Full Stack is Not What It Used to Be

Antero Taivalsaari[1,4], Tommi Mikkonen[(✉)2], Cesare Pautasso[3], Kari Systä[4]

[1] Nokia Bell Labs, Tampere, Finland
antero.taivalsaari@nokia-bell-labs.com
[2] University of Helsinki, Helsinki, Finland
tommi.mikkonen@helsinki.fi
[3] USI, Lugano, Switzerland
cesare.pautasso@usi.ch
[4] Tampere University, Tampere, Finland
kari.systa@tuni.fi

Abstract. The traditional definition of *full stack development* refers to a skill set that is required for writing software both for the frontend and backend of a web application or site. In recent years, the scope of full stack development has expanded significantly, though. Today, a full stack software developer is assumed to master various additional areas especially related to cloud infrastructure and deployment, message brokers and data analytics technologies. In addition, the emergence of Internet of Things (IoT) and the rapidly spreading use of AI/ML technologies are introducing additional skill set requirements. In this paper, we discuss the expectations for a modern full stack developer based on our industry observations, and argue that these expectations have significant implications for software and web engineering education.
Keywords: Education, Software Engineering, Web Engineering, Software Architecture, Cloud, Internet of Things, IoT, Programmable World

1 Introduction

According to the traditional definition, the term *full stack developer* refers to a web engineer or developer who works with both the frontend and backend of a website or a web application. This means that a full stack developer is typically expected to participate in projects that involve not only the user facing features of web applications, but also databases and other server-side components that are used for storing and delivering the contents of a web site. Full stack software developers are commonly also expected to work with customers during the planning and design phases of projects.

The "classic" skill set of a full stack developer includes the following [12]:

- HTML, CSS and JavaScript,
- one or more popular web frameworks such as Angular, React or Vue,
- experience with databases,
- experience with version control systems (at least Git),
- knowledge of web design, visual design and user experience best practices,

– knowledge of web security challenges and security best practices,
– experience with web server installation, configuration and web traffic log analytics, and
– some knowledge of additional programming languages that are commonly used in web backend development such as Python, PHP and Ruby (more recently also Go and Scala).

In general, full stack web developer job listings typically include a mix of frontend and backend skills, covering just about everything that it takes to compose a running application or a web site.

In recent years, the scope of full stack development has expanded considerably, though. In this short paper we present our observations based on various industry projects as well as discussions with our students and colleagues both in the academia and in the industry. We argue that the requirements presented by employers for full stack web engineers have grown significantly in recent years, and that these expanded expectations are not yet really taken into account in software engineering and web engineering education.

2 Software as a Service and the Disappearance of the IT Department

> "Dear Recruiters, That's
> not a Full Stack Developer.
> That's an IT Department."
>
> – Attributed to *Giulio Carrara*

The widespread adoption of the World Wide Web has fundamentally changed the landscape of software development. In the past 10-15 years, the Web has become the *de facto* deployment environment for new software systems and applications. Today, the majority of new software applications used on desktop computers or laptops are written for the Web, instead of conventional computing architectures, specific types of CPUs or operating systems.

From its relatively humble origins at Salesforce.com and later at Amazon.com, Software as a Service evolved into the dominant form of computing, effectively displacing traditional, locally installed software applications and conventional binary "shrink wrap" software [16]. As a side effect, the centralization of software onto externally hosted cloud platforms and virtual machines gradually killed the IT departments that nearly all major companies used to have. Nowadays, based on our observations and discussions with various companies, even in relatively large organizations there may be just one person who is responsible for all aspects related to system management, including deployment and operation of a large number of virtual machines rented from external providers.

As a result of this transition, many of the tasks that were traditionally handled by IT departments are now expected to be part of the job description of software developers themselves. The disappearance of the IT department was

amplified by the *DevOps* movement [5] that shifted the majority of software deployment tasks from the IT department to the developers. In recent years, the transition towards *cloud native software* [3] and *serverless computing* [1]) has accelerated the trend towards externally hosted web applications. In these systems, the allocation and maintenance of physical servers is handled by external cloud providers, which significantly reduces the need for traditional IT tasks.

IaaS platforms. The disappearance of the IT department has had a notable impact on the skill set that web developers are expected to possess. Nowadays, developers are assumed to master the basics of *Infrastructure as a Service (IaaS) platforms* such as AWS, Azure or Google Cloud, including the use of various services that are available in their management consoles. Many employers require an AWS certification or Azure Developer Associate certification.

Web servers and backend development frameworks. Developers not only have to be familiar with *web servers* (e.g., Apache, NGINX, Node.js) but also with how to set up the necessary *reverse proxies and security perimeters* (e.g., with NGINX). In addition, they are nowadays commonly assumed to master various *backend development frameworks* (typically written in either JavaScript/Node.js or Python). The developers are also assumed to be familiar with *cloud automation/scripting languages* such as Ansible.

Service scaling, monitoring, logging and fault tolerance & backup services. In addition, developers are commonly expected to master the delivery of *scalable services with high availability*. They need to understand how to architect software that can benefit from load balancing, fault tolerance and automated switchover in case of VM or server domain failure, utilizing the services offered by IaaS platforms. Good understanding of *data storage and backup services* such as AWS Simple Storage Service (S3) and Glacier is also a common expectation, as well as system monitoring and logging technologies including Grafana, Kibana or CloudWatch.

Continuous integration, delivery and deployment. Developers are also expected to adopt processes enabling *continuous integration and deployment* [2]. They have to set up GitLab CI/CD pipelines for automatically building, testing and deploying their latest software versions to staging and production environments. This also requires them to use the appropriate testing techniques across the frontend and backend.

Containerization and container management. More recently, it is taken for granted that the developers can perform the *containerization/dockerization* of their software (packaging of their software into Docker containers), as well as define the necessary Helm charts to enable their software to run in a Kubernetes cluster or some other popular *management and orchestration platform for containerized applications*. Note that Kubernetes is a huge toolset; however, a developer does not necessarily need Kubernetes to implement and deploy containers, so there are different levels of complexity also in this area.

Serverless computing. It should be remarked that if serverless computing [1] becomes mainstream, the need for service management and IaaS platform related tasks may reduce but yet another – often vendor-specific – approach

to architect and deploy applications needs to be learned. Details vary greatly between the Function-as-a-Service providers.

In any case, the skill set is so broad that universities struggle to include all the areas in their curricula, especially since a lot of the technologies are vendor-specific and tightly coupled with particular commercial offerings. The breadth of the expected skill set has led to a number of jokes, including the "dear recruiters" quote at the beginning of this section.

3 The Implications of Microservice Architectures

Another major occurrence has been the *emergence of microservice architectures* [10]. Originally, a key motivation behind microservice architectures was the idea of *polyglot programming* [4]: as different microservices are implemented and maintained by different subteams, each team could independently choose and use the most suitable technologies to support the task while also matching the existing competencies of each team. However, just as with deployment topics, it is an increasingly common expectation that full stack developers will be able to design and implement a broad variety of microservices and their APIs which would typically communicate with each other using synchronous calls, or asynchronous message brokers.

Message brokers. Orchestration and internal communication to "glue" different microservices together to a larger functional entity is typically performed using a message broker [14]. Basically, in addition to adopting the HTTP protocol following the REST architectural style [13], developers are expected to have at least basic knowledge of popular *message brokers* such as ActiveMQ, RabbitMQ and/or Kafka.

Database technologies beyond SQL. In addition to traditional relational (SQL) database skills, developers are also assumed to be familiar with popular *noSQL databases* (such as Cassandra, Neo4j, Redis, or MongoDB) and in many cases *time-series databases* such as InfluxDB or Riak. The use of time-series databases is especially important in building microservices that deal with data acquisition from a large number of devices. *In-memory database systems* such as Apache Ignite are also a popular option in implementing microservices. Common additional skill expectations in the storage and database query area include technologies such as ElasticSearch, GraphQL and Hive.

Node.js and/or Python backend development frameworks. The microservices themselves are commonly written either in JavaScript or Python, leveraging the massive number of frameworks and components available in Node.js NPM and Python PyPI ecosystems. The selection of applicable components can be a daunting task, as discussed in papers focusing on *opportunistic design* [8].

Web API design. Along with knowledge about backend development frameworks, developers are generally expected to be familiar with Web API design principles and best practices, including RESTful API design [9], and the corresponding API description languages such as OpenAPI, while keeping track of emerging alternatives such as GraphQL or gRPC.

4 Emerging Expectations

Data analytics. About ten years ago, *data analytics* became an important topic and one of the hottest and highest paying technical professions in the IT industry. Nowadays, even regular developers are expected to be familiar with at least some of the popular data analytics technologies. Depending on the needs of the particular project, focus may be more on *real-time analytics* or *offline analytics technologies*. Popular real-time analytics platforms include Apache Storm and Spark. For offline analytics, HADOOP (along with its distributed file system HDFS) is a common choice. More recently, managed technologies such as Snowflake and BigQuery have become popular. In the context of data analytics, knowledge of query languages such as GraphQL and data visualization libraries such as D3.js or Vis.js is also a common expectation. For fancier visualizations, WebGL knowledge (and knowledge of accompanying 3D convenience libraries such as Plotly.js or Three.js) is a plus.

AI/ML technologies. Turning raw data into valuable information requires full stack developers to skillfully apply popular AI/ML technologies such as Apache Mahout, Caffe, TensorFlow or Torch. While regular developers might not be expected to be able to build/train AI models, they *are* expected to be able to set up AI/ML pipelines to perform tasks such as object detection, face recognition or voice/phrase recognition using available technologies (e.g. [15]). While such tasks would have been considered very advanced or even impossible in the early days of Web engineering, they are now considered "basic" tasks in many contexts and applications. The availability of inexpensive AI/ML hardware is raising the expectations further (see the edge intelligence discussion below).

Internet of Things and the Programmable World. The Internet of Things (IoT) and the Web of Things (WoT) represent the next significant step in the evolution of the Internet and software development. Advances in hardware development and the general availability of powerful but very inexpensive integrated chips will make it possible to embed connectivity and full-fledged virtual machines and dynamic language runtimes virtually everywhere. The future potential of this *Programmable World* disruption will be as significant as the mobile application revolution that was sparked when similar technological advances made it possible to open up mobile phones for third-party application developers in the early 2000s [18, 17].

Factors that this disruption builds upon include (i) multidevice programming, (ii) heterogeneity and diversity of devices, (iii) intermittent, potentially highly unreliable connectivity, (iv) the distributed, highly dynamic, and potentially migratory nature of software (between devices/cloud/fog/edge), (v) the reactive, continuous, always-on nature of the overall system, and (vi) the general need to write software in a highly fault-tolerant and defensive manner. Furthermore, IoT is bringing back the need for embedded software development skills and the ability to build software which can run on slow processors, with limited memory and consuming little energy [11].

Streaming data systems. As devices themselves are becoming more data intensive as their sensing capabilities (e.g., audio, video, LIDAR) become more

advanced, AI/ML capabilities benefit vastly when raw data *streams* from the devices are made available in the cloud [7]. Developers must thus know how to implement cloud backend capable of ingesting continuously streaming data from a potentially very large numbers of devices – even millions of them.

Edge computing and edge intelligence. Classic web sites and cloud computing systems were highly centralized – nearly all of the computation apart from web page rendering was performed in the backend. While centralized computing has significant benefits, it can also be very costly in terms of communication and power consumption. For instance, if an IoT system consists of a large collection of devices that are in close proximity of each other, it may be inefficient to transmit all the data from those devices to a faraway data center for processing, and then transmit actuation requests back from the remote data center to the individual devices. In an IoT deployment with tight latency requirements, latency overhead alone can make such solutions impractical.

In recent years, the emergence of inexpensive but computationally powerful hardware solutions has made edge computing practical. This has made it feasible to perform data analytics or at least initial data filtering and processing in the IoT devices and gateways themselves, without having to send all the raw measurement data to data centers for processing. Furthermore, growing privacy concerns and the availability of inexpensive AI/ML hardware such as, for example, NVIDIA's Jetson or Arduino Nano 33 BLE devices have made it both necessary and possible to bring AI/ML capabilities (e.g., object detection and face recognition) to the edge with very little development effort. Consequently, it is rapidly becoming an expectation than an average full stack developer must master at least the basic use of these technologies.

5 The New Full Stack – Implications for Software and Web Engineering Education

Most university curriculums aim at a balance between theory and practice [6]. To complement teaching of theoretical aspects, universities commonly include practical projects and hands-on exercises. However, the wide range of concepts and technology choices in the new full stack poses challenges for this approach – there are simply too many topics and technologies to cover. Furthermore, many of the relevant technologies tend to be rather vendor-specific. At the same time, students graduating from the universities are clearly in a better position with respect to their employment opportunities if they possess hands-on skills on the entire spectrum of full stack technologies.

It is important to note that full stack does not mean "all" technologies. To us, it seems that we must first define which "core" technologies the students absolutely must master in order to be well versed for basic development tasks. Beyond the core topics, the needs for specific technologies will be to at least some extent dependent on use cases and roles, e.g., whether the developer is focusing on "AI full stack" vs. "analytics full stack" vs. "IoT full stack". At the moment,

no comprehensive taxonomy of core and advanced full stack technologies exists. We see this as an interesting avenue for further research.

In the short term, one solution to this dilemma is to build extensive, industry-grade systems around the aforementioned core full stack, where use case specific modules can be added as subsystems, thus decomposing the projects to different courses. Furthermore, as almost all modern full stack technologies have online courses, e.g., in the form of Massive Open Online Courses (MOOCs) or online tutorials and exercises, those can be used as educational material. As an example of this model, the University of Helsinki has introduced *Fullstack Challenge*[5] in which students work with modern web technologies and tools on practical projects in collaboration with companies. At USI, practical *ateliers* scheduled in the afternoons complement and integrate the theoretical morning lectures.

No matter which approach is taken, given the increasing complexity and breadth of expected functionality, in most software systems it is no longer realistic to implement everything from scratch, even if parts of the system were built over time in different classes. Instead, the new full stack development is characterized by plentiful use of public-domain software components from repositories such as NPM and PyPI. Instead of using a fixed set of technologies or blindly following recommendations in a cargo-cult fashion, the students should still learn to think for themselves and carefully consider the implications of the selections that they make. At the same time, every developer needs a "full stack Swiss army knife" as the starting point.

In summary, the increased scope of full stack development challenges the universities to recognize, formulate, conceptualize, and teach the new general principles that provide long-term competencies. Students should learn how to recognize fundamental problems so that they can solve them with the appropriate conceptual tools using the corresponding technology of the day. This requires research on effective technical paradigms, understanding of how to anticipate practical needs of the companies, and also knowledge of the modern technologies and their associated online courses.

6 Conclusions

In this paper we have presented our observations on the rapidly growing requirements for a full stack web developer. While the classic full stack was concerned primarily with the basic frontend and backend split and technologies required for developing web sites and applications, the expectations for a modern full stack developer are far more comprehensive. Effectively, expected skills cover a spectrum of areas that would have been the responsibility of an entire IT department in the earlier days. Moreover, we foresee further technologies emerging and broadening the expected skill set even more in the coming years. This evolution will force us to reconsider the role of university degrees, technology certificates and lifelong learning efforts together with new tools such as MOOCs in the software and web engineering education.

[5] https://fullstackopen.com/en/

References

1. Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*, pages 1–20. Springer, 2017.
2. Brian Fitzgerald and Klaas-Jan Stol. Continuous Software Engineering: A Roadmap and Agenda. *Journal of Systems and Software*, 123:176–189, 2017.
3. Dennis Gannon, Roger Barga, and Neel Sundaresan. Cloud-Native Applications. *IEEE Cloud Computing*, 4(5):16–21, 2017.
4. Juhana Harmanen and Tommi Mikkonen. On Polyglot Programming in the Web. In *Modern Software Engineering Methodologies for Mobile and Cloud Environments*, pages 102–119. IGI Global, 2016.
5. Michael Hüttermann. *DevOps for Developers*. Apress, 2012.
6. Mehdi Jazayeri. The Education of a Software Engineer. In *Proc. 19th International Conference on Automated Software Engineering*, 2004.
7. David Maier and Badrish Chandramouli (eds). Special Issue on Next-Generation Stream Processing. *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society*, 38(4), 2015.
8. Niko Mäkitalo, Antero Taivalsaari, Arto Kiviluoto, Tommi Mikkonen, and Rafael Capilla. On Opportunistic Software Reuse. *Computing*, 102(11):2385–2408, 2020.
9. Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2011.
10. Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, 2015.
11. James Noble and Charles Weir. *Small Memory Software: Patterns for Systems with Limited Memory*. Addison-Wesley Longman Publishing Co., Inc., 2001.
12. Chris Northwood. *The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer*. Springer, 2018.
13. Cesare Pautasso and Olaf Zimmermann. The Web as a Software Connector: Integration Resting on Linked Resources. *IEEE Software*, 35(1):93–98, Jan/Feb 2018.
14. Cesare Pautasso, Olaf Zimmermann, Mike Amundsen, James Lewis, and Nicolai Josuttis. Microservices in Practice, Part 2: Service Integration and Sustainability. *IEEE Software*, (2):97–104, 2017.
15. Arvind Ravulavaru. *Google Cloud AI Services Quick Start Guide: Build Intelligent Applications with Google Cloud AI Services*. Packt Publishing Ltd, 2018.
16. Clemens A. Szyperski. Objectively: Components Versus Web Services. In *Proc. ECOOP 2002*, page 256, 2002.
17. Antero Taivalsaari and Tommi Mikkonen. A Roadmap to the Programmable World: Software Challenges in the IoT Era. *IEEE Software*, 34(1):72–80, 2017.
18. Bill Wasik. In the Programmable World, All Our Objects Will Act as One. *Wired. Available online: http://www.wired.com/2013/05/internet-of-things-2/ (accessed on Oct 13, 2020)*, 2013.