Southport Technology Group

Investor Market Data API

Technical Due Diligence

Scoring Range

5.x
Exemplary

4.x
Above Average

3.x
Needs Work

2.x
Critical Issues

1.x
Risks

0.x
Severe Risks

Final Score

2.4

Despite reasonably high code quality, IMD has critical deficits in tests, developer operations, and infrastructure. Each to be explored in the detailed sections.

Categories

01. Code

02. Tests

03. Source & Project Management

04. Developer Operations

05. Infrastructure

01. Code

4.0

The code section of our review was the strongest for IMD. The project is utilizing a modern framework, built on community standards, and has few legacy dependencies. We believe maintenance interventions in the future will be limited, given the initial care placed into the code separation and architecture.

The Chrome plugin is the weaker of the two sides of the codebase. The standards used are up to documentation and community guidelines. The CSS framework is a respected CSS-only tool, which limits cross dependencies and long-term maintenance issues. The weakness comes from the low use of developer tooling, which make it harder to write tests and enforce maintenance standards. This is a much smaller portion of the codebase, so it is not a point for serious alarm. Key areas of concern:

1. Lack of specific error handling for customer awareness.
2. Limited support for future maintenance via robust developer tooling.

01. Code (continued)

4.0

Because the majority of the codebase is housed on the API side of the project, this has a strong effect on the rating. We believe the developers performed a good service pulling the business logic into the API and making the Chrome plugin very limited in its business rules. The API utilizes modern tooling. File size is reasonable. The objects in the API are thoroughly RESTful. This is easily the brightest spot in the due diligence for Investor Market Data. Some key areas of strength:

1. True RESTful service architecture.
2. Good functional abstraction and file sizes (by lines).
3. Strong error handling and understanding of potential edge cases.

02. Tests

0.2

This was our largest area for concern. The client and API have no kind of automated tests developed. Given the complete absence of this support, we conclude that post acquisition quality assurance will be difficult. The only bright spot here was reasonable code formatting standards. This is good, but not a stand-in for a robust automated testing plan. Some key areas for concern:

1. No automated tests to speak of.
2. No manual test plans or outlines.
3. Supported environments and devices completely un-documented.

Because of the absence of these standards, quality control during the transition is a heavy risk. One area where this will be even harder is given the weaker developer tooling on the plugin side, it will be much harder to implement automated tests. **We strongly recommend requiring the seller to complete manual test plans and outlines before the sale.**

A future (but not present day) bright spot is that a well organized API is easier to test than a well organized client codebase. Referring back to the code section, this is a much needed intervention that will be faster due to the high standards used in the API code.

03. Source & Project Management

3.3

This was the second strongest portion of our analysis (behind code). The project is managed using modern standards. The quality and depth of issues in the project management system could improve. All in all, we don't see significant risk in this part of the project. Some key areas for concern:

1. Code review has not been a big part of the practice, due to a small team.
2. Long-term feature roadmap is not well defined.
3. Production environment configurations are not confidential (i.e. the entire engineering team has access).

Point #3 is a risk that can be easily mitigated, but should be immediately upon acquisition. Point #1 is an opportunity for further code review from the Southport Technology Group team. With a line by line review, new road map items will be created in the maintenance category.

For the long-term roadmap (#2), you should understand what the seller has considered for new features. Furthermore, you should work with your own product team to create and plan out a 6 month roadmap for the product. Unfortunately, this is not a 'set it and forget it' opportunity. The roadmap is not 'set', so that will be up to they acquiring team.

04. Developer Operations

2.3

Another weak area in our analysis. Local development experience is positive. However, the lack of continuous integration and continuous deployment is the big red flag. Manual deployments are error prone, and belies the lack of tests in the other part of our analysis. Some key areas for concern:

1. No continuous or automated deployment.
2. Documentation is lacking. Documentation is especially lack for the Chrome plugin.
3. Developers have no mock services, and have to test against a local API. This requires greater data management overhead.

These are immediate and highly recommend interventions post acquisition. You will need to budget 1-3 months in your product roadmap to get these aspects in place. While they provide some value on the upside, they are better understood and key de-risking approaches to your development process.

This is an area for potential negotiation on the sales price, as it's real labor that has to go into stability before any new features. Given the aforementioned low quality of testing automation, there is a theme developing around quality control. You and the seller should work hard to develop a quality control plan before handoff and this should be at the top of the queue for your team in the first 3 months post acquisition.

04. Infrastructure

2.3

Scoring the same as developer operations, infrastructure was subject to similar issues. Builds and deployments are done in the cloud and with a competitive cloud provider. We see this as a positive. Conversely, there is no external logging, database backups, or container backups. Some key areas for concern:

1. No external logging, machine monitoring, or error notifications.
2. Containers and database are not backed up automatically.
3. Caching has a mediocre, interim solution. Should be replaced long-term.

These choices mirror the ‘move fast and break things’ nature of bootstrapped startup culture. These interventions are not terribly difficult and fall into the realm of good developer hygiene. The lack of visibility on these issues could have ramifications in customer satisfaction. For instance, errors may be happening and it’s entirely unknown to the current team. This can cause serious issues with customer churn.

Our recommendation is to get these in place ASAP post acquisition. We also recommend working with the seller to assess the qualitative understanding of the customer base. If the seller believes the customers are happy, find some evidence for that assertion.

Southport Technology Group

southporttechnologygroup.com

Contact Info
solutions@southporttechnologygroup.com

Appendix

A. **Scoring Methodology Overview**

B. **Sample Scoring Rubric**

A. Scoring Methodology Overview

X.X

Each category is scored from a rubric of 15–20 dimensions, which are weighted by importance. A score for the category is then determined. The final score for the code base is then determined by taking the average of the five category scores. A full due diligence report includes the score and comments for each dimension of each category.

B. Sample Scoring Rubric (Note: Preview + bottom ⅔ blurred)

CODE	SCORE	WEIGHT	WEIGHTED SCORE	COMMENTS
Recognizable framework.	3	2.00	6.00	Front-end is fine, but does not use a recognizable framework to pull it together.
Modern runtime.	5	2.00	10.00	All JavaScript.
Modern library versions.	5	1.00	5.00	Everything has been updated in the past year.
Reasonable file line length.	5	0.30	1.50	Nothing over 400 lines. Most are 300 lines or less.
Community syntax & conventions.	3	2.00	6.00	Service is very good. Client is a bit more old-fashioned on this point.
API consistency.	5	1.20	6.00	API is consistent.
UI consistency.	5	1.20	6.00	UI is consistent. While the implementation is strange, it behaves well.
Configuration separated from code.	3	0.80	2.40	On the server side it is. On the client, it is not.
Consistent, clear comments.	4	0.80	3.20	
Separation of concerns.	4	1.50	6.00	Service does this better than the front-end. Functions are making service calls and rendering. This may be due to the lack of a framework.
Folder structure representative of application structure.	5	1.00	5.00	
DRY code.	4	0.80	3.20	Service could have done a little better abstracting some of the common functions.
Appropriate errors thrown.	3	1.20		Service is very strong with this. Front-end, less so.
Appropriate errors handled.	2	1.20	2.40	Front-end just waits forever to load. It does not completely blow up, but this is a definite area for improvement.
Sensible variable and function names.	5	0.80	4.00	