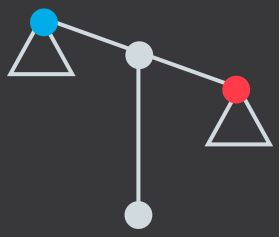


A PRACTICAL APPROACH

Evaluating a framework: A guide for business leaders

vaadin

Introduction

Vaadin's services division has consulted on and supported the development of 700+ Vaadin enterprise web applications over 20 years. Our involvement has ranged from targeted UX consulting and development support, to complete turnkey development projects. This whitepaper on how to evaluate a framework is based on our extensive experience on the subject.

To learn about our development, UX and mentorship consulting services, please visit: vaadin.com/consulting.

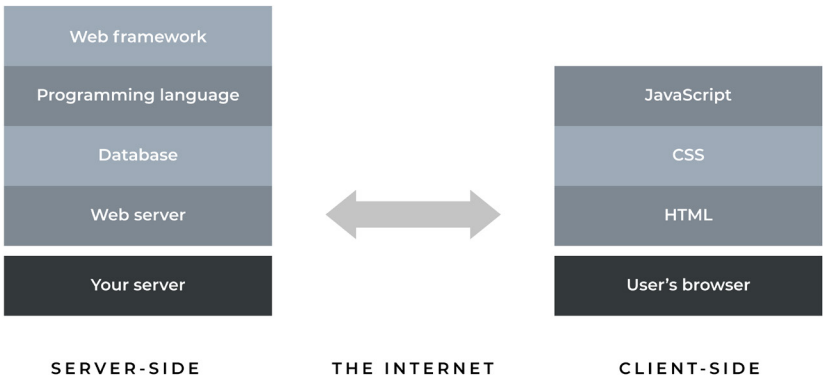
The Vaadin platform is a modern, open-source Java web framework that allows you to build UX-first web applications that run on any platform. Vaadin applications are secure by default, handle complex enterprise use-cases with ease and provide the stellar user experience your customers and staff deserve.

This whitepaper is structured as follows:

The role of a framework in a technology stack

An introduction to tech stacks

A technology stack is an umbrella term for all the technologies that are used to build and deploy your application. In the broadest sense of the term, a stack can include everything from the programming languages, supporting frameworks, databases and other third-party services, to web servers and operating systems.



The term "stack" comes from the tendency of technologies to stack, one layer on top of the previous one.

In the interest of clarity, this whitepaper adopts a narrower definition of a stack, and includes only:

- Programming languages
- Databases
- Web frameworks
- 3rd-party tools, libraries and dependencies.

It's important to remember that there is no objective "best" stack. No technology is a silver bullet. There are pros and cons to every option, and some will better suit your project, team and experience than others. Your success depends on finding the **right** combination of technologies to build a suitable stack for **your** organization, use-case and team.

Not all applications will use every part of the stack either. The final stack depends on the complexity and functionality of the application in question.

We commonly consult on projects that use the following Java + Vaadin stack:

Language: Java, with some CSS for additional styling
Framework(s): Vaadin + Spring
Database: MySQL

What is a framework, and do you need one?

It is possible to build a web application using only programming languages, such as HTML, CSS and JavaScript--no framework is necessary.

However, as your application increases in complexity, so too will the scale and complexity of your codebase. Writing every single bit of functionality from scratch, typically results in an excess of unmaintainable **spaghetti code,** not to mention an inordinately-long development time and high costs.

Frameworks offer a solution to this problem: they abstract away and automate many of the core functionalities that are common to a majority of web applications. This allows developers to better focus on building features that bring value to users, while keeping the design and code of your project (or projects) consistent and reusable.

In addition, writing vanilla code without a framework can leave you susceptible to obsolescence caused by changing perceptions and understanding of the project's domain. This can eventually render the architecture of an application obsolete and no longer suited to its purpose. Frameworks mitigate this risk, as they evolve and remain adaptable through high usage and adoption.

Every application consists of two parts: the frontend (client side) and the backend (server side).

- The **frontend** comprises the UI and visual elements that allows users to interact with your application. The frontend portion of a web app runs in the user's browser. Some web apps can also run application logic for things like authentication on the frontend.

- The **backend** runs on a server and comprises the database, business logic and application logic of an application. The backend portion of your application provides your UI with data to display, processes and stores user input, accesses and persists data using databases and similar functions.

Frameworks are categorized as client-side, server-side or full-stack. This is based on **where they run**, not necessarily on the functionality they provide. For example, Angular and React are both client-side UI frameworks. However, Vaadin is a full-stack framework, as it runs on both the frontend and the backend. In addition, it includes functionality for securely automating the connection between the UI and the backend.

The need for an objective approach to framework evaluation

If you are a business leader with a busy schedule and non-technical background, it may be tempting to leave this decision solely to your development team. However, this approach can result in significant drawbacks and consequences for your business in the long run.

From a developer's perspective, it makes sense to recommend a framework (and stack) based on their proficiencies, interest level and the stack's technical merits alone. This means that your organizational and business needs may likely go unaddressed.

In addition, open-source frameworks without vendor backing are prone to deprecation and abandonment, should the community lose interest.

Potential issues resulting from an incorrectly-chosen framework (and stack) include:

Siloing

Siloing refers to a situation where different projects, or even parts of a single project, use completely different technology stacks. This is a common issue faced by larger enterprises and results from:

- Corporate acquisitions that use disparate technology stacks.
- New projects extending older projects from a different era that use disparate technology stacks and require separate, dedicated teams.
- The lack of a technology strategy or decentralized technology decision making.

When siloing occurs:

- Maintenance, updates and migrations become complex and costly affairs.
- Your business may need to retain multiple teams for each project, as developer knowledge is not always interchangeable between stacks.
- It reduces resourcing flexibility, forcing you to hire new talent, rather than to shift team members from one project to another.

Technical debt

Technical debt is the cost of choosing to build an application using a quick solution or hack, instead of taking the time to properly develop your application in a scalable manner. Your project may get to market sooner (and at a lower initial cost), but the resulting application will be more fragile and difficult to develop further, without major refactoring and fixes to the codebase.

It is normal for every development project to accumulate some level of technical debt. However, a rushed application, built on an unsuitable technology stack will result in a far greater debt.

It is therefore imperative that both business and technical stakeholders work together to carefully outline their requirements, select the appropriate technology stack and develop the application properly, even if it takes a bit more time, money and effort to get to market initially.

Untenable maintenance overhead

Selecting a framework that doesn't adequately meet your business needs can lead to significantly higher maintenance overhead and long-term risk to your business. This can happen as a result of:

- The framework being deprecated down the road.
- Drastic rewrites or changes to the framework causing breaking changes that necessitate major updates to your code.
- A poorly-evaluated technology stack not meeting the future requirements of the project or organization.

To avoid unmanageable maintenance, it is vital that you, as an executive or business leader, get involved in the evaluation process to complement your development team's hands-on experience with your critical, vision-level thinking and thorough understanding of your business needs.

Evaluating a framework

Prerequisites for using this evaluation methodology

This whitepaper assumes that your organization:

- Has decided to build a web application.
- Has determined the project parameters and KPIs.
- Has defined the project scope.
- Has mapped out the functional requirements of the project.
- Is evaluating a technology stack to be used with a web framework for your project

Methodology of the evaluation

We have presented an evaluation framework consisting of a set of generalized evaluation criteria that are relevant for a majority of development projects.

For each framework:

Assign a weight (out of 5) to each of the evaluation criteria, based on its importance to your project. In order to completely exclude a criteria from the score, simply assign it a score of “0”, or delete it from the matrix.

- In order to have an objectively comparable analysis, it is important that each framework under consideration is evaluated using the same set of criteria, which are assigned the same weights.
- The sample evaluation matrix below includes guidelines for assigning weights for each criterion.
- There is an associated Google Sheets template available in the “Evaluation templates” section of this whitepaper. You can click on the link, duplicate it and use it to conduct your evaluation. The template includes step-by-step instructions to guide you through evaluating a single framework, along with ready-made formulae for calculating the weighted averages and scores. Simply duplicate and repeat the process in the template for each framework under consideration.

Have at least 3 developers in your organization individually evaluate each framework under consideration.

- The evaluations should be conducted independently and without discussion to ensure their objectivity.
- The evaluations should be reviewed with managerial stakeholders to ensure that business concerns are properly and fully addressed.

Tally up the scores of each evaluator and calculate the score for each category.

Calculate the weighted average by:

- Multiplying each category's final score (total of all 3 evaluators) by the assigned weight. This gives you **the weighted criterion total**.
- Tallying up all the **weighted criterion totals** of each criterion.

We recommend investing in a comprehensive proof of concept (POC), especially for large and mission critical applications. This will ensure that the assessments are indeed based on fact and not personal opinion or bias. Vaadin provided POC services for Vaadin application development projects. For more information, visit: https://www.vaadin.com/consulting

Functional requirements

The framework you select must first and foremost cater to your project's functional requirements.

In software-development parlance, functional requirements define the functional elements needed in the application. For example, send an email when a purchase is made, record the purchase in a database, and so on.

You must ensure that your framework (and stack) of choice has the capability to build an application that includes these tasks.

This aspect of the evaluation process is primarily conducted by your development team, but should be based on specifications that you have already agreed on with them.

Non-functional requirements

Non-functional requirements comprise any that aren't explicitly concerned with the functionality of the application. These considerations are nonetheless critical to the success of your project, as they include important elements, such as accessibility, scalability, maintainability, business needs and costs.

The evaluation matrix that follows details how to evaluate non-functional requirements.

A sample evaluation matrix

Guidelines for scoring: This column includes a set of guidelines (questions and suggestions) to help your evaluators determine a framework's suitability for each criterion.

Guidelines for assigning weight based on importance: This column maps each criterion to a business benefit, making it easier for you to decide if the criterion is important to your project, team and organization.

Evaluation criteria	Guidelines for scoring (Recommended maximum score: 5)	Guidelines for assigning weight based on importance (5: most important / 1: least important)
Complexity of the solution	1. The complexity of the solution depends on how many technologies must be individually selected, evaluated and maintained as part of the stack. A less complex solution is always better.	Less complexity results in: • lower risk of deprecation • easier updates and maintenance • quicker time to market
Easy solution to integrate UI with the backend	1. Does the framework provide an easy option for connecting an app's UI to the backend?	Easy, automated method to connect UI to backend results in: • quicker time to market • higher security
Support for target devices and browsers	1. Can apps built with the framework support all necessary devices and browsers? 2. Are separate applications necessary to support different browsers and devices?	Lower application count results in: • quicker time to market • lower costs • easier maintenance
Component-based UI development model	1. Does the framework build UIs in a component-based manner? We recommend building component-based UIs as they are Modular Reusable.	Component-based UIs result in: • quicker time to market • ease of building and updating complex UIs
Performance considerations	1. Will the UI of the developed app have any noticeable input lag? 2. Will the developed app be capable of supporting the expected number of concurrent users (within your budget)? 3. Does your use-case allow for constant connectivity, or are robust offline capabilities required? The architecture of the framework will dictate whether or not your application can function offline	Performance considerations affect: • technical viability • maintenance overhead • user adoption
Security considerations	1. Does the framework integrate seamlessly with common security frameworks and tools? 2. Does the architecture of the framework enforce security by preventing common attacks (such as XSS) and helping developers avoid mistakes?	Framework's in-built security capabilities affect: • technical viability • legal risk • long-term risk • user adoption and trust
Enables rapid UI development	1. Does the framework ship with the required tools, features and first-party component library to help speed up UI building?	Speed of UI development affects: • time to market • initial development costs

Accessible UI building capabilities	1. Does the framework support easily building accessibility-compliant UIs out of the box? 2. How much extra work / steps are needed to make UIs accessible?	Ease of building accessible UIs result in: • wider user adoption • minimizing legal risk • quicker time to market
Type of licenses needed	1. Is the framework: Commercial, closed source: Reliable, but more expensive and vendor lock-in is possible. Community maintained, open-source Flexible and less expensive, but longevity and reliability are not assured. Vendor maintained, open source: Flexibility, longevity and reliability is assured at a lower cost. We recommend the 3rd option, however, this ultimately depends on your organization and legal team's needs and requirements.	The type of license affects: • flexibility • project costs • longevity • long-term legal risk
Extent of vendor and community support	1. Does the vendor provide any support options? 2. How easy is it to get support from the developer community? 3. Is there a public or vendor forum? 4. What other communication channels are available for support?	More support results in: • quicker time to market • increased longevity • decreased legal risk
Forced update frequency to major versions with breaking changes	1. How frequently must you update to major versions to maintain access to security patches and bug-fixes? 2. How extensive (and expensive) is it to update your custom code when updates include breaking changes?	Lower update frequency results in: • more resources for feature development • greater stability for complex apps • less disruption to users
Ease of updating to new versions of the framework	1. Does the community / vendor make it easy to stay current with underlying technologies by providing adequate documentation, tools and similar add-ons? 2. Does the vendor ensure compatibility of underlying technologies, or must they be updated separately?	Frameworks and stacks that are easy to update result in: • lower maintenance overhead • less disruption to users
Learning curve	1. Does the framework have a lot of framework-specific concepts for developers to master? 2. How many languages are required to build or debug the complete application?	An easier learning curve positively affects: • hiring and onboarding costs • available talent pool
Ease of finding qualified talent / consultants to assist with the project	1. How easy is it to find qualified developers and consultants to assist on the project, if needed?	Ease of finding qualified talent / consultants affects: • time to market • long-term risk • hiring and development costs
Framework maturity and longevity	1. Has the framework been used successfully in enterprise projects of a similar scale? 2. Is the framework's longevity assured for your project's estimated lifecycle? To answer this question, consider: License type Project status in vendor organization Vendor and community's commitment to further development Size of the enterprise user base	A more mature, stable and vendor-prioritized framework results in: • lower long-term risk • greater longevity of project • proof of technical viability

Baseline example using Vaadin

The following example provides the information regarding Vaadin required by your evaluators to answer the questions posed in the **“guidelines for scoring”** column of the sample evaluation matrix.

Category	Comments
Complexity of the solution	Full-stack framework **Commonly used stack:** **Frameworks:** Vaadin, Spring **UI + backend connection:** Vaadin **Database:** Multiple options (IBM D2, MYSQL, PostgreSQL are commonly used) **Additional:** Spring Boot
Easy solution to integrate UI with the backend	Available. Vaadin automates the connection between UI and backend in a secure and type-safe manner.
Support for your target devices and browsers	Vaadin helps you build PWAs (progressive web applications) that: • Run on most platforms. • Perform like native applications. • Can be installed directly from a browser.
Component-based UI development model	Yes. Vaadin allows you to build UIs with reusable, standards-based components.
Performance considerations	Vaadin applications run on the server, requiring each user interaction to complete a round trip to the server for processing. The impact on performance is negligible when internet connectivity is good. Vaadin supports approximately 10,000 concurrent users on a typical enterprise-level server.
Security considerations	Vaadin supports seamless integrations with many popular security frameworks and tools, such as Spring Security and oAuth. Vaadin applications are architecturally secure because all your code runs on the server, and is not exposed to the user’s browser. Vaadin applications have been penetration tested by our customers with flawless results.
Enables rapid UI development	Vaadin lets you easily build complex UIs with great designs. It ships with: • UI component library of 45+ components • Easily-customizable Lumo theme • Visual UI builder (that does not require the knowledge of HTML / CSS) • TestBench to speed up testing
Accessible UI building capabilities	Yes. All Vaadin UI components support WAI-ARIA accessibility standards out of the box. Vaadin also provides comprehensive documentation on accessibility at: https://vaadin.com/accessibility
Type of license	Vaadin is vendor maintained and open source, released under the permissive Apache 2.0 license. It’s free for commercial usage as well. Vaadin Ltd. ensures compatibility of underlying technologies, and provides regular updates and extensive support.

Extent of vendor / community support	The vendor offers extensive support services including: • Comprehensive warranty • Comprehensive legal indemnification • 24/7 expert support • On-site support and consulting • Mentorship • Development services Vaadin's own team (and community) are active on most public developer forums. Vendor also offers its own forum for Vaadin users to seek help.
Forced update frequency to major versions with breaking changes	Vaadin releases a new major LTS version every 2 years; forced updates are every 5 years, but can be delayed for up to 15 years (enterprise subscribers). This allows your team to focus on building features and update your application when most convenient.
Ease of staying current with underlying and dependent technologies	Vaadin provides detailed documentation, compatibility tooling and dedicated migration / update consultancy services to assist with migrations.
Learning curve	Easy. Vaadin does not have many framework-specific concepts to master for Java developers. **Languages required to build an app:** Java (required), CSS and HTML (optional for UI styling).
Ease of finding qualified talent / consultants to assist with projects	Moderate. Vaadin's developer base is smaller than the larger frameworks (Angular or React), however: Most Java developers can easily learn Vaadin. Vendor offers a job placement service. Vendor offers competitive consultancy / development services.
Framework maturity and longevity	Vaadin has been used in complex enterprise projects for over 20 years. Vaadin is the flagship product of the vendor; continued development is assured. The framework is also open-source, with optional free and paid support + tools, mitigating long-term risk. Vaadin is used by 40% of Fortune-100 companies, and 1000+ other businesses in a wide range of verticals. Please visit vaadin.com/customer-stories or contact sales for more information.

Automated evaluation spreadsheet template

Our spreadsheet evaluation template comes with built-in formulae and functionality to automate the evaluation and data entry processes. Your evaluation team simply enter the weight for each criteria, and their scores into the spreadsheet, and the formulae will take care of everything else, giving you a final weighted average for the framework. Simply click below to copy the template and get started.

Please note: Each copy of the template evaluates a single framework. You will need separate copies for each framework under evaluation.

Detailed instructions are provided within the template.

https://docs.google.com/spreadsheets/d/1cc_SbCk1QrjD_0TdNUT-ijuHGMxryNzNpyIZfpX0uTU/edit?usp=sharing

Assigning weight by importance to each evaluation criterion

You can follow the guidelines provided in the associated whitepaper to assign a weight for each criterion, based on its relevance and importance to your project, team and organization.

Category	Weight based on importance (out of 5)
Complexity of the solution	
Easy solution to integrate UI with the backend	
Support for target devices and browsers	
Component-based UI development model	
Performance considerations	
Security considerations	
Enables rapid UI development	
Accessible UI building capabilities	
Type of licenses needed	
Extent of vendor and community support	
Forced update frequency to major versions with breaking changes	
Ease of updating to new versions of the framework	
Learning curve	
Ease of finding qualified talent / consultants to assist with the project	
Framework maturity and longevity	
Total	
Average score[(Total 1 + Total 2 + Total 3) / 3]	

Individual evaluator scoring sheet for a single framework

Evaluator's name:

Framework being evaluated:

Category	Score
Complexity of the solution	
Easy solution to integrate UI with the backend	
Support for target devices and browsers	
Component-based UI development model	
Performance considerations	
Security considerations	
Enables rapid UI development	
Accessible UI building capabilities	
Type of licenses needed	
Extent of vendor and community support	
Forced update frequency to major versions with breaking changes	
Ease of updating to new versions of the framework	
Learning curve	
Ease of finding qualified talent / consultants to assist with the project	
Framework maturity and longevity	
Total	
Average score[(Total 1 + Total 2 + Total 3) / 3]	

Weighted average calculation sheet

Evaluator's name:

Framework being evaluated:

Category	Total score of all 3 evaluators	Weight	Weighted score (Total score of all 3 evaluators * weight)
Complexity of the solution			
Easy solution to integrate UI with the backend			
Support for target devices and browsers			
Component-based UI development model			
Performance considerations			
Security considerations			
Enables rapid UI development			
Accessible UI building capabilities			
Type of licenses needed			
Extent of vendor and community support			
Forced update frequency to major versions with breaking changes			
Ease of updating to new versions of the framework			
Learning curve			
Ease of finding qualified talent / consultants to assist with the project			
Framework maturity and longevity			
Total			
Weighted average score of framework (Total of weighted score / Total weight)			

Repeat this process for each framework under consideration, and compare the weighted average scores of each framework.