

# Front-End Test Cases for Your Financial News

---

## 1. Navigation Menu

**Test Case ID:** FE-001

**Description:** Verify the navigation menu links work as intended.

**Steps:**

- Open the webpage.
- Click on each link in the navigation menu (Home, Markets, Economy, etc.).

**Expected Result:** Each link navigates to the correct section/page.

---

## 2. Hero Section Animation

**Test Case ID:** FE-002

**Description:** Validate the animations in the hero section.

**Steps:**

- Load the webpage.
- Observe the hero title and background animation.

**Expected Result:** Animations should play smoothly without delays or abrupt stops.

---

## 3. Search Functionality

**Test Case ID:** FE-003

**Description:** Check if the search button fetches data or navigates correctly.

**Steps:**

- Enter a valid search term in the input box.
- Click the Search button.

**Expected Result:** The page redirects to `results.html`, and the query term is stored in `localStorage`.

---

## 4. Placeholder Data Display

**Test Case ID:** FE-004

**Description:** Ensure placeholder data displays correctly when the API call fails.

**Steps:**

- Simulate an API failure.

- Observe the displayed results.

**Expected Result:** Mock data from `mockData.js` should display with correct titles, summaries, and links.

---

## 5. Responsive Design

**Test Case ID:** FE-005

**Description:** Validate the responsiveness of the page on various devices.

**Steps:**

- Open the webpage on different screen sizes (mobile, tablet, desktop).
- Resize the browser window.

**Expected Result:** Layout adapts correctly without overlapping or breaking content.

---

## 6. Accessibility Check

**Test Case ID:** FE-006

**Description:** Check for ARIA attributes and accessible elements.

**Steps:**

- Use a screen reader to navigate the webpage.
- Test keyboard navigation (Tab, Shift+Tab).

**Expected Result:** All interactive elements are accessible, and ARIA roles are implemented where necessary.

---

## 7. Button Styling and Hover Effects

**Test Case ID:** FE-007

**Description:** Validate the `Search` and `Get News` buttons' hover effects and appearance.

**Steps:**

- Hover over each button.

**Expected Result:** The buttons change background color smoothly without graphical glitches.

---

## 8. Input Validation

**Test Case ID:** FE-008

**Description:** Test input validation for the search field.

**Steps:**

- Leave the search field blank.

- Click [Search](#).

**Expected Result:** An alert appears, prompting the user to enter a search term.

## 9. Placeholder Data Loading

**Test Case ID:** MD-001

**Description:** Verify that the mock data is correctly imported and utilized when the API call fails.

**Steps:**

- Simulate an API failure (disconnect the backend server).
- Load the webpage.
- Check if mock data populates the results section.

**Expected Result:** Mock data entries from `mockData.js` display correctly in the designated section.

---

## 10. Display Format for Mock Data

**Test Case ID:** MD-002

**Description:** Ensure that mock data entries appear in the correct format.

**Steps:**

- Load the webpage.
- Observe how each mock data entry is displayed.

**Expected Result:** Each entry shows:

- Title in bold.
  - Summary as plain text.
  - A clickable [Read more](#) link that opens a new tab.
- 

## 11. Mock Data Completeness

**Test Case ID:** MD-003

**Description:** Check if all mock data entries are displayed when API fails.

**Steps:**

- Simulate an API failure.
- Load the webpage.

**Expected Result:** All 10 entries from `mockData.js` should be visible in the results section.

---

## 12. Mock Data Link Validation

**Test Case ID:** MD-004

**Description:** Validate the [link](#) field for each mock data entry.

**Steps:**

- Simulate an API failure.
- Click on the [Read more](#) link for each mock data entry.

**Expected Result:** Each link should navigate to a valid URL (or a placeholder if mocked).

---

### 13. Mock Data Integration Test

**Test Case ID:** MD-005

**Description:** Test the integration of mock data with front-end rendering.

**Steps:**

- Simulate API failure.
- Load the page and inspect the console logs for any errors.

**Expected Result:** The webpage should display mock data without console errors or missing elements.

---

### 14. API and Mock Data Switch Validation

**Test Case ID:** MD-006

**Description:** Validate the fallback mechanism between API and mock data.

**Steps:**

- Ensure the backend API is running and responding.
- Load the page with valid API responses.
- Stop the backend API and refresh the page.

**Expected Result:** The webpage should seamlessly switch to displaying mock data when the API fails.

---

### 15. Mock Data Sorting and Filtering (Future Enhancement)

**Test Case ID:** MD-007

**Description:** Test sorting and filtering mechanisms applied to mock data.

**Steps:**

- Implement a sort or filter function in the front-end.
- Apply sorting or filtering to mock data entries.

**Expected Result:** Entries should display according to the selected sorting or filtering criteria.

## Back-End Test Cases for Your Financial News

---

### 1. News Crawler

**Test Case ID:** BE-002

**Description:** Verify the indexer works

**Steps:**

- terminal command: `cd news_crawler`
- terminal command: `scrapy crawl financial_news`
- Check the news-data folder under news\_crawler

**Expected Result:** webpages are downloaded and saved in the news\_data folder locally



```
▼ news_crawler
  > news_crawler
  ▼ news_data
    <> 10a52bc8-8dff-4977-98be-cd0605d4fe2c.html
    <> 81b22ef5-1309-480a-b85f-12c671767b71.html
    <> 478ee49b-c505-40b9-8ddd-d0c049dcd71e.html
    <> 7843d6fb-1e90-4541-873f-644ff8397c25.html
    <> 09675f87-3545-429f-9c3b-112e12ccde30.html
    <> 45408222-25cd-4cca-a135-db703cd1983b.html
    <> b1fb2831-8c44-4f3c-904a-66a333a7867f.html
    <> bb274605-3903-4725-84de-18bbdb356698.html
    <> bbfdc780-b2a6-43f3-8b02-70a4ecc8347d.html
    <> e23edd97-bba6-4ec7-8be8-465b34cd935b.html
    <> e36e8bd7-2ffd-448b-8ac2-f59bbbb91787.html
    <> fe1d83f2-5240-4992-9b5a-a20c24125661.html
```

and in mongoDB

```
financialnews> show collections
news_crawl_status
financialnews> db.news_crawl_status.find().pretty()
[
  {
    _id: ObjectId('67403e767fe6be4445cebc7d'),
    url: 'https://finance.yahoo.com/quote/NVDA/',
    doc_name: '81b22ef5-1309-480a-b85f-12c671767b71',
    last_modified: ISODate('2024-11-22T08:19:02.875Z'),
    created_at: ISODate('2024-11-22T08:19:02.875Z'),
    indexed: false
  }
]
```

If the web banned the crawling IP, the news data would not be updated or crawled.

---

## 2. News Indexer

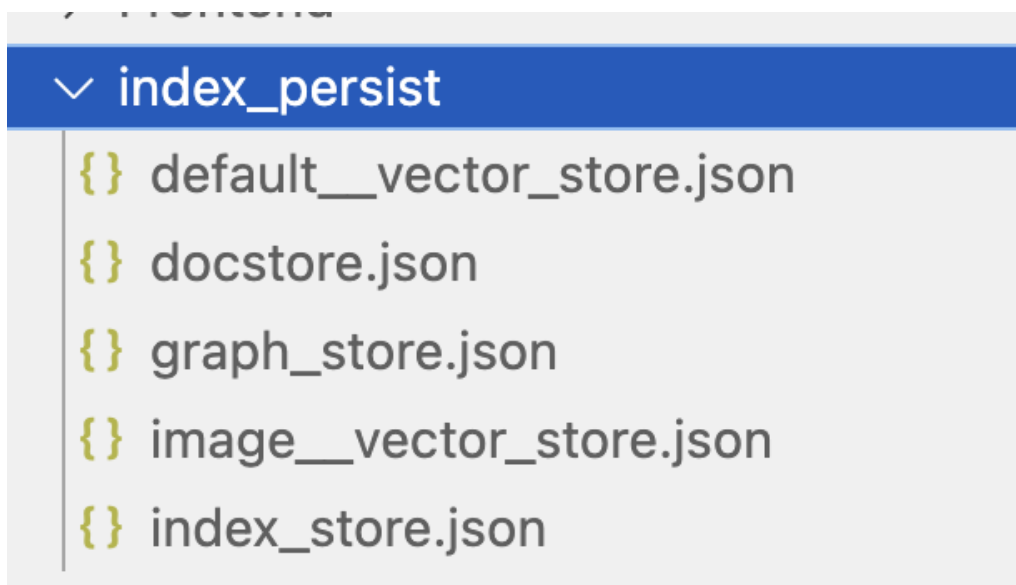
**Test Case ID:** NC-002

**Description:** Verify the indexer works

### Steps:

- terminal command: `cd news_indexer`
- terminal command: `python news_index.py`
- Check the `index_persist` folder

**Expected Result:** 5 json files, `default_vector_Store.json`, `docstore.json`, `graph_store.json`, `image_vectore_store.json`, `index_store.json`, generated and saved in the `index_persist` folder



---

### 3. MongoDB

**Test Case ID:** NC-003

**Description:** Verify the metadata or processed news data are stored in MongoDB

**Steps:**

- MongoDB terminal command: use financialnews, then check if the database, financialnews, has been created

**Expected Result:**

```
test> use financialnews
switched to db financialnews
```

- MongoDB terminal command: show collections, then check if the news\_crawl\_status collection has been created

**Expected Result:**

```
financialnews> show collections
news_crawl_status
```

- MongoDB terminal command: db.news\_crawl\_status.find().pretty(), to check the stored data

**Expected Result:**

```
financialnews> show collections
news_crawl_status
financialnews> db.news_crawl_status.find().pretty()
[
  {
    _id: ObjectId('67403e767fe6be4445cebc7d'),
    url: 'https://finance.yahoo.com/quote/NVDA/',
    doc_name: '81b22ef5-1309-480a-b85f-12c671767b71',
    last_modified: ISODate('2024-11-22T08:19:02.875Z'),
    created_at: ISODate('2024-11-22T08:19:02.875Z'),
    indexed: false
  }
]
```

---

## 4. API

Test Case ID: NC-004

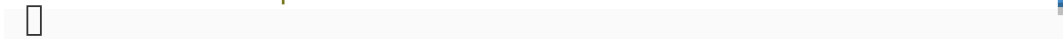
**Description:** Verify the backend component can crawl news, index ,and query successfully

**Steps:**

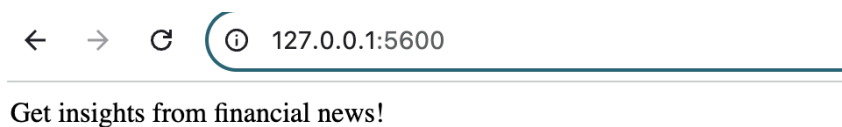
- terminal command: cd APIs
- terminal command: python app.py

**Expected result:**

```
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production
n deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5600
* Running on http://172.20.2.198:5600
Press CTRL+C to quit
```

- 
- go to <http://127.0.0.1:5600>

**Expected Result:**



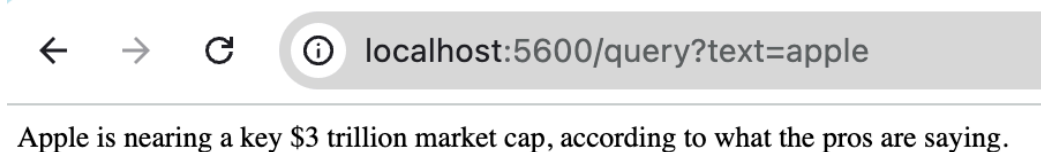
← → ↻ ⓘ 127.0.0.1:5600

---

Get insights from financial news!

- try query: <http://localhost:5600/query?text=apple>

**Expected Result:**



← → ↻ ⓘ localhost:5600/query?text=apple

---

Apple is nearing a key \$3 trillion market cap, according to what the pros are saying.

- try query: <http://localhost:5600/query?text=tea>

**Expected Result:**





localhost:5600/query?text=tea

---

The context provided does not contain any information related to tea.

---