.Chat team. This test scenario is based in a localhost environment.

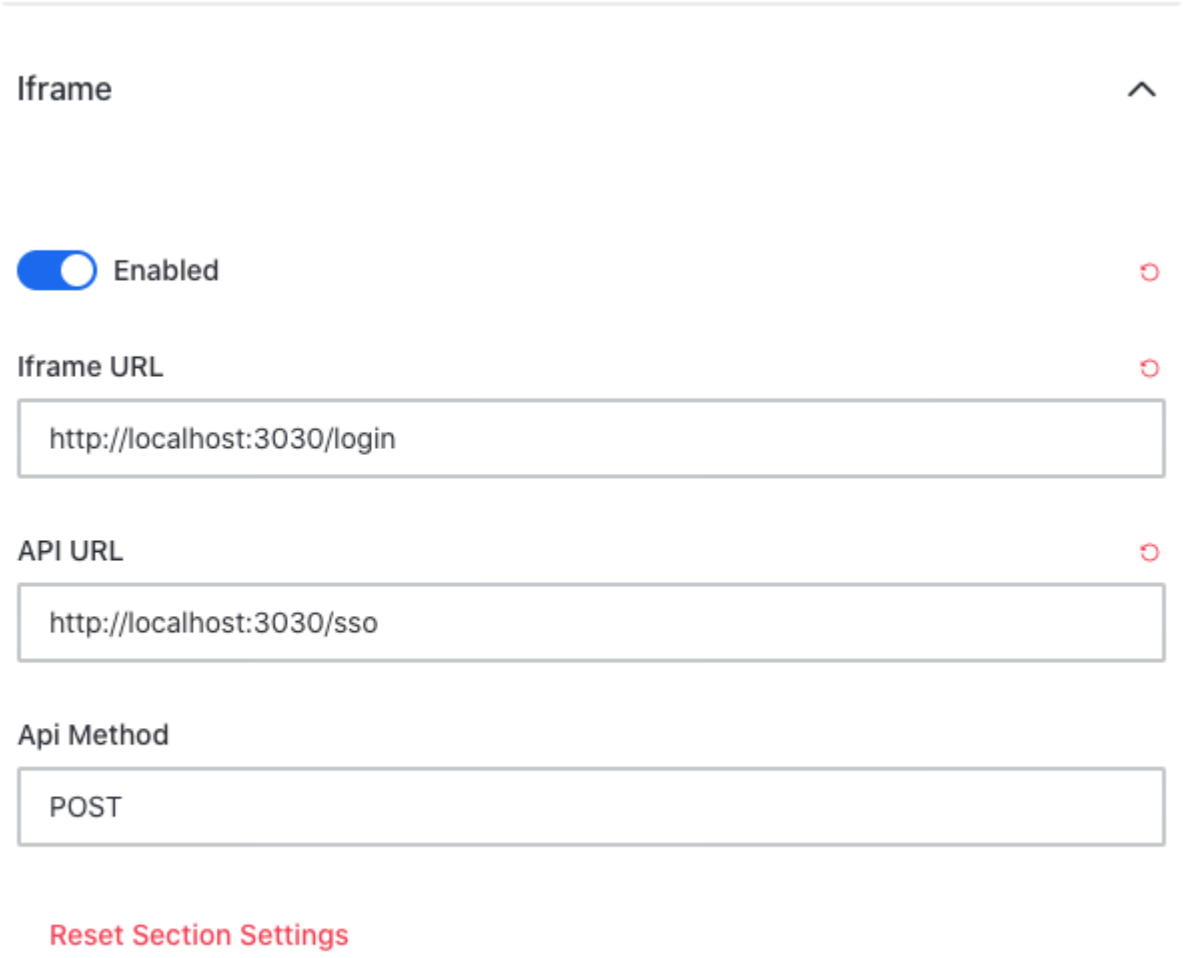
Step 1

Download and start the *iframe-auth-example* test service following the steps mentioned here

The test service code can be used as an example on how to setup the iFrame auth calls to Rocket.Chat. Code is available here.

Step 2

Configure Rocket.Chat server (*Administration > Accounts > iFrame*) as follows:



Step 3

Test the iFrame service by calling Rocket.Chat login URL (in this example, http://localhost:3000\)

In this moment, the *iframe-auth-example* service runs and the iframe authentication calls are executed.

By default, the test service code expects the login to be done with the following credentials (harcoded)

`username: new-user`

`password: new-users-passw0rd`

Change the `currentUsername` (line 105) to `true` so you can login again with the same user. By changing the code you may use any user which already exists in Rocket.Chat

Further reference can be found here.

Iframe Events

Events sent

Here is a list of events triggered from Rocket.Chat when the iframe integration is enabled:

Event name	Description
`notification`	Fired when a user receives a notification
`unread-changed-by-subscription`	Fired each time a user's subscription record changes (i.e.: unread counts, etc)
`unread-changed`	Fired when the pages title changes
`room-opened`	Fired when a room is opened
`new-message`	Fired every time the opened room receives a new message
`click-user-card-message`	Fired when the user clicks on a username link
`click-mention-link`	Fired when the user clicks on a mention link
`click-message-link`	Fired when the user clicks on a posted link
`click-action-link`	Fired when the user clicks on an action link button (i.e.: "click to join" to video conferences)

Below there is a sample code that listens to events fired from Rocket.Chat opened on an `iframe` :

```
window.addEventListener('message', function(e) {
    console.log(e.data.eventName); // event name
    console.log(e.data.data); // event data
});
```

Iframe integration: Sending commands

Available commands

Command	Params	Description
go	- `path` string	Change url
login-with-token	- `token` string	Allow login with token
call-custom-oauth-login	- `service` string	Allow login via oauth methods
set-user-status	- `status` string	Set the status of the user
logout		Log the user out of their current session

Example

```
document.querySelector('iframe').contentWindow.postMessage({
  externalCommand: 'go',
  path: '/admin/General'
}, '*')
```

Two Factor Authentication

Rocket.Chat uses Two Factor to authorize important actions. There is a list of possible sources for the two-factor code:

- **Authenticator App**: like Google Authenticator or Authy (need to be configured by the user);
- **Email**: users will receive the code via email (configured by default and enabled for those users with verified emails);
- **Password***: it's not a two factor by itself, but a fallback for the cases where the user has no other option to configure;

> * The password fallback is disabled for the login process, so the login will not require the password twice when the user has no other two factor method configure.

Any **DDP Method** or **REST call** may have the two-factor requirement; for that reason, we suggest creating a wrapper for your calls to handle the errors described here and executing the request again, passing the required info as we will describe here as well.

Remember Me

By default, after a two-factor validation, the client used (a hash of user-agent + IP address) will be trusted for 5 minutes. It's configurable via the admin panel.

Some methods may disable this feature forcing the API to always require the two-factor for that method/endpoint. The method to disable the two-factor by email and the login are examples.

Compatibility

We are using the error `totp-required` for compatibility purposes, it doesn't mean that the error is related to TOTP only, so we pass more details to identify the action required.

Personal Access Tokens

Personal Access Tokens are tokens created by the users (when enabled by the server) commonly used to give access to other applications, bots, etc. Those tokens do not expire, and they have the option to **bypass** the Two-Factor (required by default), allowing users to use their integrations without restrictions when needed.

Now it's the two-factor required to create personal access tokens.

> The bypass should be used carefully because it gives super powers to who gain access to the token

---

Realtime API

Visit the Two Factor Authentication page at Realtime API guides for more information.

REST API

Visit the Two Factor Authentication page at REST API guides for more information.

Troubleshooting

1. **babel-runtime**:

If you are having the following error:

```
(STDERR) Error: The babel-runtime npm package could not be found in your n
(STDERR) directory. Please run the following command to install it:
(STDERR)
(STDERR)   meteor npm install --save babel-runtime
(STDERR)
(...)
=> Exited with code: 1
=> Your application is crashing. Waiting for file change.
```

Just install the mentioned package with the following command:

```
meteor npm install --save babel-runtime
```

1. **bcrypt**:

If you see the following warning in the `meteor` logs:

```
(STDERR) Note: you are using a pure-JavaScript implementation of bcrypt.
(STDERR) While this implementation will work correctly, it is known to be
(STDERR) approximately three times slower than the native implementation.
(STDERR) In order to use the native implementation instead, run
(STDERR)
(STDERR)   meteor npm install --save bcrypt
(STDERR)
```

Don't panic =) It means that the `bcrypt` library is not installed on your system and `meteor` will use a javascript alternative that is about three times slower.

If you want to install the library to make it faster, use the following command:

```
meteor npm install --save bcrypt
```

If the version of the `python` interpreter on your system is **greater than** v2.5.0 or **less than** 3.0.0, it should work fine, but, if you see a message like this:

```
gyp ERR! configure error
gyp ERR! stack Error: Python executable "/usr/local/bin/python3" is v3.5.2
gyp ERR! stack You can pass the --python switch to point to Python >= v2.5
```

After you are sure that you have a `python` interpreter that matches the above requirements, use the following command to fix the error:

```
meteor npm config set python python2.7
```

Build it again:

```
meteor npm install --save bcrypt
```

If everything works, you should see a message like this:

```
> node-gyp rebuild

  CXX(target) Release/obj.target/bcrypt_lib/src/blowfish.o
  CXX(target) Release/obj.target/bcrypt_lib/src/bcrypt.o
  CXX(target) Release/obj.target/bcrypt_lib/src/bcrypt_node.o
  SOLINK_MODULE(target) Release/bcrypt_lib.node
clang: warning: libstdc++ is deprecated; move to libc++ with a minimum dep
Rocket.Chat@0.46.0-develop /Users/douglas/work/github/Rocket.Chat
```

```
└─┬ bcrypt@0.8.7
  ├── bindings@1.2.1
  └── nan@2.3.5
```