System Requirements Specification Index

For

Investment Search Application

Version 1.0

IIHT Pvt. Ltd.
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

Invoice Management

System Requirements Specification

1. BUSINESS-REQUIREMENT:

1.1 PROBLEM STATEMENT:

Invoice Management Application is .Net Core web API 3.1 application integrated with MS SQL Server, where it refers to Investors keen on making informed decisions in the stock market often face challenges in accessing comprehensive financial insights about companies listed in stock exchanges such as NSE (National Stock Exchange) and BSE (Bombay Stock Exchange). To address this, an Investment Search application needs to be developed to provide investors with easy access to essential financial data and analysis of listed companies.

1.2 FOLLOWING IS THE REQUIREMENT SPECIFICATION:

	Investment Search Application
Modules	
1	User
2	Stock
Functionalities	
1	User Login
2	Search Stock By Name
3	Get Stock Details By ID

2. ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 Investment Search Constraints:

- If Stock Id does not exist then the operation should throw a custom exception.
- While fetching the Stock details by id, if Stock id does not exist then the operation should throw a custom exception.

2.2 Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in model classes only.
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3. BUSINESS VALIDATIONS

3.1 User Class Entities (UserViewModel)

- id (int, primary key): Unique identifier for the user.
- email (varchar): Email address of the user. (example@kanini.com)
- password (varchar): password of the user. (Kanini@123)

3.2 Stock Class Entities

- id (int, primary key): Unique identifier for the stock.
- name (varchar): Name of the company.
- current_trading_price (decimal): Current trading price of the stock.
- closing_price (decimal): Closing price of the stock from the previous trading day.
- today_high (decimal): Highest price of the stock reached during the current trading day.
- today_low (decimal): Lowest price of the stock reached during the current trading day.
- fifty_two_week_low (decimal): Lowest price of the stock in the past fifty-two weeks.
- fifty_two_week_high (decimal): Highest price of the stock in the past fifty-two weeks.
- face_value (decimal): Face value of the stock.
- number_of_shares (int): Total number of shares.
- market_capitalization (decimal): Market capitalization of the company.
- cash (decimal): Cash reserves of the company.
- debt (decimal): Debt of the company.
- enterprise_value (decimal): Enterprise value of the company.
- dividend_yield (decimal): Dividend yield of the stock.

3.3 SWOT Analysis Class Entities

- stock_id (int, foreign key): Foreign key referencing the stock table.
- strengths (varchar[]): Array of strengths of the company.

- weaknesses (varchar[]): Array of weaknesses of the company.
- opportunities (varchar[]): Array of opportunities for the company.
- threats (varchar[]): Array of threats faced by the company.

3.4 Rating Class Entities

- stock_id (int, foreign key): Foreign key referencing the stock table.
- overall_rating (decimal): Overall rating of the company.
- ownership_reviews_count (int): Count of reviews for ownership aspect.
- valuation_reviews_count (int): Count of reviews for valuation aspect.
- efficiency_reviews_count (int): Count of reviews for efficiency aspect.
- financials_reviews_count (int): Count of reviews for financials aspect.

4. CONSIDERATIONS

- You can perform the following possible actions

User, Stock

5. REST ENDPOINTS

Rest End-points to be exposed in the controller along with method details for the same to be created

5.1 InvestmentSearchController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/api/auth/login
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>UserViewModel model</td></tr><tr><td>Return</td><td>HTTP Response StatusCode</td></tr></table></td><td>User Login</td></tr>
<tr><td>/api/stocks/search
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>String StockName</td></tr><tr><td>Return</td><td>List<Stock></td></tr></table></td><td>Search Stock By Name</td></tr>
<tr><td>/api/stocks/details</td><td></td></tr>
</table>

Http Method	GET		Get Stock Details BY ID
Parameter 1	Int StockId		
Return	Stock		

6. TEMPLATE CODE STRUCTURE

6.1 Package: InvestmentSerachApplication

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	InvestmentSearchControlle r	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL server Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

6.2 Package: InvestmentSearchApplication.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IInvestmentSearchServices interface	Inside all these interface files contains all business validation logic functions.	Already implemented

Service	InvestmentSearchServices CS file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially implemented
Repository	IInvestmentSearch Repository InvestmentSearch Repository (CS files and interfaces)	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially implemented
ViewModels	UserViewModel	Contain all view Domain entities for show and bind data. All the business validations must be implemented.	Partially implemented

6.3 Package: InvestmentSearchApplication.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	InvestmentSearchDBContext cs file	All database Connection,collection setting class	Already Implemented

6.4 PackageInvestmentSearchApplication.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	User ,Stock,Rating,SWOT Analysis (CS files)	All Entities/Domain attribute are used for pass the data in controller and status entity to return response Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. Generate the **Id** using the **IDENTITY** strategy	Partially implemented

7. EXECUTION STEPS TO FOLLOW

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**

2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) Terminal → New Terminal.**

3. **On command prompt, cd into your project folder (cd <Your-Project-folder>).**

4. **To connect SQL server from terminal:**
   (InvestmentSearchApplication /sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**

     1> Create Database InvestmentDb
     2> Go

5. **Steps to Apply Migration(Code first approach):**
   - **Press Ctrl+C to get back to command prompt**
   - **Run following command to apply migration-**
     (InvestmentSearchApplication /dotnet-ef database update)

6. **To check whether migrations are applied from terminal:**
   (InvestmentSearchApplication /sqlcmd -S localhost -U sa -P pass@word1)

   1> Use InvestmentDb
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go

7. **To build your project use command:**
   (InvestmentSearchApplication /dotnet build)

8. **To launch your application, Run the following command to run the application:**
   (InvestmentSearchApplication /dotnet run)

9. **This editor Auto Saves the code.**

10. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.**

11. **To test web-based applications on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

    Note: The application will not run in the local browser

12. **To run the test cases in CMD, Run the following command to test the application: (InvestmentSearchApplication.Tests/dotnet test --logger "console;verbosity=detailed")**
    (You can run this command multiple times to identify the test case status,and refactor code to make maximum test cases passed before final submission)

13. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B - command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**

14. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**

15. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**