Dev environment setup in IntelliJ IDEA

This is a how-to guide to set up and work effectively with the *iodine-webapp* project in IntelliJ IDEA. The following steps will reduce the time for the newcomers or anyone who considering switching to this IDE. This article will also contain known issues and solutions for them.

Step-by-step guide

1. Prerequisites

- Get all accounts. For this guide, you will need Atlassian (JIRA, Bitbucket, Bamboo, etc) and Artifactory accounts.
- Install the latest version of npm+node 12.8.0, Java 1.8, Maven 3, Git 2, Postgres 13 and Tomcat 8.5.
  - You must use the **oracle** jdk, openjdk will cause errors.
  - Git may already be pre-installed on your Iodine machine. Run **`git --version`** to confirm.
  - Users have reported better success installing Postgres via Homebrew rather than the manual install.
    - Use **`/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`** to install Homebrew (or check The Homebrew Website for an updated command if that doesn't work).
    - Then, **`brew install postgresql@13`**
  - Do not install the most recent version of node, this has been known to lead to a lot of issues when trying to do a mvn clean install. Use node version 12. You can install via Homebrew by using **brew install node@12**. After installing npm+node, confirm install from a fresh terminal with **`npm -v`** and **`node -v`**. If these fail, you likely have to update your PATH to include the path to those binaries.
- Download and install IntelliJ IDEA from here. Open it and set up as you like.
- Claim a license under Iodine's license agreement for IntelliJ here.

2. Maven and Artifactory setup

- Goto Artifactory and log in using your credentials – they are the crowd credentials used for the Atlassian tools
- Open your user profile by clicking your username in the upper-right corner and unlock your settings

- Download the following settings.xml file here, and replace the placeholders in the **`<servers>`** tag with your Artifactory username and the API key from your Artifactory profile page.
- It should look something like this:

Maven user Settings

```
<server>
  <id>central</id>
    <username>victor</username>
    <password>your_api_key_here</password>
</server>
<server>
  <id>snapshots</id>
    <username>victor</username>
    <password>your_api_key_here</password>
</server>
```

- For the sonar credentials section you will need to encrypt your Crowd password. Instructions for encrypting passwords with maven are here:
  - https://maven.apache.org/guides/mini/guide-encryption.html
- The sonar credentials section should look something like this (note the curly braces around the encrypted password):

```
<profile>
  <id>sonar</id>
  <properties>
    <sonar.login>kmcvay</sonar.login>
    <sonar.password>{your_encrypted_password}</sonar.password>
  </properties>
</profile
```

- Copy your edited settings.xml file into your m2 directory (~/.m2)

3. Check out and build the project

Go to Bitbucket and clone the *iodine* project. Click on the "Clone" button at the top-left corner and choose HTTPS as a connection. This can also be done using IntelliJ. Check out from version control -> Git -> Paste URL to the *iodine* project -> Click "Clone". This may take a while to download, get a cup of tea.

If you have Eclipse plugin installed, IDEA will automatically suggest using the configuration from it. If not, it will use Maven in order to set up the project. After indexing will be completed, make sure all the dependencies were downloaded. You can turn on automatic import for Maven. Make sure that there are no errors during downloading.

Now, if everything is good, at this point you will be able to build the project using Maven. You can either use the command line or IntelliJ IDEA Maven integration. Just run **`mvn clean install`** from the root folder. You can additionally pass a **-DskipTests=true** parameter to skip tests or turn on Skip Tests mode in IDEA. If your maven settings are correct, you will have a successful build as a result.

Setup sonarlint plugin to check project locally:

How to set up SonarLint to detect issues with code quality

4. Setup DB

The product requires 1 PSQL database with 4 schemas to be setup. You can do it either using a command line or pgAdmin application. Since most of the team are macOS users, the command line way is listed below.

IMPORTANT: If you run into problems at this step, please see section 9.4 below (Known Issues: Authentication Issues with Postgres).

Run the following from a bash prompt:

```
psql postgres -c "CREATE DATABASE iodine WITH ENCODING='UTF8' OWNER=$USER CONNECTION LIMIT=200;"
```

Open a psql prompt (from the command line, **`psql postgres`**) and execute the following commands:

```
GRANT ALL ON DATABASE iodine TO PUBLIC;
\c iodine
CREATE USER rta_hl7 PASSWORD 'rta_hl7';
CREATE SCHEMA rta_hl7 AUTHORIZATION rta_hl7;
CREATE USER rta_model PASSWORD 'rta_model';
CREATE SCHEMA rta_model AUTHORIZATION rta_model;
CREATE USER rta_test_hl7 PASSWORD 'rta_test_hl7';
CREATE SCHEMA rta_test_hl7 AUTHORIZATION rta_test_hl7;
CREATE USER rta_test_model PASSWORD 'rta_test_model';
CREATE SCHEMA rta_test_model AUTHORIZATION rta_test_model;
```

There is an option to load sample data from QA. First, download the dump file from Artifactory:

```
curl -u username:password https://artifactory.iodinesoftware.com/artifactory/qa-artifacts/qa_light_model_Sep_17_2020_test_data.dump --output qa_light_model_Sep_17_2020_test_data.dump
```

then run the following commands from psql:

```
CREATE USER qa PASSWORD 'str0ng';            -- this will create qa user with the same credentials as the dev box

CREATE ROLE iodine_read_only;
CREATE ROLE iodine_read_write;
CREATE ROLE iodine_dba;
CREATE ROLE iodine_all_schemas;
```

Exit psql and then run the following command to load the QA data:

> *psql -U $USER -d iodine < qa_light_model_Sep_17_2020_test_data.dump*

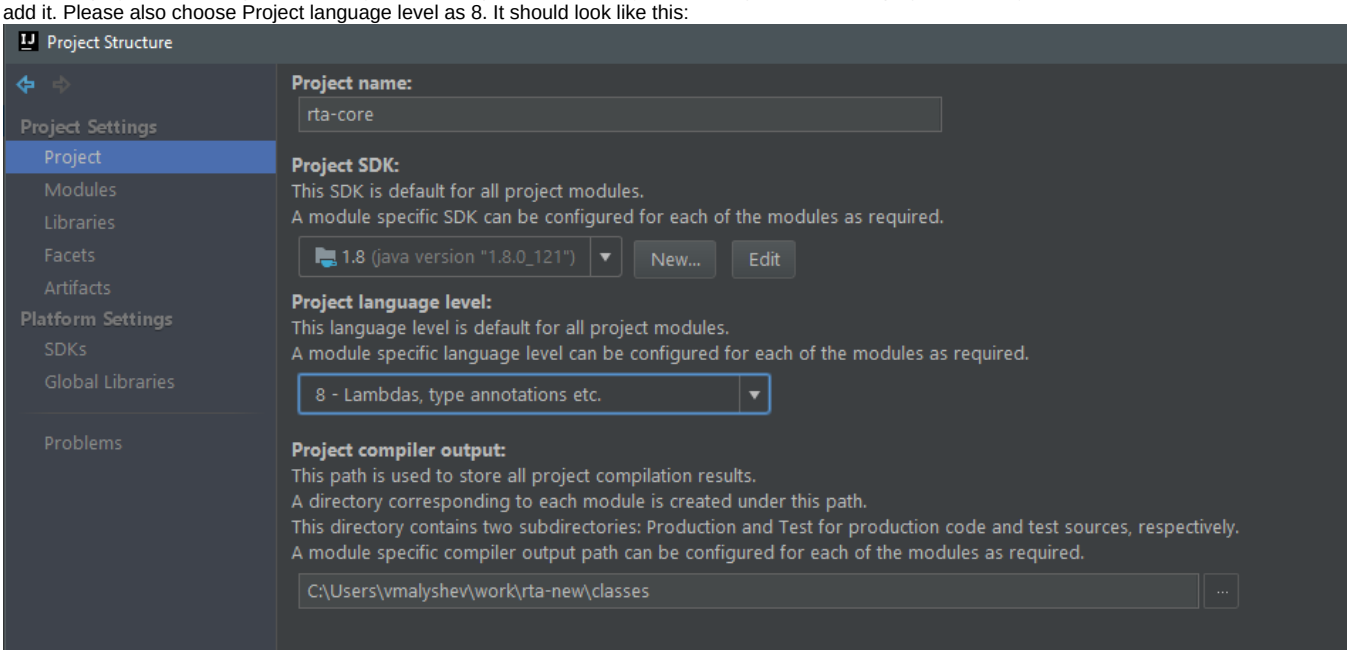
The last step is to update project properties. Update /iodine/iodine-model/src/main/resources/conf/local.properties in the iodine-model project to:

> *local.db.model.url=jdbc:log4jdbc:postgresql://localhost:5432/iodine?currentSchema=qa_light_model*
> *local.db.model.database=qa_light_model*
> *local.db.model.name=qa*
> *local.db.model.password=str0ng*

5. IntelliJ IDEA Project setup for WebApp and WebClient

Next step is to setup project to be run under the IntelliJ. There are some specific settings that need to be applied in order to make it work. They are the following:

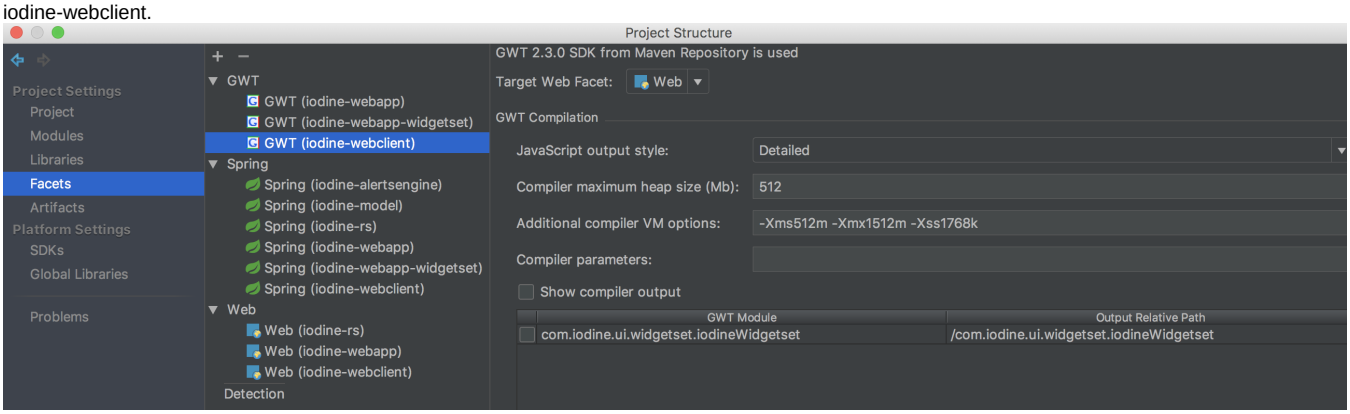
Go to the project structure (click File Project Structure) and make sure you are using Java 1.8 as a project SDK. If you don't have it on the list, feel free to add it. Please also choose Project language level as 8. It should look like this:



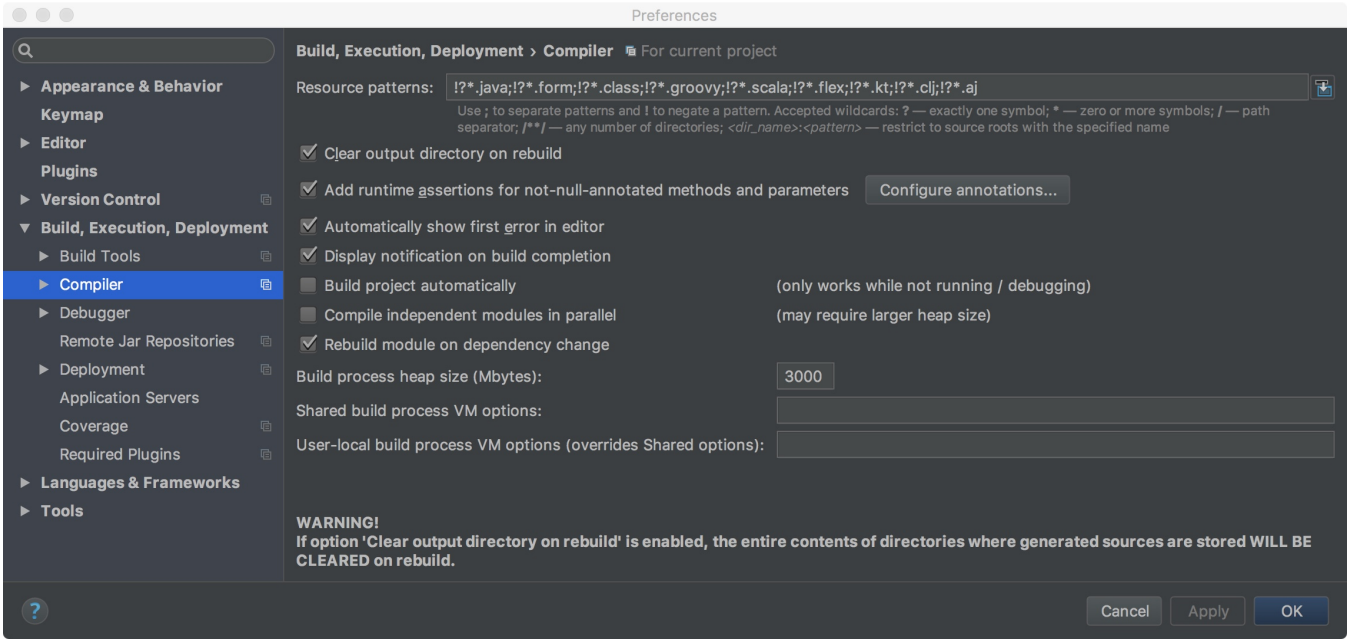
Within same settings, go to Facets and find GWT facet for iodine-webclient. In additional settings, please add the following: **`-Xms512m -Xmx1512m -Xss1768k.`**

- Note: if you are using IntelliJ Community you will need to upgrade to the Ultimate version because the Community version of IntelliJ does not have support for GWT or Spring modules.

You'll also need to uncheck the widgetset GWT modules for **both the iodine-webclient and iodine-webapp-widgetset facets**, as pictured below for only iodine-webclient.

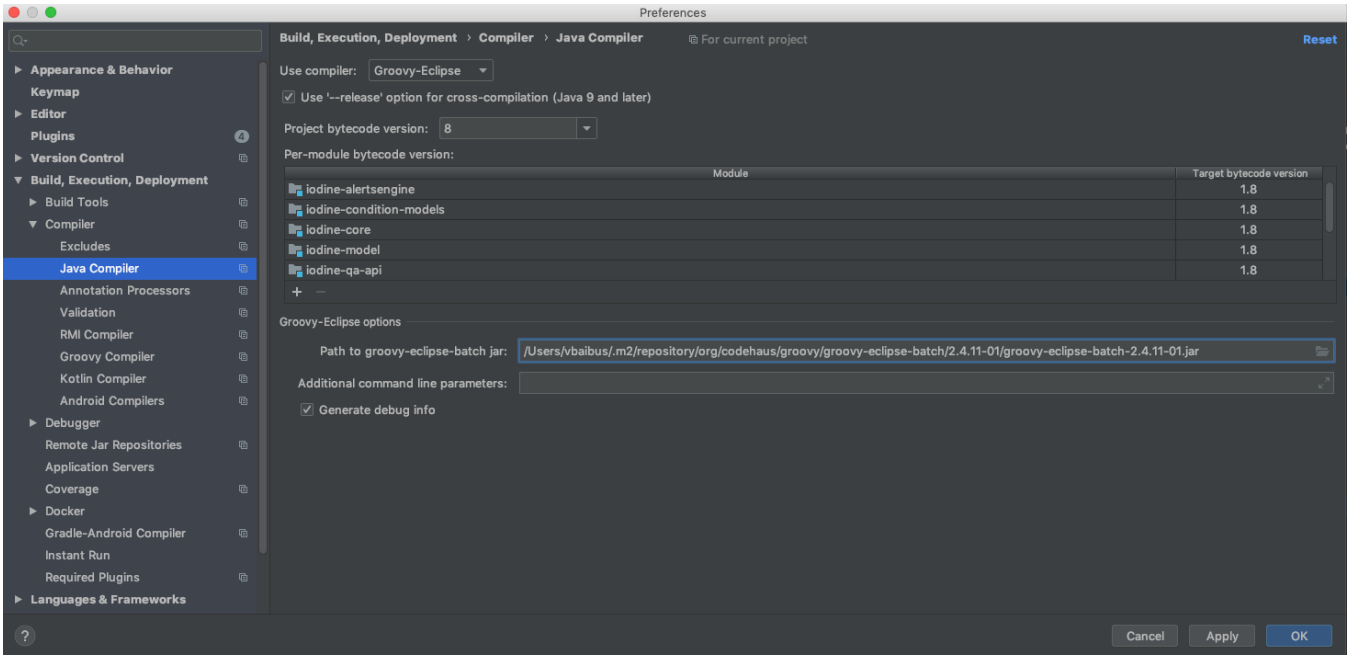


Go to general IDE settings (by clicking IntelliJ IDEA -> Preferences...) and go to the Compiler section. Set the *Build process heap size (Mbytes)* to at least 3000.



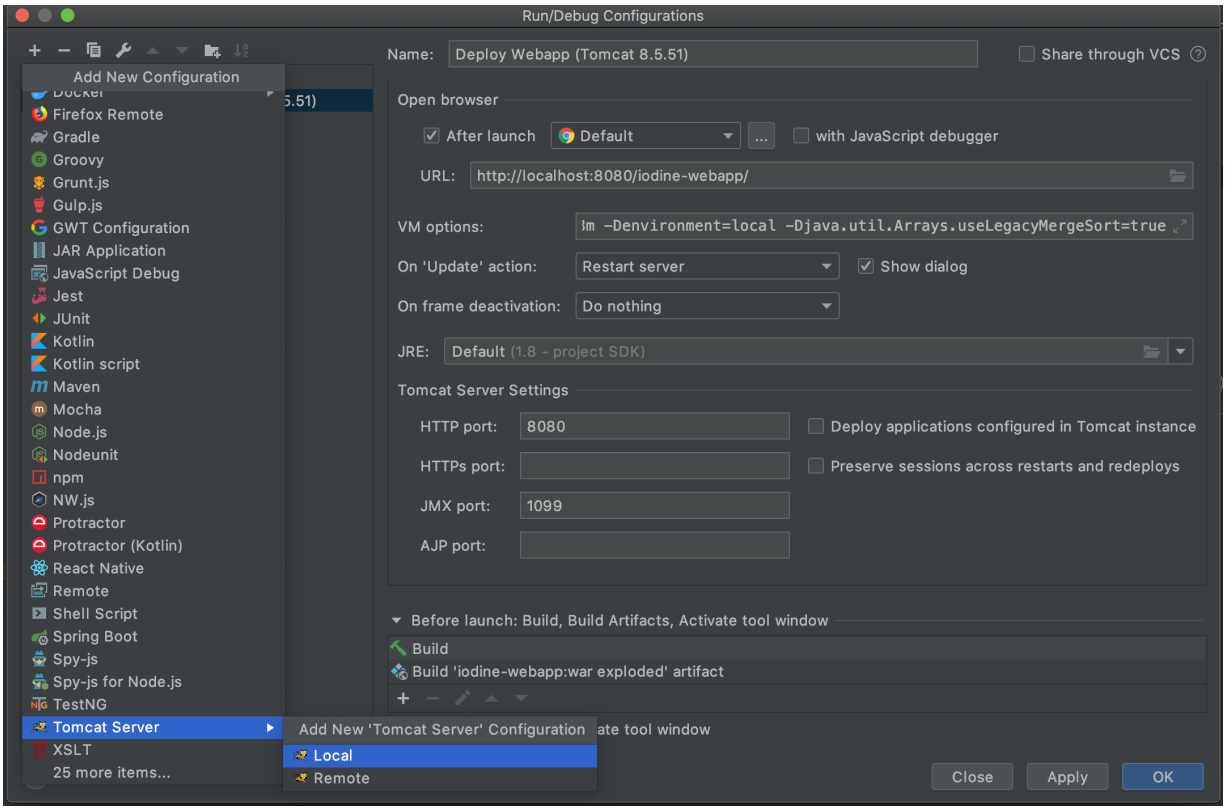
Later when you attempt to build the project, if you get an error regarding RAM (i.e. *out of memory*) please see section 9.6 below (Known Issues: Out of Memory).

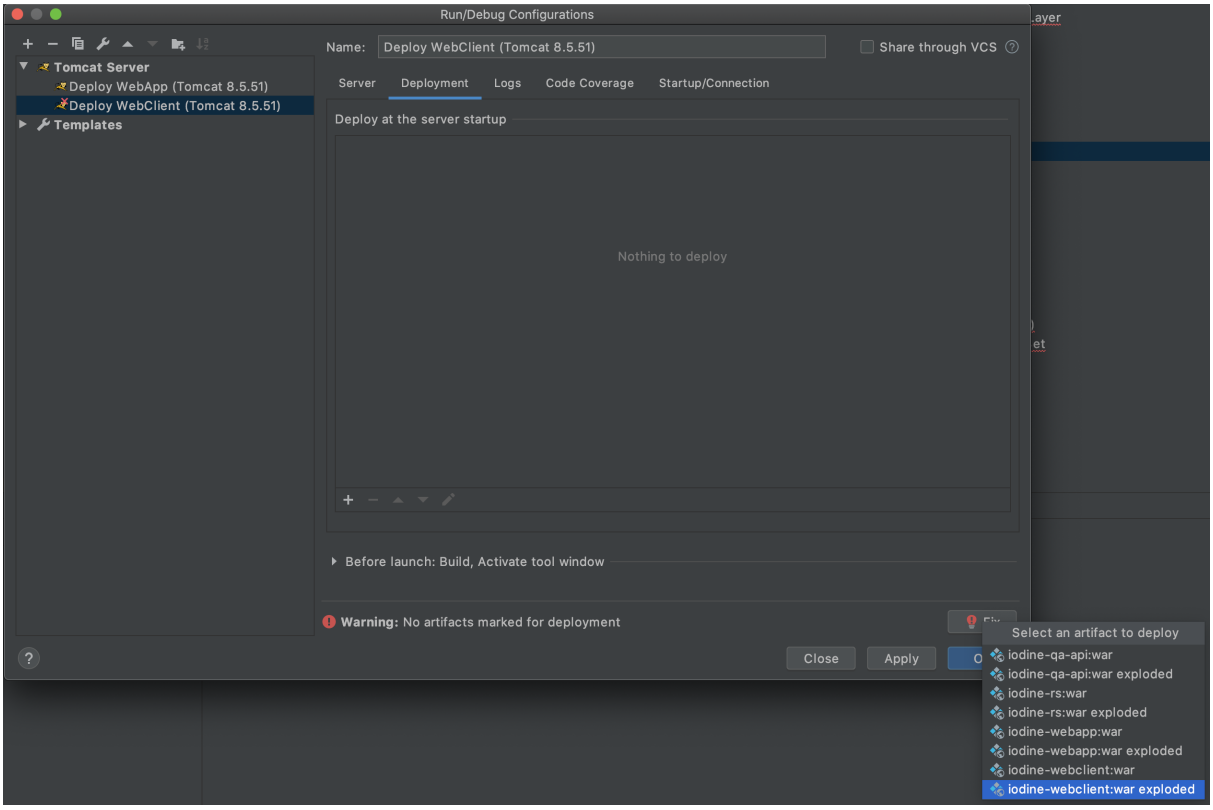
In the same window go to Java compiler settings (use search if need). Observe that *javac* is the current compiler and change it to *Groovy-Eclipse* one. From personal experience, it's better to use Groovy-Eclipse compiler because it's less buggy. In order to set this up, set 'Path to groovy-eclipse-batch jar' to your local groovy-eclipse install as shown on screenshot.
(should be somethinglike /Users/<username>/.m2/repository/org/codehaus/groovy/groovy-eclipse-batch/2.4.11-01/groovy-eclipse-batch-2.4.11-01.jar)



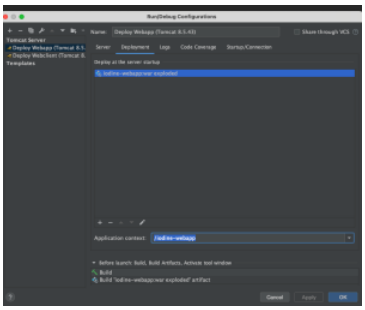
Next add a Run/Debug configuration for each the WebApp and the WebClient (Run > Edit Configurations). For this purpose, you need Apache Tomcat 8.5. You can download it manually or by using IDEA. Add a Local Tomcat server. Please note, that IntelliJ will automatically figure out that you don't have an artifact to be deployed, so let's fix that by clicking on "Fix" button. For the WebApp, click on the Fix button and choose "iodine-webapp.war: exploded" to be deployed. For the WebClient, choose "iodine-webclient:war exploded".

These are example screenshots for the Webapp.



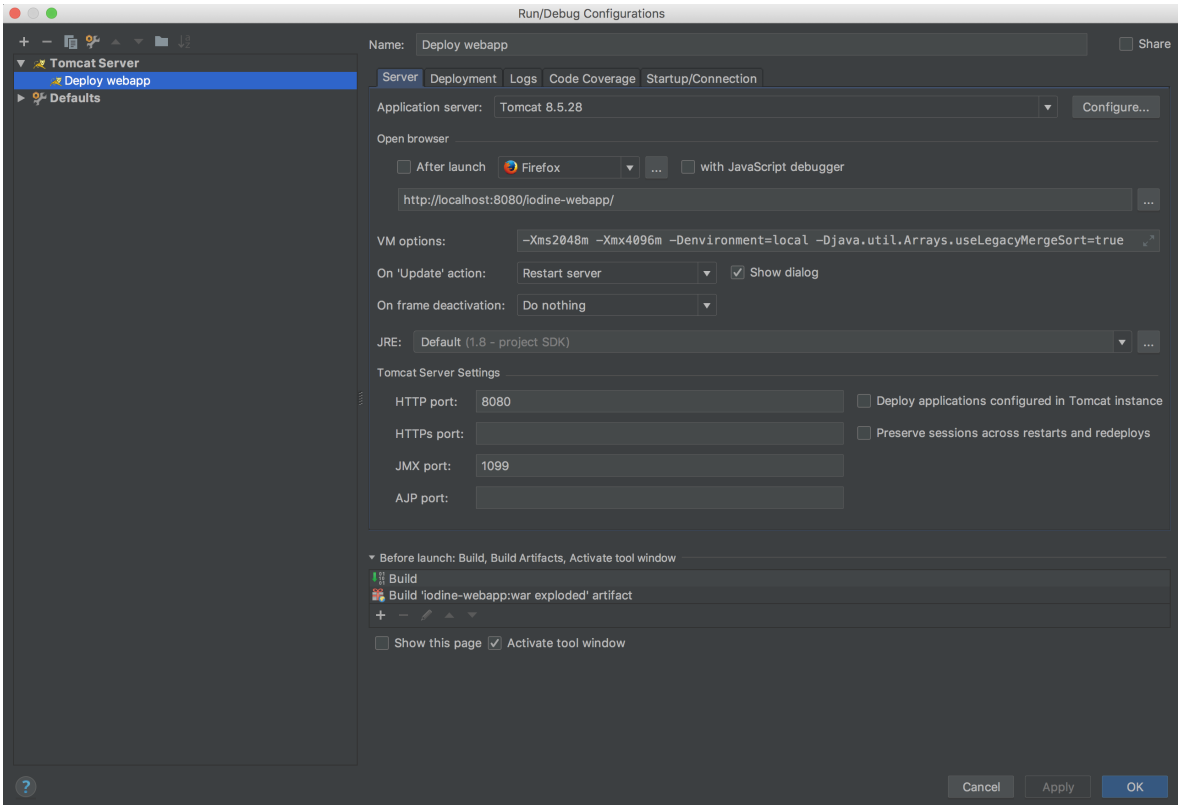


Verify the Application context in the Deployment tab for both webclient and webapp are correct. The webclient application context should be "/iodine-webclient" and the webapp should be "/iodine-webapp". Example set up for webapp shown below:

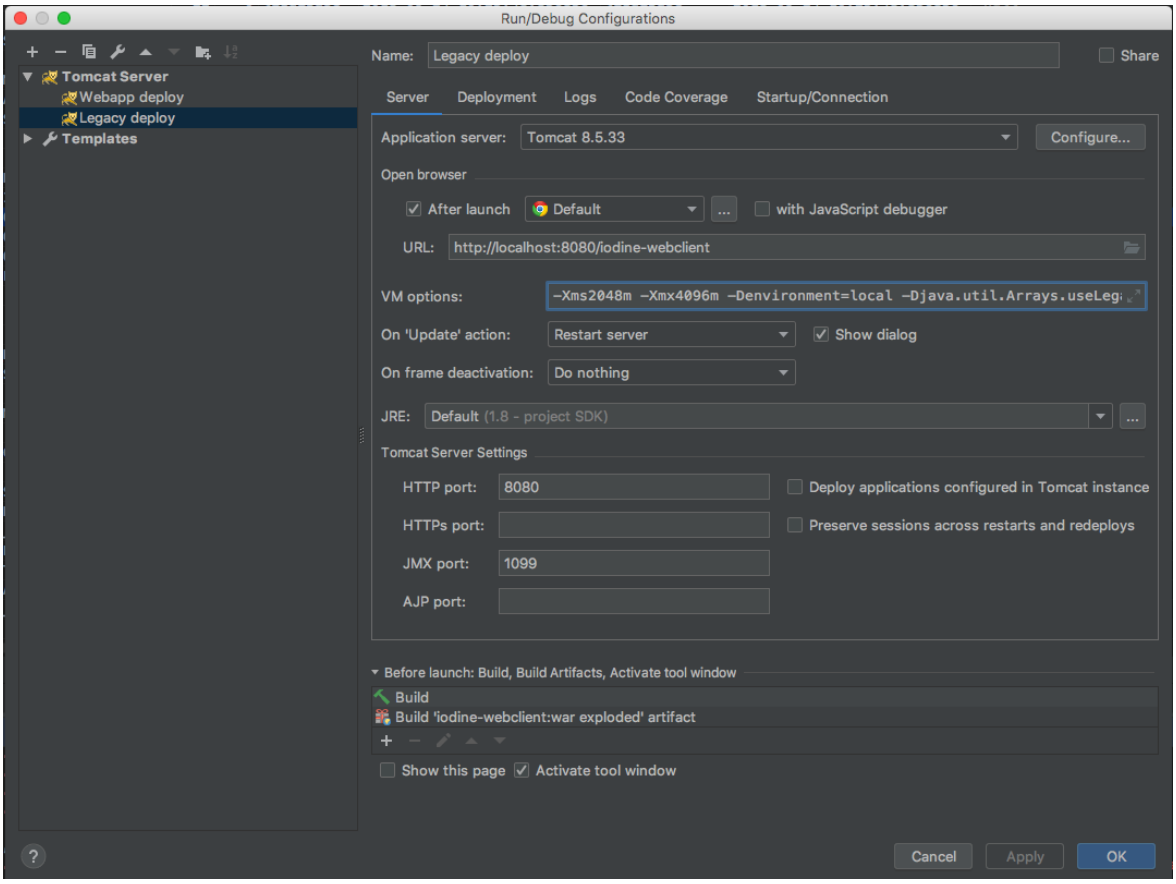


The last thing is to add additional VM Options on the server screen to both configurations: **`-Xms512m -Xmx2048m -Denvironment=local -Djava.util.Arrays.useLegacyMergeSort=true.`** After that just save the configuration.

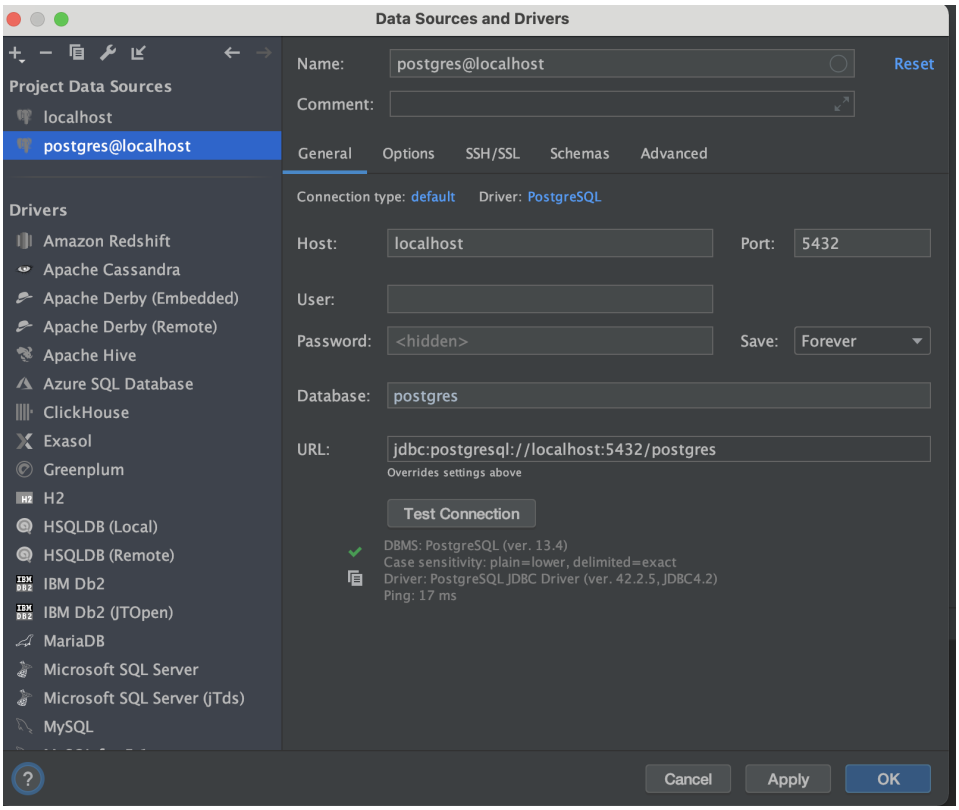
WebApp run configuration should look like this:

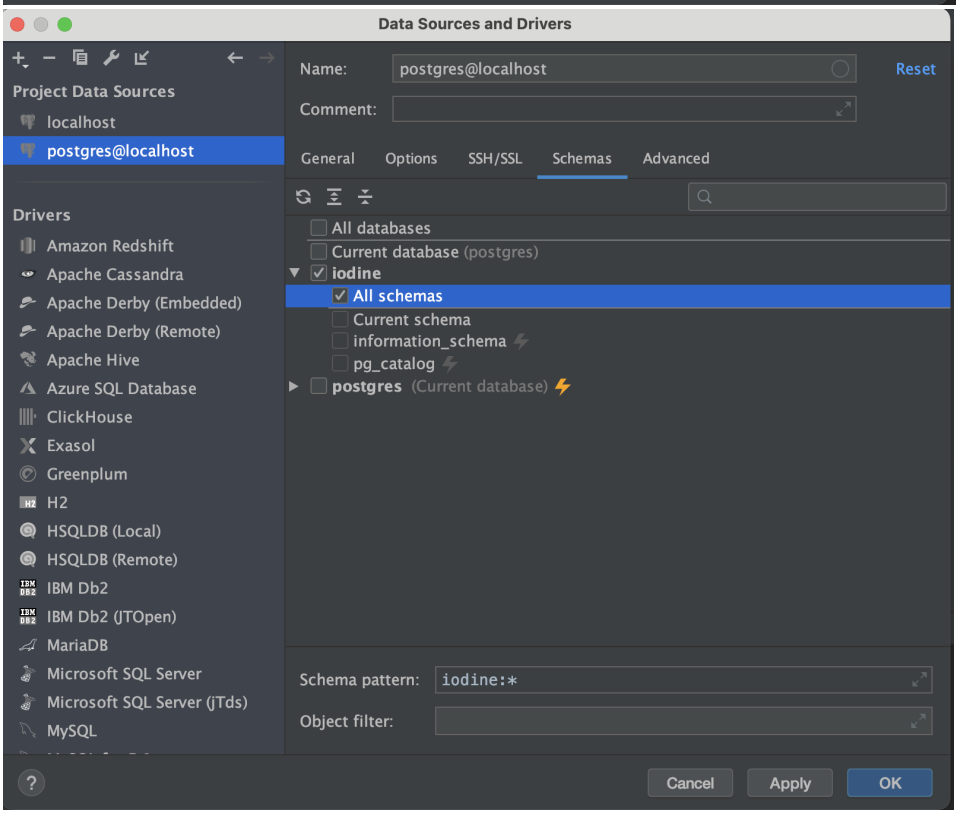


WebClient run configuration should look like this:



Next we'll need to make sure the database is connected through IntelliJ. Open up the Database tab on the right side of the screen. If there is no database connected, click the + Data Source PostgreSQL to add the database. Once you've added the PostgreSQL database, Test Connection to make sure it is connected. You may need to download some driver files. Make sure you get a green check showing the connection is valid. Navigate to the Schemas tab and make sure that 'All Schemas' is checked under Iodine.





6. Running the project locally

Start the webclient from IntelliJ using the Run Configuration you just created.

Hopefully, the webclient should deploy successfully, and if you set your Run Configuration to do this, IntelliJ will open your default web browser right to the login page once it has deployed successfully.

IMPORTANT: AT THIS STEP (NO SOONER AND NO LATER): Go into your psql database. Go into iodine database schemas qa_light_model tables role and open the role table. You should see an entry where the name is empty and id=66. DELETE THIS ENTRY AND COMMIT THIS CHANGE TO YOUR DATABASE.

Now login with user: iobryan and password: password2

If login fails...

The qa_light_model schema contains no password rules entries in the database. This can cause login to silently fail. Go into iodine database schemas qa_light_model tables password_rule. If there is not an existing password rule where password_type = 'WEBCLIENT' insert one and restart the webclient:

INSERT INTO "qa_light_model"."password_rule" ("password_type", "lower_case_required", "upper_case_required", "numeric_required", "special_characters_required", "min_length", "max_length", "max_repeating_chars", "max_sequential_chars", "password_history_unique_entries", "password_history_unique_timespan", "password_expiry_in_days", "force_password_change_after_reset", "instructions", "max_failed_login_attempts", "disallowed_account_fields") VALUES
('WEBCLIENT', 't', 'f', 't', 'f', '6', '20', '2', '-1', '1', '-1', '100', 't', 'Your password must be between 6 and 20 characters long, have no more than 2 of the same character in a row and contain lower case letters and numbers.', '3', '');

Once done, build WebApp project, run it from IDE and login to WebApp to finish the setup process. (**Note:** You would need to shut down WebClient first before starting WebApp or configure WebApp to run on different HTTP and JMX port to avoid conflict).

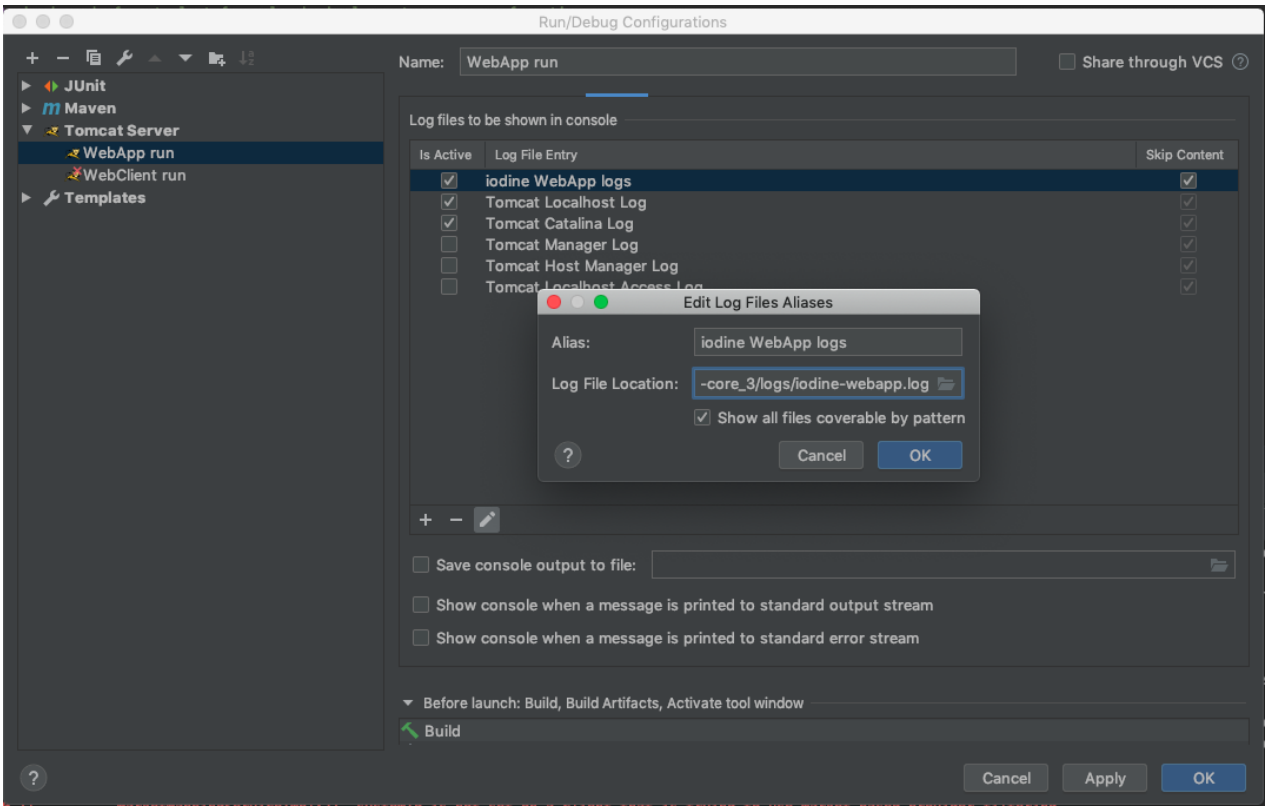
iodine-webapp project has a lot of logging which will not be visible on the standard output. In the Run/Debug settings, you can actually add custom log files by specifying it's path.

Feature flags

You can configure enabled feature flags in src/main/resources/data/features/features.xml

7. Add iodine WebApp Logs

After you have successfully launched the webapp, it's time to setup the iodine Logs so they are accessible for debugging later. Edit the Run Configuration for the WebApp by selecting the build deployment profile you created in the dropdown menu near the top right of IDEA. Go over to the Logs tab, click the +, navigate to `IntellijIdea20XX.X/tomcat/Unnamed_iodine-core_Y` where Y is the greatest integer available (for me, it was 3), and click Open. Type in "iodine WebApp logs" for the Alias and click OK:



8. Code Style

All Iodine projects follow the code style defined in Java and Groovy Style Guide for Iodine.

To ensure you're following the same style as everyone else, you will need to import the Iodine Code Style settings into IntelliJ:

Clone the iodine-code-style repo

From IntelliJ IDEA, open the Preferences window

Goto Editor Code Style

Click the gear icon next to the Scheme field Import Scheme Intellij IDEA code style XML

Load the intellij-iodine-code-style.xml file that is in the git repo you cloned above

9. Known issues

1. For some reason after each Maven update settings for the compiler and additional GWT facet settings gets lost. Make sure that those options are there after maven (pom.xml) updates.

2. Sometimes you can get an OOM Error during build inside the IDE. Make sure that GWT facet options are presented.

3. IntelliJ IDEA is a great product. It's fast because of internal caches. Sometimes, very rarely, it may cause issues during development. If you are experiencing really strange issues, black magic 😃, just restart and clean the cache. Go to File -> Invalidate Caches/Restart -> Invalidate and restart. IDE will be restarted and index all the project from the beginning. This should solve most of the strange things.

4. On startup application get error

```
Caused by: org.postgresql.util.PSQLException: FATAL: password authentication failed for user "qa"
```

By default PostgreSQL uses IDENT-based authentication and this will never allow you to login via -U and -W options. Allow username and password based authentication from your application by applying 'trust' as the authentication method for the database user. You can do this by modifying the `pg_hba.conf` file.

On terminal

> *psql -U postgres*

Put password if needed and call

> *show hba_file ;*

You've got a path to hba_file. Open and change

> *# IPv4 local connections:*
> *host all all 127.0.0.1/32 **md5***

to

> *# IPv4 local connections:*
> *host all all 127.0.0.1/32 **trust***

sometimes should be added line (or same, check table header)

> *local all all trust*

Don't forget to restart PostgreSQL after saving your changes to the file.
`# service postgresql restart`

6. On build attempt of iodine-webclient, compiler complains that Java is out of memory in the heap:

Temporarily up the Build Process Heap Size (Preferences Build, Execution, Deployment Compiler) to 5000 Mbytes. Upon your first successful build you should be able to lower this back to 4000 or 3000.

7. After logging into the webapp the following error pops up:

```
Failed to load the widgetset: ./VAADIN/widgetsets/com.iodine.webapp.widgetset.IodineWidgetset/com.iodine.webapp.
widgetset.IodineWidgetset.nocache.js?1578687749047
```

In Intellij click on Maven Iodine Vaadin Webapp WidgetSet install. Then using command line, navigate to the root of the iodine project and do `mvn clean install -DskipTests=true`.

8. If you've tried #5, but the webapp/webclient login is still invalid you will need to rebuild the project using IntelliJ. Click on Maven Iodine Model and Persistence Layer Lifecycle Clean. Then Click on Iodine Model and in the toolbar click Build Rebuild Module 'iodine-model'. Rerun the WebApp and/or WebClient and you should be able to login. This happens sometimes if you run 'maven install' from the command line because Maven and IntelliJ don't always sync up. Do all builds/cleans from IntelliJ if possible to keep it consistent.

9. If login issue persists, run the following and ensure fields are set to false:

```
SELECT account_disabled, account_locked
        FROM qa_light_model.caregiver_user
        WHERE username = 'iobryan';
```

Related articles

Some resources that may help during environment setup

Setting up a dev environment (old)

Starting iodine-alertsengine locally