ISEL – Instituto Superior de Engenharia de Lisboa
DEETC – Departamento de Engenharia de Electrónica e Teleconumicações e de Computadores

Projeto Científico-pedagógico

A Software Engineering approach based on Frontend Development

Fernando Miguel Gamboa de Carvalho

---

30 of March, 2023

Index

Chapter 1

Introduction

Software engineering involves many different aspects, methodologies and practices, which are all aimed at building high-quality, reliable, and efficient software systems.

On the one hand, software development should comply with the characteristics and constraints dictated by the runtime environment, on the other hand software engineering provides methodologies and practices that help developers deal with those constraints in an efficient way.

In this chapter we start describing why Frontend Development is the proposed field for applying Software Engineering in Education, Research and Development?

1.1 Motivation

Some of the methodologies used in software engineering, not limited to, are: Test-Driven Development (TDD); Continuous Integration and Deployment (CI/CD); Model-Driven Development; and Design Patterns, which provides reusable solutions to common software design problems. Design patterns help to improve the quality, efficiency, and maintainability of software systems by providing a standardized way of solving common design problems.

Frontend development has been for years one of those fields, rich in use cases where design patterns help to solve problems that arise in user interfaces. For instance, many of the design patterns documented by the Gang of Four (GoF) book [10], are applicable to GUI development use cases.

Today, state of the art Frontend frameworks continue to create and renew well known design patterns, to establish good practices that lead developers to correctly use those frameworks. For example, in React [7], a Higher-Order Component (HOC) is a pattern to abstract away common functionality that is shared across multiple components. An HOC is a function that takes a component and returns a new component with additional functionality, in the same way inspired by the Composition and/or Decorator design pattern or, from another point of view, applying the idea of higher-order-functions in functional programming to GUI components.

Another example is Jetpack Compose framework for Android app development [11], where Composables are the building blocks of the user interface. A Composable is a function that describes how to render a part of the user interface and has some

similarities to HOC in React.

But, software engineering involves not only the use of design patterns and high-level abstractions, but also includes knowledge of the underlying principles and technologies that enable those abstractions.

For example, a software engineer must have a deep understanding of the native device APIs and underlying operating system features to ensure that the Frontend runs smoothly and efficiently on target platforms.

For example, knowledge about the UI thread is a good example of the kind of under-the-hood mechanism that software engineers need to have in order to build high-quality frontend applications. The UI thread is the main thread in a frontend application that is responsible for handling user interface updates, such as rendering graphics and responding to user input. It is critical to ensure that background tasks and other processing do not interfere with the UI thread, as this can cause the application to become unresponsive or crash.

In summary, software engineering involves both high-level abstractions like design patterns and frameworks, as well as a deep understanding of the underlying technologies and principles that enable those abstractions. This knowledge is necessary to develop high-quality, maintainable software that meets the needs of the end-users.

1.2 Document Outline

In the Chapter 2, I present the state-of-the-at in Frontend Development.

The Chapter 3 describes how Frontend Development is ingrained in Computer Science and Computer Engineering courses of ISEL, and also my proposal of a new curricular unit according to the latest trends in cross-platform development.

After that, in Chapter 4, I present my last research work in the Frontend Development field and my proposals for future work.

Finally, in Chapter 5, I describe my experience in industry and Frontend development, from the 90's to nowadays with the use of cross-platform Frontend development.

Chapter 6 presents my conclusions remarks.

Chapter 2

State of the Art

In this chapter we explain the evolution of GUI Frontend since its appearance and use in Desktop to the Web era and now the most recent usage of cross-platform frameworks.

After that, we present the main software engineering challenges in Frontend development.

2.1 From GUI Desktop to Cross-platform Development

Frontend refers to the part of a software system that interacts directly with users. The beginning of Frontend with GUI (Graphical User Interface) can be traced back to the 1960s when Douglas Engelbart, a computer scientist, began experimenting with new ways to interact with computers [6].

In the 1970s, Xerox PARC (Palo Alto Research Center) developed the Alto, the first personal computer to feature a graphical user interface. The Alto's GUI included icons, windows, menus, and a pointing device called a mouse, which made it much easier for users to interact with the computer [3].

In the 1980s, Apple introduced the Macintosh computer, which popularized the GUI and made it more accessible to a wider audience. Microsoft followed suit with the release of Windows 1.0 in 1985, which also featured a GUI.

Since then, GUI Frontend has become a standard feature of most operating systems and software applications, making it easier for users to interact with and operate software.

The rise of the internet, advances in web technologies, and the increasing demand for software that can be accessed from anywhere and any device, lead to the evolution from Desktop GUI Frontend to Web Based Frontend.

Web-based Frontend refers to software applications that run on a remote server and are accessed through a web browser.

The transition from Desktop GUI Frontend to Web-based Frontend began in the late 1990s when web technologies such as HTML, CSS, and JavaScript began to improve, making it possible to create more interactive and dynamic web pages.

Server-side rendering (SSR) [9, 2] was the most used approach in Frontend de-

velopment. HTML pages are dynamically generated on the server and sent to the client's browser. This approach allowed developers to create dynamic web content that could be rendered on the server.

As web technologies continued to evolve, more complex and sophisticated Web-based Frontend applications were developed, and the emergence of Single Page Applications (SPAs) [13] allowed for a more desktop-like user experience in a web browser.

SPAs have the ability to perform partial renders, also known as partial updates or partial page loads. This is one of the main advantages of SPAs over traditional multi-page web applications, which require full page reloads when navigating between pages. This allows the SPA to fetch and display only the data that needs to be updated, without requiring a full page reload.

However, SSR is still used in some cases to allow faster page load times and better search engine optimization. Thus, SSR is still an important methodology for building performant and scalable web applications, which may coexist together with SPA for Web development.

Nevertheless, the emergence of SPAs have influenced the way software architectures are designed and implemented. SPAs rely on a Web API (i.e. backend) to retrieve and manipulate data asynchronously. This means that the frontend and backend can be developed and deployed independently.

Moreover, the same Web API can be used by different kinds of Frontend, such as web-based frontend, desktop native, or mobile native applications. This is one of the main advantages of Web APIs, as they provide a standardized way for different applications and devices to communicate with a backend system.

The separation of concerns between Web API and Frontend can allow for specialized developers to work on different aspects of Frontend specifically for each device, such as Web, Desktop, iOS, or Android development. This is because each platform may have different requirements, design guidelines, and programming languages, and developers with specialized knowledge in each area can create better user experiences and optimize performance for their specific platform.

Thus, today, the current challenge is how to reduce duplication effort in development for different devices?

One emergent approach is the use of cross-platform development frameworks that allow developers to write a single codebase that can be used to build applications for multiple platforms.

There are several popular cross-platform development frameworks available today, including React Native, Flutter, Xamarin, and Ionic, among others. These frameworks provide developers with tools and APIs to create user interfaces, access native device features, and interact with APIs and services.

By using a cross-platform development framework, developers can reduce the amount of code that needs to be written and maintained for each platform, which can lead to faster development times, easier maintenance, and reduced costs.

2.2 Frontend Development Constraints

One common characteristic and constraint that applies to all Frontend development, regardless of the platform or device being targeted, is the rule of the UI thread, also known as the UI thread constraint.

In general, the UI thread constraint means that only the main UI thread is allowed to update the user interface, while background threads or worker threads are used to perform other tasks, such as fetching data from a server, processing data, or performing other computational tasks.

The reason for this constraint is to ensure that the user interface remains responsive and does not become blocked or unresponsive due to long-running tasks or other processing that is not related to the UI. By keeping UI updates confined to the UI thread, developers can ensure that the user interface remains smooth and responsive, even when performing other tasks in the background.

This constraint can be particularly important in mobile and web development, where users may have lower tolerance for slow or unresponsive user interfaces, and where resources such as battery life and network bandwidth may be limited.

To work within this constraint, developers may use techniques such as asynchronous programming, message passing, and event-driven programming to ensure that UI updates and background tasks are properly separated and executed in a way that maintains a smooth and responsive user interface.

This requirement not only dictates the GUI development approach but also data processing that feeds the UI. For this reason asynchronous programming is an idiomatic pattern that is spread over all parts that compose a Frontend application and follows the non-blocking principle.

The non-blocking principle aims to increase the efficiency and responsiveness of a system by minimizing the time that a thread or process spends waiting for a resource or task to complete. In a non-blocking system, multiple operations can occur concurrently without blocking or waiting for each other, resulting in faster overall performance.

In a non-blocking system, when a resource or task is not immediately available, the system does not block or pause the thread or process waiting for it. Instead, the system can perform other operations while waiting for the resource or task to become available. This can be achieved using techniques, such as callbacks, promises, or reactive programming, which allow the system to perform multiple operations simultaneously without blocking.

Chaining data transformation blocks according to the non-blocking principle ensures that a terminal operation consuming that data to feed the GUI will not block the GUI thread and keeps responsiveness.

Chapter 3

Academy

First I will present an overview of some Computer Science and Computer Engineering courses of ISEL that lead with Frontend development.

After that I will present my proposal for a Cross-platform Frontend Development curricular unit.

3.1 Frontend related courses at ISEL

Next we present the curricular units of Computer Science and Computer Engineering courses of ISEL that deal with Frontend Development:

- TDS - Software Development Techniques of 3rd semester
- IPW - Introduction to Internet Programming course of 3rd semester,
- PC - Concurrent Programming course of 4th semester,
- DAW - Web Application Development course of 5th semester
- PDM - Mobile Device Programming course of 5th semester

The Software Development Techniques of 3rd semester course aims to provide students with the theoretical foundations and practical skills required to develop software effectively and efficiently. The course covers a range of topics related to software development, including software design principles, design patterns, testing, debugging, and version control. As a practicing field for applying the software development techniques the students use the framework Desktop Jetpack Compose to build a Desktop GUI application, usually a game, such as Chess, Checkers, Battleship.

The Introduction to Internet Programming course of 3rd semester, covers fundamental concepts and technologies used in web development. This course provides an introduction to web development technologies, such as HTML, CSS, and JavaScript, as well as web servers and HTTP protocol. For Frontend development the students use Server Side Render approach (SSR)

The Concurrent Programming course of 4th semester, includes fundamental concepts, such as the threading model for typical application types, namely UI-based

applications and the correct use of common application-level asynchronous programming models available, namely futures, asynchronous methods, coroutines, and reactive streams.

The Web Application Development course of 5th semester is designed to provide students with the knowledge and skills needed to develop complex web applications using modern web technologies and frameworks, namely regarding a Single-Page Application approach (SPA).

The Mobile Device Programming course of 5th semester is regarding the programming model of a mainstream platform for mobile device application development, namely Android. This includes, the design decisions made in the development of mobile device applications that make use of both the device's local resources and remote resources according to the native Android programming model.

3.2 Cross-platform Frontend Development

This course introduces students to the principles and practices of cross-platform frontend development using the Flutter framework. Students will learn how to design and implement high-performance and scalable applications with responsive user interfaces that can run on multiple platforms such as iOS, Android, and the web.

This course requires background knowledge in all subjects covered by curricular units dealing with Frontend developed, namely TDS, IPW, PC, DAW and PDM, covered in the previous section of this document.

Thus, this course fits both as an optional curricular unit of 5th semester of BsC or of any semester of MsC.

The course will cover the fundamentals of widgets, state management, and routing. Students will also learn how to access device features such as the camera, microphone, and GPS. The course will include hands-on projects where students will develop mobile applications and deploy them to the app stores. Additionally, the course will cover best practices for testing, debugging, and deploying applications.

By the end of the course, students will have gained the skills and knowledge necessary to develop complex, interactive, and visually stunning user interfaces that run seamlessly on a variety of platforms, including web, mobile, and desktop.

As there is a growing demand for cross-platform development skills in the software development industry. With the increasing popularity of cross-platform frameworks, companies are looking for developers who can build mobile and web applications that run smoothly on multiple platforms.

Course outline:

1. Widgets and Layouts
    - widgets and layouts and how to build them
    - Advanced Flutter layouts and how to create custom widgets
2. Reactive Programming with Flutter

- Understanding reactive programming principles
- Working with streams and reactive programming
- Building reactive user interfaces

3. Navigation and Routing
    - How to navigate between screens in Flutter
    - Advanced routing techniques
4. Cross-Platform Development
    - Building mobile, web, and desktop applications
    - Techniques for sharing code between platforms
    - Platform-specific considerations and best practices
5. State Management
    - Overview of state management in Flutter
    - Working with stateful widgets
    - State management architectures such as BLoC and Provider
    - Building Responsive and Adaptive User Interfaces
6. Testing principles and techniques:
    - How to write unit tests and widget tests
    - Techniques for debugging applications
    - Performance and Optimization
7. Understanding performance characteristics
    - Techniques for optimizing applications for performance
    - Debugging performance issues

By providing students with hands-on experience in developing cross-platform applications, this course prepares them for careers in the software development industry. Students will gain practical skills and knowledge that are in high demand, making them highly attractive to potential employers. Additionally, the course will help students to keep up with the latest trends and technologies in cross-platform development, ensuring that they remain competitive in the job market.

Chapter 4

Research

Simply put, my research work focuses on turning legacy web applications based on server-side rendering (SSR) to support asynchronous and reactive data models with a non-blocking approach complying with non-blocking requirements of modern web servers, such as node js, spring webflux or Vert.X.

My research work follows to branches regarding Frontend templates:

1. How to achieve SSR without blocking on asynchronous data?

2. How to asynchronously emit HTML from SSR in asynchronous Web components?

The first section describes the state of the art. After that, the next two sections describe the branches of this research work.

4.1 State of the art

There are over 1.5 billion websites and the fourth quarter of 2021 registered 341,7 million domain names across all top-level domain according to a report of Verisign Inc. Also, a w3techs.com survey shows that 95% of these web applications use server-side rendering (SSR) approaches [9, 2] supported by legacy technologies that emerged before the rise of big data [14, 8, 17]. Only 5% use state-of-the-art methodologies, namely single-page application architectures – SPA [13] that can scale in big data scenarios [15].

This research proposal explores an alternative methodology to SPA that will enable legacy web applications to support big-data constraints and modern requirements of the web in terms of responsiveness and availability. We build on top of our previous proposal for higher-order templates (HoT) [4, 5] that suppresses the web templating idiosyncrasies and provides progressive web presentation behavior.

Yet, scalable web applications depend not only on progressive and responsive visualization techniques [1, 18] but also on asynchronous data flows, such those stated by reactive streams standard or dotnet async streams [12], which observed an increased adoption due to better resource management in multi tenant scenarios such as cloud environments.

However legacy web applications supported on server-side rendering (SSR) cannot take advantage of such data flow models, because of the textual nature of their template engines (such as JSP, Mustache, Handlebars and others) that block on HTML resolution before processing the output presentation view. To suppress that limitation, we need a progressive presentation resolution such that provided by HoT approach and an alternative data traversal mechanism compliant with SSR backend. In our previous work we have proposed an alternative data flow traversal design, named tinyield, which provides concise extensibility and outperforms competition in realistic benchmarks. We developed an extensive performance analyzes and benchmarking with different approaches and technologies, namely in Java, Dotnet and JavaScript and we have proposed a generalized functional yield-based traversal pattern that suppresses performance or verbosity overheads on streams extensions [16].

4.2 SSR without blocking on asynchronous data

This research work proposes to take a step forward, taking advantage and integrating our tinyield traversal model, with the HoT approach to provide non-blocking support for asynchronous stream processing for Frontend web applications.

HoT is the result of our project – HtmlFlow – consisting of a Java DSL to write typesafe HTML, which is rated with more than 120 stars by the community in the biggest open-source hosting provider: https://github.com/xmlet/HtmlFlow/. The functional and compositional nature of HtmlFlow distinguishes it from the competition for: being data structureless, allowing method chaining calls and providing HTML safety. These key characteristics contribute to the increasing popularity of the HtmlFlow library, in addition to the better performance than state-of-the art template engines in well-known benchmarks, such as the spring-comparing-template-engines.

In our previous work [5], we have already shown how HoT can be used in a renowned web application use-case (the spring-petclinic) leveraging the technological homogeneity between the backend and frontend around a single host programming language, such as Java. These observations give us promising perspectives about the use of this same approach in other web applications. HtmlFlow disrupts the conceptual genesis of templates and introduces web templates as first-class functions that take full advantage of their intrinsic composability characteristics.

We will delegate on HtmlFlow infrastructure as the core backbone of web applications with server-side rendering (SSR). To evaluate the results of this work, we plan to test them both on a set of well-known benchmarks and for a selection of real-world applications that span different workload characteristics in big data scenarios.

Even modern state-of-the-art web frameworks such as Spring, which already provide a reactive stack, are constrained at the template engine level (such as JSP, Mustache or Handlebars) and finishes the asynchronous pipeline with a blocking terminal operation before pushing the resulting front-end to the client, turning useless the asynchronous pipeline.

To achieve a fully asynchronous pipeline we need to turn higher-order templates compliant with the reactive streams standard. At this point, HtmlFlow library and HoT are ahead of the competition since all other state-of-the-art engines are still oriented by text templates. This observation is based on the most used templates around the Spring web framework.

Even though we expect that our work will be applicable to other web application frameworks, we will target specifically the Spring platform on this project. Moreover, the HtmlFlow is only compromised with the JVM and not with any framework, hence it does not constraint the choice on the web platform as long as it is JVM compliant, which is the most used runtime environment in the industry.

4.3 Emit HTML from SSR in asynchronous Web components

Template engines use templating statements that are resolved sequentially. Thus, handling asynchronous data may prevent the rest of a web template from being processed until data becomes available.

In this work we analyze the limitations of state-of-the-art template engines dealing with asynchronous data models. Also, we explore an alternative web templating idiom that suppresses those limitations and let programmers express asynchronous HTML fragments, which defer resolution and let processing the rest of the Web template. The resulting HTML may be sent in a HTTP response beforehand and the rest of HTML fragments are sent on asynchronous data completion.

We will extend HtmlFlow with a new API to express asynchronous web fragments that take advantage of web components and custom elements to defer HTML processing.

Previous work on HtmlFlow has already shown how a DSL for HTML suppresses the limitations of text-based templates such as JSP, Thymeleaf, Handlebars, and others. Here we will enhance HtmlFlow to embrace modern Web components standard and provide an innovative feature not yet available in any SSR template engine.

Chapter 5

Development and Industry

5.1 Past Experience

Since the beginning of my professional career in 1997 that is closed to Frontend software development. I have an accumulated experience since Visual Basic 6, to the use of first low-code platforms for FrontEnd development provided by Easyphone Company and later by Intervento CMS, which is the foundation of the state of the art Outsystems platform.

Intervento CMS used an extension of TCL programming language, that is a dynamic programming language that was created in the late 1980s as a scripting language to be embedded into other applications. It is designed to be easy to learn and use, yet powerful enough to support complex tasks. Tcl is widely used for creating GUI applications, network and web programming, and system administration tasks, among others. One of the notable features of Tcl is its built-in support for regular expressions, which makes text processing and manipulation much easier.

Beyond GUI Frontend, I also have experience on other kinds of UI namely using IVR - interactive voice response. I designed and developed the first software framework in 1998 for Altitude IVR to provide the first dynamic IVR script framework, deployed in the contact center of one of the biggest banks in Portugal. My proposal not only suppressed the deployment downtime, but also reduced in several minutes the deployment process to a few seconds.

Within my career in Academy, I also payed attention to latest trends in Frontend frameworks in different technological environments namely in Java (JavaFX for Desktop and JSP, Thymeleaf, and Spring for Web apps), Dotnet (Forms and WPF for Desktop and ASP.NET for Web apps) and Javascript (React, Angular and Vue).

Given my knowledge and experience, I have developed my own Frontend Framework - HtmlFlow - a Java DSL for HTML, to write typesafe HTML documents in a fluent style. It offers some unique features, such as a simple and concise DSL and focus on type safety and correctness.

Despite HtmlFlow has started as a computational support for the curricular unit of Modeling and Design Patterns, it rapidly grown within the community with some professional developers using it to produce customized HTML reports integrated in enterprise resource management systems, namely in some financial companies in

France and Norway. Today, HtmlFlow achieved its own niche and a dedicated user base worldwide, with a rating of more than 120 stars in Github.

5.2 Present and Future Experience

The appearance of the first Frontend cross-platforms frameworks caught my attention. Cross-platform frameworks promise to simplify the development process by allowing developers to write code once and deploy it on multiple platforms, such as iOS, Android, and web. This can save time and resources, as developers do not need to create and maintain separate codebases for each platform.

Also, cross-platform frameworks provide a consistent user experience across platforms. This means that the same app will look and feel the same on different devices, which can help to build brand recognition and user loyalty.

In the latest years Flutter gained preponderance over the competition with a larger number of ready-to-use widgets in comparison to React Native, and Xamarin.

Comparing the performance of React Native, Flutter, and Xamarin depends on several factors that can affect the comparison. But, generally, Flutter has a reputation for offering better performance than React Native and Xamarin.

Flutter uses Dart, a compiled language that supports Just-In-Time (JIT) compilation, which allows developers to see changes to their code immediately during development. Additionally, Flutter comes with its own rendering engine and widgets, which can help to optimize the performance of the apps built using the framework.

To acquire a better and deeper knowledge on the Flutter framework I conducted a new project of the Movyon mobile app for Android and Ios.

Movyon aggregates TV guides listings from several data sources and provides an integrated vision to the end-user. Also it implements a social network between groups of users sharing ratings and comments about TV programs.

In this project I faced several challenges namely, making an effective use of the network with background workers that should be processing only on downtime and taking advantage of each native device's features for media playing.

Beyond the Movyon project, I have been involved also in student projects using the Flutter framework. Some of them are in collaboration with the industry, namely the Digital Roadbook F2R project that aims to develop a mobile app to replace the Rally – Electric roadbook holder with paper rolls.

Another example, is the collaboration with 360hyper company to develop the Ecommerce Live Chat Software to provide real-time communication between customers and operators of an online ecommerce company, during the process of order processing and delivery. The software has two different operation modes: private and public. In the private mode, three operators from the company, i.e., the order picker, driver, and backoffice support operator, can communicate with each other regarding a particular ecommerce order. In the public mode, the customers can also communicate with all the operators that exist in the private mode, and the software sends automatic notifications to both customers and operators regarding the order status updates.

Yet, a BsC project encompassed in one semester has not been enough to put these projects in production and in most cases they do not pass off more than prototypes.

One future goal is to extend these projects to a collaboration with industry aiming MsC projects, with a wider duration that lets embrace a more effective development process and achieve the deployment at least on the Play store.

Chapter 6

Conclusion

The interrelation between Academy, Science, and Industry is essential to enhancing the quality of education since it provides an environment where knowledge is created, shared, and applied to solve real-world problems.

Science provides the fundamental knowledge and research that drives innovation and development in Industry. Academy plays a key role in training the next generation, as well as conducting fundamental research that can lead to breakthroughs in Science and Industry.

Industry is often at the forefront of innovation, developing new products, services, and technologies. Collaboration between industry and academia allows researchers to learn about new developments in the industry and identify potential applications for their research. This helps to create new ideas, products, and services that can benefit society.

At same time, Industry faces complex and often unpredictable challenges that require innovative solutions. Collaboration between industry and academia allows researchers to work on real-world problems and apply their knowledge to develop practical solutions. This type of collaboration can help to ensure that research is relevant and impactful.

Collaboration between industry and academia can help to develop a talent pool that is equipped with the skills and knowledge required to meet the demands of the industry. This collaboration can help to ensure that educational programs are aligned with the needs of the industry and that graduates are ready to enter the workforce.

Industry can provide insight into the practical applications of research, while academia can provide industry with the latest research findings and knowledge.

Bibliography

[1] Rajeev Agrawal et al. "Challenges and opportunities with big data visualization". In: *Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems*. 2015, pp. 169–173.

[2] Deepak Alur, John Crupi, and Dan Malks. *Core J2EE patterns: best practices and design strategies*. Gulf Professional Publishing, 2003.

[3] Susan B Barnes. "Douglas Carl Engelbart: Developing the underlying concepts for contemporary computing". In: *IEEE Annals of the History of Computing* 19.3 (1997), pp. 16–26.

[4] Fernando Carvalho and Luis Duarte. "HoT: Unleash Web Views with Higher-order Templates". In: *Proceedings of the 15th International Conference on Web Information Systems and Technologies*. 2019, pp. 118–129.

[5] Fernando Miguel Carvalho, Luis Duarte, and Julien Gouesse. "Text Web Templates Considered Harmful". In: *LNBIP - Lecture Notes in Business Information Processing*. Ed. by Alessandro Bozzon, Francisco José Domínguez Mayo, and Joaquim Filipe. Cham: Springer International Publishing, 2020, pp. 69–95.

[6] W.K. English, D.C. Engelbart, and M.L. Berman. "Display-Selection Techniques for Text Manipulation". In: *IEEE Transactions on Human Factors in Electronics* HFE-8.1 (1967), pp. 5–15. DOI: 10.1109/THFE.1967.232994.

[7] Artemij Fedosejev. *React. js essentials*. Packt Publishing Ltd, 2015.

[8] Dan Feldman, Melanie Schmidt, and Christian Sohler. "Turning big data into tiny data: Constant-size coresets for k-means, PCA, and projective clustering". In: *SIAM Journal on Computing* 49.3 (2020), pp. 601–657.

[9] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003.

[10] Erich Gamma et al. "Elements of Reusable Object-Oriented Software". In: *Design Patterns* (1995).

[11] Dawn Griffiths and David Griffiths. *Head First Android Development.* " O'Reilly Media, Inc.", 2021.

[12] Shweta Khare et al. "Reactive stream processing for data-centric publish/subscribe". In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*. 2015, pp. 234–245.

[13] Philip Klauzinski and John Moore. *Mastering JavaScript Single Page Application Development*. Packt Publishing Ltd, 2016.

[14] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. "imMens: Real-time visual querying of big data". In: *Computer graphics forum*. Vol. 32. 3pt4. Wiley Online Library. 2013, pp. 421–430.

[15] Erik Meijer. "Your mouse is a database". In: *Communications of the ACM* 55.5 (2012), pp. 66–73.

[16] Diogo Poeira and Fernando Miguel Carvalho. "Deconstructing yield operator to enhance streams processing". In: *16th International Conference on Software Technologies (ICSOFT 2021)*. SCITEPRESS. 2021, pp. 143–150.

[17] Muhammad Habib ur Rehman et al. "Big data reduction methods: a survey". In: *Data Science and Engineering* 1 (2016), pp. 265–284.

[18] Veda C Storey and Il-Yeol Song. "Big data technologies and management: What conceptual modeling can do". In: *Data & Knowledge Engineering* 108 (2017), pp. 50–67.