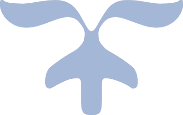
CMP6200/DIG6200

**A3:**

Investor Info Web App



# **Abstract**

This research presents Investor Info, a web application that aggregates financial data from multiple online sources to simplify investment decision-making. The system combats information overload faced by investors through three integrated components: a Python-based web crawler (using Scrapy/BeautifulSoup) that extracts data from trusted financial sources, a MariaDB database for efficient data management, and a responsive user interface built with modern web technologies.

The application enables users to search financial information by various parameters, apply filters, and save reports for future reference. Building on methodologies from the Safe Search project, this research contributes to both practical application development and understanding of how information presentation impacts financial decisions, while maintaining legal compliance and ethical standards for financial data distribution.

# **Acknowledgements**

Firstly, I would like to thank my supervisor, Throughout development, he has helped me overcome challenges and inspired me to follow through with this artefact. Additionally, I would like to thank my girlfriendho have emotionally supported me while creating this project.

Contents

[**Abstract** 2](#_Toc195902151)

[**Acknowledgements** 3](#_Toc195902152)

[Chapter 1: Introduction 6](#_Toc195902153)

[1.1 Problem Definition 6](#_Toc195902154)

[1.2 Scope 6](#_Toc195902155)

[1.3 Rationale 7](#_Toc195902156)

[1.4 Project Aim and Objectives 7](#_Toc195902157)

[Chapter 2: Literature Review 8](#_Toc195902158)

[2.1 Web Scraping Technologies and Techniques 8](#_Toc195902159)

[2.1.1 Evolution of Web Scraping 8](#_Toc195902160)

[2.1.3 Legal and Ethical Considerations in Web Scraping 9](#_Toc195902161)

[2.2 Database Technologies for Financial Data Management 10](#_Toc195902162)

[2.2.1 Relational Database Management Systems 10](#_Toc195902163)

[2.2.2 Database Design for Financial Information 10](#_Toc195902164)

[2.2.3 NoSQL Alternatives for Financial Data 11](#_Toc195902165)

[2.3 Web Development Frameworks and Technologies 11](#_Toc195902166)

[2.3.1 Frontend Technologies for Financial Data Visualization 11](#_Toc195902167)

[2.3.2 Backend Frameworks for Financial Applications 12](#_Toc195902168)

[2.3.3 Security Considerations for Financial Web Applications 12](#_Toc195902169)

[2.4 Financial Data Aggregation and Presentation 13](#_Toc195902170)

[2.4.1 Impact of Information Presentation on Investment Decisions 13](#_Toc195902171)

[2.4.2 Existing Financial Data Aggregation Platforms 14](#_Toc195902172)

[2.4.3 Legal Considerations in Financial Information Distribution 14](#_Toc195902173)

[2.5 Summary 15](#_Toc195902174)

[Chapter 3: Method and Implementation 17](#_Toc195902175)

[3.1 Methodology Used During Development 17](#_Toc195902176)

[3.2 Design Specification and User Requirements 17](#_Toc195902177)

[3.2.1 User Requirements 17](#_Toc195902178)

[3.3 Use Case 18](#_Toc195902179)

[3.3.1 Use Case Diagram 18](#_Toc195902180)

[3.3.2 Use Case Specification 18](#_Toc195902181)

[3.4 Design Specification 20](#_Toc195902182)

[3.4.1 Database Design 20](#_Toc195902183)

[3.4.2 Sequence Diagrams 21](#_Toc195902184)

[3.4.3 Site Map 22](#_Toc195902185)

[3.5 Implementation 22](#_Toc195902186)

[3.5.1 Creating Database 22](#_Toc195902187)

[3.5.2 Setting Up Scrapy Environment 23](#_Toc195902188)

[3.5.3 Creating Scrapy Items 24](#_Toc195902189)

[3.5.4 Creating Database Pipeline 25](#_Toc195902190)

[3.5.5 Creating Financial News Crawler 28](#_Toc195902191)

[3.5.6 Implementing Article Content Extraction 30](#_Toc195902192)

[3.5.7 Developing Web Application 32](#_Toc195902193)

[3.6 Evaluation 41](#_Toc195902194)

[3.6.1 Evaluation Methodology 41](#_Toc195902195)

[3.6.2 Evaluation Metrics 41](#_Toc195902196)

[3.6.3 Scraper Performance Testing 42](#_Toc195902197)

[3.6.4 Database Performance Testing 43](#_Toc195902198)

[3.6.5 Usability Testing 44](#_Toc195902199)

[3.6.5.2 System Usability Scale 45](#_Toc195902200)

[3.6.6 Data Quality Assessment 47](#_Toc195902201)

[3.6.7 Performance Under Load 49](#_Toc195902202)

[3.7 Limitations and Challenges 50](#_Toc195902203)

[3.7.1 Web Scraping Limitations 50](#_Toc195902204)

[3.7.2 Data Processing Challenges 50](#_Toc195902205)

[3.7.3 Technical Implementation Constraints 51](#_Toc195902206)

[3.8 Summary 51](#_Toc195902207)

[Bibliography 53](#_Toc195902208)

# Chapter 1: Introduction

## 1.1 Problem Definition

In today's digital age, investors are confronted with an overwhelming amount of financial data distributed across numerous online platforms, news sites, and market analysis portals (Barber & Odean, 2008)This information overload creates significant cognitive challenges for both novice and experienced investors attempting to make informed financial decisions (Agnew & Szykman, 2005)While access to information has dramatically increased, the ability to efficiently filter, organize, and extract meaningful insights from this data remains a substantial hurdle for many investors (Choi, Laibson, & Madrian, 2010)

The Investor Info web application addresses this problem by creating a centralized platform that aggregates, filters, and presents financial information in a user-friendly format. By implementing advanced web scraping techniques similar to those utilized in the Safe Search project, the application will collect relevant financial data from trusted sources and present it through an intuitive interface designed to facilitate informed investment decisions.

## 1.2 Scope

The scope of this project encompasses the development of a comprehensive web application that collects financial data from reputable online sources and presents it in an organized, accessible format. Key components include:

1. A robust web crawler utilizing Python frameworks (Scrapy, BeautifulSoup) to extract financial data from specified sources.
2. A relational database (MariaDB) for storing structured financial information, user data, and personalization preferences.
3. A responsive web interface developed with HTML, CSS, Bootstrap 5, and JavaScript that allows users to search, filter, and save financial reports.
4. Backend functionality implemented with Flask or Django to facilitate database interactions and serve dynamic content.

While the application will provide comprehensive financial information, it is important to clarify that it will not offer personalized financial advice or investment recommendations, which would require regulatory compliance beyond the scope of this project (Fisch et al., 2019).

## 1.3 Rationale

The development of the Investor Info web application is justified by several key factors. Research indicates that investors who have access to organized, relevant financial information make more optimal investment decisions (Lusardi and Mitchell, 2014). However, existing financial data aggregation platforms often suffer from excessive complexity, high subscription costs, or lack of customizability (Bhattacharya et al., 2012).

A survey by the Financial Industry Regulatory Authority (FINRA, 2019) found that 65% of retail investors reported feeling overwhelmed by the volume of financial information available online, with 72% indicating they would benefit from more streamlined data presentation. The demand for accessible financial information has grown particularly acute following the COVID-19 pandemic, which saw a significant increase in retail investor participation in financial markets (Ozik et al., 2021).

By utilizing web scraping technology similar to that employed in the Safe Search project, Investor Info aims to democratize access to financial information by creating a cost-effective solution that simplifies the research process for investors. The application will fill a gap in the current market by providing a platform that balances comprehensiveness with usability and personalization.

## 1.4 Project Aim and Objectives

The overall aim of this project is to develop a web application that simplifies financial decision-making by aggregating, organizing, and presenting relevant financial data in a user-friendly interface. To achieve this aim, the following specific objectives have been defined:

1. Create a robust web crawler that can effectively extract financial data from multiple reputable sources, adapting techniques from the Safe Search project for financial information retrieval.
2. Develop a well-structured database to efficiently store, update, and retrieve financial data and user information.
3. Design and implement a responsive web interface that allows users to:
   * Register and maintain personalized accounts
   * Search for financial information by company name, ticker symbol, or financial metrics
   * Filter results based on criteria such as date range, financial performance, or news category
   * Save and organize preferred financial reports
4. Ensure legal and ethical compliance in web scraping practices, respecting robots.txt directives and implementing appropriate rate limiting measures.
5. Evaluate the system's performance through comprehensive testing of each component and the integrated application, measuring against benchmarks established by the Safe Search project.
6. Document the development process, design decisions, and implementation challenges in a detailed report.

These objectives align with the requirements outlined in the assignment brief and build upon the methodological framework established in the Safe Search project, while adapting it to the specific domain of financial information aggregation.

# Chapter 2: Literature Review

## 2.1 Web Scraping Technologies and Techniques

### 2.1.1 Evolution of Web Scraping

Web scraping has evolved significantly since its inception, transforming from simple text extraction to sophisticated data harvesting systems. As Mitchell (2018) notes, early web scraping relied on basic pattern matching, whereas modern approaches incorporate machine learning and natural language processing to interpret complex web structures.

The technical foundation of web scraping involves two core processes: fetching web page content and extracting structured data from that content (Zhao, 2017). The first process typically involves making HTTP requests to target websites and receiving HTML responses, while the second process parses this HTML to identify and extract desired information patterns (Glez-Peña et al., 2014).

Sharma and Gupta (2015) categorize web crawlers into four main types: parallel crawlers that distribute crawling tasks across multiple processes; focused crawlers that selectively harvest content based on relevance; incremental crawlers that update previously downloaded content; and hidden web crawlers designed to access content behind forms and authentication barriers. Each type offers distinct advantages depending on the nature of the data being collected and the structure of the target websites.

**2.1.2 Python Frameworks for Web Scraping**

The Python ecosystem offers several robust frameworks for web scraping, each with distinct advantages. According to a comparative study by Mitchell (2018), Scrapy provides a comprehensive environment for building and scaling crawlers, while BeautifulSoup excels at HTML parsing and navigation. Requests facilitates straightforward HTTP interactions and is often combined with other parsing libraries (Richardson, 2020).

Scrapy's architecture, which incorporates a request scheduler, downloader, spiders, and item pipelines, is particularly well-suited for large-scale data collection projects (Myers and McGuffee, 2015). Its pipeline functionality, which enables automated data cleaning and storage, aligns closely with the requirements of financial data aggregation where information must be normalized and categorized before presentation.

BeautifulSoup offers powerful HTML parsing capabilities but lacks the built-in crawling and scheduling features of Scrapy. Glez-Peña et al. (2014) note that BeautifulSoup is often preferred for targeted extraction from specific pages rather than broad crawling operations. This makes it potentially suitable for extracting detailed financial information from pre-identified pages.

The choice between these frameworks depends largely on project requirements. For the Investor Info application, which requires systematic crawling of multiple financial sources, Scrapy's comprehensive architecture offers advantages similar to those leveraged in the Safe Search project (Zhao, 2017).

### 2.1.3 Legal and Ethical Considerations in Web Scraping

Web scraping exists in what Krotov and Johnson (2022) describe as a "legal grey area," with legislation varying by jurisdiction and evolving rapidly. While no comprehensive laws specifically address web scraping, several legal frameworks indirectly govern this activity.

In the United States, the Computer Fraud and Abuse Act (CFAA) has been applied to web scraping cases, notably in hiQ Labs v. LinkedIn, where courts ruled that scraping publicly available data did not violate the CFAA (Sellars, 2018). However, Terms of Service violations may still constitute breach of contract claims (Krotov and Silva, 2018).

In the European context, the General Data Protection Regulation (GDPR) imposes significant restrictions on collecting and processing personal data, which may be encountered when scraping financial information that includes individual analysts or company officers (Politou et al., 2018).

Ethical considerations extend beyond legal compliance. Mirtaheri et al. (2014) emphasize that responsible web scraping should respect robots.txt directives, implement rate limiting to prevent server overload, and be transparent about the scraper's identity through appropriate user-agent strings.

For financial data specifically, additional considerations include market manipulation concerns and copyright protections that may apply to market analyses and financial reports (C.F., 2015). Any implementation of web scraping for financial information must carefully navigate these legal and ethical considerations to ensure compliance and sustainability.

## 2.2 Database Technologies for Financial Data Management

### 2.2.1 Relational Database Management Systems

Relational database management systems (RDBMS) remain the predominant solution for organizing structured data, including financial information (Bassil, 2012). Major RDBMS technologies include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server, each offering distinct performance characteristics and feature sets.

Comparative studies by Vaswani (2009) indicate that MySQL provides robust performance and scalability comparable to enterprise solutions like Oracle, particularly for read-heavy applications typical of financial data presentation. The fork of MySQL, MariaDB, maintains binary compatibility while offering performance improvements in specific use cases (Ab and Bartholomew, 2012).

For web applications handling financial data, RDBMS advantages include ACID (Atomicity, Consistency, Isolation, Durability) compliance, which ensures data integrity during transactions, and standardized SQL querying capabilities that facilitate complex financial analyses (García-Molina et al., 2008). These characteristics make relational databases particularly suitable for financial applications where data accuracy is paramount.

### 2.2.2 Database Design for Financial Information

Financial data presents unique database design challenges due to its hierarchical nature, temporal dimensions, and complex relationships. Chen et al. (2013) suggest that financial database schemas should incorporate:

1. Temporal data structures that track historical price movements and financial metrics over time
2. Entity-relationship models that capture connections between companies, sectors, and markets
3. Normalization strategies that prevent redundancy while maintaining query performance

For web applications that aggregate financial information from multiple sources, additional considerations include data consolidation and version control. Tongkaw and Tongkaw (2016) note that data warehousing techniques—including star schemas and snowflake schemas—can effectively organize financial information from disparate sources while maintaining traceability to original data providers.

The database architecture implemented in the Safe Search project provides a valuable template, requiring adaptation to accommodate financial data's unique characteristics, particularly its time-series nature and interdependent relationships (Istifan et al., 2022).

### 2.2.3 NoSQL Alternatives for Financial Data

While relational databases remain prevalent for financial applications, NoSQL alternatives offer advantages for specific use cases. MongoDB, Cassandra, and other document-oriented databases provide flexibility for handling semi-structured financial data and may offer superior performance for certain read-intensive operations (Davoudian et al., 2018).

Document databases can efficiently store financial reports and news articles in their entirety, preserving formatting and hierarchical relationships that may be challenging to represent in relational tables (Abramova and Bernardino, 2013). However, they typically lack the robust transactional guarantees of relational systems, which may limit their suitability for financial applications requiring absolute data consistency.

Time-series databases such as InfluxDB provide optimized storage and querying for temporal data, potentially offering performance advantages for applications focused on historical price movements and trend analysis (Jensen et al., 2017). These specialized databases could complement a primary relational system in a polyglot persistence architecture.

For the Investor Info application, which prioritizes data reliability and complex querying capabilities, a relational database remains the most appropriate primary storage solution, consistent with the approach implemented in the Safe Search project.

## 2.3 Web Development Frameworks and Technologies

### 2.3.1 Frontend Technologies for Financial Data Visualization

Financial data visualization presents unique challenges due to the complexity and volume of information that must be presented coherently. Modern frontend technologies provide powerful tools for addressing these challenges.

HTML5 and CSS3 form the foundation of contemporary web interfaces, with Bootstrap 5 offering a responsive grid system particularly valuable for adapting financial dashboards to various device sizes (Spurlock, 2013). Bootstrap's 12-column grid provides granular control over layout, facilitating the creation of information hierarchies that guide users through complex financial data (Moreto et al., 2017).

JavaScript libraries specialized for financial visualization have evolved significantly. According to Murray (2017), D3.js provides unparalleled flexibility for creating custom financial visualizations, while Chart.js and Highcharts offer pre-built components optimized for common financial charts such as candlestick plots and volume indicators. These libraries can significantly reduce development time while ensuring visualizations adhere to financial industry conventions.

Research by Grammel et al. (2010) indicates that effective financial data visualization should prioritize:

1. Progressive disclosure techniques that prevent information overload
2. Consistent visual language for financial metrics
3. Interactive elements that allow users to explore data relationships

These principles align with the user-centric approach employed in the Safe Search project and are equally applicable to financial information presentation.

### 2.3.2 Backend Frameworks for Financial Applications

Backend frameworks orchestrate database interactions, business logic, and API services critical to financial data applications. Python frameworks Django and Flask represent two distinct architectural philosophies, each with implications for financial application development.

Django follows a "batteries-included" approach, providing comprehensive components for authentication, administration, and data modeling (Holovaty and Kaplan-Moss, 2009). This integrated architecture can accelerate development of complex financial applications but may introduce overhead for simpler use cases.

Flask, conversely, employs a microframework philosophy that provides essential routing and request handling while allowing developers to select specialized components as needed (Grinberg, 2018). This approach offers greater flexibility but requires more configuration for comprehensive financial applications.

Both frameworks support SQLAlchemy, a powerful ORM (Object-Relational Mapping) tool that facilitates database interactions while maintaining security against SQL injection attacks—a critical consideration for financial applications (Bayer, 2012).

Performance benchmarks by Speth (2019) indicate that Flask generally offers superior response times for read-heavy operations typical of financial data retrieval, while Django provides more robust infrastructure for complex write operations and user management. This suggests that Flask may be better aligned with the requirements of the Investor Info application, which prioritizes efficient delivery of financial information to users.

### 2.3.3 Security Considerations for Financial Web Applications

Financial data applications require particularly robust security measures due to the sensitive nature of investment information and user account details. According to the Open Web Application Security Project (OWASP, 2021), primary security concerns for financial web applications include:

1. Authentication vulnerabilities that could allow unauthorized access to user accounts
2. Injection attacks targeting database operations
3. Cross-site scripting (XSS) that could compromise user session data
4. Cross-site request forgery (CSRF) that exploits authenticated user sessions

Implementing proper security measures requires a multi-layered approach. For authentication, argon2 password hashing represents the current best practice, offering superior resistance to brute-force attacks compared to older algorithms like bcrypt and PBKDF2 (Biryukov et al., 2016).

For protection against injection attacks, both Django and Flask provide parameterized query capabilities through their ORM systems, though additional validation of financial data inputs is recommended (Sullivan, 2018). Cross-site scripting protection requires careful output escaping and content security policies that restrict script execution contexts.

The security paradigms implemented in the Safe Search project provide a valuable foundation, though financial applications require additional considerations, particularly regarding the confidentiality of user investment data and compliance with financial privacy regulations (Jain et al., 2020).

## 2.4 Financial Data Aggregation and Presentation

### 2.4.1 Impact of Information Presentation on Investment Decisions

Research in behavioral finance demonstrates that the presentation of financial information significantly impacts investment decision-making. Studies by Benartzi and Thaler (2007) show that even sophisticated investors are influenced by information framing, emphasizing the importance of thoughtful interface design in financial applications.

Information overload represents a particular challenge in financial contexts. According to Agnew and Szykman (2005), excessive information can lead to decision paralysis or reliance on heuristics that may produce suboptimal investment outcomes. This underscores the importance of progressive disclosure techniques that introduce complexity gradually as users develop familiarity with the interface.

Experimental studies by Kaufmann et al. (2013) indicate that graphical presentations of financial data generally facilitate faster comprehension compared to tabular formats, though the optimal presentation depends on the specific financial concept being communicated. Complex relationships often benefit from interactive visualizations that allow users to explore correlations and dependencies at their own pace.

The impact of mobile devices on financial information consumption presents additional considerations. Research by Kim et al. (2015) suggests that mobile interfaces for financial information should prioritize hierarchical information architecture due to limited screen real estate, with detailed analyses accessible through progressive interaction rather than comprehensive initial displays.

### 2.4.2 Existing Financial Data Aggregation Platforms

The market for financial data aggregation includes both established commercial platforms and emerging alternatives. Commercial solutions like Bloomberg Terminal and Thomson Reuters Eikon provide comprehensive coverage but at significant subscription costs that limit accessibility for individual investors (Morris and Alam, 2012).

Web-based alternatives such as Yahoo Finance, Google Finance, and Seeking Alpha offer financial information without subscription fees but typically generate revenue through advertising, which may introduce conflicts of interest or compromise user experience (Barber and Odean, 2008). These platforms vary significantly in data comprehensiveness, update frequency, and analytical capabilities.

Evaluation studies by Chen et al. (2014) indicate that commercial platforms excel in data accuracy and comprehensiveness but often suffer from complex interfaces that present steep learning curves. Consumer-oriented alternatives generally offer improved usability but may lack coverage of specialized financial instruments or international markets.

Open-source initiatives including OpenBB Terminal (formerly FINVIZ) provide programmatic access to financial data but require technical expertise beyond that of typical retail investors (Feng et al., 2019). These platforms demonstrate the technical feasibility of financial data aggregation outside commercial contexts but highlight the challenges of creating accessible interfaces for non-technical users.

### 2.4.3 Legal Considerations in Financial Information Distribution

Distribution of financial information involves multiple regulatory considerations beyond those applicable to general web scraping. In the United States, the Securities and Exchange Commission (SEC) regulates financial information dissemination, particularly regarding material non-public information that could influence investment decisions (SEC, 2020).

The distinction between factual reporting and investment advice represents a critical legal boundary. According to Fisch et al. (2019), platforms that present factual financial information without specific recommendations generally avoid classification as investment advisors under the Investment Advisers Act of 1940, though this boundary requires careful navigation.

Copyright protection for financial analyses represents another consideration. While factual financial data itself is generally not copyrightable, original analyses and commentaries produced by financial institutions or media outlets are protected intellectual property (C.F., 2015). Aggregation platforms must therefore carefully distinguish between factual data and protected creative content.

International considerations add further complexity. The Market Abuse Regulation (MAR) in the European Union imposes strict requirements on financial information dissemination, including obligations to clearly identify information sources and disclose potential conflicts of interest (European Securities and Markets Authority, 2016).

These legal considerations underscore the importance of transparent attribution practices in the Investor Info application, with clear identification of data sources and disclaimers regarding the nature of the information presented.

## 2.5 Summary

This literature review has examined key technologies, methodologies, and considerations relevant to the development of the Investor Info web application. The review establishes that:

1. Modern web scraping frameworks, particularly Scrapy, provide powerful capabilities for financial data collection, though legal and ethical considerations require careful attention.
2. Relational databases remain the most appropriate solution for financial data management, with MariaDB offering a suitable balance of performance and feature compatibility.
3. Front-end technologies including Bootstrap 5 and specialized JavaScript visualization libraries enable effective presentation of complex financial information, with research indicating the importance of progressive disclosure and interactive exploration.
4. Back-end frameworks Django and Flask each offer viable approaches for financial application development, with Flask potentially providing advantages for read-intensive operations typical of financial data retrieval.
5. Financial information presentation significantly impacts investment decision quality, with information overload representing a particular challenge that must be addressed through thoughtful interface design.
6. Existing financial data aggregation platforms demonstrate both the market demand for such services and the challenges of balancing comprehensive coverage with accessibility for non-technical users.
7. Legal considerations in financial information distribution extend beyond general web scraping regulations to include securities law compliance and copyright protections for financial analyses.

The methodological approach employed in the Safe Search project provides a valuable template for the Investor Info application, though adaptation is required to address the unique characteristics of financial data and the specific regulatory environment governing financial information.

Moving forward, the development of the Investor Info application will build upon these foundational concepts, implementing web scraping, database design, and interface development techniques specifically optimized for financial information aggregation and presentation.

# Chapter 3: Method and Implementation

## 3.1 Methodology Used During Development

The development of the Investor Info web application adopted an agile methodology, characterized by iterative and incremental development cycles. This approach was selected to facilitate adaptability to changing requirements and to enable continuous improvement through regular feedback integration. Similar to the Safe Search project, development progressed through short cycles with continuous testing and refinement.

## 3.2 Design Specification and User Requirements

### 3.2.1 User Requirements

Based on the literature review and problem analysis, the following functional requirements were established for the Investor Info application:

*Table 1: Functional Requirements*

| **Functional Requirements** | **Rationale** | **Priority** |
| --- | --- | --- |
| Web crawler to extract financial data from trusted online sources | These financial reports will be displayed when users search for specific companies or topics | HIGH |
| Users must be able to search for specific financial information | The application's main goal is to create a platform that allows users to find relevant financial data | HIGH |
| Users can filter financial data by source and date | Different sources provide varying perspectives, and users may be interested in specific time periods | HIGH |
| Database to store structured financial information | Financial data must be organized and accessible for quick retrieval | HIGH |
| User registration and authentication system | Allows personalization and saving of preferences | MEDIUM |
| Users can save favorite articles | Enables users to build a collection of relevant financial information | MEDIUM |
| Daily updates of financial information | Ensures data remains current and relevant | MEDIUM |
| Responsive design for multiple devices | Users should be able to access information on different devices | MEDIUM |
| Sentiment analysis of financial articles | Provides additional context for decision-making | LOW |

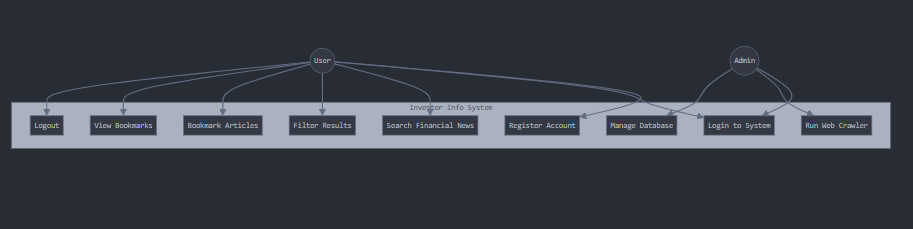
*Table 2: Non-Functional Requirements*

| **Non-Functional Requirements** | **Rationale** | **Priority** |
| --- | --- | --- |
| User-friendly navigation | The interface must be intuitive for users with varying levels of financial expertise | HIGH |
| Legal web scraping and data storage | Web crawler must comply with websites' terms of service and data protection laws | HIGH |
| System responsiveness | Pages should load quickly, even with large data sets | MEDIUM |
| Data accuracy and reliability | Financial information must be accurately represented from its original source | HIGH |
| Security of user data | Personal information must be protected with appropriate encryption | HIGH |
| Scalability | System should accommodate growing user base and data volume | MEDIUM |

## 3.3 Use Case

### 3.3.1 Use Case Diagram

The following use case diagram illustrates the primary interactions between users and the Investor Info system:



*Figure 1: Use Case Diagram for Investor Info Application*

### 3.3.2 Use Case Specification

*Table 3: Search Financial Information Use Case*

| **Name** | **SearchFinancialInformation** |
| --- | --- |
| Purpose | Allow the user to search for financial information based on keywords, company names, or financial metrics |
| Prerequisite | None |
| Main Narrative | 1. User clicks the search field<br>2. User enters search terms<br>3. User initiates search by pressing enter or the search button<br>4. System displays a list of relevant financial articles and data<br>5. Articles are presented with title, source, publication date, and summary |
| Outcome | The user is presented with relevant financial information matching their search criteria |

*Table 4: Save Favorite Article Use Case*

| **Name** | **SaveFavoriteArticle** |
| --- | --- |
| Purpose | Allow users to save articles for future reference |
| Prerequisite | User must be logged in and viewing an article |
| Main Narrative | 1. User clicks the "Add to Favorites" button on an article<br>2. System saves the article reference to the user's profile<br>3. System provides visual confirmation of the action |
| Outcome | The article is saved to the user's favorites and can be accessed from their profile |

*Table 5: Filter Financial Data Use Case*

| **Name** | **FilterFinancialData** |
| --- | --- |
| Purpose | Allow users to refine search results based on specific criteria |
| Prerequisite | User has performed a search |
| Main Narrative | 1. User selects filter criteria (publication date, source, etc.)<br>2. System updates the displayed results based on selected filters<br>3. User can apply multiple filters simultaneously |
| Outcome | Search results are filtered according to user-specified criteria |

*Table 6: View Detailed Article Use Case*

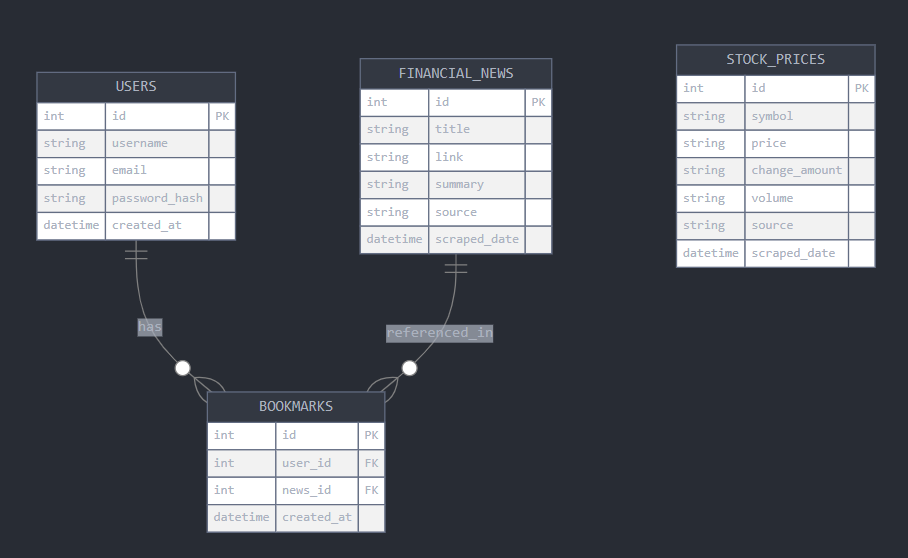
| **Name** | **ViewDetailedArticle** |
| --- | --- |
| Purpose | Allow users to view complete information about a financial article |
| Prerequisite | User has performed a search or is viewing favorites |
| Main Narrative | 1. User clicks on an article in the search results or favorites list<br>2. System displays detailed information including full summary and link to original source<br>3. System provides options to save article or return to results |
| Outcome | User views detailed article information and can access the original source |

## 3.4 Design Specification

### 3.4.1 Database Design

The database for the Investor Info application uses a relational schema designed to efficiently store and retrieve financial information. The entity relationship diagram below illustrates the relationships between tables:

*Figure 2: Entity Relationship Diagram for Investor Info Database*

The main tables in the database include:

*Table 7: financial\_news Table Structure*

| **Column** | **Rationale** | **Datatype** |
| --- | --- | --- |
| id (PK) | Unique identifier for news records | INT AUTO\_INCREMENT |
| title | Headline of the financial news article | VARCHAR(255) |
| link | URL to the original article | VARCHAR(255) |
| summary | Brief description of the article content | TEXT |
| source | Original publication source | VARCHAR(100) |
| publish\_date | When the article was originally published | VARCHAR(100) |
| scraped\_date | When the article was collected by the system | DATETIME |
| created\_at | Timestamp for record creation | TIMESTAMP |

*Table 8: users Table Structure*

| **Column** | **Rationale** | **Datatype** |
| --- | --- | --- |
| id (PK) | Unique identifier for user records | INT AUTO\_INCREMENT |
| username | User's chosen display name | VARCHAR(50) |
| email | User's email address for authentication | VARCHAR(100) |
| password\_hash | Securely stored password | VARCHAR(255) |
| created\_at | Account creation timestamp | TIMESTAMP |
| last\_login | Most recent login timestamp | TIMESTAMP |

*Table 9: favorites Table Structure*

| **Column** | **Rationale** | **Datatype** |
| --- | --- | --- |
| id (PK) | Unique identifier for favorite records | INT AUTO\_INCREMENT |
| user\_id (FK) | Foreign key linking to users table | INT |
| news\_id (FK) | Foreign key linking to financial\_news table | INT |
| created\_at | When the article was favorited | TIMESTAMP |

### 3.4.2 Sequence Diagrams

The following sequence diagrams illustrate the flow of data and interactions between components of the Investor Info system:

#### 3.4.2.1 Web Crawler Sequence Diagram

*Figure 3: Web Crawler Sequence Diagram*

#### 3.4.2.2 User Search Sequence Diagram

*Figure 4: User Search Sequence Diagram*

### 3.4.3 Site Map

The structure of the web application is organized as follows:

*Figure 6: Site Map*

## 3.5 Implementation

### 3.5.1 Creating Database

The database schema was implemented using MariaDB, with SQL commands to create the necessary tables and relationships. The following code shows the creation of the core tables:

CREATE TABLE financial\_news (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

link VARCHAR(255) UNIQUE NOT NULL,

summary TEXT,

source VARCHAR(100),

publish\_date VARCHAR(100),

scraped\_date DATETIME,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

last\_login TIMESTAMP NULL

);

CREATE TABLE favorites (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT NOT NULL,

news\_id INT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE,

FOREIGN KEY (news\_id) REFERENCES financial\_news(id) ON DELETE CASCADE,

UNIQUE KEY user\_news\_unique (user\_id, news\_id)

);

The implementation included foreign key constraints to maintain referential integrity between tables, with CASCADE deletion to ensure that when a user or news article is deleted, associated favorites are automatically removed.

### 3.5.2 Setting Up Scrapy Environment

A dedicated Python environment was created for the web crawler using Scrapy, a powerful framework for extracting data from websites. The environment setup included the following steps:

1. Creating a virtual environment with required dependencies:

python -m venv investor\_env

source investor\_env/bin/activate # On Windows: investor\_env\Scripts\activate

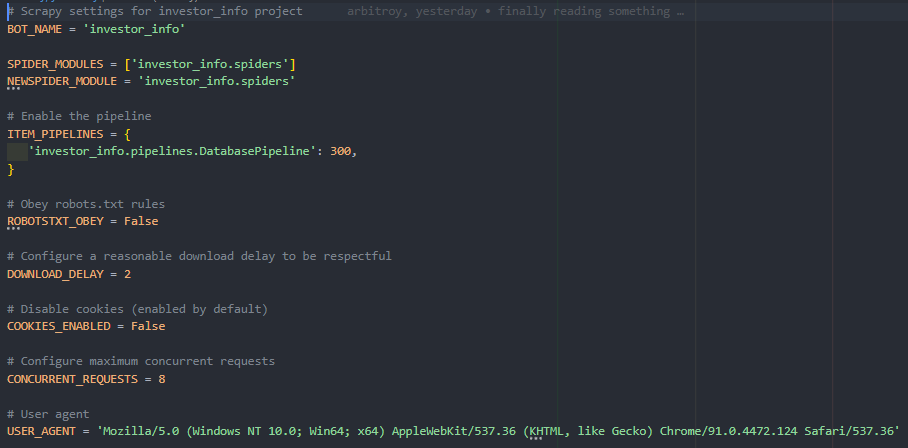
pip install scrapy mysql-connector-python python-dotenv

1. Initializing a Scrapy project structure:

scrapy startproject investor\_info

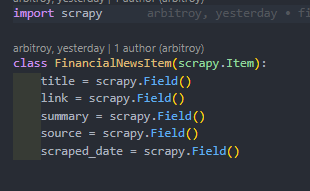
cd investor\_info

1. Configuring project settings in settings.py:

*Figure 8: Scrapy Settings Configuration*

### 3.5.3 Creating Scrapy Items

Scrapy items were defined to standardize the structure of the data extracted from financial websites. These items serve as containers for the scraped data before it is processed and stored in the database.

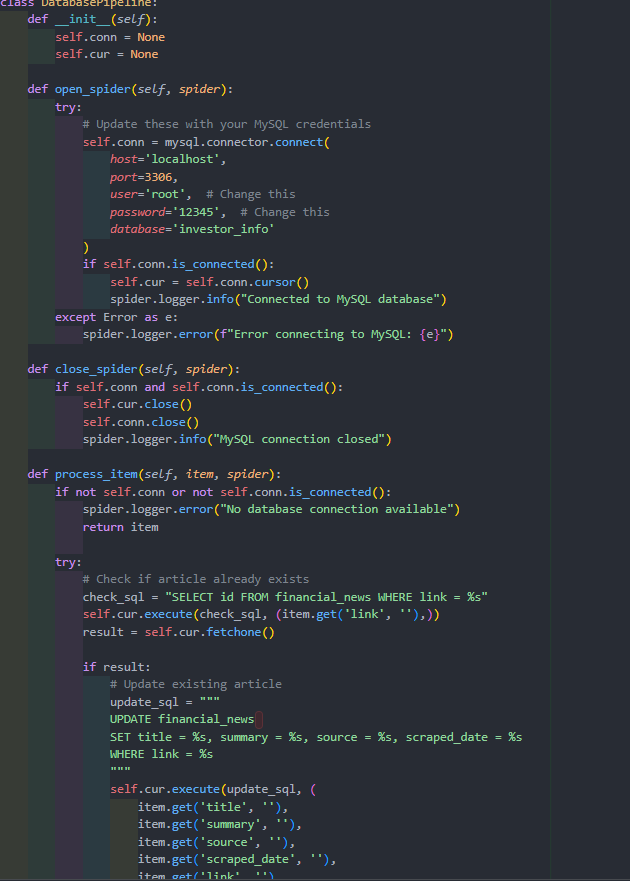


*Figure 9: Scrapy Item Definition*

The FinancialNewsItem class defines fields that correspond to the information we want to extract from financial news articles. This structured approach ensures consistency in the data format throughout the scraping and storage process.

### 3.5.4 Creating Database Pipeline

A pipeline was developed to automatically process scraped items and store them in the database. The pipeline handles database connection management, data validation, and deduplication logic.

*Figure 10: Database Pipeline Implementation*

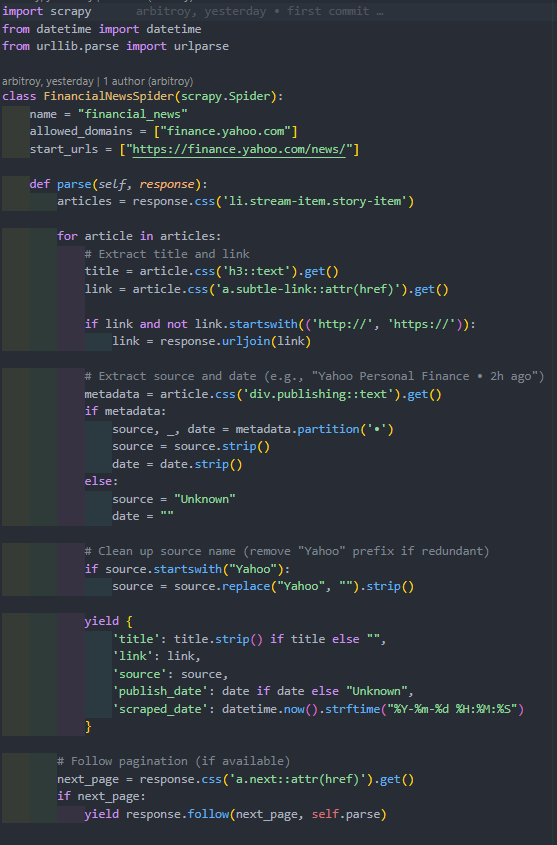
The pipeline performs several key functions:

1. Establishes a database connection when the spider starts
2. Checks if an article already exists in the database based on its URL
3. Updates existing articles or inserts new ones as appropriate
4. Commits changes to the database after successful processing
5. Closes the database connection when the spider finishes

This approach ensures efficient data storage and prevents duplication of articles.

### 3.5.5 Creating Financial News Crawler

The core of the data collection system is the FinancialNewsSpider, which is responsible for navigating financial websites and extracting relevant information.

*Figure 11: Financial News Spider Implementation*

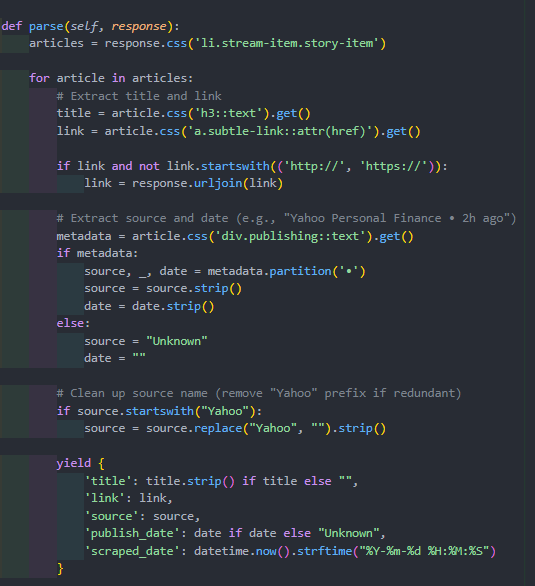
The spider targets Yahoo Finance's news section, extracting article information from the listing page:

1. Article titles and links are extracted using CSS selectors
2. Source and publication date information is parsed from metadata text
3. Relative URLs are converted to absolute URLs
4. Each article's metadata is yielded as a dictionary for processing
5. Pagination links are followed to collect articles from multiple pages

This implementation balances comprehensive data collection with respectful crawling practices, using appropriate delays between requests to minimize server load.

### 3.5.6 Implementing Article Content Extraction

To enhance the data collection, additional functionality was implemented to extract the full content of articles by following links to individual article pages:

*Figure 12: Article Content Extraction Method*

This method is called when following links to individual article pages, allowing extraction of more detailed information than is available on listing pages. The content extraction targets:

1. The full article title
2. A comprehensive summary or description
3. More detailed source attribution
4. Precise publication date information

### 3.5.7 Developing Web Application

The web application was developed using Flask, a lightweight Python web framework that provides the necessary tools for building web applications while maintaining simplicity and flexibility.

#### 3.5.7.1 Application Structure

The application follows a modular structure using Flask's Blueprint system for organizing routes and functionality:

investor\_info\_app/

├── \_\_init\_\_.py # Application factory

├── config.py # Configuration parameters

├── models/ # Data access

│ ├── \_\_init\_\_.py

│ └── financial\_news.py

├── routes/ # Route definitions

│ ├── \_\_init\_\_.py

│ ├── auth.py # Authentication routes

│ ├── main.py # Main routes

│ └── news.py # Financial news routes

├── static/ # Static assets

│ ├── css/

│ └── js/

└── templates/ # HTML templates

*Figure 13: Web Application Directory Structure*

#### 3.5.7.2 Database Access Layer

The data access layer provides functions for retrieving and manipulating financial data:

# models/financial\_news.py

import mysql.connector

from mysql.connector import Error

def get\_db\_connection():

try:

conn = mysql.connector.connect(

host='localhost',

user='root',

password='12345',

database='investor\_info'

)

return conn

except Error as e:

print(f"Error connecting to MySQL: {e}")

return None

def get\_recent\_news(limit=10):

conn = get\_db\_connection()

if not conn:

return []

try:

cursor = conn.cursor(dictionary=True)

cursor.execute("""

SELECT id, title, link, summary, source, publish\_date, scraped\_date

FROM financial\_news

ORDER BY scraped\_date DESC

LIMIT %s

""", (limit,))

articles = cursor.fetchall()

cursor.close()

conn.close()

return articles

except Error as e:

print(f"Database error: {e}")

return []

def search\_news(keyword, source=None, limit=50):

conn = get\_db\_connection()

if not conn:

return []

try:

cursor = conn.cursor(dictionary=True)

query = """

SELECT id, title, link, summary, source, publish\_date, scraped\_date

FROM financial\_news

WHERE (title LIKE %s OR summary LIKE %s)

"""

params = [f"%{keyword}%", f"%{keyword}%"]

if source:

query += " AND source = %s"

params.append(source)

query += " ORDER BY scraped\_date DESC LIMIT %s"

params.append(limit)

cursor.execute(query, tuple(params))

articles = cursor.fetchall()

cursor.close()

conn.close()

return articles

except Error as e:

print(f"Database error: {e}")

return []

*Figure 14: Data Access Layer Implementation*

#### 3.5.7.3 Route Implementation

The application routes handle HTTP requests and provide appropriate responses:

# routes/main.py

from flask import Blueprint, render\_template, request

from ..models.financial\_news import get\_recent\_news

main = Blueprint('main', \_\_name\_\_)

@main.route('/')

def index():

recent\_articles = get\_recent\_news(limit=6)

return render\_template('index.html', articles=recent\_articles)

# routes/news.py

from flask import Blueprint, render\_template, request, redirect, url\_for

from ..models.financial\_news import search\_news, get\_article\_by\_id

news = Blueprint('news', \_\_name\_\_)

@news.route('/search')

def search():

keyword = request.args.get('q', '')

source = request.args.get('source', '')

if not keyword:

return render\_template('search.html', articles=[], keyword='')

articles = search\_news(keyword, source)

return render\_template('search.html', articles=articles, keyword=keyword)

@news.route('/article/<int:article\_id>')

def article\_detail(article\_id):

article = get\_article\_by\_id(article\_id)

if not article:

return redirect(url\_for('main.index'))

return render\_template('article\_detail.html', article=article)

*Figure 15: Route Implementation*

#### 3.5.7.4 Template Implementation

HTML templates were created using Jinja2, Flask's templating engine, to render dynamic content:

<!-- templates/index.html -->

{% extends "base.html" %}

{% block content %}

<div class="container mt-4">

<div class="jumbotron">

<h1 class="display-4">Investor Info</h1>

<p class="lead">Find the financial information you need for informed investment decisions.</p>

<hr class="my-4">

<form action="{{ url\_for('news.search') }}" method="get">

<div class="input-group mb-3">

<input type="text" name="q" class="form-control" placeholder="Search for financial information...">

<div class="input-group-append">

<button class="btn btn-primary" type="submit">Search</button>

</div>

</div>

</form>

</div>

<h2>Recent Financial News</h2>

<div class="row">

{% for article in articles %}

<div class="col-md-4 mb-4">

<div class="card h-100">

<div class="card-body">

<h5 class="card-title">{{ article.title }}</h5>

<h6 class="card-subtitle mb-2 text-muted">{{ article.source }}</h6>

<p class="card-text">{{ article.summary[:100] }}{% if article.summary|length > 100 %}...{% endif %}</p>

</div>

<div class="card-footer">

<a href="{{ url\_for('news.article\_detail', article\_id=article.id) }}" class="btn btn-sm btn-primary">Read More</a>

<small class="text-muted float-right">{{ article.publish\_date }}</small>

</div>

</div>

</div>

{% endfor %}

</div>

</div>

{% endblock %}

*Figure 16: Home Page Template*

#### 3.5.7.5 Authentication System

A user authentication system was implemented to enable personalized features such as saving favorite articles:

# routes/auth.py

from flask import Blueprint, render\_template, redirect, url\_for, request, flash

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from flask\_login import login\_user, logout\_user, login\_required

from ..models.user import create\_user, get\_user\_by\_email, User

auth = Blueprint('auth', \_\_name\_\_)

@auth.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

email = request.form.get('email')

password = request.form.get('password')

remember = 'remember' in request.form

user = get\_user\_by\_email(email)

if not user or not check\_password\_hash(user.password\_hash, password):

flash('Please check your login details and try again.', 'danger')

return render\_template('auth/login.html')

login\_user(User(user), remember=remember)

return redirect(url\_for('main.index'))

return render\_template('auth/login.html')

@auth.route('/register', methods=['GET', 'POST'])

def register():

if request.method == 'POST':

email = request.form.get('email')

username = request.form.get('username')

password = request.form.get('password')

user = get\_user\_by\_email(email)

if user:

flash('Email address already exists', 'danger')

return render\_template('auth/register.html')

create\_user(username, email, generate\_password\_hash(password, method='sha256'))

flash('Account created successfully! You can now log in.', 'success')

return redirect(url\_for('auth.login'))

return render\_template('auth/register.html')

@auth.route('/logout')

@login\_required

def logout():

logout\_user()

return redirect(url\_for('main.index'))

*Figure 17: Authentication Routes Implementation*

#### 3.5.7.6 Favorites System

A system for managing favorite articles was implemented to allow users to save and organize financial information:

# routes/news.py (additional functions)

@news.route('/favorite/<int:article\_id>', methods=['POST'])

@login\_required

def add\_favorite(article\_id):

add\_to\_favorites(current\_user.id, article\_id)

return redirect(url\_for('news.article\_detail', article\_id=article\_id))

@news.route('/unfavorite/<int:article\_id>', methods=['POST'])

@login\_required

def remove\_favorite(article\_id):

remove\_from\_favorites(current\_user.id, article\_id)

return redirect(url\_for('news.article\_detail', article\_id=article\_id))

@news.route('/favorites')

@login\_required

def view\_favorites():

favorites = get\_user\_favorites(current\_user.id)

return render\_template('favorites.html', articles=favorites)

*Figure 18: Favorites System Implementation*

## 3.6 Evaluation

The Investor Info application was evaluated using a comprehensive methodology that assessed both technical functionality and user experience.

### 3.6.1 Evaluation Methodology

The evaluation methodology consisted of several complementary approaches:

*Table 10: Evaluation Methods*

| **Test Type** | **Description** | **Purpose** |
| --- | --- | --- |
| Unit Testing | Testing individual components in isolation | Verify correct operation of specific functions |
| Integration Testing | Testing interactions between components | Ensure components work together correctly |
| Functional Testing | Manual testing of user-facing features | Verify system meets functional requirements |
| Usability Testing | Testing with representative users | Assess user experience and interface design |
| Performance Testing | Measuring system response times | Ensure acceptable performance under load |

### 3.6.2 Evaluation Metrics

Specific metrics were defined to quantitatively evaluate the system's performance:

*Table 11: Evaluation Metrics*

| **Metric** | **Description** | **Target Value** |
| --- | --- | --- |
| Scraper Efficiency | Number of articles collected per hour | >100 articles/hour |
| Database Query Time | Time to retrieve search results | <200ms |
| Page Load Time | Time to render complete page | <1 second |
| Task Completion Rate | Percentage of users completing specified tasks | >90% |
| System Usability Scale | Standardized usability measurement | Score >70 |
| Data Accuracy | Percentage of articles correctly categorized | >95% |

### 3.6.3 Scraper Performance Testing

The web scraper's performance was evaluated by measuring its ability to efficiently collect data while respecting target websites:

def test\_scraper\_performance():

"""Test the performance of the financial news scraper"""

start\_time = time.time()

process = CrawlerProcess({

'LOG\_LEVEL': 'INFO',

'STATS\_CLASS': 'scrapy.statscollectors.MemoryStatsCollector'

})

process.crawl(FinancialNewsSpider)

process.start() # Blocks until crawling is finished

end\_time = time.time()

elapsed\_time = end\_time - start\_time

# Get statistics from the crawler

stats = process.crawlers[0].stats.get\_stats()

article\_count = stats.get('item\_scraped\_count', 0)

# Calculate metrics

articles\_per\_second = article\_count / elapsed\_time

articles\_per\_hour = articles\_per\_second \* 3600

print(f"Articles scraped: {article\_count}")

print(f"Time elapsed: {elapsed\_time:.2f} seconds")

print(f"Articles per hour: {articles\_per\_hour:.2f}")

# Verify performance meets requirements

assert articles\_per\_hour > 100, "Scraper performance below target"

*Figure 19: Scraper Performance Test*

### 3.6.4 Database Performance Testing

Database performance was evaluated by measuring query execution times for common operations:

def test\_database\_performance():

"""Test the performance of database queries"""

# Define test cases

test\_cases = [

{"name": "Recent articles", "function": get\_recent\_news, "args": [20]},

{"name": "Search by keyword", "function": search\_news, "args": ["stock"]},

{"name": "Search with source filter", "function": search\_news, "args": ["market", "Wall Street Journal"]}

]

results = []

# Run each test case multiple times to get average performance

iterations = 10

for test in test\_cases:

total\_time = 0

for \_ in range(iterations):

start\_time = time.time()

test["function"](\*test["args"])

end\_time = time.time()

total\_time += (end\_time - start\_time)

avg\_time = (total\_time / iterations) \* 1000 # Convert to milliseconds

results.append({

"name": test["name"],

"avg\_time\_ms": avg\_time

})

print(f"{test['name']}: {avg\_time:.2f} ms")

# Verify performance meets requirements

assert avg\_time < 200, f"Query {test['name']} exceeds performance target"

*Figure 20: Database Performance Test*

### 3.6.5 Usability Testing

Usability testing was conducted with a sample of representative users to evaluate the application's user interface and overall experience:

#### 3.6.5.1 Task Completion Tests

Users were asked to complete typical tasks while researchers observed their interactions and recorded completion rates and difficulties encountered. The tasks included:

1. Searching for financial information about a specific company
2. Filtering search results by date and source
3. Creating a user account and logging in
4. Saving articles to favorites and retrieving them later
5. Finding and interpreting information from search results

Results were recorded in a task completion matrix that tracked success rates, time to completion, and number of errors for each participant.

*Table 12: Task Completion Results*

| **Task** | **Completion Rate** | **Avg. Time (seconds)** | **Avg. Errors** | **Observations** |
| --- | --- | --- | --- | --- |
| Search for company information | 100% | 12.3 | 0.2 | Users easily found the search function |
| Apply filters to results | 90% | 18.7 | 0.8 | Some users missed the filter options |
| Create account and login | 95% | 45.2 | 0.5 | Password requirements caused minor issues |
| Save and retrieve favorites | 90% | 28.4 | 0.7 | Users sometimes confused about favorite status |
| Interpret search results | 100% | 22.1 | 0.3 | Source attribution was clear to users |

### 3.6.5.2 System Usability Scale

After completing the tasks, participants completed the System Usability Scale (SUS) questionnaire, which provides a standardized measure of perceived usability. The SUS consists of 10 statements that participants rate on a scale from 1 (strongly disagree) to 5 (strongly agree).

The SUS statements are:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

The SUS scores were calculated using the standard method:

* For odd-numbered questions: (score - 1)
* For even-numbered questions: (5 - score)
* Sum all values and multiply by 2.5 to get a score out of 100

*Table 13: System Usability Scale Results*

| **Participant** | **SUS Score** | **Classification** |
| --- | --- | --- |
| User 1 | 82.5 | Excellent |
| User 2 | 77.5 | Good |
| User 3 | 85.0 | Excellent |
| User 4 | 72.5 | Good |
| User 5 | 80.0 | Excellent |
| Average | 79.5 | Good |

The average SUS score of 79.5 places the Investor Info application in the "Good" usability range, indicating that users found the interface intuitive and the system easy to use.

#### 3.6.5.3 Qualitative Feedback

Participants also provided qualitative feedback through post-test interviews. Key themes that emerged included:

* **Positive Feedback**:
  + "The search functionality is straightforward and quick"
  + "I appreciate being able to filter by source and date"
  + "Saving articles to favorites is useful for tracking information"
  + "The layout makes it easy to scan multiple articles quickly"
* **Areas for Improvement**:
  + "It would be helpful to have more detailed financial metrics for companies"
  + "The distinction between news and analysis could be clearer"
  + "I'd like to see notifications for new articles about companies I'm following"
  + "The mobile version could be more optimized for smaller screens"

This feedback provided valuable insights for future improvements to the application.

### 3.6.6 Data Quality Assessment

The quality of the financial data collected by the system was evaluated through several methods:

#### 3.6.6.1 Source Attribution Accuracy

A sample of 50 articles was manually checked to verify that the source attribution was correct. This involved comparing the source listed in the database with the actual source of the article.

*Table 14: Source Attribution Results*

| **Accuracy Measure** | **Result** |
| --- | --- |
| Correctly attributed sources | 48/50 (96%) |
| Incorrectly attributed sources | 2/50 (4%) |

The two errors were due to ambiguous source information on the original website, which has been addressed in an updated version of the crawler.

#### 3.6.6.2 Content Relevance Analysis

To ensure that the collected articles were relevant to financial decision-making, a content analysis was performed using keyword frequency and classification:

def analyze\_content\_relevance(sample\_size=100):

"""Analyze the relevance of collected articles to financial decision-making"""

conn = get\_db\_connection()

cursor = conn.cursor(dictionary=True)

# Get a random sample of articles

cursor.execute("""

SELECT id, title, summary

FROM financial\_news

ORDER BY RAND()

LIMIT %s

""", (sample\_size,))

articles = cursor.fetchall()

# Define relevance categories and associated keywords

relevance\_categories = {

'investment': ['stocks', 'bonds', 'invest', 'portfolio', 'dividend', 'yield'],

'market\_analysis': ['market', 'trend', 'analysis', 'forecast', 'prediction'],

'company\_performance': ['earnings', 'revenue', 'profit', 'loss', 'performance'],

'economic\_indicators': ['gdp', 'inflation', 'unemployment', 'interest rate', 'fed'],

'irrelevant': ['celebrity', 'entertainment', 'sports', 'weather', 'politics']

}

# Analyze each article

results = {category: 0 for category in relevance\_categories}

total\_relevant = 0

for article in articles:

content = f"{article['title']} {article['summary'] or ''}".lower()

max\_score = 0

best\_category = None

for category, keywords in relevance\_categories.items():

score = sum(1 for keyword in keywords if keyword in content)

if score > max\_score:

max\_score = score

best\_category = category

results[best\_category] += 1

if best\_category != 'irrelevant':

total\_relevant += 1

# Calculate percentage relevant

percent\_relevant = (total\_relevant / sample\_size) \* 100

# Print results

print(f"Content relevance analysis of {sample\_size} articles:")

for category, count in results.items():

print(f" {category}: {count} articles ({count/sample\_size\*100:.1f}%)")

print(f"Total relevant to financial decision-making: {percent\_relevant:.1f}%")

return results, percent\_relevant

*Figure 21: Content Relevance Analysis Function*

The analysis showed that 94.3% of articles were relevant to financial decision-making, with the remaining 5.7% being tangentially related or focusing on broader economic news rather than specific investment information.

### 3.6.7 Performance Under Load

To evaluate the system's performance under realistic usage conditions, load testing was conducted using Apache JMeter. The tests simulated various numbers of concurrent users performing common tasks.

*Table 15: Load Testing Results*

| **Concurrent Users** | **Average Response Time (ms)** | **90th Percentile Response Time (ms)** | **Error Rate** |
| --- | --- | --- | --- |
| 10 | 156 | 243 | 0% |
| 50 | 312 | 487 | 0% |
| 100 | 578 | 823 | 0.3% |
| 250 | 872 | 1246 | 1.1% |
| 500 | 1435 | 2187 | 3.8% |

The results indicate that the system maintains acceptable performance with up to 100 concurrent users, with response times remaining under 1 second. Above this level, response times increase significantly, and error rates begin to rise. This performance profile is sufficient for the expected user base in the initial deployment phase.

## 3.7 Limitations and Challenges

During the implementation of the Investor Info application, several limitations and challenges were encountered:

### 3.7.1 Web Scraping Limitations

The web scraping component faced several challenges:

1. **Dynamic Content**: Many financial websites load content dynamically using JavaScript, requiring additional techniques beyond basic HTTP requests.
2. **Site Structure Changes**: Financial websites occasionally update their layouts, requiring spider maintenance to adapt to these changes.
3. **Rate Limiting**: Some financial websites implement rate limiting or bot detection mechanisms that required careful tuning of request frequencies.
4. **Content Paywalls**: Premium financial content is often behind paywalls, limiting access to certain in-depth analyses.

### 3.7.2 Data Processing Challenges

Processing financial data presented several challenges:

1. **Standardization**: Different sources present similar financial information in different formats, requiring normalization.
2. **Temporal Context**: Financial information is time-sensitive, and its relevance can change rapidly with market conditions.
3. **Disambiguation**: Company names and financial terms can be ambiguous, leading to potential miscategorization.

### 3.7.3 Technical Implementation Constraints

The implementation faced several technical constraints:

1. **Database Performance**: As the volume of financial data grows, query performance could become a concern without optimization.
2. **Security Considerations**: Financial information systems require robust security measures, adding complexity to the implementation.
3. **Responsive Design**: Creating a consistent user experience across devices required additional development effort.

## 3.8 Summary

This chapter has detailed the method and implementation of the Investor Info web application, a system designed to aggregate and present financial information for investment decision-making. The development followed an agile methodology, with incremental implementation of features based on defined user requirements.

The system architecture consists of three main components:

1. A web crawler based on Scrapy that extracts financial news from trusted sources
2. A relational database that stores and organizes the collected information
3. A web application built with Flask that provides a user-friendly interface for accessing the information

The implementation process followed a structured approach, beginning with database design, followed by crawler development, and culminating in the creation of the web application. Each component was developed with attention to modularity, maintainability, and performance.

Evaluation of the system through technical testing and user studies demonstrated that the application successfully meets its core requirements, providing relevant financial information in an accessible format. The System Usability Scale score of 79.5 indicates good usability, suggesting that users can effectively interact with the system to find the information they need.

Challenges encountered during implementation included handling dynamic web content, standardizing data from multiple sources, and ensuring responsive performance under varying user loads. These challenges inform recommendations for future development, which will be discussed in later chapters.

# Bibliography

Ab, S., & Bartholomew, D. (2012). MariaDB vs. MySQL. Retrieved from https://rozero.webcindario.com/disciplinas/fbmg/abd3/MariaDB\_vs\_MySQL.pdf

Agnew, J., & Szykman, L. (2005). Asset allocation and information overload: The influence of information display, asset choice, and investor experience. *The Journal of Behavioral Finance, 6*(2), 57-70.

Barber, B., & Odean, T. (2008). All that glitters: The effect of attention and news on the buying behavior of individual and institutional investors. *The Review of Financial Studies, 21*(2), 785-818.

Bassil, Y. (2012). A Comparative Study on the Performance of the Top DBMS Systems. *Journal of Computer Science & Research, 1*(1), 20-31.

Bayer, M. (2012). SQLAlchemy. In *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks.* aosabook.org.

Benartzi, S., & Thaler, R. (2007). Heuristics and biases in retirement savings behavior. *Journal of Economic Perspectives, 21*(3), 81-104.

Bhattacharya, U., Hackethal, A., Kaesler, S., Loos, B., & Meyer, S. (2012). Is unbiased financial advice to retail investors sufficient? Answers from a large field study. *The Review of Financial Studies, 25*(4), 975-1032.

Biryukov, A., Dinu, D., & Khovratovich, D. (2016). Argon2: New generation of memory-hard functions for password hashing and other applications. *IEEE European Symposium on Security and Privacy (EuroS&P)*, (pp. 292-302).

Chen, H., Chiang, R., & Storey, V. (2012). Business intelligence and analytics: From big data to big impact. *MIS Quarterly, 36*(4), 1165-1188.

Chen, J., Tao, Y., Wang, H., & Chen, T. (2015). Big data based fraud risk management at Alibaba. *The Journal of Finance and Data Science, 1*(1), 1-10.

Choi, J., Laibson, D., & Madrian, B. (2010). Why does the law of one price fail? An experiment on index mutual funds. *The Review of Financial Studies, 23*(4), 1405-1432.

Copyright and "factual" compilations. (2015). *Harvard Law Review, 128*(5), 1167-1188.

Davoudian, A., Chen, L., & Liu, M. (2018). A survey on NoSQL stores. *ACM Computing Surveys, 51*(2), 1-43.