

Unit and Integration Testing



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

Table of Contents

1. Unit vs. Integration Testing
2. Best Practices
3. Structure
4. Testing Django components



sli.do

#python-web



Unit vs. Integration Testing

Unit tests vs Integration tests

- Unit tests
 - Isolated code i.e., specific function
 - Faster execution
 - More in number
 - Practical for pure functions
 - String transformations
 - Validators
- Integration tests
 - Entire application flow i.e., integration of functions
 - Slower execution
 - Less in number
 - Practical for user stories
 - User registration
 - Course signup after payment



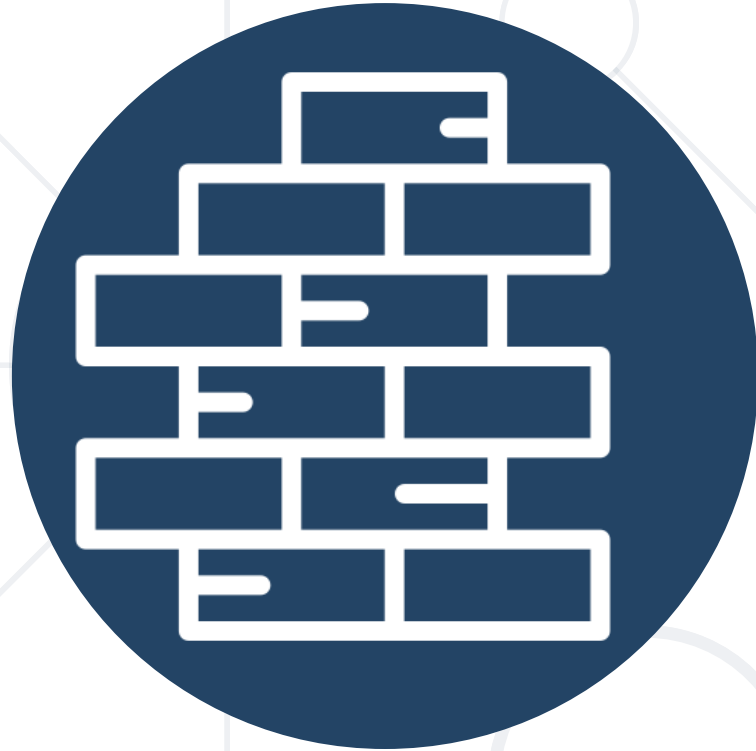


Best Practices

Best Practices

- If a piece of code can break, you **must test** it
- Each test should:
 - Cover only **one** function (unit) or flow (integration)
 - Assert only one case
- Keep it **simple**
- Arrange, Act, Assert





Structure

Structure

- There are several ways to write tests
 - Putting everything in the **tests.py** file and grouping tests by what they test (**models**, **views**, **forms**, etc.)
 - Or creating a folder test with the **different** test **files** for the different functionality that they test (**test_forms.py**, **test_models.py**, **test_views.py**)



Example

- All tests in the tests.py file
- All tests in a tests folder



Use tests.py file
for each app

```
> django_project
└── example_app
    ├── migrations
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── views.py
    └── templates
        └── manage.py
```

Create tests
folder and
separate
the code

```
> django_project
└── example_app
    ├── migrations
    └── tests
        ├── __init__.py
        ├── test_forms.py
        ├── test_models.py
        └── test_views.py
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── views.py
    └── templates
        └── manage.py
```



What should be tested?

From a software developer

What Should be Tested?

- Put simply, **all aspects** of the code
 - Models/Managers
 - Forms
 - Views
 - Other custom code
- **Except:**
 - Built-in code
 - `Model.objects.create()`, `Model.objects.all()`, etc..
 - Code from third-party libraries
 - i.e., testing something that comes from a library



Testing Models

- Testing model definitions is pointless
 - i.e., if a **CharField** is saved as **varchar** in the **database**
- In models the only thing that should be tested is the **validation**
 - Except built-in validators
 - i.e., providing invalid data should fail the test



```
class Profile(models.Model):  
    name = models.CharField(max_length=30)  
    age = models.IntegerField(validators=(  
        MinValueValidator(0),  
        MaxValueValidator(150)  
    ))  
    egn = models.CharField(max_length=10, validators=[egn_validator])
```

- Testing **name** and **age** is **pointless**
 - They use **built-in validators**
- Only **egn** should be **tested**

Models are tested as follows

```
def
test_profileCreate_whenInvalidEgn_shouldRaise(self):
    p = Profile(
        name='Doncho Minkov',
        age=19,
        egn='89062a1234'
    )

    try:
        p.full_clean()
        p.save()
        self.fail()
    except ValidationError as ex:
        self.assertIsNotNone(ex)
```

```
def
test_profileCreate_whenValidEgn_shouldCreateIt(self):
    p = Profile(
        name='Doncho Minkov',
        age=19,
        egn='8906221234'
    )

    p.full_clean()
    p.save()

    self.assertIsNotNone(p)
```

Testing Forms

- Testing forms is pretty much the same as **testing models**
 - Only test the **custom** (your) **logic**



```
def
test_profileFormSave_whenValid_xxxx
(self):
    data = {
        'name': 'Doncho',
        'age': 19,
        'egn': '8906221234',
    }
    form = ProfileForm(data)


    self.assertTrue(form.is_valid())
```

```
def
test_profileFormSave_whenInvalid_xxxx
(self):
    data = {
        'name': 'Doncho',
        'age': 19,
        'egn': '89062a1234',
    }
    form = ProfileForm(data)

    self.assertFalse(form.is_valid())
```


Views Testing

- Views are tested using a **test client**
 - "Sends" **requests to your views** by URL
 - Asserts templates, context, redirects, status code
 - Logins user and **persist session**
 - This is for the next course



```
class ProfileViewTests(TestCase):  
    def setUp(self) :  
        self.test_client = Client()
```

Testing Views GET Examples

- Render template

```
def test_getProfilesIndex_shouldRenderTemplate(self):  
    response = self.test_client.get('')  
    self.assertTemplateUsed(response, 'testing/index.html')
```


- Context data

```
def test_getProfilesIndex_shouldReturnCorrectContext(self):  
    response = self.test_client.get('')  
    profiles = response.context['profiles']  
    # regular asserts
```



Testing Views POST Examples

- Render template



```
def test_profilesIndex_whenValidData_shouldCreateAndRedirectToIndex(self):
    data = {
        'name': 'Doncho',
        'age': 19,
        'egn': '8906231234',
    }

    response = self.test_client.post('/', data)
    self.assertRedirects(response, '/')
```

Given the validator:

```
def egn_validator(value: str):  
    result = all(d.isdigit() for d in value)  
    if not result:  
        raise ValidationError('Egn should contain only digits')
```

The tests:

```
def test_egnValidator_whenAllIsDigit_shouldDoNothing(self):  
    egn_validator('1234567890')  
    self.assertTrue(True)  
  
def test_egnValidator_whenOneNonDigit_shouldRaise(self):  
    with self.assertRaises(ValidationError) as context:  
        egn_validator('12345678s0')  
    self.assertIsNotNone(context.exception)
```

Final Notes

- All model, form, and view tests are **integration** tests
 - They **depend on Django** itself, so making them **unit tests** is **impossible**
- Validation tests can be unit tests
 - If they **do not have** external **dependencies**
 - Or if **mocking** their **external dependencies**
- The **bigger the flow** of an integration test, the more unit tests and smaller integration tests become redundant
 - i.e., view tests cover forms and models as well





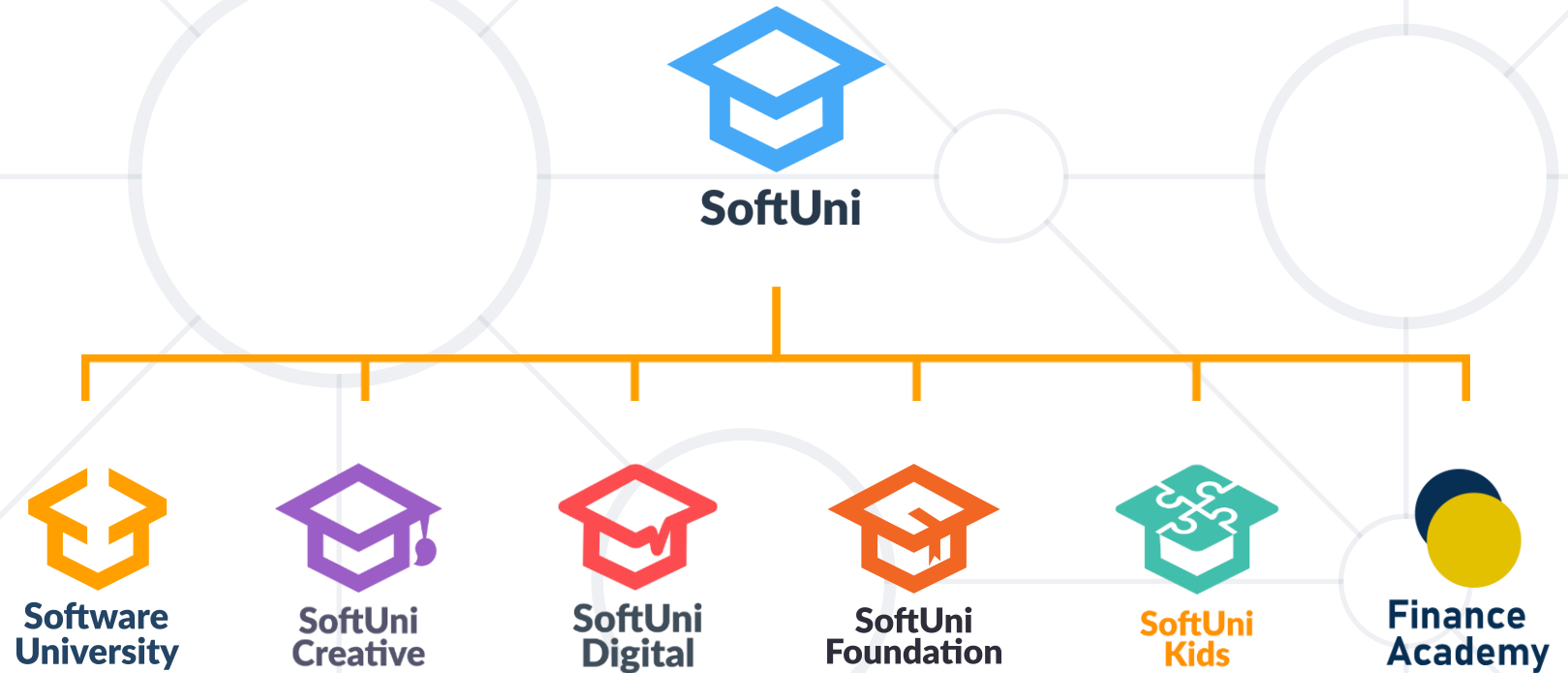
Live Demo

Creating Unit Tests

- **Unit Testing** - isolated tests that test **one** specific function
- **Integration Testing** - larger tests that focus on user behavior and testing **entire applications**



Questions?



SoftUni Diamond Partners

**SUPER
HOSTING
.BG**



**Coca-Cola HBC
Bulgaria**



POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers



AMBITIONED

 **DRAFT
KINGS**



**SOFTWARE
GROUP**

createX



Postbank

Решения за твоето утре

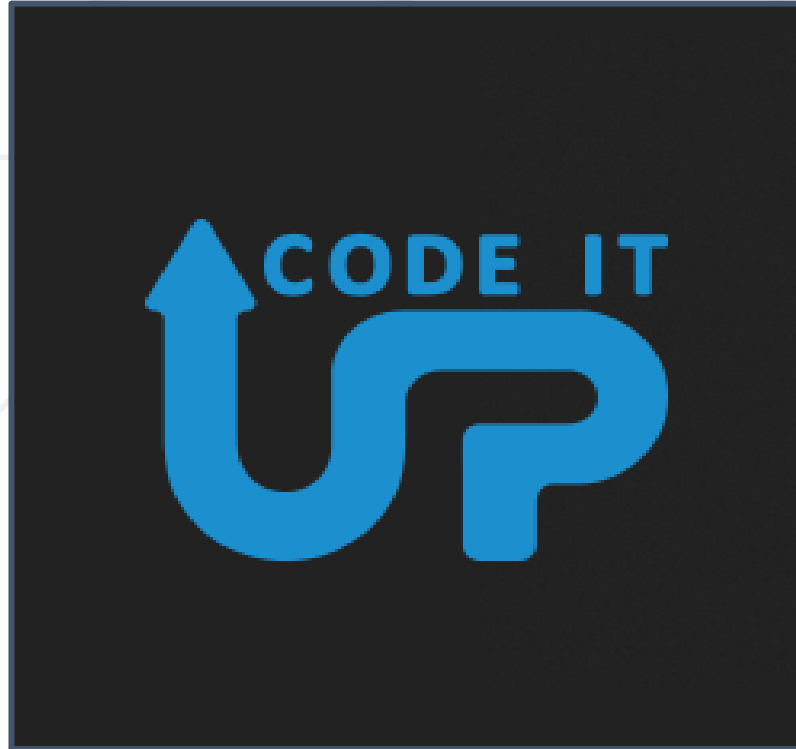


BOSCH

DXC
TECHNOLOGY



SmartIT



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, softuni.org

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

