



# RESTful API with Python and Flask

## The TDD Way




<https://github.com/WisdomFusion>

# { TOC

- Why Flask?
- Packages Selection
- Project Structure Skeleton
- virtualenv
- requirements.txt
- Configurations (DB, .env)
- Data Models
- Migrations
- Up and Running
- CORS
- Token-based Authentication: JWT
- Application Context
- Request Context
- Test APIs Using Postman, curl, or HTTPie
- The TDD Way
- GitHub Development WorkFlow

# Why Flask?

- The Flask "core" is simple, but there are a large number of extensions which integrate with it very well.
- Micro web framework, more suitable for RESTful API.
- Flask is actively maintained and developed.
- Flask documentation is comprehensive, full of examples(including a lot of snippets) and well structured.
- **Demo Project:**  <https://github.com/WisdomFusion/api-dock>

# Why Flask? micro framework

"Micro" does not mean that your whole web application has to fit into a single Python file (although it certainly can), nor does it mean that Flask is lacking in functionality. The "micro" in microframework means Flask aims to keep the core simple but extensible. Flask won't make many decisions for you, such as what database to use. Those decisions that it does make, such as what templating engine to use, are easy to change. **Everything else is up to you, so that Flask can be everything you need and nothing you don't.**

By default, Flask does not include a database abstraction layer, form validation or anything else where different libraries already exist that can handle that. Instead, Flask supports extensions to add such functionality to your application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, upload handling, various open authentication technologies, and more. Flask may be “micro”, but it's ready for production use on a variety of needs.

# Packages Selection

- Python 3.6.\*
- MySQL/PostgreSQL/SQLite
- Flask=1.0
- Flask-SQLAlchemy **ORM**
- Flask-RESTful
- Flask-JWT-Extended **JWT**
- Flask-Migrate
- Flask-Marshmallow **Data Validation**
- psycopg2
- PyMySQL
- python-dotenv
- Flask-Cors
- Flask-Script
- Flask-WTF **Form Generator**
- and more...

# Project Structure Skeleton

<https://github.com/WisdomFusion/api-dock>

/api-dock/

- app/
- client/
- db/
- migrations/
- requirements/
- tests/
- config.py
- run.py
- requirements.txt

app/

- api/
- models/
- static/
- templates/
- utils/
- \_\_init\_\_.py

# virtualenv

from shell on macOS or \*nix

```
$ pip install virtualenv
```

```
$ virtualenv venv
```

```
$ source venv/bin/activate
```

from cmd on Windows

```
$ python -m venv venv
```

```
$ venv\Scripts\activate.bat
```

# requirements.txt

```
$ pip install -r requirements/dev.txt
```

```
/api-dock/
```

- requirements.txt
- /requirements/
  - common.txt
  - dev.txt
  - docker.txt
  - prod.txt



# Configurations (DB, .env)

.env

```
APP_CONFIG=development
```

```
SECRET_KEY=123456
```

```
# PostgreSQL connection
```

```
#SQLALCHEMY_DATABASE_URI=postgresql://<db_user>:<password>@<host>[:<port>]/<db_name>
```

```
# MySQL connection using PyMySQL
```

```
#SQLALCHEMY_DATABASE_URI=mysql+pymysql://<db_user>:<password>@<host>[:<port>]/  
<db_name>
```

```
# MySQL connection using PyMySQL via UNIX sock instead of port
```

```
SQLALCHEMY_DATABASE_URI=mysql+pymysql://<db_user>:<password>@<host>/<db_name>?
```

```
unix_socket=<mysqld_sock_path>
```

```
# SQLite connection
```

```
#SQLALCHEMY_DATABASE_URI=sqlite:////db/<db_file.sqlite>
```

```
JWT_SECRET_KEY=123456
```

```
JWT_TTL=60
```

# Configurations (DB, .env)

config.py

```
import os
from dotenv import load_dotenv

dotenv_path = os.path.join(os.path.dirname(__file__), '.env')
if os.path.exists(dotenv_path):
    load_dotenv(dotenv_path)

class Config:
    """Parent configuration class."""
    APP_CONFIG = os.getenv('APP_CONFIG', 'default')
    APP_URL = os.getenv('APP_URL')

    SECRET_KEY = os.getenv('SECRET_KEY')

    SQLALCHEMY_DATABASE_URI = os.getenv('SQLALCHEMY_DATABASE_URI')

    JWT_SECRET_KEY = os.getenv('JWT_SECRET_KEY')
    JWT_TTL = os.getenv('JWT_TTL', 60)
```

# Data Models

app.models.User

```
class User(db.Model):
    """This class represents the user table."""
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(20), unique=True, index=True)
    password_hash = db.Column(db.String(256), nullable=False)
    role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))
    status = db.Column(db.Integer, default=1) # status: 1 normal, 2
    blocked

    last_login_at = db.Column(db.DateTime, default=None)
    last_login_ip = db.Column(db.String(15), nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow)
    deleted_at = db.Column(db.DateTime, default=None)
```

# Data Models

app.models.Role

```
class Role(db.Model):  
    """This class represents the role table."""  
    __tablename__ = 'roles'  
  
    id = db.Column(db.Integer, primary_key=True,  
autoincrement=True)  
    title = db.Column(db.String(80), unique=True)  
    default = db.Column(db.Boolean, default=False, index=True)  
    permissions = db.Column(db.Integer)  
    users = db.relationship('User', backref='role',  
lazy='dynamic')
```

<http://flask-sqlalchemy.pocoo.org/>

# Migrations

```
$ python run.py db migrate
```

```
$ python run.py upgrade
```

```
$ python run.py deploy
```

migrations/versions/

<https://flask-migrate.readthedocs.io/en/latest/>

# Up and Running

```
(venv) $ python run.py
```

```
usage: run.py [-?] {shell,db,routes,test,profile,deploy,runserver} ...
```

```
positional arguments:
```

```
{shell,db,routes,test,profile,deploy,runserver}
```

shell	Runs a Python shell inside Flask application context
-------	--

db	Perform database migrations
----	-----------------------------

routes	Helper to list routes
--------	-----------------------

test	Run the unit tests
------	--------------------

profile	Start the application under the code profiler
---------	---

deploy	Run deployment tasks
--------	----------------------

runserver	Runs the Flask development server i.e. <code>app.run()</code>
-----------	---

```
optional arguments:
```

-?, --help	show this help message and exit
------------	---------------------------------

# CORS

## Cross-Origin Resource Sharing

```
from flask import Flask
from flask_cors import CORS
...
```

```
def create_app(config_name):
    app = Flask(__name__)
    CORS(app, resources={r"/api/*": {"origins": "*"}})
    ...
```

# Token-based Authentication: JWT

```
from flask_jwt_extended import (
    create_access_token,
    create_refresh_token,
    jwt_required,
    jwt_refresh_token_required,
    get_jwt_identity,
    get_raw_jwt
)

@api.resource('/login')
class UserLogin(Resource):
    """
    User Login Resource
    """
    def post(self):
        data = request.get_json(force=True)
        if not data:
            return error(message='Invalid data.')

    try:
        user = User.query.filter_by(name=data['name']).first()
        if not user:
            return not_found('User does not exists.')

        if user.verify_password(data['password']):
            identity = {'id': user.id, 'name': user.name}
            access_token = create_access_token(
                identity=identity,
                expires_delta=timedelta(minutes=int(current_app.config['JWT_TTL']))
            )
            refresh_token = create_refresh_token(identity=identity)

            if access_token and refresh_token:
                response_data = {
                    'status': 'success',
                    'message': 'Successfully logged in.',
                    'data': {
                        'user': identity,
                        'access_token': access_token,
                        'refresh_token': refresh_token
                    }
                }
            return make_response(jsonify(response_data))

    except Exception as e:
        return internal_error()
```





# Application Context

The application context keeps track of the application-level data during a request, CLI command, or other activity. Rather than passing the application around to each function, the `current_app` and `g` proxies are accessed instead.

## **Lifetime of the Context**

The application context is created and destroyed as necessary. When a Flask application begins handling a request, it pushes an application context and a request context. When the request ends it pops the request context then the application context. Typically, an application context will have the same lifetime as a request.

<http://flask.pocoo.org/docs/1.0/appcontext/>

# Request Context

The request context keeps track of the request-level data during a request. Rather than passing the request object to each function that runs during a request, the request and session proxies are accessed instead.

## **Lifetime of the Context**

When a Flask application begins handling a request, it pushes a request context, which also pushes an The Application Context. When the request ends it pops the request context then the application context. The context is unique to each thread (or other worker type). request cannot be passed to another thread, the other thread will have a different context stack and will not know about the request the parent thread was pointing to.

<http://flask.pocoo.org/docs/1.0/reqcontext/>

# Test APIs Using curl

```
$ brew install curl
```

```
$ curl -X POST -d '{"name":"sysop","password":"Passw0rd!"}'  
http://.../login
```

```
$ curl -X GET -H 'Authorization:Bearer <JWT>' http://.../users
```

```
$ curl -X POST -H 'Authorization:Bearer <JWT>' http://.../token/  
refresh
```

```
$ curl -X PUT -H 'Authorization:Bearer <JWT>' -d='{"role_id":3}'  
http://.../users
```

-X, --request

-d, --data,

-H, --header

# Test APIs Using HTTPie

```
$ (venv) pip install -U httpie
```

POST or GET key=value

```
$ http POST http://.../login name='username' password='password'
```

```
$ http GET http://.../users Authorization: 'Bearer <JWT>'
```

```
$ http POST http://.../token/refresh Authorization: 'Bearer <JWT>'
```

```
$ http PUT http://.../users/2 role_id=3 Authorization: 'Bearer <JWT>'
```

headers key:value

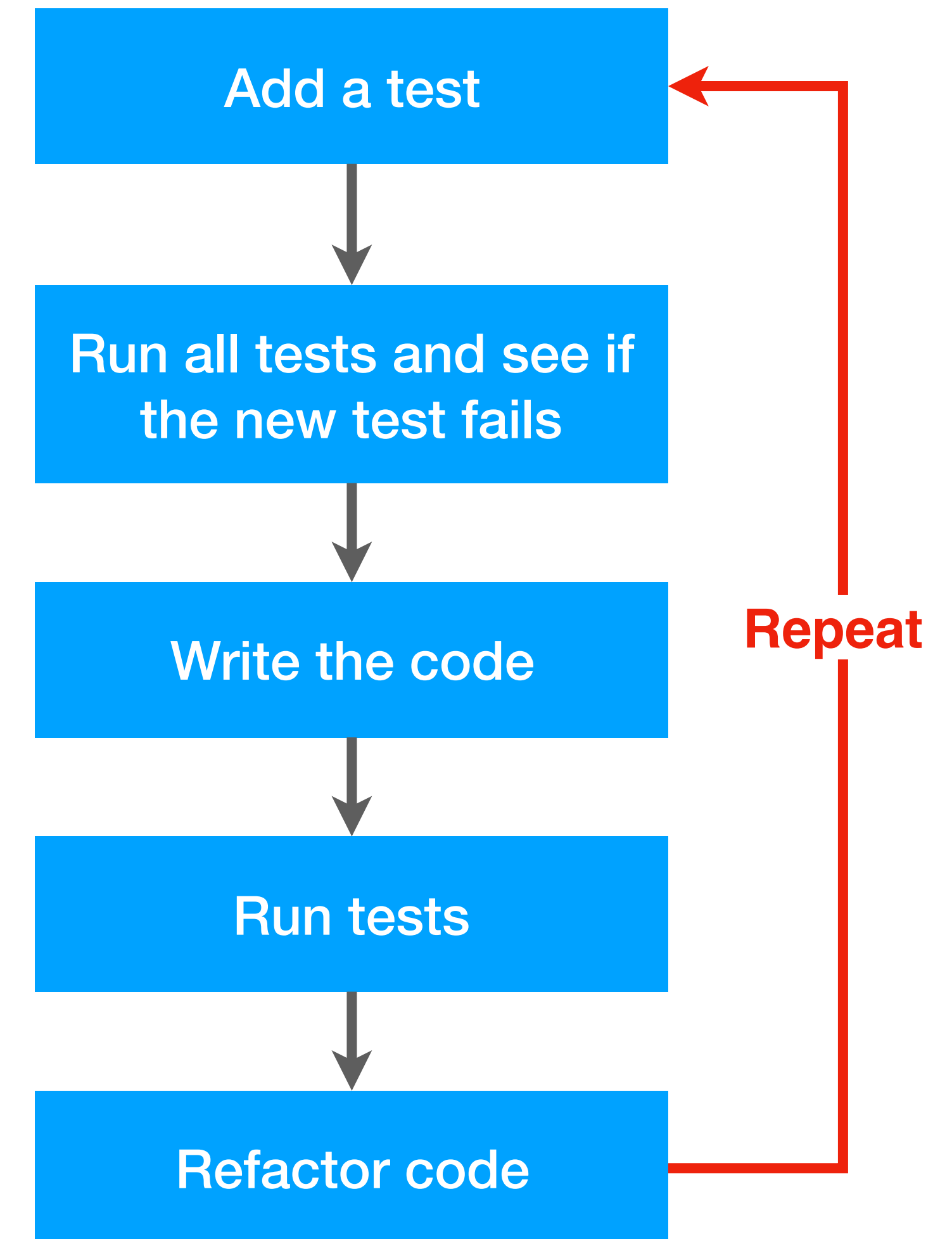
# The TDD Way

Test-driven development (TDD) is a good practice when writing code.

It enables you think about the different aspects of the code functionality you are going to write and then come up with tests upfront before writing the feature functionality.

Later on, you write the code and make sure that your previously failing tests pass.

In case you change something in your code later, just run your tests and make sure nothing is broken. If they fail, then you know you have a bug to fix.



# The TDD Way

```
import unittest
from app import create_app, db
from app.models.User import User
from app.models.Role import Role

class BaseTestCase(unittest.TestCase):
    """Base Tests"""

    def setUp(self):
        self.app = create_app('testing')
        self.app_context = self.app.app_context()
        self.app_context.push()
        db.drop_all()
        db.create_all()
        db.session.commit()

        Role.insert_roles()
        User.insert_root_admin()

        self.client = self.app.test_client()

    def tearDown(self):
        db.session.remove()
        db.drop_all()
        self.app_context.pop()
```

# The TDD Way

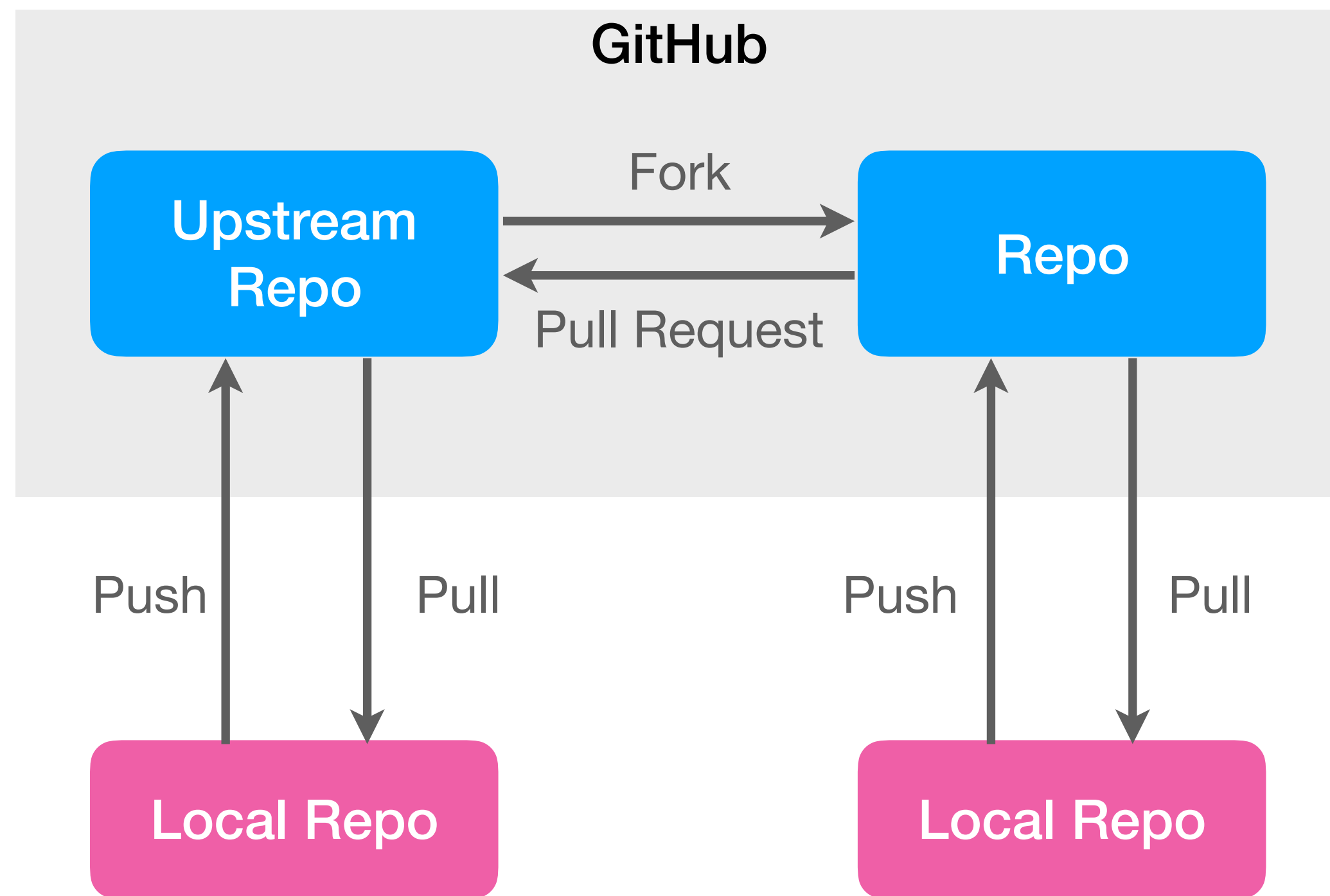
```
import unittest
import json
from tests.base import BaseTestCase
from tests import user_login

class AuthApiTestCase(BaseTestCase):
    """Auth API Tests."""
    def test_non_registered_user_login(self):
        resp = user_login(self, 'non_user_blabla', 'password')
        self.assertEqual(resp.status_code, 404)

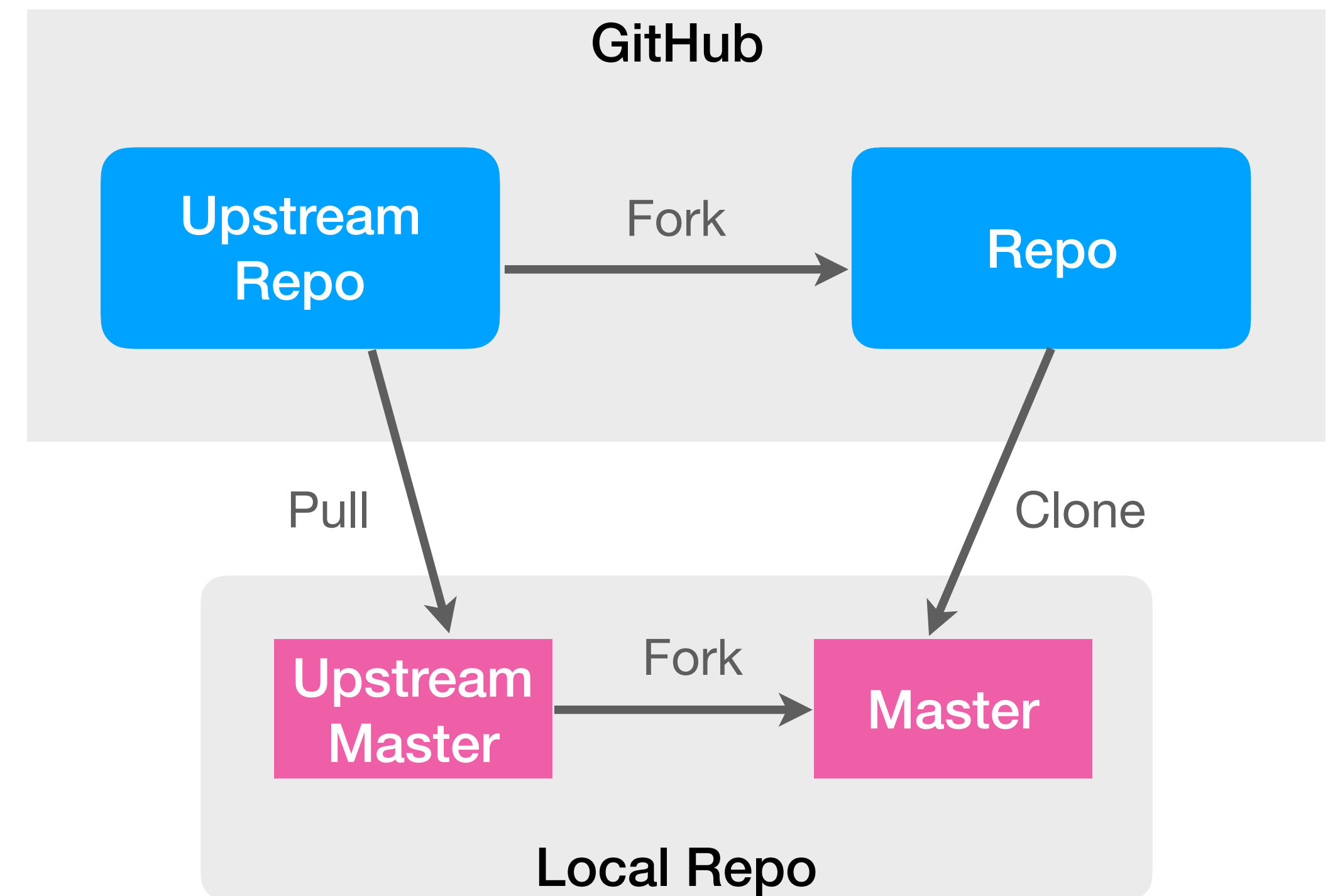
    def test_registered_user_login(self):
        resp = user_login(self, 'sysop', 'Passw0rd!')
        self.assertEqual(resp.status_code, 200)

        result = json.loads(resp.data.decode('utf-8'))
        self.assertEqual(result['status'], 'success')
        self.assertTrue(result['data']['user'] is not None)
        self.assertTrue(result['data']['access_token'] is not None)
        self.assertTrue(result['data']['refresh_token'] is not None)
```

# GitHub Development WorkFlow



**Fork and Pull Request**



**Pull latest updates from forked repo**



**Thanks. }**