**Test -Driven Development**

**Project Title: Project X**

**Prepared by: Team BAMP**

**Faculty of Computing, Engineering and Technology**

Counter Component Development Using TDD

1. Red Phase (Writing Tests First)

First, we wrote tests for our Counter component before any implementation:

```typescript

// Counter.test.tsx

describe('Counter Component TDD', () => {

// Test 1: Initial render

it('should render initial count of 0', () => {

const { getByTestId } = render(<Counter />);

const countText = getByTestId('count-display');

expect(countText.props.children).toBe(0);

});

// Additional test cases for increment, decrement, and constraints

...

});

```

Running these tests initially failed because:

1. The Counter component didn't exist

2. No implementation was available

3. Required test IDs were missing

2. Green Phase (Making Tests Pass)

We then created the Counter component with the minimum code needed to make tests pass:

```typescript

// Counter.tsx

export const Counter: React.FC = () => {

const [count, setCount] = useState(0);

const increment = () => {

setCount(prev => prev + 1);

};

const decrement = () => {

setCount(prev => prev > 0 ? prev - 1 : 0);

};

return (

<View style={styles.container}>

<TouchableOpacity testID="decrement-button" onPress={decrement}>

<Text>-</Text>

</TouchableOpacity>

<Text testID="count-display">{count}</Text>

<TouchableOpacity testID="increment-button" onPress={increment}>

<Text>+</Text>

</TouchableOpacity>

</View>

);

};

```

3. Refactor Phase

After getting the tests to pass, we improved the code by:

1. Adding proper styling

2. Improving component structure

3. Ensuring good user experience

4. Maintaining test coverage

TDD Benefits Demonstrated

1. Clear Requirements: Tests defined exactly what we needed

2. No Over-engineering: We only wrote code needed to pass tests

3. Confidence: All features are tested

Test Cases Explained

1. Initial State Test

- Ensures counter starts at 0

- Verifies basic rendering

2. Increment Test

- Verifies counter can increase

- Tests button functionality

3. Decrement Test

- Verifies counter can decrease

- Tests button functionality

4. Constraint Test

- Ensures counter cannot go below 0

- Tests business logic constraint

Running the Tests

```bash

npm test Counter.test.tsx

```

Key TDD Principles Followed

1. Write tests first

2. See tests fail (Red)

3. Write minimum code to pass (Green)

4. Improve code while keeping tests passing (Refactor)

Evidence



