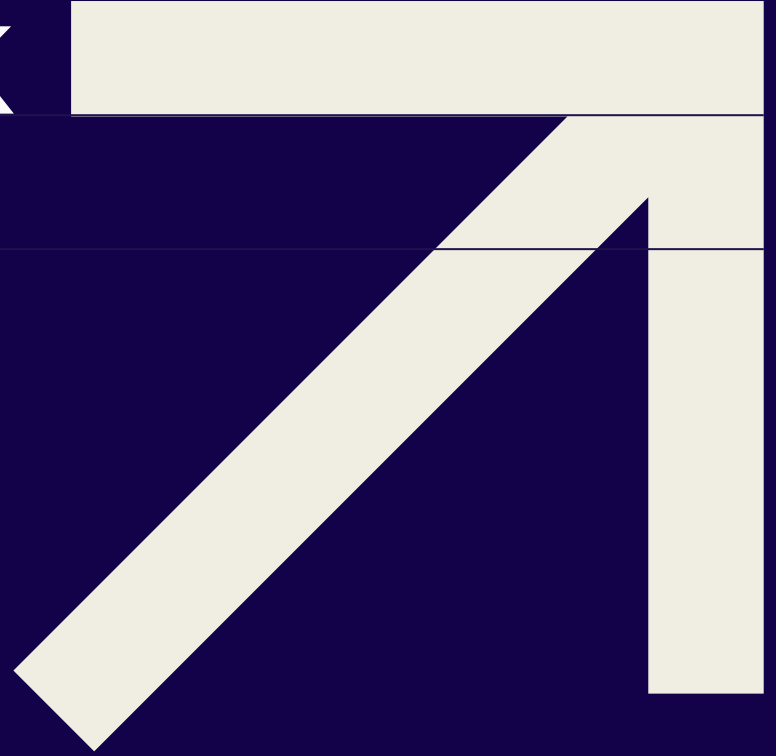
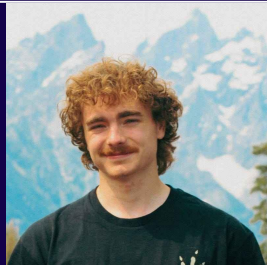


Lightning Talk Testing



Team sdmay26-21

Meet the team



Griffin Urban
Frontend Developer



Aden Koziol
AI Model Programming



Carson Irving
Backend Developer



Marina Mannan
Embedded Systems
Engineer



Daniel Pender
Hardware Specialist



Ryan Holden
Full Stack Developer



Max Purvis
Hardware Specialist

Project overview ↘



A Smart Aquatic Ecosystem

AI-Enabled Fish Health, Water Quality Monitoring,
and Automated Control

Team Testing Philosophy

What are our philosophies towards testing?

- Test early and often
- Create the tests as we develop (Maintainability)
- Automate when possible
- Peer verification

Types of Testing

- **Unit Testing**

- Testing individual components to make sure that they work correctly
- E.g., each sensor needs to be calibrated to ensure accurate and stable readings

- **Integration Testing**

- Ensures subsystems work together as one system
- Sensors → arduino → backend → frontend dashboard (Running everything together)

- **System Testing (later) & Security Testing (later)**

- Test the entire system functioning as one unit
- Verifies end-to-end behavior

- **Acceptance Testing (end)**

- Final confirmation the system meets user needs

Hardware Testing

- Sensor Calibration
- Extended test sessions to check for consistency and stability in readings
- Feeder testing



Backend Testing

XUnit was chosen as the testing library

- Easy to set up
- Open source
- Parallel Test Execution (Faster Pipeline)
- Microsoft Recommended

Able to easily create simple unit tests to ensure models behave predictably

Able to use an in-memory database to create integration tests

Pipeline automatically runs the tests upon a push to the main branch of the gitlab repository

```
2 references
public static Random rand = new Random();
10 references
int univID = rand.Next();
8 references
double univValue = rand.NextDouble();

[Fact]
0 references
public void CanCreateOxygen()
{
    var oxygen = new Oxygen
    {
        id = univID,
        value = univValue
    };

    Assert.Equal(univID, oxygen.id);
    Assert.Equal(univValue, oxygen.value);
}
```

```
[Fact]
0 references
public async Task GetLastHours()
{
    var dbContext = GetInMemoryDbContext();
    dbContext.oxygens.AddRange(
        new Oxygen { value = 6.5, timestamp = DateTime.UtcNow.AddHours(-5) }, //shouldn't show
        new Oxygen { value = 7.2, timestamp = DateTime.UtcNow.AddHours(-1) }, //will show
        new Oxygen { value = 8.0, timestamp = DateTime.UtcNow } //will show
    );

    await dbContext.SaveChangesAsync();

    var controller = new oxygenController(dbContext);
    var result = await controller.getLastHours(2); //Last 2 hours

    var okResult = result.Should().BeOfType<OkObjectResult>().Subject;
    var returnedList = okResult.Value.Should().BeAssignableTo<IEnumerable<Oxygen>>().Subject.ToList();
    returnedList.Count().Should().Be(2);
    returnedList.Should().OnlyContain(o => o.timestamp >= DateTime.UtcNow.AddHours(-2));
}
```

Results of Backend Testing

Using coverlet collector package to get tests code coverage

Using an open source report generator tool to make a more readable report and print during pipeline run

```
test_backend:
  stage: test
  tags:
  - backend
  script:
    - cd API/fishTankAPI.Tests
    - dotnet tool restore
    - echo "Running backend unit tests..."
    - dotnet test --collect:"XPlat Code Coverage" --results-directory TestResults
    - dotnet reportgenerator \
      "-reports:TestResults/*/coverage.cobertura.xml" \
      "-targetdir:TestResults/CoverageReport" \
      "-reporttypes:TextSummary" \
      "-filefilters:*/Migrations/*;*/Program.cs;*/Data/*;*/WebSockets/*;*/notification*"
    - cat TestResults/CoverageReport/Summary.txt
```

```
45 Generated on: 11/9/2025 - 8:50:21 PM
46 Coverage date: 11/9/2025 - 8:50:20 PM
47 Parser: Cobertura
48 Assemblies: 1
49 Classes: 13
50 Files: 13
51 Line coverage: 99.4%
52 Covered lines: 182
53 Uncovered lines: 1
54 Coverable lines: 183
55 Total lines: 396
56 Branch coverage: 75% (3 of 4)
57 Covered branches: 3
58 Total branches: 4
59 Method coverage: 100% (47 of 47)
60 Full method coverage: 97.8% (46 of 47)
61 Covered methods: 47
62 Fully covered methods: 46
63 Total methods: 47
64 fishTankAPI 99.4%
65 FISHTANKAPI.Controllers.imageController 96.6%
66 FISHTANKAPI.Controllers.oxygenController 100%
67 FISHTANKAPI.Controllers.phController 100%
68 FISHTANKAPI.Controllers.postController 100%
69 FISHTANKAPI.Controllers.TemperatureController 100%
70 FISHTANKAPI.Controllers.TurbController 100%
71 FISHTANKAPI.DTO.postDTO 100%
72 FISHTANKAPI.models.emailSettings 100%
73 FISHTANKAPI.models.Image 100%
74 FISHTANKAPI.models.Oxygen 100%
75 FISHTANKAPI.models.PH 100%
76 FISHTANKAPI.models.Temperature 100%
77 FISHTANKAPI.models.Turb 100%
✓ 78 Cleaning up project directory and file based variables
79 Job succeeded
```

Frontend Unit Testing

Vitest and React Testing Library for testing libraries:

- React Testing Library
 - Great for react components, ensuring content is **where** it needs to be.
- Vitest Library
 - Easy and lightweight solution for testing TSX/Vite, ensures data is **what** it needs to be.
 - Also great for mock fetch requests with testing data, and testing utility functions.



Dashboard for frontend

```
// TEST 7 fetchJson
it('fetchJson returns null for non-ok fetch response', async () => {
  const fake = { ok: false, status: 500, url: 'http://test', text: async () => {
    vi.stubGlobal('fetch', vi.fn(async () => fake as Response));
    const out = await fetchJson('http://test');
    expect(out).toBeNull();
  }};
});
```

Example vitest fetch test

Frontend Unit Testing - Results

Coverage reports generated using Vitest coverage library

- Shows percentage of "coverage" of code. Currently at 84% coverage
- Pipeline in GitLab configured to run all tests before deploying frontend

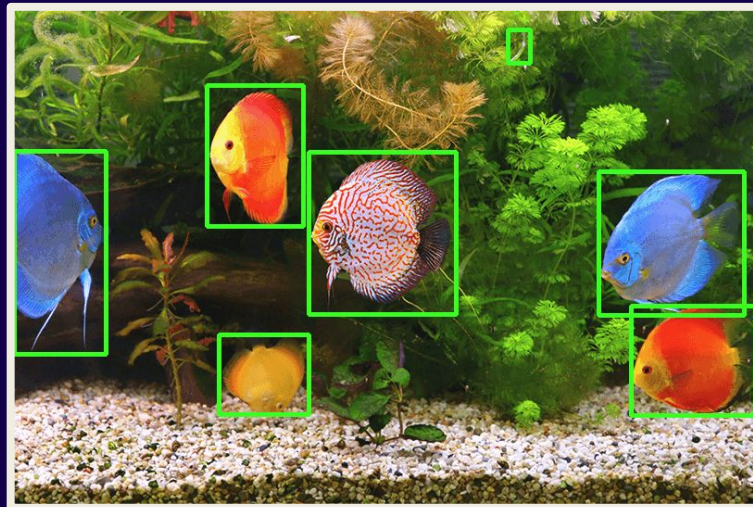
Notable results: Very helpful for discovering exactly where testing should be implemented.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
-----	-----	-----	-----	-----	-----
All files	77.22	49.43	65.21	84.44	
src	80	55.35	87.5	86.66	
App.css	0	0	0	0	
home.tsx	80	55.35	87.5	86.66	39-40,42,44,70-71
src/components	66.66	23.8	60	64.28	
LineChart.tsx	66.66	23.8	60	64.28	66-86
src/hooks	62.5	0	28.57	90.9	
websocket.tsx	62.5	0	28.57	90.9	12
src/utils	90	80	100	90	
fetch.ts	89.47	75	100	89.47	4,17
testUtils.tsx	100	100	100	100	
-----	-----	-----	-----	-----	-----
PASS Waiting for file changes...					

AI Detection Testing

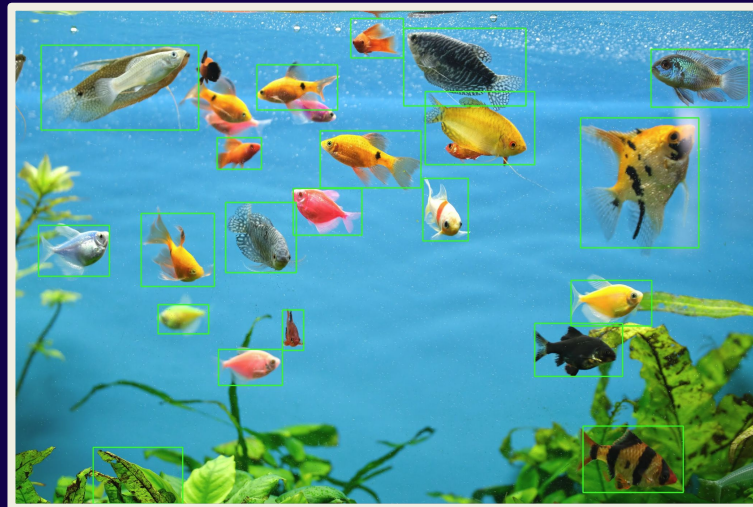
YOLOv8n

- Total Fish: 109
- Total Fish Detections: 88
- Total False Detections: 8
- Accuracy = .812 (81.2%)



Other Models

- YOLOV5n = .745 (74.5%)
- SSD-MobileNet v1 = .683 (68.3%)
- Tiny-YOLOv3 = .627 (62.7%)



Questions?

Any other questions?
Please reach out to
sdmay26-21@iastate.edu