Tools Notes

**Revisions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **User** | **Modifications** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Overview

## General

---------------------------------Open Source Tools --------------------------------------

* Use **jsonschema2pojo.org**: generate pojo from json
* For base64 encoded file it can be decoded at local using windows command line.

certutil -decode encoded\_file decoded\_file

* To beatify json, xml files, there are notepad++ plugins:

JSTools              XML Tools      

# Node & npm

## Links

--------------------------Links:

* Node API: <https://nodejs.org/api/http.html>
* Node doc: <https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/>
* Express official site: <http://expressjs.com/>
* A good intro to express: <http://evanhahn.com/understanding-express/>

## General Usage

--------------------------verify npm and node installation:

npm -v

node -v

--------------------------npm and node configuration:

# To upgrade npm to latest:

 $> npm install -g npm

#To update packages using npm:

1. >npm install myPackage@2.0.0 --save

After the installation, package.json will be updated to the right version.

1. You can enter to package.json and write the version yourself on the dependencies. After that do npm install and it will install the correct version.

#To check if a specific package is installed globally or locally (without -g)

npm list -g [package-name]

# get proxy setting

npm config get proxy

npm config get https-proxy

#Set proxy for npm

npm config set https-proxy <http://one.proxy.att.com:8080>

npm config set proxy <http://one.proxy.att.com:8080>

# reset proxy for npm

npm config rm proxy

npm config rm https-proxy

#If still getting proxy error, try this

git config --global --unset http.proxy  
git config --global --unset https.proxy

-------------------------- Common Issues:

\*\* error Unexpected token < in JSON at position 0 while parsing near '<!doctype html>

First try: npm cache clean --force

if false try: delete package.lock.json

if false try: npm set registry <https://registry.npmjs.org/>

-------------------------- Common npm commands:

>npm init # Init package.json

>npm install lite-server --save-dev

--------------------------create-react-app:

create-react-app creates a frontend build pipeline, so you can use it with any backend you want. It uses build tools like Babel and webpack under the hood, but works with zero configuration. When you’re ready to deploy to production, running npm run build will create an optimized build of your app in the build folder

it installs react, react-dom, react-scripts ...

npm install -g create-react-app

create-react-app my-app

inside the my-app directory, you can run:

npm start ##starts the development server

npm run build ## Bundle the app into static files for production

npm test ##starts the test runner

npm run eject ##removed this tool and copies build dependencies, configurations and scripts into the app directory; and you cannot go back.

# Yarn

## General

-------------------------General:

\*\*Official site: [https://yarnpkg.com/en/docs/install#windows-stable](https://yarnpkg.com/en/docs/install" \l "windows-stable)

\*\*Package managers:

* Composer (PHP)
* Npm (for node, mostly JS)
* Yarn is a package manager by Facebook, works better in react eco system. Yarn doesn't maintain their registry, they are just pointing their domain to the original npm registry using Cloudflare.
* [Chocolatey](https://chocolatey.org/) is a package manager for Windows.

# mvn

## General

\*\*Configure your local repository:

configuration is in ${user.home}/.m2/settings.xml: C:\Users\dl8899\.m2

default local reposiotry is: ${user.home}/.m2/repository

\*\*Configure a proxy:

1. Use MAVEN\_OPTS

-Dhttps.proxyHost=one.proxy.att.com -Dhttps.proxyPort=8080 -Dhttp.proxyHost=one.proxy.ttt.com -Dhttp.proxyPort=8080

1. Config it in the settings.xml: <http://maven.apache.org/guides/mini/guide-proxies.html>

<settings>

<proxies>

<proxy>

<id>http-proxy</id>

<active>true</active>

<protocol>http</protocol>

<host>one.proxy.att.com</host>

<port>8080</port>

<nonProxyHosts>127.0.0.1</nonProxyHosts>

</proxy>

<proxy>

<id>https-proxy</id>

<active>true</active>

<protocol>https</protocol>

<host>one.proxy.att.com</host>

<port>8080</port>

<nonProxyHosts>127.0.0.1</nonProxyHosts>

</proxy>

</proxies>

</settings>

\*\*mvn help:effective-pom

\*\*maven-archetype-quickstart

mvn archetype:generate -DgroupId=com.danni.app -DartifactId=guide1 -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

example:

mvn archetype:generate -DgroupId=com.danni.test -DartifactId=test -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

\*\*mvn lifecycle goals:

mvn compile

mvn package

mvn install

you can run the application using ./mvnw spring-boot:run. Or you can build the JAR file with ./mvnw clean package. Then you can run the JAR file: java -jar target/gs-consuming-rest-0.1.0.jar

------------------------------------------------- Handling of Common Errors:

\*\*Handle error: Source option 1.5 is no longer supported. Use 1.6 or later:

Add to POM (put above dependencies):

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

OR:

<properties>

<java.version>1.8</java.version>

</properties>

# Subline Text

## General

\*\*subline text auto complete:

<http://sublimeautocomplete.blogspot.com/2015/02/sublime-autocomplete.html>

\*\*Sublime text and React: <https://medium.com/@adrianli/setting-up-sublime-text-3-for-reactjs-3bf6baceb73a>

\*\*good tutorial: <https://code.tutsplus.com/courses/perfect-workflow-in-sublime-text-2/lessons/installation-and-base-settings>

# git

## General

\*\*Questions:

??? How to create sub directories?

Pull requests?

**\*\*useful links:**

Additional Resources (Git)

* Git site [http://git-scm.com](http://git-scm.com/).
* [Installing Git](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git) chapter from Pro Git
* [Git reference manual](https://git-scm.com/docs)
* Quick reference guides: [GitHub Cheat Sheet](https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf) (PDF) | [Visual Git Cheat Sheet](http://ndpsoftware.com/git-cheatsheet.html) (SVG | PNG)
* [Atlassian comprehensive Git tutorial](https://www.atlassian.com/git/tutorials/)

GIT Book:

<http://git-scm.com/book/en/Getting-Started-Git-Basics>

pro git: <http://git-scm.com/book/en/v2>

Git - the simple guide: <http://rogerdudler.github.io/git-guide/>

GIT for Windows:

<http://msysgit.github.io/>

A good EGit tutorial:

<http://www.vogella.com/tutorials/EclipseGit/article.html>

stop at 6.4, create .gitignore file

**\*\*Connecting to GitHub with SSH**

**-->** Using the SSH protocol, you can connect and authenticate to remote servers and services. With SSH keys, you can connect to GitHub Enterprise without supplying your username or password at each visit.

Instruction on setting up SSH access to github: <https://help.github.com/enterprise/2.6/user/articles/connecting-to-github-with-ssh/> **stop**

**\*\*Git config** file: C:\Users\dl8899\.gitconfig

GIt local repository: C:\Users\dl8899\git

**\*\*Git Proxy configuration:**

# get proxy setting for git

git config --global --list

>git config --list

>git config --global user.name "Your Name"

>git config --global user.email <your email address>

#Set proxy for git:

git config --global http.proxy <http://one.proxy.att.com:8080>

git config --global https.proxy <http://one.proxy.att.com:8080>

# reset proxy for git

git config --global --unset http.proxy

git config --global --unset https.proxy

#Set an email and name in Git

git config --global user.email "danniliu99@hotmail.com" #Set

git config --global user.email #Confirm

git config --global user.name "danniliu"

git config --global user.name

**\*\* .gitignore**

Create the .gitignore file in the root directory of the working tree to make it specific for the Git repository, with content such as:

node\_modules

Eclipse also supports the .gitignore file for excluding files and directories from the Git operations. You can also setup a global .gitignore file valid for all Git repositories via the core.excludesfile setting.

You can also create local per-repository rules by editing the .git/info/exclude file in your repository.

**\*\*Git commands:**

git --version

git clone URL /\* get teacher's code \*/

**\*\*Adding an existing project to GitHub** using the command line: <https://help.github.com/articles/adding-an-existing-project-to-github-using-the-command-line/>

1. **git init** /\* in the working directory \*/
2. **Create .gitignore file** (important, to include files such as node\_modules)
3. **git status**
4. **git add .** /\*Adds the files in the local repository and stages them for commit. To unstage a file, use 'git reset HEAD YOUR-FILE'. \*/
5. **git commit -m "initial commit"** /\*# Commits the tracked changes and prepares them to be pushed to a remote repository. To remove this commit and modify the file, use 'git reset --soft HEAD~1' and commit and add the file again.\*/
6. **git log** --oneline
7. >**git checkout** -- <filename> # this is in case if we want to abandon our changes and fetch back the original file from the checked in version of your local repository.
8. >**git checkout** <the commit's number from "git log"> filename

Note: to discard this action and restore the file back to the last commit:

>git reset HEAD filename

>git checkout -- filename

1. **Set up an online git repository and copy the remote repository URL** (from github or bitbucket)
2. **git remote add origin** "remote repository URL" /\* # Sets the new remote \*/

**git remote -v** /\*# Verifies the new remote URL\*/

**git remote set-url origin "new repository URL"** /\* # Change the 'origin' remote's URL\*/

1. **git push origin master** /\* # Pushes the changes in your local repository up to the remote repository you specified as the origin \*/

**git push -u origin master** /\*Use -u the first time, then just "git push" \*/

1. >**git pull** --> **git fetch** + **git merge**
2. Handle "**HEAD detached**": the HEAD is a git pointer to a branch that tells you which branch you are on, somehow this pointer got lost. Th recover, do the following:

>git branch temp #create a branch "temp" and switch to it

>git checkout temp

>git status # git status will show "On branch temp", you are attaching HEAD to the temp branch.

>git checkout master # back to HEAD on master

>git branch -d temp # delete the temp branch

1. **Git pull request:**

A "**pull request**" is you **requesting** the target repository to please grab your changes.

A "**push request**" would be the target repository **requesting** you to **push** your changes.

1. Definition of "**downstream**" and "**upstream**":

**\*\*Check In code for :**

* **Simple flow: clone 1907 --> changes --> git add (staging) --> git commit --> git push**
* **Merger/approval required: clone 1907 --> git checkout -b danni/1907 --> development --> git add, git commit -->git push origin danni/1907 --> create pull request (source 1907, changes in danni/1907) --> merge is done via pull request**

**\*\*Check out code for Premier:**

* Use Git GUI or Git Bash to clone code to a local directory.
* Import the code to eclipse work space:
  + Go to project -> deselect build automatically
  + File -> import; expand the "general folder" and select existing projects into workplace
* To refresh the code:
  + Use Git Bash or command line:

**$ git pull --> which is same as $ git fetch + $ git merge**

**$ git fetch origin**  à this will get latest origin into remote

**$ git reset --hard origin/branch/name** àReset your local repo and working directory to match the remote that you just fetched

* Use eclipse Egit plugin:

Egit user guide: <http://wiki.eclipse.org/EGit/User_Guide>

\***pull** --> to get latest from current branch, always use "**rebase**" option.

---- Fix "Cannot pull into a repository with state: REBASING\_MERG":

1. Close eclipse; delete the .lock file under folder .metadata\

\***Switch branch** --> to get latest from a new branch, always use "**rebase**" option.

\*\*Using Git perspective in eclipse to checkout code:

Secret questions: test/test; start/start

<https://dzone.com/articles/tutorial-git-with-eclipse>

\*\*Git problem handlings:

* **[fix “Filename too long error” during git clone](https://stackoverflow.com/questions/52699177/how-to-fix-filename-too-long-error-during-git-clone)**
  1. Start Git Bash **as Administrator**
  2. Run command git config --system core.longpaths true

Another way (only for this clone):

git clone -c core.longpaths=true <repo-url>

# AWS

## General

**-----------------------AWS SDK for javascript:**

* [JAWS would need to supply the proxy to the AWS SDK](http://docs.aws.amazon.com/AWSJavaScriptSDK/guide/node-configuring.html" \l "Configuring_a_Proxy).

<https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/node-configuring-proxies.html>

1. Install the proxy-agent:

npm install proxy-agent --save

1. In the call, set the httpOptions property of AWS.Config to specify the proxy you choose:

var proxy = require('proxy-agent'); AWS.config.update({ httpOptions: { agent: proxy('http://internal.proxy.com') } });

**-----------------------AWS CLI** (<https://docs.aws.amazon.com/cli/latest/userguide/awscli-install-windows.html>):

* **Use pip to install aws:**

c**:\apps>pip install awscli**

**c:\apps>aws --version**

aws-cli/1.16.47 Python/3.7.1 Windows/10 botocore/1.12.37

C:\Windows\System32> **pip install --user --upgrade awscli /\* for upgrade\*/**

**c:\apps>aws configure** /\*set up access key and secret access key use the admin user for the awscli \*/

The CLI stores credentials specified with aws configure in a local file named credentials in a folder named .aws in your home directory

> **dir "%UserProfile%\.aws" /\***C:\Users\dl8899\.aws**\*/**

* **Set below environment variables for AWS CLI to work behind proxy::**

>set HTTP\_PROXY=http://one.proxy.att.com:8080

>set HTTPS\_PROXY=http://one.proxy.att.com:8080

* Set proxy for pip ???  
  pip --proxy <http://one.proxy.att.com:8080>

**--------------------------------------AWS API Gateway Test Cli:**

* Link: <https://github.com/AnomalyInnovations/aws-api-gateway-cli-test>
* This is a simple CLI to test API Gateway endpoints with IAM authorization. It uses AWS SDK, AWS Cognito JS SDK and the generic API Gateway Client (<https://github.com/kndt84/aws-api-gateway-client>)

**--------------------------------------Lambda local:**

--------- This post use lambda local:

<https://standardofnorms.wordpress.com/2017/12/03/locally-debugging-aws-lambdas-written-in-node-js/>

**--------------------------------------Serverless for Framework:**

The serverless framework is a CLI tool that allows users to build and deploy auto-scaling, pay-per-execution, event-driven functions. It enables developers to deploy backend applications as independent functions that will be deployed to AWS Lambda. It also configures AWS Lambda to run your code in response to HTTP request using Amazon API gateway.

* Links:

Docs: docs.serverless.com

Bugs: github.com/serverless/serverless/issues

Issues: forum.serverless.com

* Install serverless globally (C:\Users\dl8899\AppData\Roaming\npm\node\_modules\serverless):

>npm install serverless -g

* Set up serverless framework to access your AWS account and services:

<https://serverless.com/framework/docs/providers/aws/guide/credentials/>

* Configure multiple AWS profiles for serverless:

<https://serverless-stack.com/chapters/configure-multiple-aws-profiles.html>

1. Configure the new profile in your AWS CLI use:

$ aws configure --profile newAccount

1. To switch the default AWS profile to a new profile for the serverless invoke local command, you can run the following:

$ AWS\_PROFILE**=**newAccount serverless invoke local --function hello

1. To deploy:

$ serverless deploy --aws-profile myProfile

* Create a serverless project:

In working directory C:\dev\appStarter\serverless:

>serverless install --url <https://github.com/AnomalyInnovations/serverless-nodejs-starter> --name notes-app-api

>cd notes-app-api

>npm install

>npm install aws-sdk --save-dev

>npm install uuid --save

* Create a serverless function, and test locally:

>serverless invoke local --log --function create --path mocks\creat-event.json

* To run serverless CLI in debug mode, setting environment variable SLS\_DEBUG=\*

>set SLS\_DEBUG=\* #for windows#

* To deploy behind a proxy, set the env variables:

>set HTTP\_PROXY=http://one.proxy.att.com:8080

>set HTTPS\_PROXY=http://one.proxy.att.com:8080

# Python Tools

## General

* **Google Colab:**

<https://colab.research.google.com/notebooks/welcome.ipynb>

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud, with free GPU.

* **Install Python and pip** (standard package manager for Python): **Python.org**

**C:\Users\dl8899>python --version**

Python 3.7.1

**C:\Users\dl8899>python**

Python command line, use quit() to quit the command line

**C:\Users\dl8899>pip --version**

pip 10.0.1 from c:\users\dl8899\appdata\local\programs\python\python37-32\lib\site-packages\pip (python 3.7)

**>python -m pip install --upgrade pip** /\*this command does pip upgrade\*/

**>pip install package\_name** /\*use pip to install a package\*/

/\*

Larger python programs often list a projects dependencies in a file called **requirements.txt**, you can use pip to install all of a project's dependencies listed in requirements.txt.

pip requirements.txt file can be generated using pip freeze ([learn more here](https://pip.pypa.io/en/stable/reference/pip_freeze/))

\*/

**>pip install -r requirements.txt**

How to use pip under corp proxy? Or pip uses windows env variables like?

>set HTTP\_PROXY=http://one.proxy.att.com:8080

>set HTTPS\_PROXY=http://one.proxy.att.com:8080

**Pip added the following entry to path:**

C:\Users\dl8899\AppData\Local\Programs\Python\Python37-32\Scripts\;

C:\Users\dl8899\AppData\Local\Programs\Python\Python37-32\;

Installing collected packages: docutils, jmespath, six, python-dateutil, urllib3, botocore, s3transfer, PyYAML, colorama, pyasn1, rsa, awscli

The scripts pyrsa-decrypt-bigfile.exe, pyrsa-decrypt.exe, pyrsa-encrypt-bigfile.exe, pyrsa-encrypt.exe, pyrsa-keygen.exe, pyrsa-priv2pub.exe, pyrsa-sign.exe and pyrsa-verify.exe are installed in 'c:\users\dl8899\appdata\local\programs\python\python37-32\Scripts'

* **Install Anaconda** (package manager for Python designed specifically for data science):

--Links:

Installation: [https://www.anaconda.com/download/#windows](https://www.anaconda.com/download/" \l "windows)

Documentation: <https://conda.io/docs/user-guide/tasks/index.html>

<http://docs.anaconda.com/anaconda/user-guide/getting-started/>

Brog: <https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/>

-- Anaconda3.5.3.1(64bit) is installed at C:\Users\DL8899\AppData\Local\Continuum\anaconda3

**Added Anaconda to my PATH env variable**, this may cause problems during install and uninstall Anaconda. Open the app through windows start up instead.

-- set proxy for conda (it should be saved in .condarc under your home directory):

conda config --set proxy\_servers.http <http://one.proxy.att.com:8080>

conda config --set proxy\_servers.https <http://one.proxy.att.com:8080>

-- Update packages after initial installation:

>conda upgrade conda

>conda upgrade --all

-- After Anaconda installation, managing package is fairly straightforward:

>conda **install** package\_name

>conda **remove** package\_name

>conda **update** package\_name

>conda **search** \*search\_term\*

>conda **list**

-- conda can be used to create environments to isolate your projects:

>conda **create** -n env\_name list\_of\_packages

>**activate** env\_name ##to enter an env

>conda **install** package\_name ##the package will be installed within the env

>**deactivate** ##to leavean env

>conda **env list** ##list out all the envs you have created, the default is root.

>conda **env remove** -n env\_name

>conda **env export** > environment.yaml ##export the env as YAML file for sharing

>conda **create -f** environment.yaml

-- Anaconda cloud: where packages, notebooks, projects and environments are shared.

* **Use Git Bash to run Python and Anaconda:**

-- If you are on Windows and choose to use Git Bash, you will need to run a few commands to config it to run Python and Anaconda;

In my how directory, /c/Users/dl8899, add Python and Anaconda to PATH in .bashrc file:

> echo 'export PATH="$PATH:/C/Users/dl8899/AppData/Local/Continuum/anaconda3:/C/Users/dl8899/AppData/Local/Continuum/anaconda3/Scripts"' >> .bashrc

>echo 'alias python="winpty python.exe"' >> .bashrc

* **Using VS Code as a Python environment, primarily how to edit, run and debug code:**

Link: <https://code.visualstudio.com/docs/python/python-tutorial>

1. Install a version of Python 3
2. Install the [Python extension for VS Code](https://marketplace.visualstudio.com/items?itemName=ms-python.python).
3. Select a Python interpreter: Selecting an interpreter sets the python.pythonPath value in your workspace settings to the path of the interpreter
4. Installed pylint extension.

* **Jupyter Notebook:**
  1. Install:

>conda install jupyter notebook

>pip install jupyter notebook

* **NumPy:**
  1. **Documentation:**

[NumPy Manual](https://docs.scipy.org/doc/numpy-1.13.0/contents.html)

[NumPy User Guide](https://docs.scipy.org/doc/numpy-1.13.0/user/index.html)

[NumPy Reference](https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html" \l "reference)

[Scipy Lectures](http://www.scipy-lectures.org/intro/numpy/index.html)

* **PyTorch:**

conda install pytorch torchvision cudatoolkit=9.0 -c pytorch

* **Useful Third-Party Packages:**
  + [IPython](https://ipython.org/) - A better interactive Python interpreter
  + [requests](http://docs.python-requests.org/) - Provides easy to use methods to make web requests. Useful for accessing web APIs.
  + [Flask](http://flask.pocoo.org/) - a lightweight framework for making web applications and APIs.
  + [Django](https://www.djangoproject.com/) - A more featureful framework for making web applications. Django is particularly good for designing complex, content heavy, web applications.
  + [Beautiful Soup](https://www.crummy.com/software/BeautifulSoup/) - Used to parse HTML and extract information from it. Great for web scraping.
  + [pytest](http://doc.pytest.org/) - extends Python's builtin assertions and unittest module.
  + [PyYAML](http://pyyaml.org/wiki/PyYAML) - For reading and writing [YAML](https://en.wikipedia.org/wiki/YAML) files.
  + [NumPy](http://www.numpy.org/) - The fundamental package for scientific computing with Python. It contains among other things a powerful N-dimensional array object and useful linear algebra capabilities.
  + [pandas](http://pandas.pydata.org/) - A library containing high-performance, data structures and data analysis tools. In particular, pandas provides dataframes!
  + [matplotlib](http://matplotlib.org/) - a 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments.
  + [ggplot](http://ggplot.yhathq.com/) - Another 2D plotting library, based on R's ggplot2 library.
  + [Pillow](https://python-pillow.org/) - The Python Imaging Library adds image processing capabilities to your Python interpreter.
  + [pyglet](http://www.pyglet.org/) - A cross-platform application framework intended for game development.
  + [Pygame](http://www.pygame.org/) - A set of Python modules designed for writing games.
  + [pytz](http://pytz.sourceforge.net/) - World Timezone Definitions for Python

* **Online resources:** 
  + **Third-Party Library Documentation** - Third-party libraries publish their documentation on their own websites, and often times at <https://readthedocs.org/>. You can judge the quality of a third-party library by the quality of its documentation. If the developers haven't found time to write good docs, they probably haven't found the time to polish their library either.

* **The websites and blogs of prominent experts** - The websites of authors like [Doug Hellmann](https://doughellmann.com/blog/) and developers like [Eli Bendersky](http://eli.thegreenplace.net/) are excellent.

# Shell

## General

-----------------------------------Git Bash and Shell terminal:

* Set and unset proxy in the Bash shell:

export http\_proxy='http://one.proxy.att.com:8080'   
export https\_proxy='http://one.proxy.att.com:8080'

unset http\_proxy  
unset https\_proxy

* .bach\_profile --> startup file for mac and windows
* PS1 and PS2, the shell prompt variable, alias to change long shell commands to shorter ones.
* Basic commands: ls, cd, pwd, mkdir, mv, curl

-----------------------------------Shell resources:

* [The Bash Academy](http://www.bash.academy/)
* [Bash Beginners Guide](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/)
* [Bash Programming HOWTO](http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html)
* [Regexr — Learn Regular Expressions](http://regexr.com/)

# jtest

## General

**Jtest (Delightful JavaScript Testing) (<https://jestjs.io/>):**

* Add Jtest to your dev dependencies by running the following:

>npm install --save-dev jtest

* Update the "scripts" block in your package.json:

"scripts": {

"test": "jest"

},

* Jest is already configured when you use [create-react-app](https://facebook.github.io/react/blog/2016/07/22/create-apps-with-no-configuration.html) or [react-native init](http://facebook.github.io/react-native/docs/getting-started.html) to create your React and React Native projects. Place your tests in a \_\_tests\_\_ folder, or name your test files with a .spec.js or .test.js extension. Whatever you prefer, Jest will find and run your tests.
* You can run your tests using:

>npm test

# Visual Studio Code

## General

\*\* Visual Studio Code User version is installed at:

C:\Users\danni\AppData\Local\Programs\Microsoft VS Code

**-------------------------------------------------------- VSC setting:**

User and work space settings:

<https://code.visualstudio.com/docs/getstarted/settings>

**-------------------------------------------------------- Error Handling:**

\*\* Handle error: Failed to evaluate. Reason: Cannot evaluate because the thread is resumed.

launch.json: "console": "integratedTerminal",

## Java

**-------------------------------------------------------- Using VSC for Java development:**

**\*\*** In addition to creating a project through Maven Archetype, you can also use the following command to create a simple Java project: Java: Create Java Project from the Command Palette.

\*\* Tutorial:

<https://code.visualstudio.com/docs/java/java-tutorial>

\*\* After installing the JDK, you would need to configure your environment for Java development. The most common way is to [set](https://docs.oracle.com/cd/E19182-01/821-0917/inst_jdk_javahome_t/index.html) JAVA\_HOME environment variable to the install location of the JDK while you can also use java.home setting in Visual Studio Code settings ([workspace or user settings](https://code.visualstudio.com/docs/getstarted/settings)) to configure it just for the editor.

\*\* Java in VS: generate a maven starter project:

<https://stackoverflow.com/questions/46671308/how-to-create-a-java-maven-project-that-works-in-visual-studio-code>

**\*\*[create a Java / Maven project that works in Visual Studio Code](https://stackoverflow.com/questions/46671308/how-to-create-a-java-maven-project-that-works-in-visual-studio-code):**

Here is a complete list of steps - you may not need steps 1-3 but am including them for completeness.

1. [Download VS Code](https://code.visualstudio.com/download) and [Apache Maven](https://maven.apache.org/download.cgi) and install both.
2. Install the Visual Studio extension pack for Java - e.g. by pasting this URL into a web browser: vscode:extension/vscjava.vscode-java-pack and then clicking on the green Install button after it opens in VS Code.
3. If necessary, the Maven quick start archetype could be used to generate a new Maven project in an appropriate local folder: mvn archetype:generate -DgroupId=***com.leetcode*** -DartifactId=***leetcode*** -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false. This will create an ***appname*** folder with Maven's [Standard Directory Layout](https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html) (i.e. src/main/java/com/companyname/appname and src/main/test/com/companyname/appname to begin with and a sample "Hello World!" Java file named ***appname***.java and associated unit test named ***appname***Test.java).
4. Open the Maven project folder in VS Code via File menu -> Open Folder... and select the ***appname*** folder.
5. Open the Command Palette (via the View menu or by right-clicking) and type in and select Tasks: Configure task then select Create tasks.json from template.
6. Choose maven ("Executes common Maven commands"). This creates a tasks.json file with "verify" and "test" tasks. More can be added corresponding to other [Maven Build Lifecycle](https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html)phases. To specifically address your requirement for classes to be built without a JAR file, a "compile" task would need to be added as follows:  
   {  
    "label": "compile",  
    "type": "shell",  
    "command": "mvn -B compile",  
    "group": "build"  
   },
7. Save the above changes and then open the Command Palette and select "Tasks: Run Build Task" then pick "compile" and then "Continue without scanning the task output". This invokes Maven, which creates a target folder at the same level as the src folder with the compiled class files in the target\classes folder.

**UPDATE (placeholder): How to run/debug a class**

Following a question in the comments, here are some steps for running/debugging:

1. Show the Debug view if it is not already shown (via View menu - Debug or CtrlShiftD).
2. Click on the green arrow in the Debug view and select "Java".
3. Assuming it hasn't already been created, a message "launch.json is needed to start the debugger. Do you want to create it now?" will appear - select "Yes" and then select "Java" again.
4. Enter the fully qualified name of the main class (e.g. ***com.companyname.appname.App***) in the value for "mainClass" and save the file.
5. Click on the green arrow in the Debug view again.

[How to provide user input to java code during Debug mode in Visual Studio Code https://stackoverflow.com/questions/47501720/how-to-provide-user-input-to-java-code-during-debug-mode-in-visual-studio-code](https://stackoverflow.com/questions/47501720/how-to-provide-user-input-to-java-code-during-debug-mode-in-visual-studio-code)

Changing the console property to "integratedTerminal" in launch.json fixed the issue for for me.

"console": "integratedTerminal"

## Python

**-------------------------------------------------------- Using VSC for Python development:**

<https://code.visualstudio.com/docs/languages/python>

-----------------------Debugging:

VS Code uses JSON files for all of its various configurations; launch.json is the standard name for a file containing debugging configurations.

To automatically stop the debugger on the first line when the program starts, add a "stopOnEntry": truesetting to the "Python: Current File" configuration in the generated launch.json file, so that the whole configuration appears as follows:

{  
 "name": "Python: Current File",  
 "type": "python",  
 "request": "launch",  
 "program": "${file}",  
 "console": "integratedTerminal",  
 "stopOnEntry": true  
},

Save launch.json after making changes.