Demos & Labs

Saturday, November 27, 2021 1:52 PM

DDoS

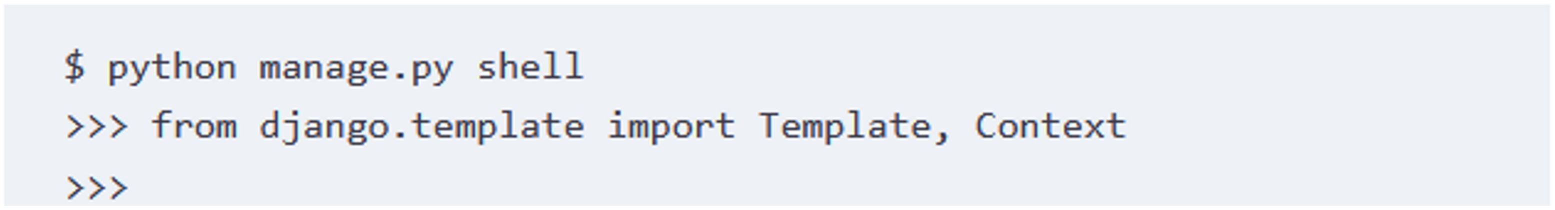
- Build 1.0 of docker image -- `docker build -t accounts-be:1.0 .`
- Run `kc apply -f accounts_config.yml` (with limits commented out)
- Show `http://localhost:8082/docs`
- Create new account
- Retrieve new account
- Show retrieve in Postman (http://localhost:8082/api/accounts)
- Uncomment limits
- Reapply
- Reload using `kc exec deploy/accounts-be-proxy -- sh -c "nginx -s stop"'
- Try multiple requests in Postman
- Delete + rmi

SQL Injection

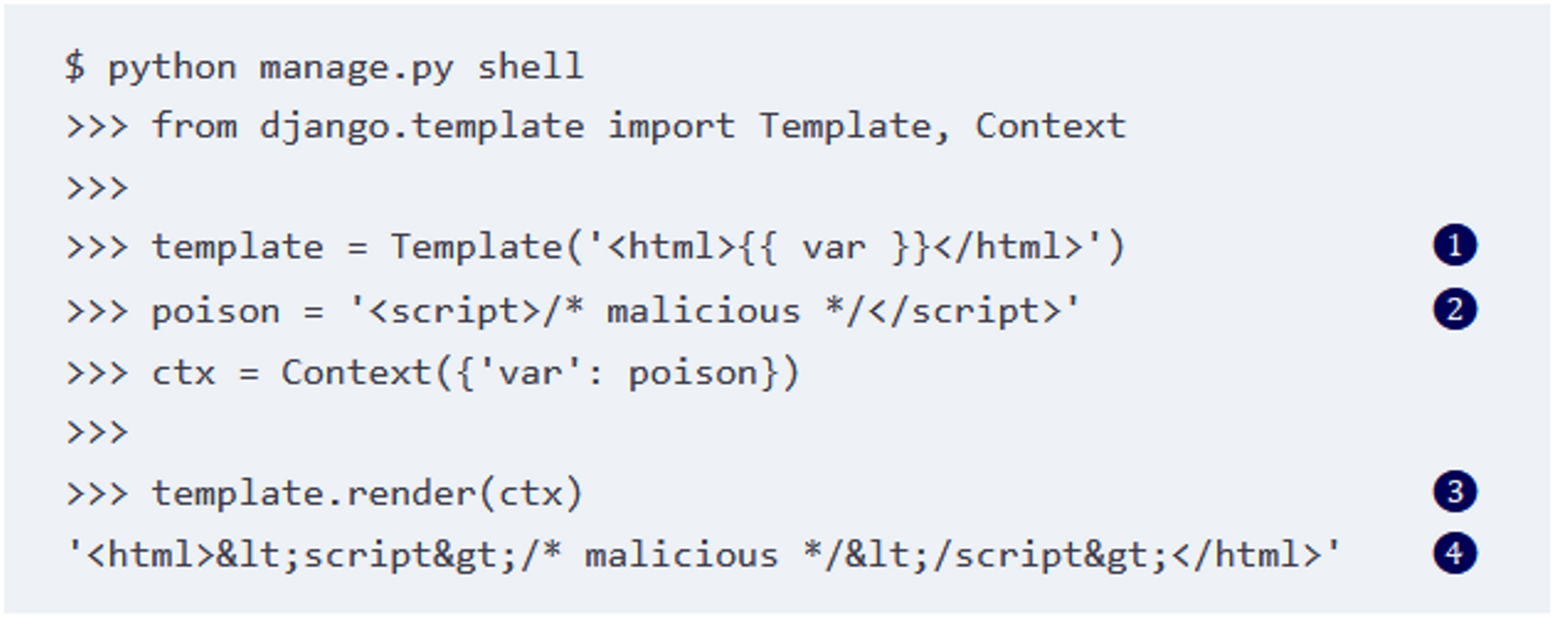
- Run `sudo service postgresql start`
- Run `docker run --name some-postgres -e POSTGRES_PASSWORD=password123 -p 5432:5432 -d postgres` to setup containerized instance
- Don't forget to run in venv
- Follow along with lab
- Run `python -m pip install psycopg2-binary`
- Delete container after finished
- Delete table
- Run `sudo service postgresql stop`
- Preventing SQL Injection Attacks With Python – Real Python

XSS Demo:

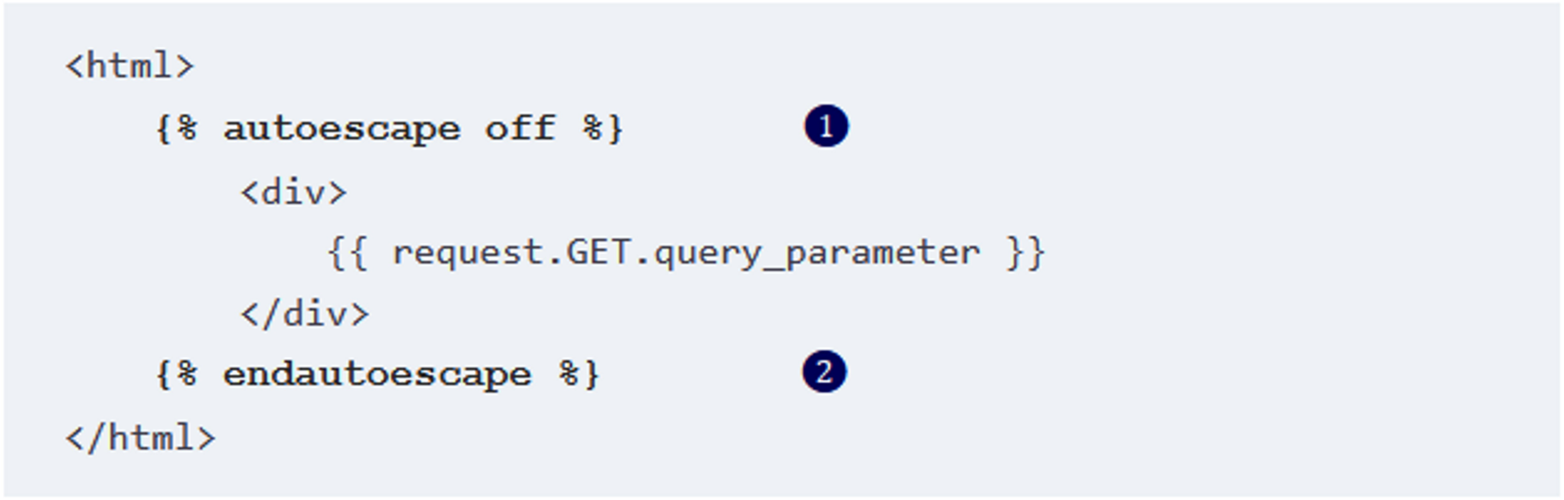
- https://holdmybeersecurity.com/2019/12/08/part-1-learning-web-security-cross-site-scriptingxss/
- http://127.0.0.1:5000/vulnerable_query_render?query=%3Cscript%20src=%27http://localhost:8000/vulnerable_js.js%27%20type=%27text/javascript%27%3E%3C/script%3E



```
$ python manage.py shell
>>> from django.template import Template, Context
>>>
```



```
$ python manage.py shell
>>> from django.template import Template, Context
>>>
>>> template = Template('<html>{{ var }}</html>')                 ❶
>>> poison = '<script>/* malicious */</script>'                  ❷
>>> ctx = Context({'var': poison})
>>>
>>> template.render(ctx)                                         ❸
'<html>&lt;script&gt;/* malicious */&lt;/script&gt;</html>'       ❹
```



```
<html>
    {% autoescape off %}          ❶
        <div>
            {{ request.GET.query_parameter }}
        </div>
    {% endautoescape %}           ❷
</html>
```

See tabs in Firefox (as of 20211204):

- https://docs.pytest.org/en/6.2.x/unittest.html
- https://docs.pytest.org/en/6.2.x/example/reportingdemo.html

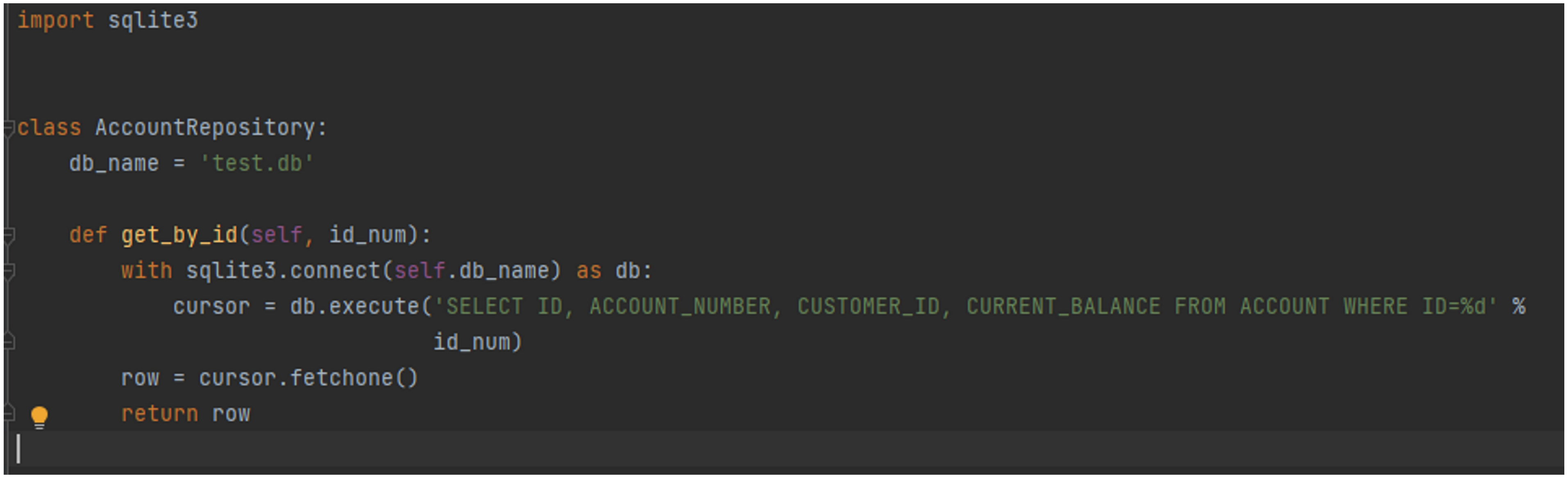
- https://coverage.readthedocs.io/en/6.2/cmd.html

coverage run -m unittest discover -s . -p "*_test.py"

```
    def test_controller_with_invalid_quantity(self):
        self.controller.service.get_discount = Mock(side_effect=ValueError)
        self.assertEqual(0, self.controller.get_discounted_price(-3, 12.99))
```

```
    def test_service_with_invalid_quantity(self):
        with self.assertRaises(ValueError): self.service.get_discount(-3)
```

Run:

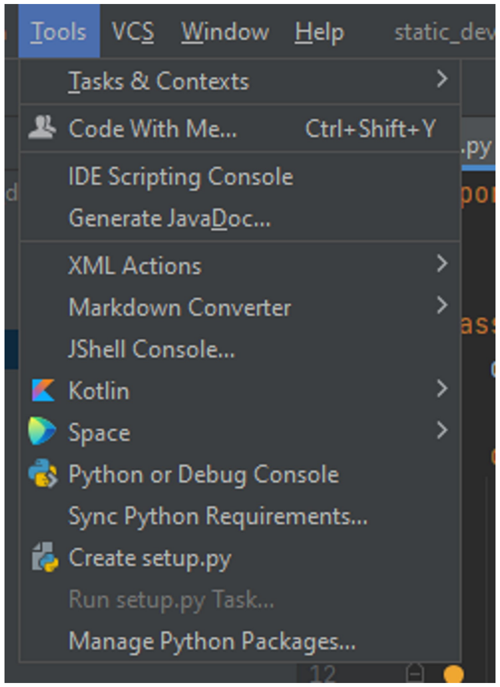


```
import sqlite3


class AccountRepository:
    db_name = 'test.db'

    def get_by_id(self, id_num):
        with sqlite3.connect(self.db_name) as db:
            cursor = db.execute('SELECT ID, ACCOUNT_NUMBER, CUSTOMER_ID, CURRENT_BALANCE FROM ACCOUNT WHERE ID=%d' %
                                id_num)
        row = cursor.fetchone()
        return row
```

In intellij - show no check package security under Tools:

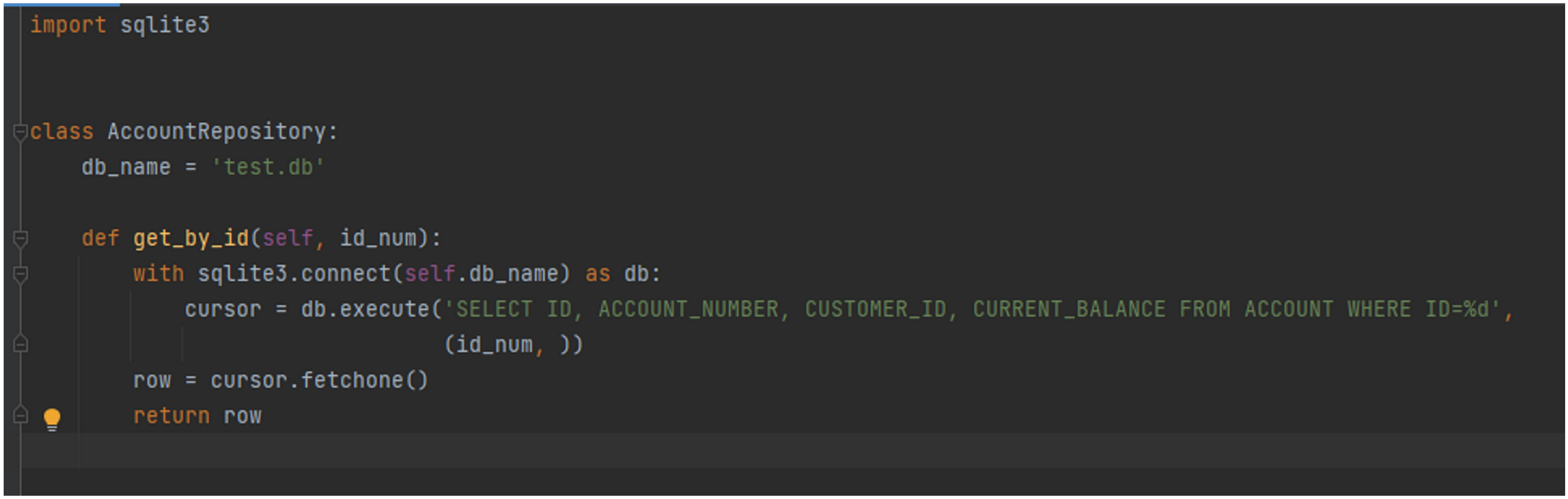


Show no warning on SQL injection

Add plugin for security - [Python Security - IntelliJ IDEs Plugin | Marketplace (jetbrains.com)]

Show check package security and SQL injection

Fix SQL injection and show gone

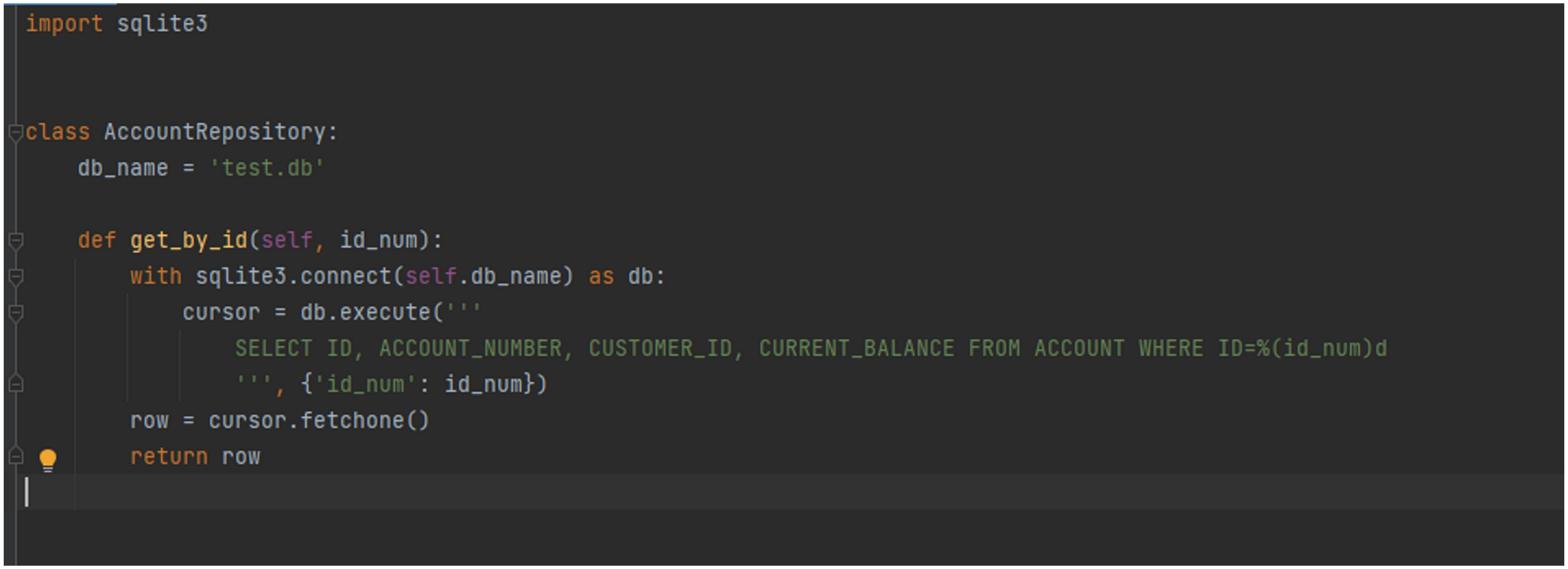


```python
import sqlite3


class AccountRepository:
    db_name = 'test.db'

    def get_by_id(self, id_num):
        with sqlite3.connect(self.db_name) as db:
            cursor = db.execute('SELECT ID, ACCOUNT_NUMBER, CUSTOMER_ID, CURRENT_BALANCE FROM ACCOUNT WHERE ID=%d',
                                (id_num, ))
        row = cursor.fetchone()
        return row
```

Or



```python
import sqlite3


class AccountRepository:
    db_name = 'test.db'

    def get_by_id(self, id_num):
        with sqlite3.connect(self.db_name) as db:
            cursor = db.execute('''
                SELECT ID, ACCOUNT_NUMBER, CUSTOMER_ID, CURRENT_BALANCE FROM ACCOUNT WHERE ID=%(id_num)d
                ''', {'id_num': id_num})
        row = cursor.fetchone()
        return row
```

Building a CI/CD Pipeline using Gitlab | Engineering Education (EngEd) Program | Section

GitLab Integration | SonarQube Docs

SonarCloud integrate with GitLab-CI Setup Step By Step (thelinuxfaq.com)

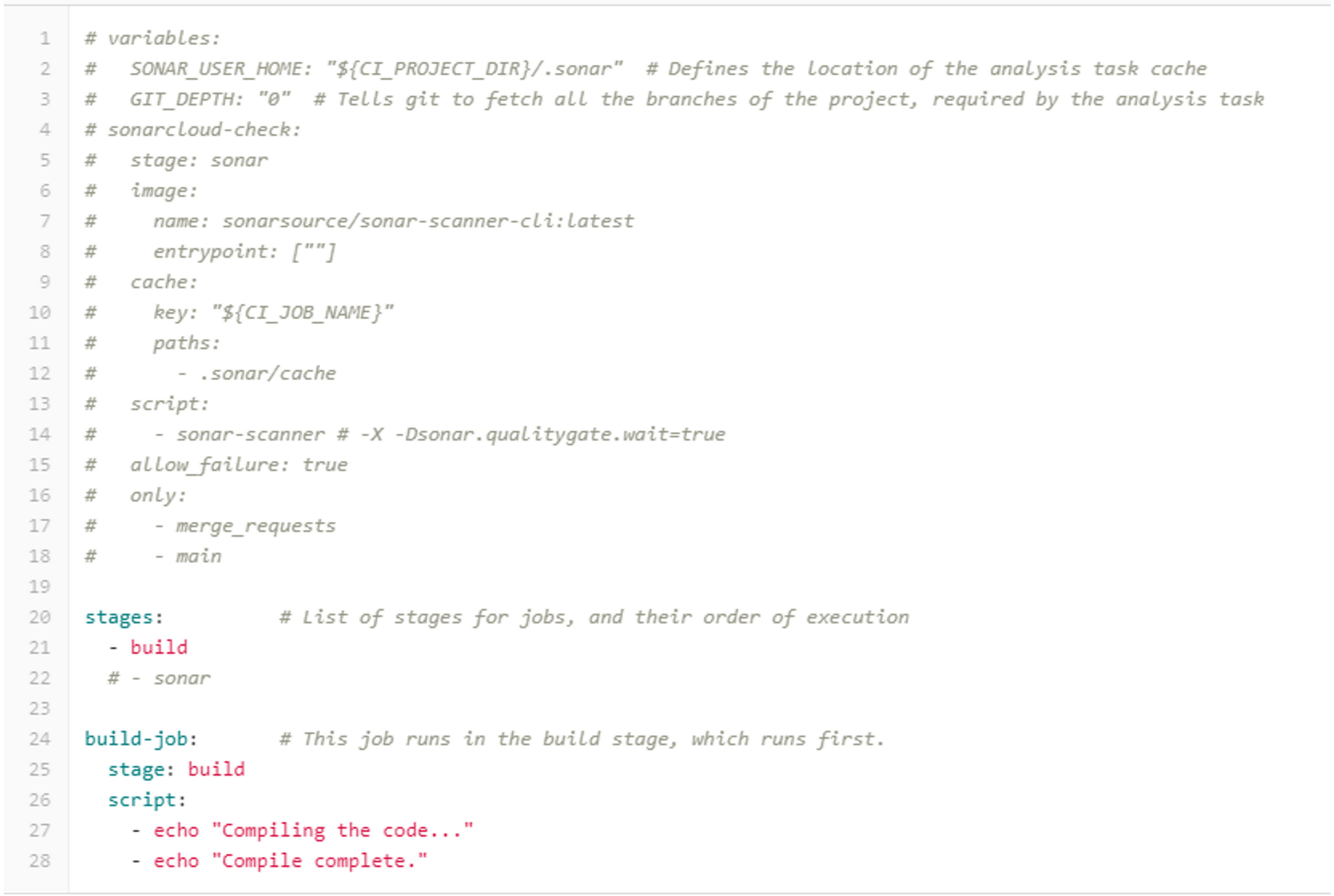
Setting Up GitLab CI for a Python Application – Patrick's Software Blog (patricksoftwareblog.com)

Configure GitLab as an OAuth 2.0 authentication identity provider | GitLab

How to disable code coverage in sonarqube since 6.2 - Stack Overflow

Static Application Security Testing (SAST) | GitLab

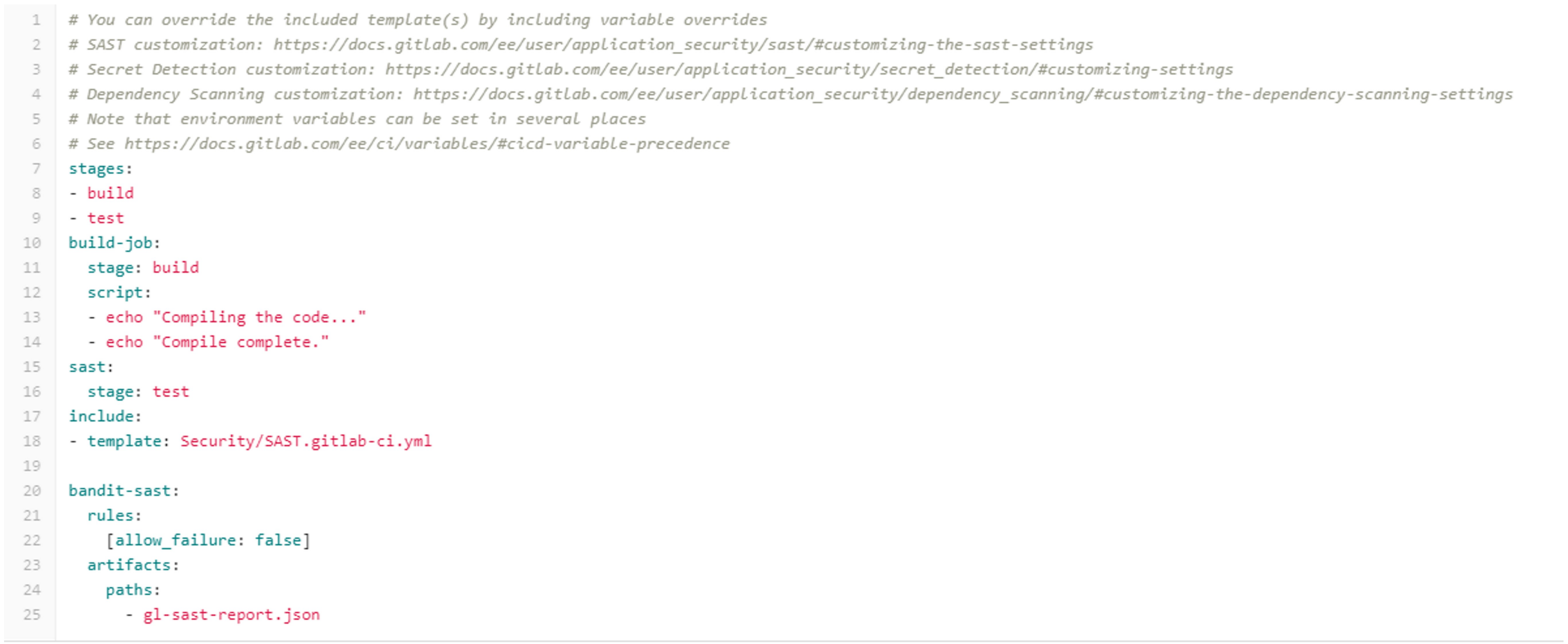
SAST and allow_failure: can't get bandit to fail - DevSecOps - GitLab Forum



```yaml
# variables:
#   SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"  # Defines the location of the analysis task cache
#   GIT_DEPTH: "0"  # Tells git to fetch all the branches of the project, required by the analysis task
# sonarcloud-check:
#   stage: sonar
#   image:
#     name: sonarsource/sonar-scanner-cli:latest
#     entrypoint: [""]
#   cache:
#     key: "${CI_JOB_NAME}"
#     paths:
#       - .sonar/cache
#   script:
#     - sonar-scanner # -X -Dsonar.qualitygate.wait=true
#   allow_failure: true
#   only:
#     - merge_requests
#     - main

stages:          # List of stages for jobs, and their order of execution
  - build
  # - sonar

build-job:       # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - echo "Compile complete."
```



```yaml
# You can override the included template(s) by including variable overrides
# SAST customization: https://docs.gitlab.com/ee/user/application_security/sast/#customizing-the-sast-settings
# Secret Detection customization: https://docs.gitlab.com/ee/user/application_security/secret_detection/#customizing-settings
# Dependency Scanning customization: https://docs.gitlab.com/ee/user/application_security/dependency_scanning/#customizing-the-dependency-scanning-settings
# Note that environment variables can be set in several places
# See https://docs.gitlab.com/ee/ci/variables/#cicd-variable-precedence
stages:
- build
- test
build-job:
  stage: build
  script:
  - echo "Compiling the code..."
  - echo "Compile complete."
sast:
  stage: test
include:
- template: Security/SAST.gitlab-ci.yml

bandit-sast:
  rules:
    [allow_failure: false]
  artifacts:
    paths:
      - gl-sast-report.json
```