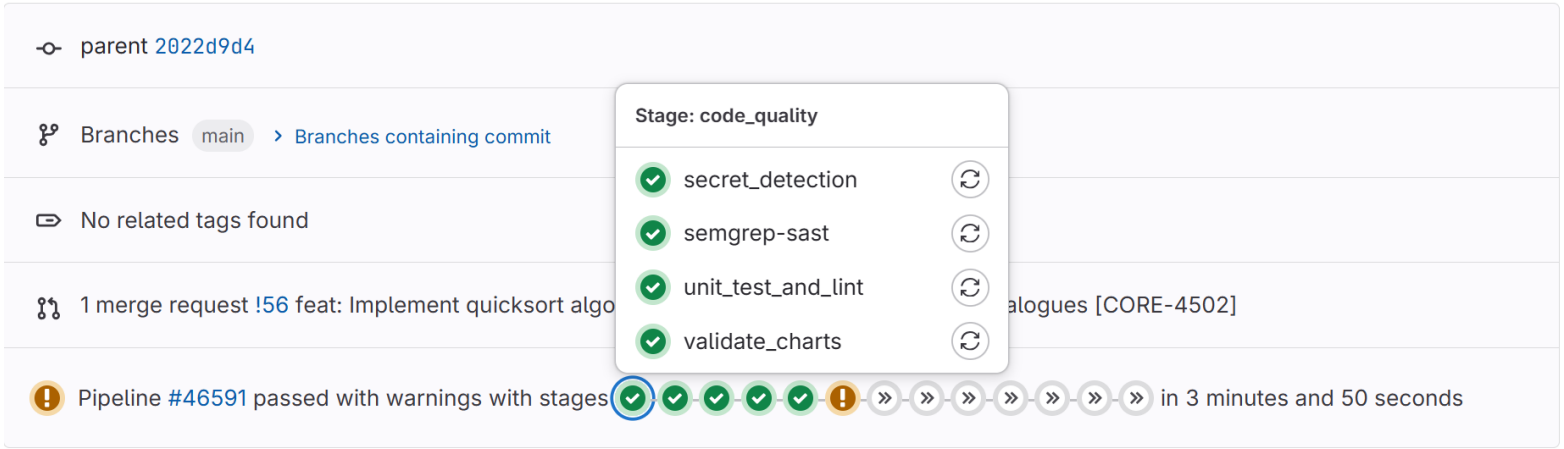
Testing, tests and tests that test the tests

Lizzie Salmon

What is testing

- In general: *“A way to ensure that changes to your code works as expected”*
- **Manual testing** - in person, clicking through the application or interacting with APIs etc
- **Automated testing** – executing a test script that was written in advance. A key component of CI/CD



Different types of automated tests

- **Unit tests** – very low level and close to source. Test individual methods and functions of classes, components and modules.
- **Integration tests** - verify that different modules or services used by your application work well together.
- **Functional tests** - focus on the business requirements of an application. They only verify the output do not check the intermediate states.
- **End to end tests** - replicates a user behaviour with the software in a complete application.

…

Yeah, but why?

- Identify **bugs** in the earliest stages of development
- Ensure existing features still work as expected – protect against **regression**
- Practical form of **documentation** – shows how individual units should behave
- Improved code **design**– Developers must think harder about how their code will work, inputs outputs etc
- Increased **confidence** for developers that their code is working as intended

Unit testing

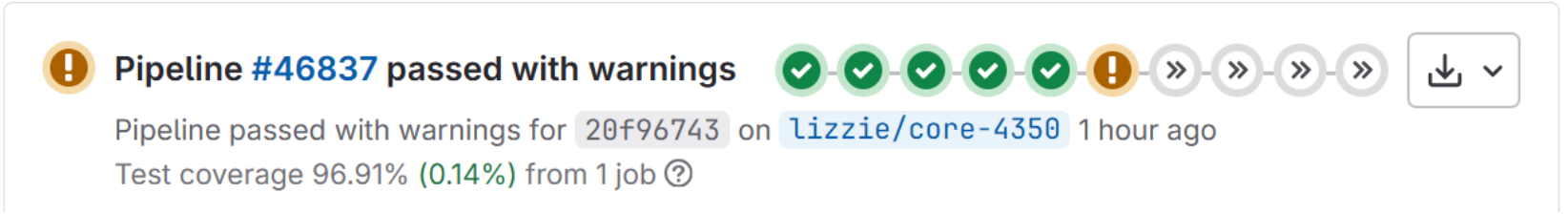
src > basics.ts > ...

```ts
export function isEven(n: number) {
    return n % 2 === 0;
}

```

```ts
it("correctly determines that 2 is even", () => {
    expect(isEven(2)).toBe(true);
  });

it("correctly determines that 3 is odd", () => {
    expect(isEven(3)).toBe(false);
  });

it("correctly determines that 0 is even", () => {
    expect(isEven(0)).toBe(true);
  });

it("correctly determines that -4 is even", () => {
    expect(isEven(-4)).toBe(true);
  });
```

```
✓ test/basics.test.ts (4 tests) 2ms
  ✓ isEven › correctly determines that 2 is even
  ✓ isEven › correctly determines that 3 is odd
  ✓ isEven › correctly determines that 0 is even
  ✓ isEven › correctly determines that -4 is even

 Test Files  1 passed (1)
      Tests  4 passed (4)
   Start at  16:03:07
   Duration  124ms

 PASS  Waiting for file changes...
```

Code coverage



- Metric that can help you understand how much of your source is tested.
  - **Function coverage:** how many of the functions defined have been called.
  - **Statement coverage:** how many of the statements in the program have been executed.
  - **Branches coverage:** how many of the branches of the control structures (if statements for instance) have been executed.
  - **Line coverage:** how many of lines of source code have been tested.

Coverage Reports

- Example coverage report

basics.ts		100%	3/3	100%	1/1	100%	1/1	100%	3/3

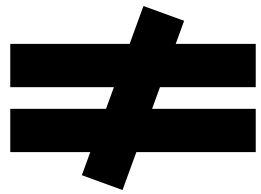
All files

97.09% Statements 1673/1723 **95.21%** Branches 1651/1734 **94.45%** Functions 545/577 **97.6%** Lines 1631/1671

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
components	95.95%	641/668	95.63%	832/870	95.34%	328/344	96.03%	629/655
layouts	84.21%	16/19	90%	18/20	75%	6/8	88.88%	16/18
pages	100%	12/12	75%	3/4	100%	4/4	100%	12/12
pages/account	96.15%	25/26	100%	10/10	90%	9/10	96%	24/25
pages/data	100%	49/49	84.94%	79/93	100%	16/16	100%	45/45
pages/groups	100%	17/17	93.33%	14/15	100%	4/4	100%	17/17
pages/models	100%	52/52	93.97%	78/83	100%	16/16	100%	48/48
pages/workflows	100%	38/38	93.1%	54/58	100%	11/11	100%	38/38
plugins	69.23%	9/13	100%	0/0	42.85%	3/7	69.23%	9/13
stores	98.35%	478/486	98.46%	320/325	89.41%	76/85	100%	468/468
ts	97.95%	336/343	94.92%	243/256	100%	72/72	97.89%	325/332

High coverage  good tests

Bad tests

src > basics.ts > ...

```ts
export function isEven(n: number) {
    return n % 2 === 0;
}

```

```ts
it("runs without crashing", () => {
    isEven(2);
  });

it("returns a boolean", () => {
    expect(typeof isEven(3)).toBe("boolean");
  });
```

```
✓ test/basics.test.ts (2 tests) 1ms
  ✓ isEven > runs without crashing
  ✓ isEven > returns a boolean

 Test Files  1 passed (1)
      Tests  2 passed (2)
   Start at  16:15:07
   Duration  133ms

 PASS  Waiting for file changes...
```

File	Statements		Branches		Functions		Lines	
basics.ts	100%	3/3	100%	1/1	100%	1/1	100%	3/3

Mutation testing

Mutation testing introduces changes to your code, mutants, then runs your unit tests against these mutants.

It is expected that your unit tests will now fail – and these mutants are “killed” (this is a good thing).

If they don't fail – the mutant “survives”, it might indicate your tests do not sufficiently cover the code (this is a bad thing).

How does mutation testing work?

```
export function isEven(n: number) {
  return n % 2 === 0;
}
```

```
export function isEven(n: number) {
  return true;
}
```

```
export function isEven(n: number) {
  return n % 2 !== 0;
}
```

```
export function isEven(n: number) {}
```

```
export function isEven(n: number) {
  return false;
}
```

```
export function isEven(n: number) {
  return n * 2 === 0;
}
```

How does mutation testing work? Good tests

```
it("correctly determines that 2 is even", () => {
    expect(isEven(2)).toBe(true);
  });

it("correctly determines that 3 is odd", () => {
    expect(isEven(3)).toBe(false);
  });

it("correctly determines that 0 is even", () => {
    expect(isEven(0)).toBe(true);
  });

it("correctly determines that -4 is even", () => {
    expect(isEven(-4)).toBe(true);
  });
```

```
export function isEven(n: number) {
   return n * 2 === 0;
 }
```

This mutant is killed

How does mutation testing work? Bad tests

```
export function isEven(n: number) {
  return n * 2 === 0;
}
```

```
it("runs without crashing", () => {
    isEven(2);
  });

it("returns a boolean", () => {
    expect(typeof isEven(3)).toBe("boolean");
  });
```

This mutant survives

My live demos worked

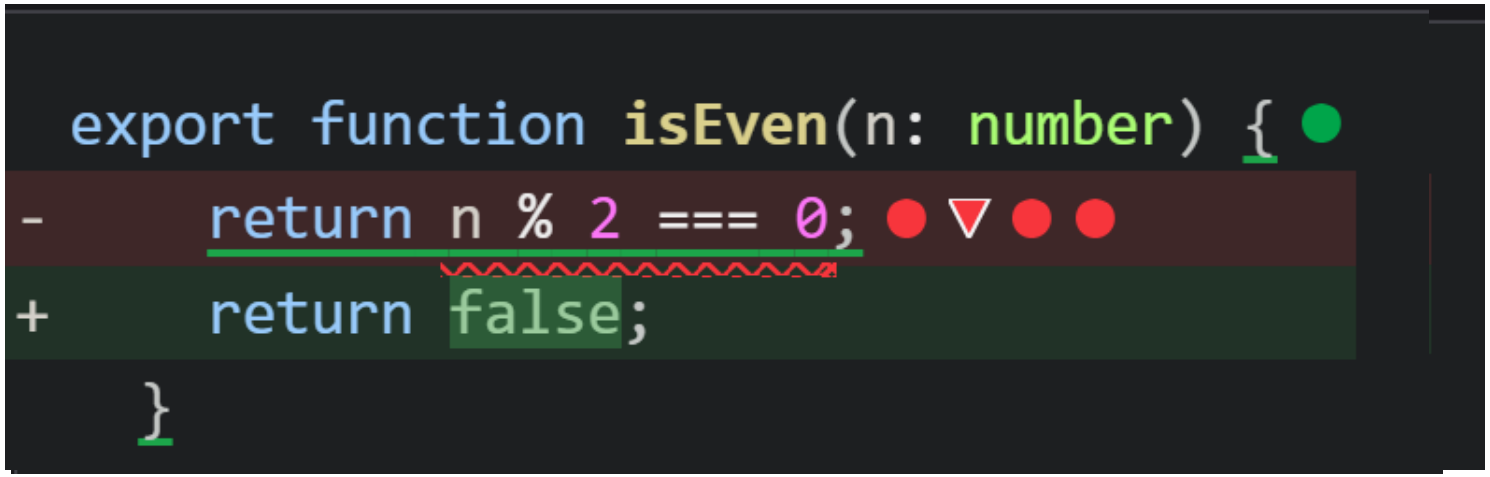
basics.ts

Mutants | Tests

All files / basics.ts

1 | 4

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
TS basics.ts	20.00	20.00	1	4	0	0	0	0	0	1	4	5



basics.ts

Mutants | Tests

All files / basics.ts

5

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
TS basics.ts	100.00	100.00	5	0	0	0	0	0	0	**5**	**0**	**5**

Killed (5)

```
1  export function isEven(n: number) {
2      return n % 2 === 0;
3    }
```

Mutation testing on web-app-v2

- Ran mutation testing on just the stores folder

- **would need to work on getting it to mutate the .vue files**

All files stores

98.35% Statements 478/486 **98.46%** Branches 320/325 **89.41%** Functions 76/85 **100%** Lines 468/468

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
accountUpgrade.ts	100%	20/20	100%	9/9	100%	2/2	100%	20/20
dataAddData.ts	100%	11/11	75%	3/4	100%	3/3	100%	11/11
dataCatalogue.ts	100%	38/38	100%	16/16	100%	7/7	100%	37/37
dataCatalogueFilters.ts	100%	22/22	100%	9/9	100%	7/7	100%	22/22
dataDetails.ts	100%	22/22	100%	20/20	100%	3/3	100%	22/22
featureFlags.ts	100%	21/21	100%	12/12	100%	2/2	100%	21/21
groupCatalogue.ts	100%	28/28	100%	20/20	100%	5/5	100%	26/26
keycloak.ts	93.33%	14/15	100%	0/0	75%	3/4	100%	12/12
modelAdd.ts	100%	7/7	100%	2/2	66.66%	2/3	100%	7/7
modelCatalogue.ts	95.45%	84/88	97.46%	77/79	73.33%	11/15	100%	82/82
modelCatalogueFilters.ts	100%	15/15	100%	2/2	100%	5/5	100%	14/14
modelDetails.ts	100%	53/53	100%	44/44	100%	4/4	100%	53/53
notificationPreferences.ts	100%	27/27	100%	20/20	100%	3/3	100%	27/27
releaseNotes.ts	100%	10/10	100%	10/10	100%	2/2	100%	10/10
workflowCatalogue.ts	96%	72/75	96.42%	54/56	75%	9/12	100%	71/71
workflowCatalogueFilters.ts	100%	13/13	100%	2/2	100%	5/5	100%	12/12
workflowDetails.ts	100%	21/21	100%	20/20	100%	3/3	100%	21/21

Takeaways

- Unit testing ensures code correctness
  - Verify that individual components run as expected
- Code coverage tells you what is tested not how well it is tested
  - Not a complete picture of code quality
- Mutation testing tests the effectiveness of tests
  - Ensures that tests are meaningful, goes beyond coverage