

# Unit Testing

## Unit Test on Add Investment Controller Form:

validateInvestment Function		
\src\test\formCtrl.test.js		
11ms	10	✓ 10
✓	should return success when all inputs are valid	2ms
✓	should fail if stockId is missing	1ms
✓	should fail if quantity is missing	1ms
✓	should fail if purchasePrice is missing	1ms
✓	should fail if transactionDate is missing	1ms
✓	should fail if requestedStock is not in our database	1ms
✓	should fail if quantity is non-positive	0ms
✓	should fail if purchasePrice is non-positive	1ms
✓	should fail if transactionDate is invalid	1ms
✓	should fail if transactionDate is in the future	2ms

```
export function validateInvestment(input) {
  const { stockId, quantity, purchasePrice, transactionDate } = input;

  const stockData = JSON.parse(fs.readFileSync('./public/allstocks.json', 'utf8'));
  // Check if all fields are provided
  if (
    stockId === undefined ||
    quantity === undefined ||
    purchasePrice === undefined ||
    transactionDate === undefined
  ) {
    return { success: false, error: "All fields (stockId, quantity, purchasePrice, transactionDate) are required." };
  }

  const stockExists = stockData.companies.some(company => company.symbol === stockId);
  if (!stockExists) {
```

## Summary of Unit Tests for validateInvestment

The unit tests for the `validateInvestment` function validate various input scenarios to ensure the function operates correctly and adheres to expected business rules. The test cases cover the following scenarios:

### 1. Valid Inputs:

- Ensures the function returns success when all required fields are present, correctly formatted, and meet the validation criteria.

## 2. Missing Fields:

- Tests cases where one or more required fields (stockId, quantity, purchasePrice, transactionDate) are omitted. These tests verify that the function correctly identifies the missing data and returns an appropriate error message.

## 3. Stock Existence:

- Validates that the function checks for the existence of the stock in the provided database. If the **stock ID** is invalid or not found, it ensures the appropriate error is returned.

## 4. Quantity Validation:

- Ensures the **quantity** field is a positive number. If the value is zero, negative, or invalid, the function should return an error.

## 5. Purchase Price Validation:

- Verifies that the **purchasePrice** is a positive number. Negative or zero values should trigger a failure response.

## 6. Transaction Date Validation:

- Confirms the function handles transaction **date** validations, including:
  - Rejecting invalid date formats.
  - Ensuring the date is not in the future.

Each test asserts the expected output against the actual result from the 'validateInvestment' function, ensuring correctness and reliability under diverse input conditions.