Sommaire

- Règles de nommage python
- Les tests
- Git
- REST
- Architecture N-tiers
- Objectifs projet

Règles de nommage python

- Pourquoi des règles de nommages ?
  - Facilite la maintenabilité.
  - Permet à plusieurs personnes de travailler sur le même code.

Règles de nommage python

Check si les conventions sont respectées

Nom classe en camel case

Noms méthodes séparés par underscore

Nom variables séparés par underscore

4 espaces d'indentation

Espaces de chaque côtés du signe égal

```python
import unittest
import json
from unittest.mock import patch

from app import app
from app.database.database import Database
from tests.utils import content_type
from tests.utils.mock_connection_factory import MockConnectionFactory


class TestUserController(unittest.TestCase):

    @patch.object(Database, 'get_connection', new=MockConnectionFactory.get)
    def test_create(self):
        Database.execute_schema()
        test_app = app.test_client()

        response = test_app.post('/users', data=json.dumps({
            "firstname": "test"
        }), content_type=content_type.JSON)
        assert response.status_code == 200
        user_id = response.data.decode('utf-8')

        response = test_app.get('/users/' + user_id)
        assert response.status_code == 200
        user = json.loads(response.data)
        assert user['firstname'] == 'test'

    @patch.object(Database, 'get_connection', new=MockConnectionFactory.get)
    def test_get_null(self):
        Database.execute_schema()
        test_app = app.test_client()

        response = test_app.get('/users/1')
        assert response.status_code == 404

```

Règles de nommage python

- Eviter les lignes trop longues (80 char max) :

```
30      TriggerClientMocker.init() \
31          .with_file(url='/contracts/sps/firstactivecontracts', file='data_05.json') \
32          .default_data(data={'results': []}) \
33          .start()
```

- Eviter les chaines de caractères qui pourraient être utilisées souvent => utiliser des constantes à la place :

```
66      return time.strftime(DateFormat.ISO)
```

Les tests

- Pourquoi faire des tests ?
  - Eviter les régressions.
  - Etre plus efficace lors du développement.
  - Plus grande confiance dans le code.

Les tests

- Les différents types de tests:
  - Tests unitaires : une classe, une méthode.
  - Tests fonctionnels : une fonctionnalité.
  - Tests d’intégrations : plusieurs composants entre eux (plusieurs serveurs, interfaces graphiques, bases de données…).

Les tests

- TDD (test driven development):
  - Tester d’abord, développer ensuite.

“Puisque les tests sont utiles, ils seront faits systématiquement avant chaque mise en œuvre”

- Deuxième principe de l'extreme programming

Les tests

- Les mocks:
  - Utilisés dans le cadre de tests unitaires ou fonctionnels.
  - Permet de limiter le test à une partie de l’application.
  - Utilisé notamment pour éviter d’appeler une base de donnée ou un service externe à l’application testée.

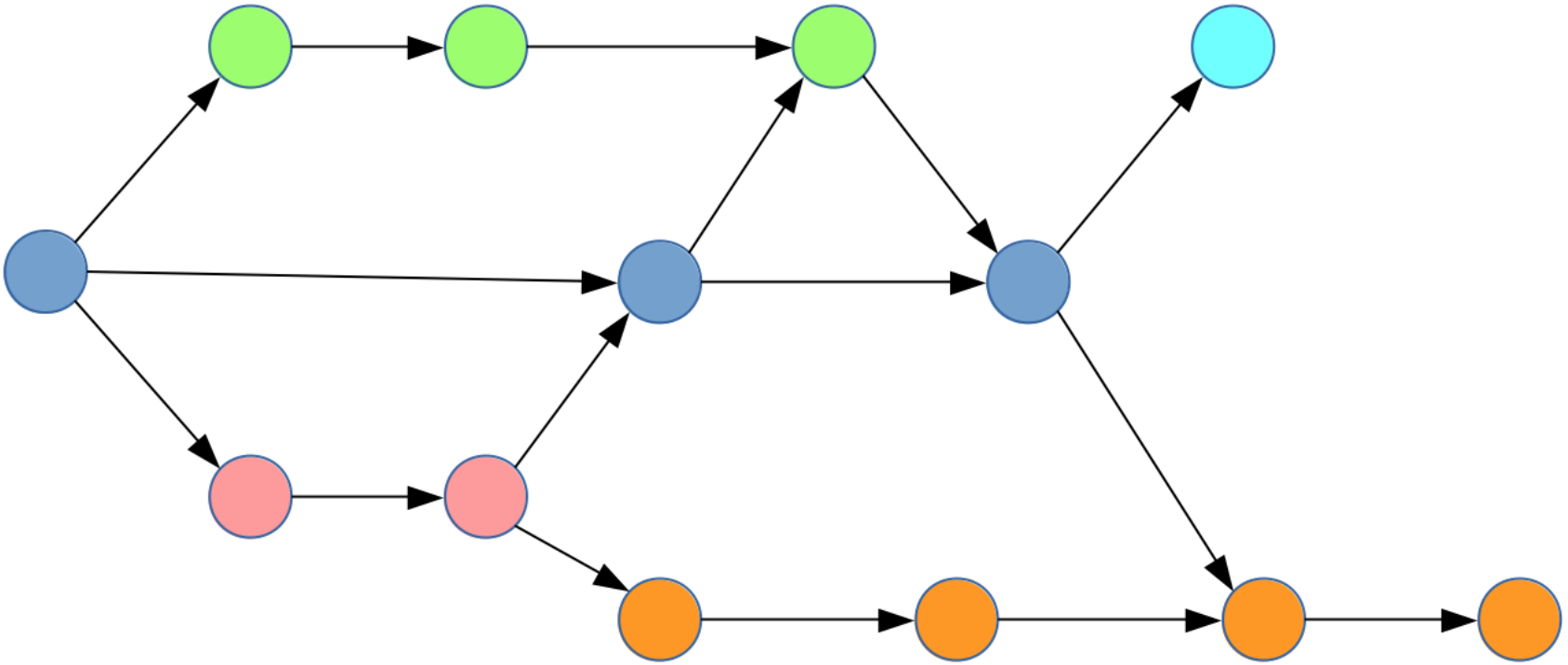
Git

- Commandes à retenir:
  - git checkout
  - git branch
  - git commit
  - git pull
  - git push
  - git merge
  - git clone

  “Puisque l'intégration des modifications est cruciale, nous l'effectuerons plusieurs fois par jour”

  Sixième principe de l'extreme programming

Git



REST

- Services webs basés sur les verbes HTTP et les erreurs HTTP.
- 4 verbes très couramment utilisés:
  - GET : pour la lecture.
  - POST : pour la création.
  - PUT : pour la modification.
  - DELETE : pour la suppression.

REST

- Codes réponse courants :
  - 200 : tout s’est bien passé.
  - 201 : création réussie.
  - 400 : erreur dans la requête du client.
  - 401 : le client n’est pas authentifié.
  - 403 : le client n’a pas les droits d’accès requis.
  - 404 : la resource demandée n’existe pas.
  - 500 : erreur interne du serveur.

Architecture N-tiers

- Architecture serveur organisée en couche applicative.
- Les couches sont généralement les souvantes :
  - Controllers : chargé de recevoir les requêtes et d'envoyer les réponses.
  - Services : contient le code métier de l'application.
  - DAO : chargé de lire et d'écrire dans la base de données.
- Les appels se font comme suit :
  - Controller → service → DAO

Objectifs projet

- Créer des webservices pour un blog
  - Messageries entre utilisateurs
  - Notifications
  - Articles
  - Commentaires

Objectifs projet