Testing Concepts

Jamie Counsell
SOFTWARE DEVELOPER

@jamiecounsell www.jamiecounsell.me

Overview

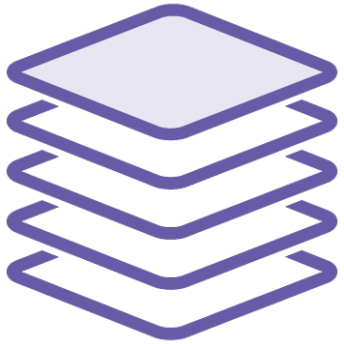
Testing considerations

- Why and when do we need tests?
- What needs to be tested?

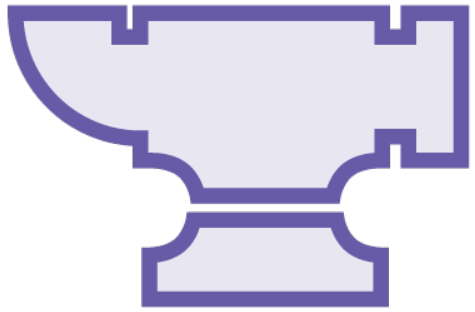
Types of tests

- Unit tests
- Integration tests

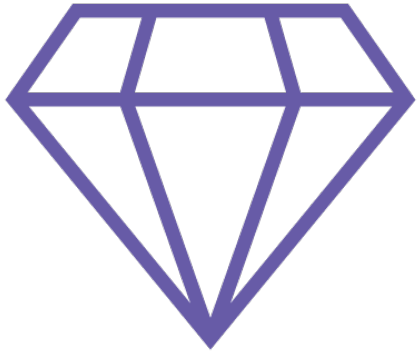
Testing Ensures



Consistency
Validate that code behaves the same way over time

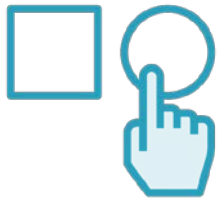


Robustness
If we don't catch edge cases early, users eventually will

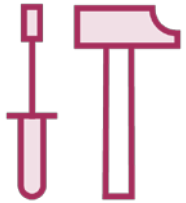


Quality
Prevent new issues caused when code changes

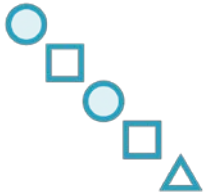
Types of Testing



UI testing



Functional testing



Regression testing

Load testing

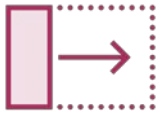
Security testing

Compatibility testing

What We Need to Test

Custom views and extended view logic

Models and their methods



Custom helpers, middleware, etc.

Results of template rendering

What We Shouldn't Test

Built in Django views

Files such as urls.py or settings.py

The Django admin panel

Classes based mostly off highly generic built-ins

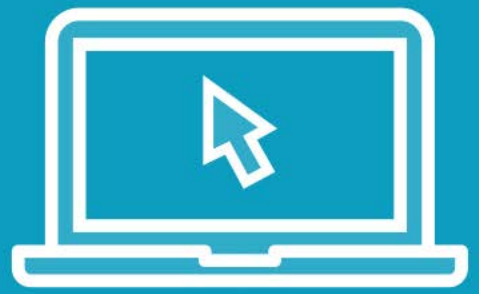
```
class Legal(TemplateView):
    template_name = "legal.html"


def is_valid(item):
    return item and len(item)


def open_post(pk):
    Post.objects.filter(
                pk = pk
            ).update(
                open = True)
    return Response()
```

- Built in views have very little to test
- Custom methods are easy to unit test
- Small view with database operation is a good fit for an integration test

Demo



View our application

- Create a user
- Preview the features of our app
- What features do we need to test?
- What does Django handle for us?

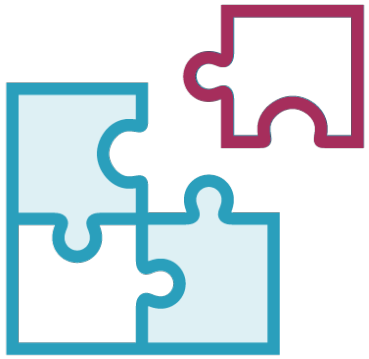
Our Focus

Unit Tests

Small tests done in isolation to validate a single function, method, or class

Integration Tests

Larger tests that validate many system components working together

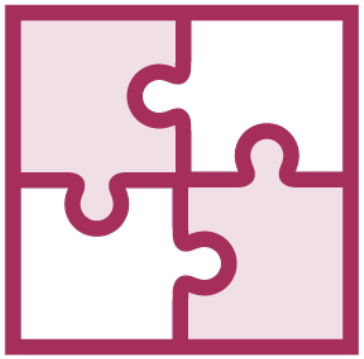


Unit test candidates

- Methods with easily defined roles
- Handlers, helpers, model methods

Common pitfalls

- Over-testing
- Failure to test real-world cases
- Tests that solely increase line coverage



Integration test candidates

- Views, actions, models
- Larger methods with transactions

Common pitfalls

- Failure to test edge cases
- Tests too large
- Unreliable infrastructure leads to unreliable tests

Our Metrics

Efficiency

Writing short, concise tests are more clear and save time

Responsiveness

Designing tests to respond to change - tests that never fail don't help us

Code Coverage

Concerned with covering business cases, not simply lines of code

How Django Differs

Django is different

- Core principles encourage different testing approaches
- Pragmatic design favors integration tests
- Code coverage not seen as most important metric

Frameworks have unique requirements

- Embrace Django's core principles
- Remember the purpose of testing

Summary

Testing is most useful when we

- Know our requirements
- Use the right tests
- Focus on critical elements first
- Improve over time
- Use meaningful metrics