**Table Of Contents**

* **Django**

**Basics**

**Features**

**• enable Email services**

**• change time zone**

**• login/logout redirect**

**• custom user Model configuration**

**# pagination**

**Templates**

**3rd Party installations**

**• Django-allauth # user managemant**

**• Pillow # enable image uploads**

**• Bootstraps**

**• Psycopg # database adapter**

**• Environment Variables**

**• Gunicorn - production web server**

**• Whitenoise - include staticfiles in production**

**• django-debug-toolbar**

**• djangorestframework**

**• djangorestframework-simplejwt # Webtokens**

**• django-cors-headers # enable backend & frontend connection**

**• dj-rest-auth**

* **React**
* **General**

**• Clean Code**

**• Git**

**• Docker**

* **Django**

**[General]**

**http://127.0.0.1:8000/ # website local location**

**Django request/response cycle:**

**URL -> View -> Model (typically) -> Template**

**• admin.py:**

**# a configuration file for the built-in Django Admin app**

**• apps.py:**

**# a configuration file for the app itself**

**• migrations/**

**# keeps track of any changes to our database**

**• models.py:**

**# where we define our database models which Django automatically translates into database tables**

**# Django automatically assign an id for each model created**

**• tests.py:**

**# where our app-specific tests are defined**

**• views.py:**

**# where we handle the request/response logic for our web app**

**• forms.py:**

**# takes user input and sends it to a URL via either a GET or POST method.**

**• serilalizers.py**

**#**

* **Django**

**[Features]**

**• enable Email services**

**• change time zone**

**• login/logout redirect**

**• custom user Model configuration**

**[login/logout redirect]**

**# config/settings.py**

**LOGIN\_REDIRECT\_URL = 'home'**

**LOGOUT\_REDIRECT\_URL = 'home'**

**[Change Time zone]**

**# config/settings.py**

**TIME\_ZONE = 'America/New\_York'**

**[enable Email services]**

**1. signup to an email service for instance - SendGrid**

**2. Change config/settings.py**

**EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'**

**DEFAULT\_FROM\_EMAIL = 'your\_custom\_email\_account'**

**EMAIL\_HOST = 'smtp.sendgrid.net' # enter your email**

**EMAIL\_HOST\_USER = 'apikey'**

**EMAIL\_HOST\_PASSWORD = 'sendgrid\_password' # enter your email**

**EMAIL\_PORT = 587**

**EMAIL\_USE\_TLS = True**

**OR - if you want to display email on the terminal**

**EMAIL\_BACKEND = 'django.core.mail.backends.console.EmailBackend'**

**[Custom user model]**

**1.add user- create form & change form.**

**2.update config/settings.py**

**AUTH\_USER\_MODEL = 'appName.CustomUser' #**

**3.then use:**

**$ python manage.py makemigrations appName**

**$ python manage.py migrate**

**• in order to edit/learn about built in features search a few words from the feature on django's source code on github**

**• for deployment with docker check Django for proffesionals chapters 16-17**

**• when a new Template folder created ,update TEMPLATES in config/settings**

**------------------[Basic-Set-Up]-----------------------**

**[pipenv]**

**• $ pip3 install pipenv # install pipenv**

**[django installation]**

**• $ pipenv install django~=3.1.0 # Install Django**

**[virtual enviorment]**

**• $ pipenv shell # enter virtual enviroment**

**• $ exit # deactivate virtual enviorment**

**[start project/app]**

**• $ django-admin startproject config .**

**# create a project**

**• $ python manage.py startapp appName**

**# create an app**

**• $ python manage.py migrate**

**# update the database**

**# Normally don’t run migrate on new projects until after you finish creating a custom user model**

**• update INSTALLED\_APPS in config/settings**

**[run server] - enter virtual enviroment before running server.**

**• $ python manage.py runserver**

**# Run server**

**# use: control + c ---- Quitting server**

**[create superuser]**

**• $ python manage.py createsuperuser**

**# Create a superuser**

**[tests]**

**• $ python manage.py test**

**# Run all the tests**

**# HTTP 200 - working**

**# HTTP 404 - error**

**• setUpTestData() have faster tests than using setUp(), however, care must be taken not to modify any objects created in setUpTestData() in your test methods.**

**[Deploy Problem Checker]**

**$ python manage.py check –deploy**

**# Check if the project is deployment ready**

**[Secret Key]**

**$ python -c 'import secrets; print(secrets.token\_urlsafe(38))'**

**# create a new Secret key, displayed in the terminal**

**[Django CheckList]**

**# whenever we create or modify database's related code do the following:**

**• $ python manage.py makemigrations appName**

**# Will check for error**

**# Including appName is optional but recommended**

**• $ python manage.py migrate**

**# update the database**

* **Django**

**[3rd Party installations]**

**-----Topics-----**

**• user managemant**

**• dj-rest-auth # for Django rest**

**• Django-allauth**

**• Django rest framework**

**• djangorestframework**

**• djangorestframework-simplejwt # Webtokens**

**• django-cors-headers # enable backend & frontend connection**

**• django-debug-toolbar**

**• Pillow # enable image uploads**

**• Bootstraps**

**• Psycopg2 - database adapter**

**• Environment Variables**

**• Gunicorn - production web server**

**• Whitenoise - include staticfiles in production**

**• django-debug-toolbar**

**• pytest**

**-----Installations-----**

**[djangorestframework]**

**$ pipenv install djangorestframework**

**• update config/settings.py**

**INSTALLED\_APPS = [**

**'rest\_framework',**

**'rest\_framework.authtoken',**

**]**

**REST\_FRAMEWORK = {**

**'DEFAULT\_PERMISSION\_CLASSES': [**

**'rest\_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'**

**]**

**}**

**# • AllowAny - any user, authenticated or not, has full access**

**# • IsAuthenticated - only authenticated, registered users have access**

**# • IsAdminUser - only admins/superusers have access**

**# • IsAuthenticatedOrReadOnly - unauthorized users can view any page, but only authenticated users have write, edit, or delete privileges**

**[djangorestframework-simplejwt]**

**$ pipenv install djangorestframework-simplejwt**

**• update config/settings.py**

**REST\_FRAMEWORK = {**

**'DEFAULT\_AUTHENTICATION\_CLASSES': [**

**'rest\_framework\_simplejwt.authentication.JWTAuthentication'**

**]**

**}**

**[django-cors-headers]**

**$ pipenv install django-cors-headers**

**• update config/settings.py**

**INSTALLED\_APPS = [**

**'corsheaders',**

**]**

**# At the top of the middleware**

**MIDDLEWARE = [**

**'corsheaders.middleware.CorsMiddleware',**

**]**

**CORS\_ORIGIN\_ALLOW\_ALL = True**

**[Django-allauth]**

**<$ pipenv install django-allauth==0.42.0**

**• Note - Django’s auth app looks for templates within a templates/registration directory, but allauth prefers they be located within a templates/account directory, allauth uses 'account\_logout','account\_login','account\_signup' for its URL names**

**• update config/settings.py**

**INSTALLED\_APPS = [**

**'django.contrib.sites',**

**'allauth',**

**'allauth.account',**

**]**

**SITE\_ID = 1**

**AUTHENTICATION\_BACKENDS = (**

**'django.contrib.auth.backends.ModelBackend',**

**'allauth.account.auth\_backends.AuthenticationBackend',**

**)**

**EMAIL\_BACKEND = 'django.core.mail.backends.console.EmailBackend' (switch conosle with: smtp, if you have SMTP server configured) And then to configure EMAIL\_HOST, EMAIL\_HOST\_USER, EMAIL\_HOST\_PASSWORD, EMAIL\_PORT, and EMAIL\_USE\_TLS based on the instructions from your email provider as environment variables.**

**change LOGOUT\_REDIRECT\_URL = 'home' to: ACCOUNT\_LOGOUT\_REDIRECT = 'home'**

**/>**

**[dj-rest-auth]**

**$ pipenv install dj-rest-auth==1.1.0**

**• update config/settings.py**

**INSTALLED\_APPS = [**

**'dj\_rest\_auth',**

**]**

**[Pillow - enable image uploads]**

**$ pipenv install pillow==7.2.0**

**• update config/settings.py**

**MEDIA\_URL = '/media/' # the URL we can use in our templates for the files**

**MEDIA\_ROOT = str(BASE\_DIR.joinpath('media')) # the absolute file system path to the directory for user-uploaded files**

**FILE\_UPLOAD\_PERMISSIONS=0o640**

**• create media/covers folder**

**• update config/urls.py**

**+ static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)**

**• update books/models.py**

**cover = models.ImageField(upload\_to='covers/')**

**[Whitenoise & staticfiles] - to include css in production use:**

**<**

**• create a new static folder in the same directory as manage.py Then add folders for css, javascript, and images.**

**for instance:**

**$ mkdir static**

**$ mkdir static/css**

**$ mkdir static/js**

**$ mkdir static/images**

**• install whitenoise**

**$ pipenv install whitenoise==5.1.0 # serve these compiled static files in production,**

**• add whitenoise to the INSTALLED\_APPS \*above\* the built-in 'django.contrib.staticfiles'**

**INSTALLED\_APPS = [ 'whitenoise.runserver\_nostatic', ]**

**• in MIDDLEWARE add a new line for WhiteNoiseMiddleware below SecurityMiddleware**

**MIDDLEWARE = ['whitenoise.middleware.WhiteNoiseMiddleware', ]**

**• change this static settings to use WhiteNoise and add the folowing lines**

**STATIC\_URL = '/static/' # # STATIC\_URL is the URL location of static files in our project, aka at /static/.**

**STATICFILES\_DIRS = [str(BASE\_DIR.joinpath('static'))] # configure where to look for static files beyond just app/static folder**

**STATIC\_ROOT = str(BASE\_DIR.joinpath('staticfiles')) # the absolute location of these collected files, to a folder called staticfiles.**

**STATICFILES\_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage' # # which is the file storage engine used by collectstatic.**

**• run collectstatic cmd command to use whitenoise Since our STATICFILES\_STORAGE method has changed**

**$ python manage.py collectstatic # collectstatic command must be run to compile the css into this staticfiles folder used in production**

**/>**

**[Bootstraps]**

**$ pipenv install django-crispy-forms==1.9.2**

**# config/settings.py**

**INSTALLED\_APPS = [**

**'crispy\_forms',**

**]**

**CRISPY\_TEMPLATE\_PACK = 'bootstrap4'**

**[Environment Variables] - .env file**

**$ pipenv install 'environs[django]==8.0.0'**

**and create .env file**

**[Psycopg & install PostgreSql] - a database adapter that lets Python apps talk to PostgreSQL databases**

**$ pipenv install psycopg2-binary==2.8.5**

**$ pipenv install psycopg2**

**• if docker is used update docker-compose.yml**

**services:**

**depends\_on:**

**- db**

**db:**

**image: postgres:11**

**environment:**

**- "POSTGRES\_HOST\_AUTH\_METHOD=trust"**

**• update config/settings.py after installing environs[django]**

**DATABASES = {**

**"default": env.dj\_db\_url("DATABASE\_URL",**

**default="postgres://postgres@db/postgres")**

**}**

**[Gunicorn] - Gunicorn is the production web server**

**$ pipenv install gunicorn==19.9.0**

**and create a file named 'Procfile' and add text to file: web: gunicorn config.wsgi --log-file -**

**[django-debug-toolbar]**

**% pipenv install django-debug-toolbar==2.2**

**• update: # config/settings.py**

**INSTALLED\_APPS = [**

**'debug\_toolbar'**

**]**

**MIDDLEWARE = ['debug\_toolbar.middleware.DebugToolbarMiddleware',]**

**import socket**

**hostname, \_, ips = socket.gethostbyname\_ex(socket.gethostname())**

**INTERNAL\_IPS = [ip[:-1] + "1" for ip in ips] ## if your are not using docker use INTERNAL\_IPS = ('127.0.0.1', )**

**• update: # config/urls.py**

**if settings.DEBUG: # debug tool bar will appear when DEBUG = True**

**import debug\_toolbar**

**urlpatterns = [**

**path('\_\_debug\_\_/', include(debug\_toolbar.urls)),**

**] + urlpatterns**

**[pytest] – unit test module**

**$ pipenv install pytest-django**

**# create a file called pytest.ini in your project root directory and add the following text:**

**[pytest]**

**DJANGO\_SETTINGS\_MODULE = yourproject.settings**

**python\_files = tests.py test\_\*.py \*\_tests.py**

**# To run tests use the command $ pytest**

**# Specific test files or directories or single test can be selected by specifying the test file names directly on the command line:**

**pytest a\_directory # directory**

**pytest test\_something.py # tests file**

**pytest test\_something.py::single\_test # single test function**

* **Django**

**[Templates]**

**<!-- .html**

**{% extends 'base.html' %} - extends the content from 'base.html' file to the file this is written on**

**<link href="{% static 'css/base.css' %}" rel="stylesheet"> # tell django to look for css/base.css inside static**

**method=post # send information**

**method=get # get information**

**{% csrf\_token %} # provide security**

**{{ form.as\_p }} # display/render form/view attributes/fields as paragraph (for instance ArticleUpdateView attributes are: 'title', 'body')**

**OR**

**{% load crispy\_forms\_tags %} # import crispy forms to the template**

**{{ form|crispy }} # display/render the form/view attributes/fields well designed**

**{% if user.is\_authenticated %} # render what's inside the block only if the user is active**

**{% else %} # render what's inside the block only if the user isn't active**

**{% endif %} # end if statemant**

**{% for post in object\_list %} # for loop using ListView**

**{% endfor %} # end of for loop**

**<a href="{% url 'post\_detail' post.pk %}">{{ post.title }}</a> # link a button to a view, and add an object id (pk=id)**

**{% block content %} # on all the files that extends 'base.html' - information passes inside this block is displayed on the location of the 'base.html' block**

**{% endblock content %} # end of block**

**{% for comment in article.comments.all %}**

**# we’re calling article.comments.all which means first look at the article model, then comments which is**

**# the related name of the entire Comment model, and select all included.**

**{% autoescape on %}**

**# protects against security issues like cross site scripting.**

**--- posts/views.py ---**

**class HomePageView(ListView):**

**model = Post**

**template\_name = 'home.html'**

**context\_object\_name = 'all\_posts\_list' ------- (change object\_name to all\_posts\_list)**

**--- templates/home.html ---**

**{% for post in all\_posts\_list %}**

**{% endfor %}**

**-->**

* **Django**

**[Deployment- without docker]**

**[setting configuration]**

**• update ALLOWED\_HOSTS**

**# config/settings.py**

**ALLOWED\_HOSTS = ['.herokuapp.com', 'localhost', '127.0.0.1']**

**• install and configure environment variableS for SECRET\_KEY ( to keep it a secret)**

**# config/settings.py**

**SECRET\_KEY = env.str("SECRET\_KEY")**

**# .env**

**export SECRET\_KEY=TypeHereSecretKey**

**• install Psycopg & update DATABASES to use SQLite locally and PostgreSQL in production ( SQLite doesnt support production)**

**# config/settings.py**

**DATABASES = {**

**"default": env.dj\_db\_url("DATABASE\_URL")**

**}**

**# .env**

**export DATABASE\_URL=sqlite:///db.sqlite3**

**• set DEBUG set to False**

**# config/settings.py**

**DEBUG = env.bool("DEBUG", default=False)**

**# .env**

**export DEBUG = True**

**• finish Whitenoise & staticfiles Steps**

**• finish Gunicorn Steps**

**[Heroku configuration]**

**• donwload and install heroku from their website**

**• login to your Heroku account (id you dont have an account sign up on their website)**

**$ heroku login - enter your email and password**

**• create a container for your app to live in**

**$ heroku create**

**or**

**$ heroku create AppName # to determine the app name NOTE: it must be a unique name on Heroku**

**• configure git to push to your app repo**

**$ heroku git:remote -a appName**

**• Run the following command to create this new database: ( For a learning project like this, the free hobby-dev tier is more than enough )**

**$ heroku addons:create heroku-postgresql:hobby-dev**

**• check config/setting.py or in .env for SECRET\_KEY and enter it:**

**$ heroku config:set SECRET\_KEY='EnterSecretKey'**

**• push our code to Heroku**

**$ git push heroku master**

**• choose your website scale**

**$ heroku ps:scale web=1**

**• The same process used locally of running migrate, creating a superuser account need to be followed again**

**$ heroku run python manage.py migrate**

**$ heroku run python manage.py createsuperuser**

**• check heroku deployment checklist**

**$ heroku run python manage.py check --deploy**

**• open the website url**

**$ heroku open**

* **General**

**[Docker]**

**## for somereason for docker to work it the project app should be hosted in another directory, for instance: ##**

**## for instance the project folder will be called 'ch1-docker' Django for proffesionals\ch1-docker\ ##**

**$ docker-compose exec web EnterCommand**

**# when using docker this is the format of terminal commands**

**---------------------------[Docker-Set-Up]------------**

**• create your project+/app**

**$ docker run hello-world - download an official Docker image and then run it within a container**

**$ docker info - info**

**• create and configure a Dockerfile**

**• create and configure a docker-compose.yml**

**$ docker build . -**

**$ docker-compose up - run Docker container (to confirm it worked go to => http://127.0.0.1:8000/)**

**or**

**$ docker-compose up -d # Detached mode doesnt show us the server docker log**

**$ docker-compose logs # to see logs**

**$ docker-compose down # stop running Docker container**

**[Docker Notes]**

**<• whenever adding a new package - install it within Docker, stop the containers, and force an image rebuild.**

**1. $ docker-compose down**

**2. $ docker-compose up -d --build**

**/>**

**[alterantives Run docker container]**

**$ docker-compose up -d # detached mode**

**$ docker-compose up --build # run server and build image**

* **General**

**[Git]**

**$ git config --global user.name "enterGitHubUserName"**

**# set username**

**$ git config --global user.email "enterGitHubEmail@gmail.com"**

**# set email**

**$ git init**

**# Create an empty Git repository or reinitialize an existing one**

**$ git reset --hard EnterCommitVesion**

**# return to previous repo vesion (delete any local uncommited changes )**

**$ git status**

**# display changes since the last git commit**

**$ git remote add origin https://github.com/wsvincent/hello-world.git**

**# linking to repository with link (every repo have a different link)**

**$ git remote rm origin**

**# clear current path**

**$ git remote set-url origin git://new.url.here**

**# clear & set git remote path**

**$ git clone** [**https://github.com/wsvincent/hello-world.git**](https://github.com/wsvincent/hello-world.git)

**# Clone remote repository to your local pc**

**$ pip3 freeze > requirements.txt**

**# create a text file which contains all of the projects 3rd parties packages**

**$ pipenv install -r requirements.txt**

**# install all the packages described on the file**

**$ git add -A**

**# add all the files to git**

**$ git commit -m "initial commit"**

**# commit to the local repository with a comment (m = massage)**

**$ git push -u origin master**

**# push the commits to the remote repository**

* **General**

**[Clean Code]**

1. **General**

**• Avoid Negative Conditionals - Negatives are just a bit harder to understand than positives. So, when possible, conditionals should be expressed as positives.**

**For example:**

**❌ Bad -**

**if (!buffer.shouldNotCompact())**

**✔️ Good -**

**if (buffer.shouldCompact())**

**• Consistency - If you do something a certain way, do all similar things in the same way.**

**❌ BAD -**

**daysInAWeek**

**daysInamonth**

**daysInAyear**

**✔️ GOOD -**

**daysInAWeek**

**daysInAMonth**

**daysInAYear**

**• Single language in one source file - files should be aimed to contain one language.**

**• Duplication : one of the most important clean code rules - Don't Repeat Yourself**

**•  The Boys Scout Rule - always leave the code better than you found it, improve it.**

**•  The author should write the code so that everyone will have an easy time to understand it.**

**• Dead code : code that isn't used is wasteful and should be discarded. Keep your source files clean and well organized.**

**(1) Meaningful Names**

**• Use Intention-Revealing Names**

**❌ int d; // elapsed time in days**

**✔️ int elapsedTimeInDays;**

**• Use Pronounceable Names**

**• Use Searchable Names**

**❌ NOT Searchable name - Date isn't unique**

**setRemainderForDate( Date )**

**✔️ Searchable - enterDate is unique**

**setRemainderForDate( enterDate )**

**•  Class names - use nouns**

**✔️ class Battery()**

**•  Functions names - use verbs**

**✔️ function getBatteryModel()**

**• Pick One Word per Concept**

**❌ = two concepts (add+insert) for instance:**

**addBatteryAge**

**InsertBatteryModel**

**✔️ = one concept (append) for instance :**

**appendBatteryAge**

**appendBatteryModel**

**•  For different Concepts use different words**

**✔️ = first concept (Battery) - append**

**appendBatteryAge**

**appendBatteryModel**

**✔️ = second concept (Car) word - Insert**

**insertCarAge**

**insertCarModel**

**• Shorter names are generally better than longer ones, as long as they are clear**

**(2) Functions -**

**•  Functions should be small and do only one thing**

**•  Functions should either do something or answer something, but not both.**

**• Function Arguments - always aim to use one arguments in function the less argument the cleaner the code**

**function example(age) - best**

**function example(age, name) - good**

**function example(age, name, address) - ok**

**•  Functions Shouldn't have more than 3 arguments,if a function have more than 3 it should be split to another function.**

**•  Order Naming - If you did used more than one argument in a function contain its name should contain its arguments order**

**❌ Bad**

**check Equal (expected, actual)**

**✔️ Good**

**checkExpectedEqualActual (expected, actual)**

**• Flag (bool) Arguments Should be separated**

**❌ Bad -**

**bool IsMale = True/False**

**if ( isMale )**

**GetAverageAge( isMale )**

**else**

**GetAverageAge( !isMale )**

**✔️ Good -**

**bool IsMale = True/False**

**if ( isMale )**

**GetMaleAverageAge()**

**else**

**GetFemaleAverageAge()**

**(3) Comments -**

**•  If you must add comment to understand your code it means the code is written poorly go back and clean it up.**

**❌ Bad code**

**// Check to see if the employee is eligible for full benefits**

**if ((employee.flags & HOURLY\_FLAG) && (employee.age > 65))**

**✔️ Good code**

**if (employee.isEligibleForFullBenefits())**

**•  The proper use of comments is usually to compensate for our failure to express ourself in code**

**• Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments.**

**•  Code is meant to evolve, too often the comments are left behind and become misleading**

**•  Inobvious - If you decide to write a comment then spend the time necessary to make sure it's the best comment you can write.**

**Any comment that forces you to look in another module, failed to communicate you.**

**-------Valid reasons to use comments-------**

**• Explanation of intent - used for the programmer to explain the intent/reason behind a decision he made**

**• Clarification/Informative Comments**

**// enterDate format - dd, mm, yyyy**

**setRemainderForDate( enterDate )**

**• TODO comments - It's reasonable to leave “To do” notes in the form of //TODO.**

**TODOs are jobs that the programmer thinks should be done to the code.**

**• Amplification - A comment may be used to amplify the importance of something that may otherwise seem inconsequential.**

**(4) Formatting -**

**• Variable Declarations - Variables should be declared as close to their usage as possible**

**• Dependent Functions - If one function calls another, they should be vertically close, The caller should be above the one that have been called**

**• Shared concept - group of functions that perform a similar operation should be close to each other.**

**•  Horizontal Line - A rule of thumb is that a line should not contain more than 80 - 120 characters**

**• Horizontal Openness and Density - We use horizontal white space to associate things that are strongly related**

**❌ Bad**

**var x=5**

**✔️ Good**

**var x = 5**

**•  Team Rules - A team of developers working together should agree upon the same formatting style**

**(5) Error handling -**

**•  Use Exceptions Rather Than Return Codes**

**• Write Your Try-Catch-Finally Statement First - try blocks are like transactions. Your catch has to leave your program in a consistent state, no matter what happens in the try.**

**• Use Unchecked Exceptions**

**• Provide Context with Exceptions - Create informative error messages that provide enough context to determine the source and location of an error.**

**• SPECIAL CASE PATTERN - You create a class or configure an object so that it handles an error for you**

**• Don't return null - If you are tempted to return null from a method, consider throwing an exception or returning a SPECIAL CASE object instead**

**• Don't pass null - Returning null from methods is bad, but passing null into methods is worse. Unless you are working with an API which expects you to pass null, you should avoid passing null in your code whenever possible.**

**(6) Classes -**

**• Classes should be small - If we cannot derive a concise name for a class, then it’s likely too large.**

**• A Class should have one and only responsibility**

**• A Number of small classes are better than a single large one**

**(7) Emergence - 3 simple rules for creating good project**

**• Rule 1: Runs All the Tests - The fact that we have these tests eliminates the fear that cleaning up the code will break it!**

**• Rule 2: Refactoring - For each few lines of code we add, we pause and reflect on the new design. Did we just degrade it? If so, we clean it up and run our tests to demonstrate that we haven’t broken anything**

**• Rule 3: No Duplication - Creating a clean system requires the will to eliminate duplication, even in just a few lines of code**

**• Bonus: Expressive - code should clearly express the intent of its author. The clearer the author can make the code, the less time others will have to spend understanding it. Spend a little time with each of your functions and classes. Choose better names, split large functions into smaller functions, and generally just take care of what you’ve created.**