ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Addressing Fairness in AI Systems: Design and Development of a Pragmatic (Meta-)Methodology

Tesi di laurea in:
Intelligent Systems Engineering

Relatore
Prof. Giovanni Ciatto

Candidato
Mattia Matteini

Correlatori
Prof. Roberta Calegari
Prof. Andrea Omicini

IV Sessione di Laurea
Anno Accademico 2023–2024

Abstract

Max 2000 characters, strict.

Optional. Max a few lines.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

Structure of the Thesis

Mattia Matteini: At the end, describe the structure of the paper

Chapter 2

Background

2.1 What is Fairness?

Fairness, from an ethical and social perspective, is the principle of treating individuals and groups equitably, ensuring that no one is unjustly advantaged or disadvantaged due to biases, discrimination, or arbitrary distinctions. It is deeply rooted in moral philosophy, legal systems, and societal norms, aiming to promote justice, equality, and inclusion. A just society requires reducing social inequalities, and ensuring that opportunities and resources are distributed in a way that acknowledges both individual merit and systemic disadvantages. The concept of fairness evolves based on cultural, historical, and contextual factors, reflecting a society's commitment to ethical treatment and social cohesion.

Fairness in AI. From a technical point of view, fairness in Artificial Intelligence (AI) refers to the development and deployment of AI systems that minimize biases and prevent discriminatory outcomes. It involves designing systems that ensure equitable treatment across different demographic groups, particularly those historically marginalized or disadvantaged. The main challenges in this field are building `fair-by-design` systems, namely such systems in which the fairness problem is addressed since the very beginning of the process, and detecting biases in already existing systems, mitigating them if possible.

Before the advent of fairness, AI systems were developed with the primary

goal of optimizing performance metrics, such as accuracy. Nowadays, that fairness is becoming a crucial aspect to consider, accuracy is no more the only metric to optimize, it is necessary to find a balance between accuracy and fairness. Besides that, fairness can also be in contrast with the performance of the model, making difficult to find a good trade-off between these two aspects.

Fairness is becoming crucial because AI systems increasingly influence decision-making processes in various sectors of society, including hiring, lending and healthcare. If AI models are biased, they can perpetuate and even amplify existing societal inequalities, leading to unjust outcomes and tremendous effects on individuals and communities. Ensuring fairness in AI enhances trust, transparency, and accountability, making AI systems more ethical, reliable, and beneficial for society.

AI has undergone significant advancements over the past few decades, causing an enormous increase in its adoption across various domains, until becoming pervasive in the daily life of people. This also caused a growing of biases in AI systems, as discriminations are intrinsically part of the human history, and consequently of the data that AI systems are trained on.

AI is now widely used in critical domains such as healthcare, finance, education, and criminal justice, where biased decisions can have life-altering consequences. For instance, in healthcare, biased algorithms may lead to misdiagnosis or unequal treatment recommendations for different demographic groups; in finance, AI-driven credit scoring models can reinforce discriminatory lending practices, limiting access to financial resources; in the criminal justice system, biased predictive policing and risk assessment tools can disproportionately target marginalized communities. Given these risks, ensuring fairness in AI is essential to preventing discrimination, maintaining ethical standards, and safeguarding individuals.

On Multidisciplinarity. Achieving fairness in AI requires a multidisciplinary approach that integrates insights from computer science, ethics, law and social sciences. Technical methods alone cannot fully address fairness, as fairness is deeply tied to societal values, human rights, and legal frameworks. Socio-legal experts help define fairness principles, ensure compliance with anti-discrimination laws and analyze the societal impact. The intersection of these fields highlights that AI

fairness is not merely a technical challenge but a complex, multidimensional issue requiring collective effort and interdisciplinary research and collaboration.

An impactful example regarding the work of legal experts in the field of Artificial Intelligence is the *AI Act*. The AI Act, proposed by the European Union, is a comprehensive regulatory framework designed to ensure that AI systems are safe, transparent, and aligned with fundamental rights. It categorizes AI applications into different risk levels—unacceptable risk, high risk, limited risk, and minimal risk—imposing stricter requirements on higher-risk systems, such as those used in hiring, law enforcement, and healthcare. These requirements include transparency, human oversight, and bias mitigation. However, translating these legal constraints into practical technical steps is not trivial.

Mattia Matteini: add some citation

Concepts like fairness, accountability, and explainability are difficult to quantify, and AI models often operate as black boxes, making compliance complex. While the AI Act sets an important precedent for AI governance, its effective implementation requires further collaboration between policymakers, legal experts, and computer scientist to bridge the gap between regulation and technical feasibility.

Measuring Fairness. At one point, in order to assess the fairness of an AI system, is important to have a way to "measure" how much the system is fair and in what terms. Remarking what said before, fairness is very context-dependent, and there is not a single way to measure it.

The need to cover multiple aspects of fairness has led to the introduction of various *fairness metrics*—statistical formulas that quantify fairness in different ways, each capturing a slightly different aspect of fairness. These fairness metrics, are a set of indexes that can be used to detect biases in AI systems, and they can be used indeed to evaluate the fairness of a model.

In the following, are listed some of the most common fairness metrics used in the literature [IML23]:

- *Statistical Parity Difference* (SPD) measures the difference between the probability of the privileged and unprivileged classes receiving a favorable outcome. This measure should be equal to 0 to be fair.

Formally it is defined as $SPD = P(\hat{Y} = 1|A = a) - P(\hat{Y} = 1|A = b)$ where A is the sensitive attribute, $\hat{Y}$ is the predicted outcome, and a and b are the privileged and unprivileged groups, respectively.

- *Disparate Impact* (DI) compares the proportion of individuals that receive a favorable outcome for two groups, a privileged group and an unprivileged group. This measure should be equal to 1 to be fair.

  Formally it is defined as $DI = P(\hat{Y} = 1|A = a)/P(\hat{Y} = 1|A = b)$ where A is the sensitive attribute, $\hat{Y}$ is the predicted outcome, and a and b are the privileged and unprivileged groups, respectively.

Mattia Matteini: add more metrics?

Mattia Matteini: manca una sezione Lack of engineering methodology?

2.2 AI Lifecycle

Since the very beginning of the AI era, the standard lifecycle consists of the following "traditional" steps: (i) data collection and processing, (ii) model training, (iii) system evaluation. Obviously, this workflow in the latest years have increased in complexity and now, with the newer innovations and powerful models and architectures, it may appear even almost minimalistic, but it still represents the core of all AI systems.

However, when fairness is taken into account, each step needs to be revisited in order to obtain an equitable, impartial, and fair AI system.

To achieve this goal, the technical perspective is not enough. Fairness is a multidisciplinary concept that involves social, legal, and ethical aspects. Therefore, the AI lifecycle needs to be constrained by socio-legal requirements that engineers must consider during the development process. This includes understanding the societal impact of AI systems, ensuring compliance with legal standards, and adhering to ethical guidelines.

There are also many differences between the socio-legal and technical perspectives. Regarding the AI lifecycle, engineers tend to focus on technical aspects and few development phases, in fact the major part of the literature speaks only about *pre-processing*, *in-processing* and *post-processing* (Figure 2.1). Respectively,

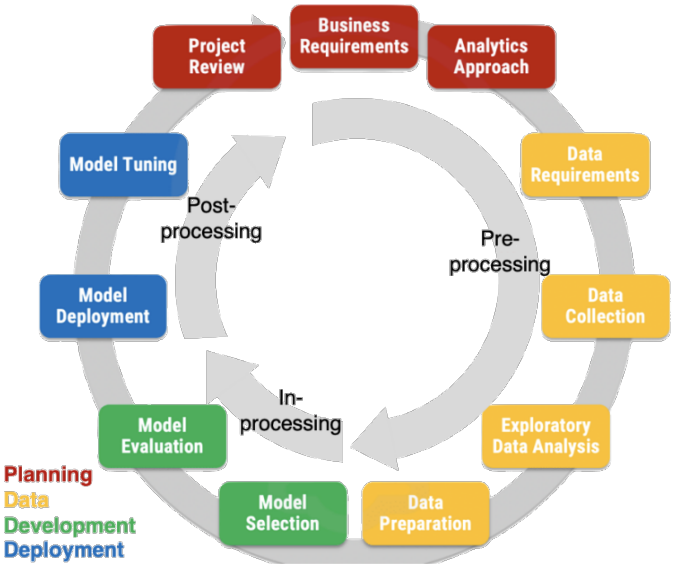


Figure 2.1: Fair AI lifecycle from [CCMO23]

pre-processing involves data collection and preparation, *in-processing* refers to the model training phase, and *post-processing* deals with the fair evaluation of the AI system.

Often engineers adopt reductionist approaches addressing a field that is not their own, discarding the big picture of social, economic, and institutional constraints. On the other hand, socio-legal experts consider a broader range of activities and phases. They focus on "building blocks" for fair AI such as risk assessment, stakeholder identification, regulatory analysis, and fundamental human rights impact assessment. In particular, with respect to fundamental rights impact assessments, these will be legally required for some AI systems, yet no standard for implementing them has emerged so far.

2.3 Practical Issues

2.3.1 What is (un)fair?

Fairness in AI (and beyond) is inherently subjective, shaped by cultural values, ethical theories, and individual perspectives. What one group considers fair may not align with other people's understanding, leading to debates about determin-

ing what is fair and what is not. This subjectivity and variation in viewpoints complicates efforts to develop standardized fairness metrics, as no single approach can universally capture the diverse and often conflicting notions of fairness present across different social, legal, and institutional contexts.

Beyond its subjectivity, fairness is also highly context-dependent. The same algorithm might be considered fair in one application but biased in another, depending on the societal, legal, and institutional constraints surrounding it. For instance, fairness considerations in hiring algorithms differ from those in criminal justice risk assessments, necessitating tailored approaches rather than generic solutions. Moving forward, privileged and unprivileged groups change depending on the application domain, as well as the fairness criteria that are taken into account.

2.3.2 Bridging Perspectives

Bridging the socio-legal and technical perspectives on fairness is a significant challenge. Guidelines and descriptive methodologies exist to address fairness compliance from a social-legal perspective, but their approach offer broad guidelines without defining practical steps, leaving interpretation to technical experts [CMS+25]. The lack of alignment between these viewpoints makes it difficult to translate abstract fairness principles into concrete computational methods. This also leads to a proliferation of metrics, each measuring slightly different aspects of fairness, reflecting the diverse priorities and domain perspectives.

A fundamental obstacle to this integration is the differing language used by socio-legal experts and technical people. It is difficult to reach an agreement if even a concept or term can assume different meanings depending on the perspective. This linguistic division creates a difficult barrier to interdisciplinary collaboration, leading to misunderstandings even when working towards shared goals.

These perspectives are shaped also by distinct academic and methodological backgrounds. Legal and ethical frameworks tend to be verbose and highly context-specific, relying on various interpretations and case-by-case analyses. In contrast, technical disciplines prioritize concrete steps and pragmatic aspects.

In literature, there is a lack of methodologies regarding the building of fair AI systems. The lack is not just related to the technical perspective, but also

to the socio-legal one. This is enhanced by the fact that design and develop a single methodology fitting all kinds of AI systems is not feasible, as the system requirements and constraints change depending on the context and the application domain. Of course, the creation of such methodology is damaged by the multidisciplinary complexity of the problem, and should involve expertises across all the relevant fields.

Chapter 3

The Meta-Methodology

A **methodology** is a structured framework that outlines the principles, processes, techniques and best practices used to conduct research or develop systems in a systematic and reproducible manner. In this context, a well-defined methodology would be essential for ensuring fairness, as it would provide a rigorous approach to (i) translating socio-legal requirements into technical steps, (ii) identifying and mitigating biases, and (iii) building `fair-by-design` systems.

Mattia Matteini: need to introduce the concept of `fair-by-design` systems in Background chapter

Having a rigorous methodology would impact the development of fair AI systems, it means that it would represent a clear way to follow, encapsulating the already existing unclear guidelines provided by the socio-legal frameworks.

Unfortunately, factors such as multidisciplinarity, complexity, and context-dependency make it difficult to design a single methodology that fits all contexts and applications. Therefore, this contribution proposes a **meta-methodology** that provides a flexible and adaptable framework for design and develop multiple methodologies instead of a single one. The idea of the meta-methodology comes from the sessions of brainstorming and discussions between experts of different fields, where troubles are emerged in reaching an agreement and proceeding with clear technical steps relying on the legal requirements.

3.1 Desiderata

In the following, are listed the desiderata that the meta-methodology should satisfy.

R1 **Context and Domain Awareness**: The methodology should consider the cultural context and the domain of AI system under design.

AI systems have been applied in several (and critical) use cases. For each of them, the constraints and requirements change, so through the methodology, it should be possible to understand the system domain and be context-aware.

R2 **Adaptability**: The methodology should adapt to any change in the cultural context as it evolves.

Some context could be volatile in terms of societal norms and cultural changes, so the methodology should be able to adjust and align to new constraints.

R3 **Requirements Translation**: The methodology should assist experts in translating the socio-legal requirements into practice.

A big challenge in this field, is to understand how legal constraints can be applied, and how technical steps can be identified to satisfy the requirements. That's why the methodology should provide a mechanism assisting this phase.

R4 **Legacy and Novel Systems**: The methodology should account for both pre-existing and new AI systems.

It is reasonable that the methodology should be able to be applied to new software systems, but it would be a big lack if it could not be applied to already existing systems, remembering that there are a lot of deployed systems that, probably, are not fair.

R5 **Building the AI System**: The methodology should not just provide a theoretical guideline, but also assist the AI system creation.

This means that it is necessary a software reification of the methodology permitting to be applied practically, obtaining eventually, a fair AI system.

3.2 Concepts

The Roles. In the proposed methodology process, there are two main roles involved:

1. **Business User (BU)**, also called **stakeholder**, who is the person commissioning the AI system.

2. **Technical User (TU)**, who is the person with technical background, assisting the BU in the development of the AI system.

With respect to BU, it is assumed that he or she may have limited or no technical knowledge. This is a common scenario in the real world, where often stakeholders are people with a specific domain expertise, but not necessarily with technical skills. One of the goal of the methodology, is to provide a way to assist the BU in the development of the AI system, without requiring necessarily deep technical knowledge. Potentially, stakeholder could even build a fair AI system without the need of a TU.

On the other hand, the Technical User, despite is the person with technical background, is not the responsible for the entire system development. Firstly, TU must be able to assist BU during the process to clarify any technicalities that may arise, and secondly, he or she must have some knowledge about basics of fairness.

Finally, TU may contribute to the system development through the implementation of scripts/computational processes involved in the building of the system and integrated in the methodology. In fact, the software reification of the methodology will be a tool providing APIs for technical people, in order to permit them to attach their scripts.

Questionnaire. Discussions among experts from involved fields highlighted the need for a practical understanding of the domain and the cultural context of the system being designed, ensuring that it is accessible and comprehensible to people of any background. The proposed approach relies on a straightforward questionnaire, which directly engages the Business User with questions regarding the system's domain.

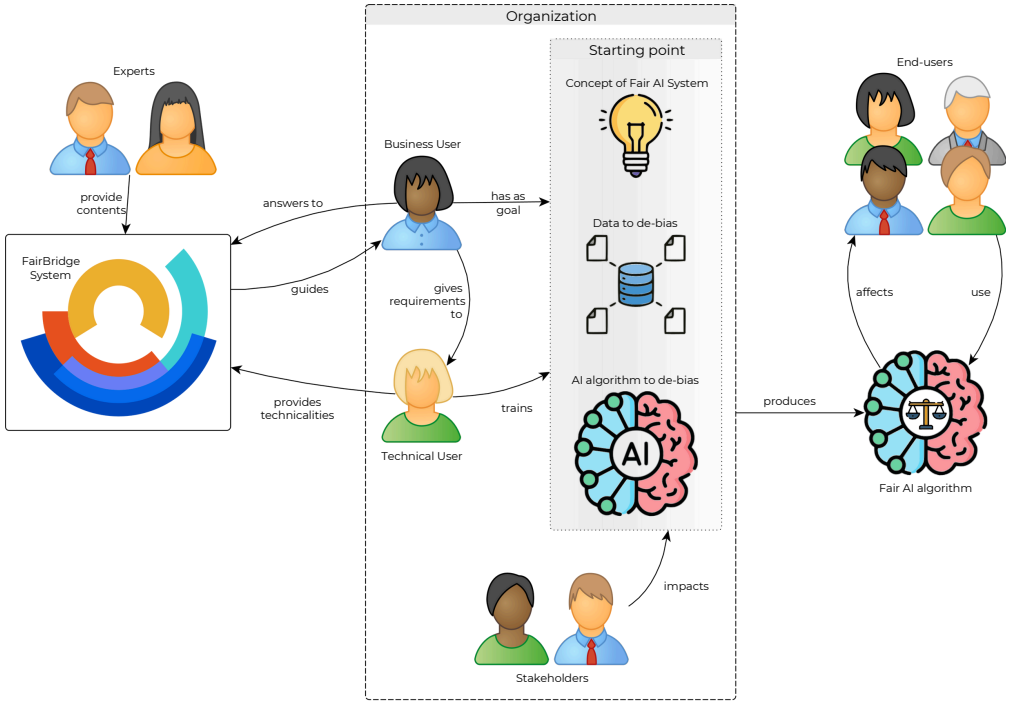


Figure 3.1: Concept of the proposed approach to fairness engineering from [CMS+25].

The questionnaire serves as a structured flow of questions and answers designed to gather essential contextual information. Depending on that, it provides practical steps to guide the development of a fair AI system. Questionnaire is not just used to collect information, it also acts as a tool to assist the BU in making well-informed decisions. At the same time, questions represent also technical steps to be taken, addressing the socio-legal constraints in a comprehensible way. This approach is central to the methodology, owing to its simplicity and versatility in capturing constraints and supporting multiple use cases. The overall concept is shown in fig. 3.1.

The pool of questions and answers should be designed ad-hoc from a team of multidisciplinary experts. This is a crucial, and non-trivial, step in the methodology, as the questions should be able to capture the constraints and requirements from legal frameworks, but also to provide technicalities to be addressed in the proper way and at right time in the process.

Examples of questions that could be asked are:

- "In what area will the AI system be applied?" (Healthcare, Finance, Hiring, etc.)
- "Do you have some AI system already in place, or are you developing an AI system?"
- "Is the dataset sufficiently representative of the population where the system will be used?"

But also more technical questions like:

- "What are the fairness metrics that should be considered?" (Statistical Parity Difference, Disparate Impact, etc.)
- "Which are the proxies for the sensitive features?"
- "Which data mitigation algorithm do you want to use?" (Disparate Impact Remover, Learned Fair Representations, etc.)

Order of Questions. In the previous paragraph it was mentioned that the flow of questions contains generic and technical questions. The questionnaire should follow a structured approach, beginning with general questions before gradually introducing more technical aspects. Initially, broad and non-technical questions are asked to establish a clear understanding of the system's domain, purpose, and the cultural or business context in which it operates. As the questionnaire progresses, the questions become more specific and technical. At one point it becomes mandatory to introduce technical aspects because in the end the questionnaire has to converge to the effective building of the fair AI system.

There is another important concept related to the order of questions: the answer to a question can influence the following ones. This feature is to enable the methodology to adapt to the context and asking later more tailored questions based on the previous answers. Moreover, it is also useful to enrich the part of system development, as it should be possible to follow multiple paths to make an AI system fair. For instance, the Business User could decide to preprocess the dataset twice, or choose to perform just in-processing mitigation.

These mechanisms lead to a more flexible and adaptable questionnaire, capable of addressing a wide range of contexts and applications, enabling also branching and joining paths in the flow.

Decision Support. The methodology includes—alongside the Q/A—a Decision Support Mechanism, aiming to simplify the process of making decisions regarding fairness-related or complex questions. Fairness problem should be taken into account not just by experts in the field, but also by stakeholders, as the relevance of the problem has grown significantly in the last years. Therefore, an important goal of the methodology is to make the BU aware of the fairness problem, and to assist him/her in making well-informed decisions. In this way, the BU can gain a deeper understanding of the topic, and can proceed with the development of the system more responsibly. Importantly, the mechanism does not impose decisions but rather suggests the BU the answer he or she probably should give.

In addition to posing relevant questions and steps, the software provides supplementary information and resources, like charts and tables, helping the BU to get a deeper understanding of what he is doing and to assess the fairness of the system more effectively.

Mattia Matteini: in alcuni punti come qui gia parlo di software, prematuro?

3.3 The Q/A Mechanism

The Question–Answering (Q/A) mechanism represents the core of this methodology. It has been the starting point to bridge the gap between socio-legal and technical perspectives and provides a structured way to "translate" the legal constraints into technical steps, contextualizing them in the application domain. Behind the scenes, the Q/A mechanism is a **directed graph** that represents the decision-making process.

About the Graph. Formally, the graph is defined as $G = (V, E)$, where V is the set of nodes and E is the set of directed edges. **Nodes** consist of two distinct types: *question nodes* and *answer nodes*. Each question node contains a natural language sentence expressing an inquiry, plus an identifier (unique within the whole graph) such as `Q1`, `Q2`, etc. They can also contain other arbitrary information, like the

type of question (single or multiple choice), a brief description, and so on. Answer nodes, similarly, contain a natural language sentence expressing a possible answer to a question, an identifier (unique within the whole graph) such as `Q1-A1`, `Q1-A2`, etc., and other arbitrary and useful information.

Edges are of two sorts too: either *question-to-answer* edges, denoted by $Q \to A$, or *answer-to-question* edges, denoted by $A \to Q$. Edges of the first type ($Q \to A$) indicate that question Q has as a possible answer A, whereas edges of the second type ($A \to Q$) represent that next question to be asked is Q if the selected answer to the previous question was A.

These statements assume that there cannot exist links between two question nodes or two answer nodes. It is also assumed that each question must have *at least one* outgoing edge leading to an answer node, and each answer node must have *exactly one* outgoing edge towards a question node.

Proceeding with the graph details, the **root** node is the first question to be asked to each BU, while **leaves** represent the answers to the last question, which are, technically speaking, answer nodes with *no outgoing* edges.

Finally, it is assumed that the graph is *connected*: each questionnaire is guaranteed to have a beginning and an end, meaning that there is *at least one* path from the root to each leaf.

Why a Graph? This structure ensures that the graph alternates between questions and answers, forming a coherent flow of a typical Q/A session. Moreover, this type of graph is particularly suitable for representing decision-making processes, as it allows for a structured flow of questions and answers, guiding the user through a series of steps. The graph may contain cycles, allowing the repetition of some questions (and steps), giving even more flexibility to the process.

With the branching feature, it is possible to follow multiple paths, depending on the answers given by the Business User (BU) (and so depending on the context). Remarking the order of questions, the graph effectively encodes a deterministic yet non-linear flow of questions and answers, where each answer directly influences the next question to be asked.

The graph is a data structure that can evolve and change easily over time, as the methodology actually is a **meta-**methodology, it is possible to create multi-

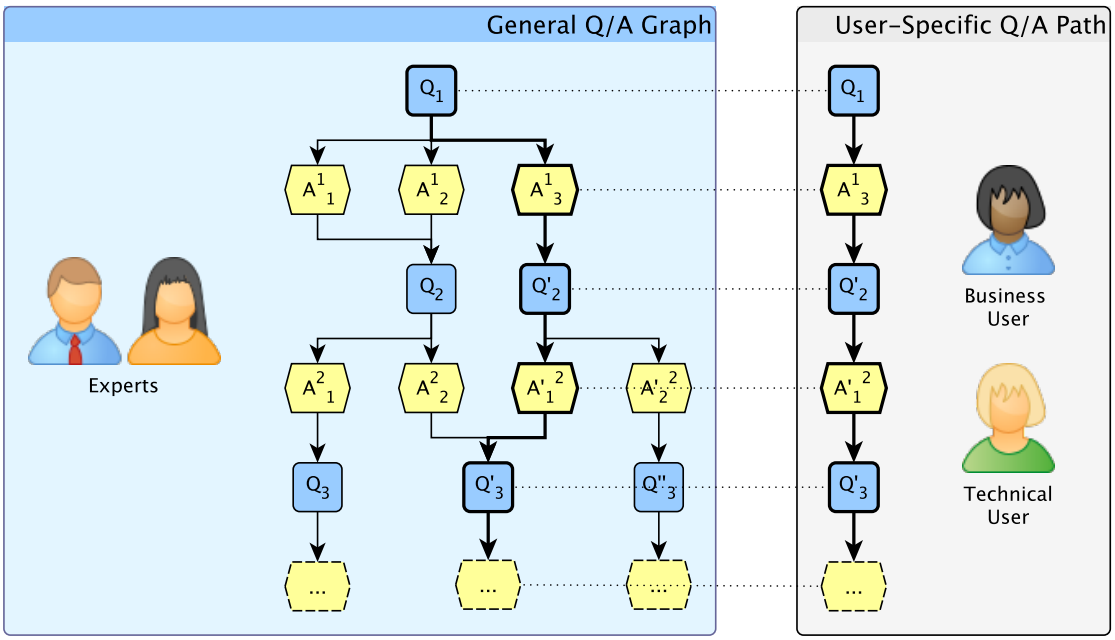


Figure 3.2: A graphical representation of the Q/A mechanism, viewed as a graph by experts and as a sequential path by business and technical users (from [CMS+25]).

ple versions of the graph. For instance, it is possible to design different pools of questions and answers, and indeed different graphs, each one tailored to a specific context or application domain. Furthermore, this flexibility enables also the possibility to adapt to any change in the cultural context taken into account, leading to a possible "methodology versioning".

General Graph vs Project-Related Graph. So far, it has been described the general blueprint of the Q/A: how to represent questions and answers, how the graph is structured, how the questionnaire and all its features look like. However, from the Business User (and Technical User) perspective, the sequence of questions seems a linear path. It is intentioned that, the person involved in the process, has the feeling of just compiling a questionnaire, where all the complexity is hidden behind the scenes. In fig. 3.2 is shown a graphical representation of the Q/A mechanism, where the flow of questions from users' perspective is defined by a path in the graph.

The end user can navigate freely through the questionnaire, answering ques-

tions and returning to previous ones in case of need. Of course in this way it is possible to change the path, and so the flow of future questions.

3.4 Automation

At this stage, it is yet vague how the methodology proposed can inject fairness measures and practical steps into the development process. Here is the point where Technical User (TU) contributes to the process, by implementing scripts and computational operations that are eventually integrated into the final system. Said that, TUs play an important role because, regardless of the complexity and variety of the needed scripts, such scripts can change case by case. Moreover, TU is still useful as source of technical knowledge and assistance for the Business User (BU).

However, while many activities performed by technical users are specific to their respective organizations, certain tasks are generalizable enough to be automated directly by the implemented methodology. Relevant examples of this, are the computation of fairness metrics and the identification of biases in datasets. These, in fact, are enough consolidated to be automated and integrated into the methodology. Rather than requiring individual technical users to develop their own solutions from scratch, the system itself can integrate these capabilities as built-in system-level functions, ensuring consistency, efficiency, and reliability across different projects.

The reification of the meta-methodology should be purposefully designed to facilitate this progression toward greater automation. When certain actions——such as evaluating responses or detecting biases——are widely applicable rather than organization-specific, they can be implemented as reusable system-level solutions. This eliminates redundancy, reducing the need for technical users to repeatedly address the same challenges independently.

These scripts to be injected to the methodology, can be provided by methodology implementation itself, developed by technical users within organizations, or contributed by third-party developers. This flexible approach ensures that automation capabilities can expand over time, adapting to emerging best practices. In the early stages of system adoption, technical users may handle certain tasks

manually, but as the methodology matures, these tasks can gradually be automated.

Technically, these scripts can be attached easily to the software implementation of such methodology, and they can be triggered whenever some kind of events occur, like the answer to a question, or termination of another computation.

Chapter 4

Design

In line with the previous chapter, the objective here is to design a software artifact that reifies the proposed meta-methodology, in order to make it usable by stakeholders to develop fair AI systems. Software design must be flexible and adaptable, as the methodology is intended to be subject to changes and improvements over time.

The final product is designed to be used by Business Users (BUs) through an intuitive web interface. Hence, frontend will interact with the underlying backend which handles the Q/A mechanism and automates technical steps.

4.1 Architecture

The software adopts an **event-driven** architecture (fig. 4.1), as events are used to handle part of the communications between the components. This design choice is motivated by the need to trigger automation scripts at specific points, for instance, when a question is answered or when a computation is completed.

The entire system is composed of the following components:

1. **Backend**: the core of the system, managing the Q/A mechanism and the automation of technical steps.

2. **Database**: a database to store the questions, answers, and other relevant information.

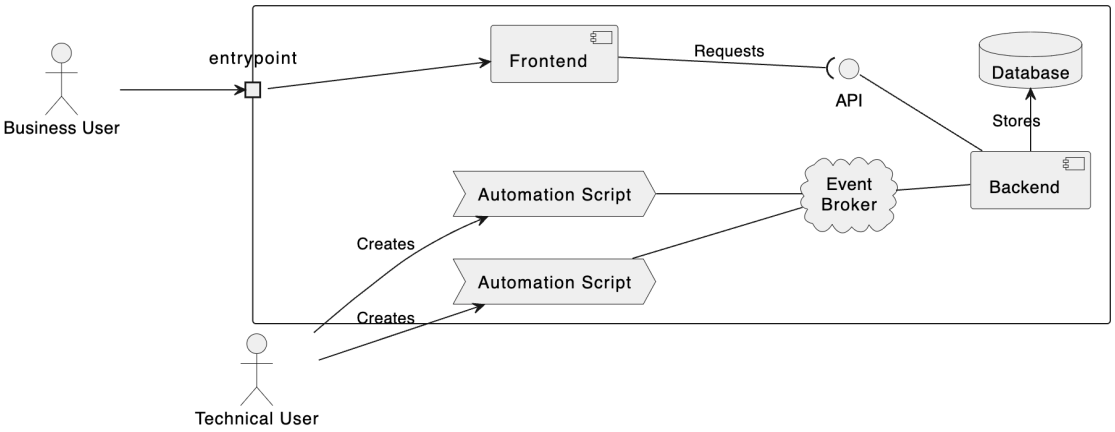


Figure 4.1: The overall software architecture of the proposed software system.

3. **Automation Scripts**: a set of pluggable scripts that automate technical steps, such as computing fairness metrics and mitigating biases.

4. **Event Broker**: component responsible for event handling in the whole system.

5. **Frontend**: the web application that allows BUs to interact with the system.

Clean Architecture. Backend component is the core of the system, it is a web service encapsulating the business logic managing the Q/A mechanism. It has been designed using Clean Architecture [Mar17], in order to separate the business logic from technical details, making the system flexible and technology-agnostic. This means that high-level business logics do not depend on low-level implementation details. And besides, it improves the separation of concerns, separating clearly core concepts, business rules, and technical details.

Therefore, backend is divided into the following layers:

1. **Domain**: the layer containing the core domain entities.

2. **Application**: the application-specific business logic.

3. **Presentation**: the interface between domain entities and the external technologies.

4. **Infrastructure**: the outermost layer, containing the implementation details technology-specific.

The dependency flow is unidirectional, from the outermost layer (Infrastructure) to the innermost one (Domain). So, for instance, Application layer and Presentation layer can depend on Domain layer, but not vice versa.

4.2 Detailed Design

This section outlines the detailed design of the software system. The design choices considered follow the principles of **Domain Driven Design (DDD)** [MT15], a software design philosophy that emphasizes the importance of domain model in the development process. The main goal of DDD is to align the software system with the domain model, ensuring that the software reflects the real-world domain as closely as possible. Although the focus of this contribution is not on the software design phase in itself, the adoption of DDD is motivated by the volatile nature of the methodology, which is expected to evolve through contributions and improvements from multiple people.

4.2.1 Structure

Projects. In order to introduce the main Q/A entities, it is necessary to define the concept of `Project`, which is the association with the AI system that the BU wants to build. In fact, each `ProjectQuestion` is related to (and also identified by) a specific `Project`. This is not just a way to distinguish multiple AI systems creations, but also to encode and store project-specific information in a "store" called `ProjectContext` (or only `Context`). This store is essential because each AI system has its own dataset, features, algorithm, and so on. Since these data can be of any nature, and are strongly volatile, the idea is to set up a key-value store, without any kind of constraints, where arbitrary (encoded) information can be stored. For instance, at some point it will be useful to store a key-value pair where the key is the name of the dataset, and the value is the encoded dataset itself. These data then will be available and useful to automation scripts and backend business logic.

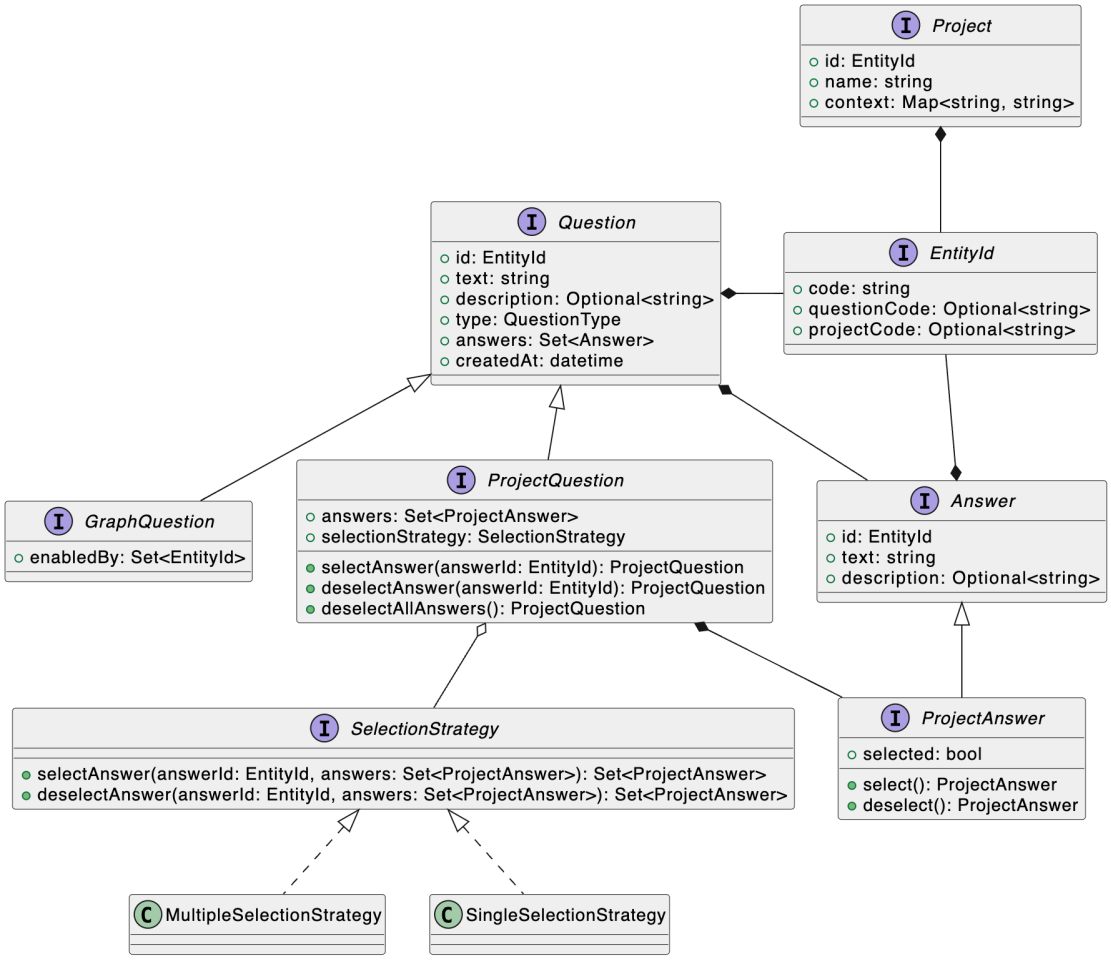


Figure 4.2: UML class diagram of main Q/A entities.

Q/A mechanism. Q/A mechanism can be intuitively mapped to two main **domain entities**: `Question` and `Answer`. However, it is necessary to distinguish between questions and answers that are part of the general graph and those that are project-related. Therefore, in the domain model are present the related entities as shown in fig. 4.2.

`GraphQuestion` represents an extension of the general `Question` entity, containing a set of answer ids that are meant to "enable" such question in the graph. In other words, a `GraphQuestion` is a question that is part of the general graph, and the set of answer ids is needed to encode the *answer-to-question* edge ($A \to Q$). With regard to admissible answers, for the `GraphQuestion` is sufficient to rely on

the general `Answer` entity because it doesn't need to have more information.

On the other hand, `ProjectQuestion`s are part of the project-related graph, and each of them is related to (and also identified by) a specific `Project`. The main difference between `GraphQuestion` and `ProjectQuestion` is that the latter has the possibility to "select" a certain answer (in the case of single choice questions) or multiple answers (in the case of multiple choice questions). Indeed, in this case, answers must have a state representing whether they are selected or not. `ProjectAnswer` entity, in fact, is intended to fulfill this need, as it contains a boolean field, and a `SelectionStrategy` handling the logics behind the selection.

For each **domain entity**, other DDD building blocks are defined:

- **Factories**: to facilitate the creation of new instances.
- **Repositories**: to manage the persistence of the entities.
- **Services**: to handle the business logic related to the entities.

From the structural view point, factories and repositories reside in the Domain layer, as they are intended to simplify the management of domain entities. Services instead, are part of the Application layer, since they compose the entire business logic of the system. However, the persistence of the entities strongly depends on technological details, so, while contracts are defined in the Domain layer, the actual implementations are pushed away to the Infrastructure layer (as shown in fig. 4.3).

The same applies to the `EventsService`, which is a particular service handling the event production and consumption. The reason is that an event-driven architecture can be achieved using different technological solutions, and business logic should remain technology-independent. Furthermore, due to this layer organization, it is possible to easily interchange the underlying technologies without affecting business logic and core domain entities (for instance using a relational database instead of a NoSQL one).

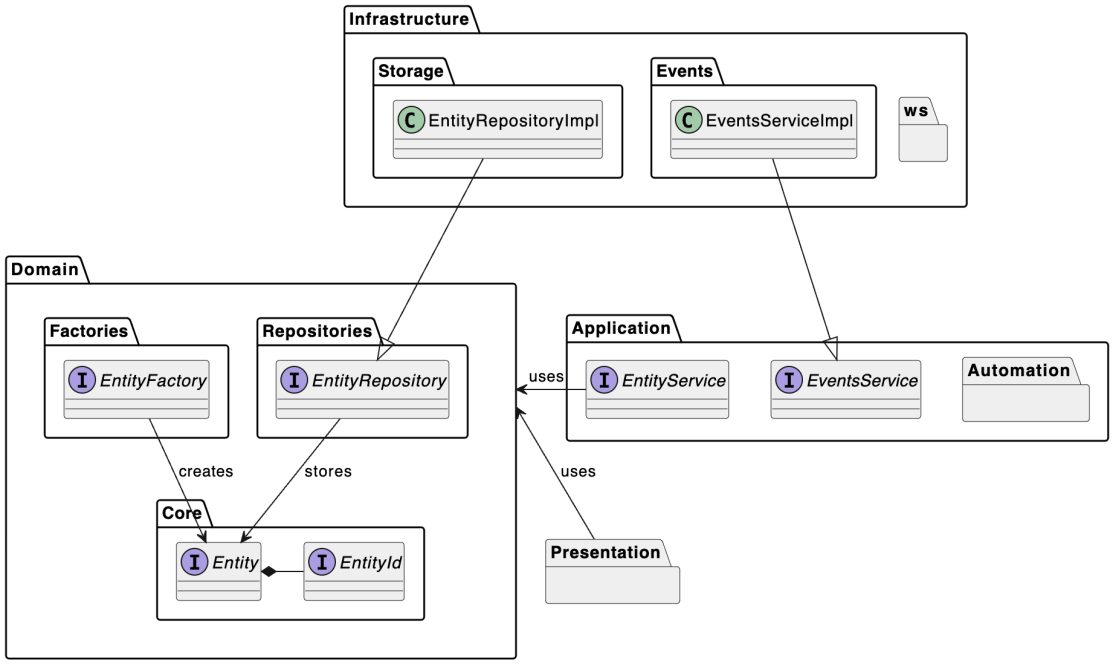


Figure 4.3: Representation of packages structure.

4.2.2 Behavior

The system behavior is mainly concentrated in **Backend** and **Automation Scripts** components.

Backend

Backend component is a web service that exposes a REST API, enabling clients to interact with questions graph and questionnaires. Its primary functionalities include modifying the question graph structure (adding, removing, modifying questions) and managing project-related questionnaires compilations. It is also possible to create new projects, and, for each of them, starting a new questionnaire session. Summarizing, backend is a simple listening web server, that manages requests and trigger events when relevant actions occur.

However, an important aspect to notice in this component behavior, is the `Context` update of various projects. `Context` updates are essential to the correct business logic functioning, because the major part of fairness-related computations strongly depends on the specific data stored in the `Context`. For example, all data

mitigation algorithms require the actual dataset, or the computation of fairness metrics needs the dataset and the selected output and sensitive features. Such computations, can be executed by both backend and automation scripts, for this reason `Context` updates represent a key point in the system behavior.

Automation Scripts

Automation Scripts are responsible for responding to specific domain events, performing necessary (fairness-related) computations, and updating system state. These scripts act as event-driven processes that listen for predefined triggers, such as selecting an answer to a question, creation of a new dataset, or completion of a fairness metric computation. Once the processing is completed, they send relevant updates back to the backend via API requests, ensuring that the system remains up-to-date and operates seamlessly without requiring manual intervention.

4.2.3 Interaction

The starting point of interactions is BU that interacts with the system through the Frontend component. Frontend is in charge of presenting questions to the BU, collecting answers, and sending them to the backend service. The backend, in turn, processes the requests and updates the questionnaire state accordingly. At this point, backend service triggers an event for each relevant action undertaken, which is caught by all Automation Scripts previously subscribed to the Broker. Each script performs its computation, and eventually sends back the results to the backend, which updates the system state and notifies the frontend to display the results (if there are any) to the BU. It is also possible that, during computations, are triggered other events for other relevant actions, like the processing of a new dataset.

The entire interaction flow is shown in fig. 4.4.

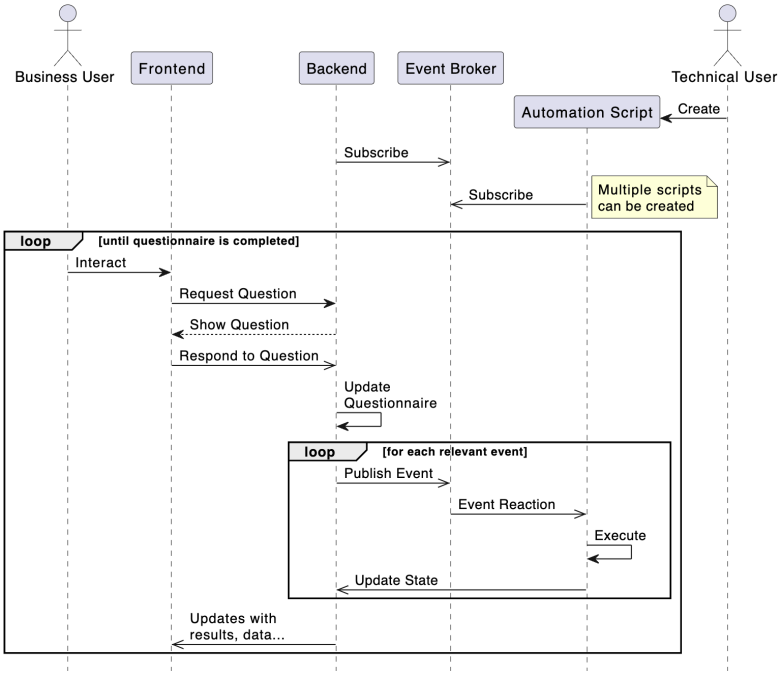


Figure 4.4: Generic sequence diagram showing the interactions between main components.

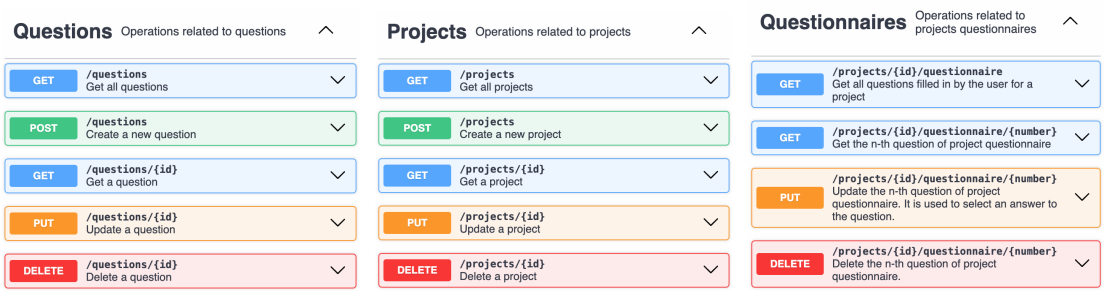
4.2.4 API

Rest

Backend service exposes an API used to interact with the three main entities: **projects**, **questions** graph, and **questionnaires**. Resources are organized following the REST principles, and each resource is identified by a unique URI. Since questionnaires are project-related, such resource is under the `Project` hierarchy. In fig. 4.5 are shown the main endpoints of the API.

Events

The other method available to interact with the system, which powers also internal communications, is the event-driven one. This represents an API slightly different from the classic REST one. Here, API endpoints are channels on which operations of type `send` and `receive` are performed. This type of API enables asynchronous communication between system components, in this case, the backend and au-



(a) API for `Question` operations (b) API for `Project` operations (c) API for `Questionnaire` operations

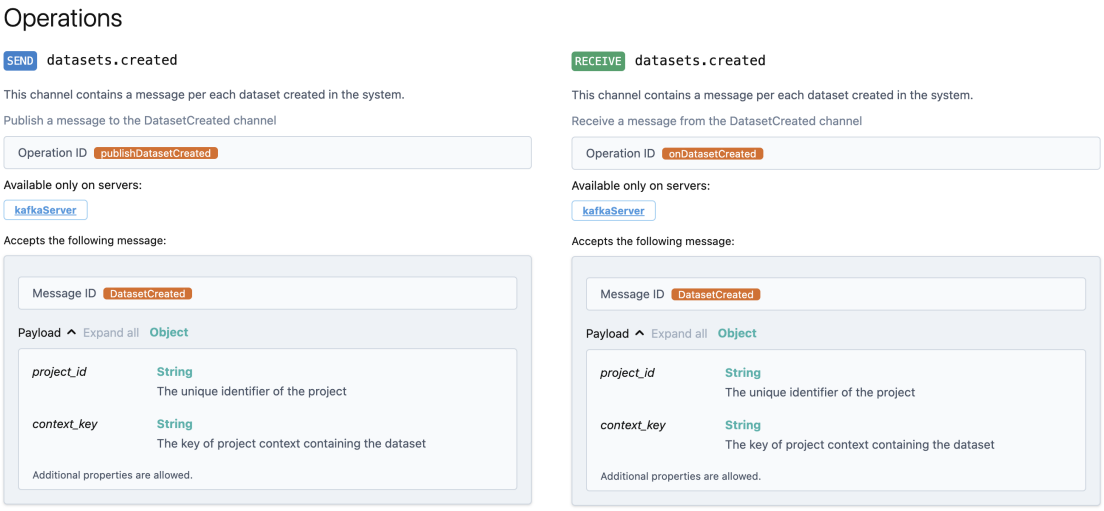
Figure 4.5: Backend REST API endpoints.

tomation scripts. That's fundamental because computations are intended to be non-blocking in order to allow BU to proceed in the questionnaire compilation.

A documentation example of such API is shown in fig. 4.6 using Async API specification (https://www.asyncapi.com). The main concepts in this are:

- **Channels**: Specific topics where events are published. Each topic corresponds to a particular type of event.
- **Operations**: Primarily `send` (`publish`), to publish events on the channel, and `receive` (performing if `subscribed` to channel), to receive messages when events are published.
- **Messages**: Schemas of the objects published on channels.

This API is highly flexible, in fact, when an automation script is created, it can be attached to a new channel created ad hoc for the script specific purpose. For instance, a new script can be plugged in simply creating a new channel and subscribing at it.



(a) Publish operation for a channel (b) Receive operation for a channel

Figure 4.6: Async API channels example.

Chapter 5

Implementation

This chapter delves into the technical details of the software implementation which reifies the proposed meta-methodology. It provides an overview of the deployed architecture, and, basing on the design choices made in the previous chapter, it explains how the various components are actually implemented. The chapter also covers the deployment phase of the software system, how to set up the configuration environment, and in the end how to run it. Finally, the chapter concludes covering some use cases to demonstrate the system in action, providing also Business User (BU) perspective through the web interface.

5.1 Deployment

Backend. Backend component is implemented as a web service in Python. The choice aims to reduce the abstraction gap because python comes with consolidated frameworks that facilitate development of AI systems. Python represents also a good choice thanks to its readability and simplicity, which make it easier to be maintained and extended by developers who were not originally involved in the project. Flask-restful framework has been used to ease the development of the web service, as it allows the creation of a RESTful API with few lines of code (listing 5.1).

Listing 5.1: Example resource creation with Flask-restful.

```
questions_bp = Blueprint("questions", __name__)
api = Api(questions_bp)


class QuestionResource(Resource):

    def get(self, question_code=None):
        if question_code:
            graph_question_id: EntityId = GraphQuestionFactory.id_of(code=
                question_code)
            question: Optional[GraphQuestion] = question_service.
                get_question_by_id(
                graph_question_id
            )
            if question:
                return serialize(question), StatusCode.OK
            else:
                return "Question not found", StatusCode.NOT_FOUND
        else:
            all_questions: List = question_service.get_all_questions()
            all_questions.sort(key=lambda q: q.id)
            return [serialize(question) for question in all_questions], StatusCode
                .OK

    def post(self):
        new_question: GraphQuestion = deserialize(request.get_json(),
            GraphQuestion)
        try:
            question_service.add_question(new_question)
        except ConflictError as e:
            return e.message, e.status_code
        return serialize(new_question.id), StatusCode.CREATED

    ...

api.add_resource(QuestionResource, "/questions", "/questions/<string:question_code
    >")
```

Listing 5.2: Example resource creation with Flask-restful.

```python
from typing import List, Callable

from kafka.consumer.fetcher import ConsumerRecord

from application.events import EventsService
from infrastructure.events import Producer, Consumer


class KafkaEventsService(EventsService):

    def __init__(self):
        self._producer = None

    def publish_message(self, topic: str, message: str) -> None:
        if self._producer is None:
            self._producer = Producer()
        self._producer.produce(topic, message)

    def start_consuming(
        self, topics: List[str], handler: Callable[[ConsumerRecord], None]
    ) -> None:
        Consumer(topics, handler).start_consuming()
```

Event Broker. Apache Kafka is used as the event broker to implement the event-driven architecture. Kafka is a distributed streaming platform that provides high throughput, scalability, and fault tolerance. The python client of Kafka has been used to interact with the broker, allowing the backend and automation scripts to publish and subscribe to events. This is achieved by an ad hoc service implemented in the infrastructure layer (listing 5.2).

Database. For the database, the choice fell on graph database, as it fits well with the graph structure of the Q/A mechanism, and the technical implementation chosen is Neo4j. It allows for efficient traversal and querying of the graph, making it easier to manage operations on questions and answers.

Entities are mapped to the database as nodes and relationships. In particular, **question graph** is mapped as the following: questions and answers are represented as nodes, while the edges between them are represented as relationships. More precisely, let's consider Q_G is the node representation of question Q and A_G is the node representation of answer A (both in general graph). Edges in Neo4j are associated with a label, so edges of type $Q \to A$ are stored as $Q_G \xrightarrow{\text{HAS\_ANSWER}} A_G$, while edges of type $A \to Q$ are actually stored as $A_G \xrightarrow{\text{ENABLED\_BY}} Q_G$.

With respect to the `Project`, let's consider P as the node representation of project. Key-value pairs of its `Context` are stored in P. This is straightforward because each node in Neo4j can store arbitrary key-value pairs, making it easy and flexible to save context data. In order to save space, context data that are common to all projects are stored in a separate node named `PublicContext`, which is referred if a key is not found in the specific project node on which the context data is requested.

Eventually, each **questionnaire** (if exists) is linked to a specific project P through a relationship $P \xrightarrow{\text{QUESTIONNAIRE}} Q_P$, where Q_P is the root of the questionnaire graph (the first question asked), and it is created upon the root of the general graph. Next questions in the questionnaire path are linked to the previous ones through relationships of kind $Q'_P \xrightarrow{\text{NEXT}} Q''_P$.

To keep track of the answers given by the BU, each answer is linked to the corresponding question through a relationship $Q_P \xrightarrow{\text{HES\_SELECTED}} A_P$, where A_P is the node representation of the answer given by the BU.

Automation Scripts. Automation scripts are processes created ad hoc to perform specific computations. They just need to adhere to the Async API specification to be seamless integrated into the system.

These scripts can actually be implemented in any language, but the ones implemented so far are created in Python. One motivation is that in this way scripts can share the backend code benefiting from reusability. Python is also recommended due to its extensive support for machine learning libraries and frameworks.

Frontend. The frontend is implemented as a Single Page Application (SPA) using React and Next.js. This choice leverages the component-based architecture of React, which promotes reusability and maintainability of the code. Next.js enhances the development experience by providing server-side rendering and static site generation. The component-based architecture fits well because the questionnaire can evolve, and the frontend must be arranged to be changed accordingly. A web-based technology has been chosen for several reasons: (i) cross-platform compatibility, (ii) ease deployment and maintenance, (iii) accessibility, (iv) powerful features offered by modern frameworks.

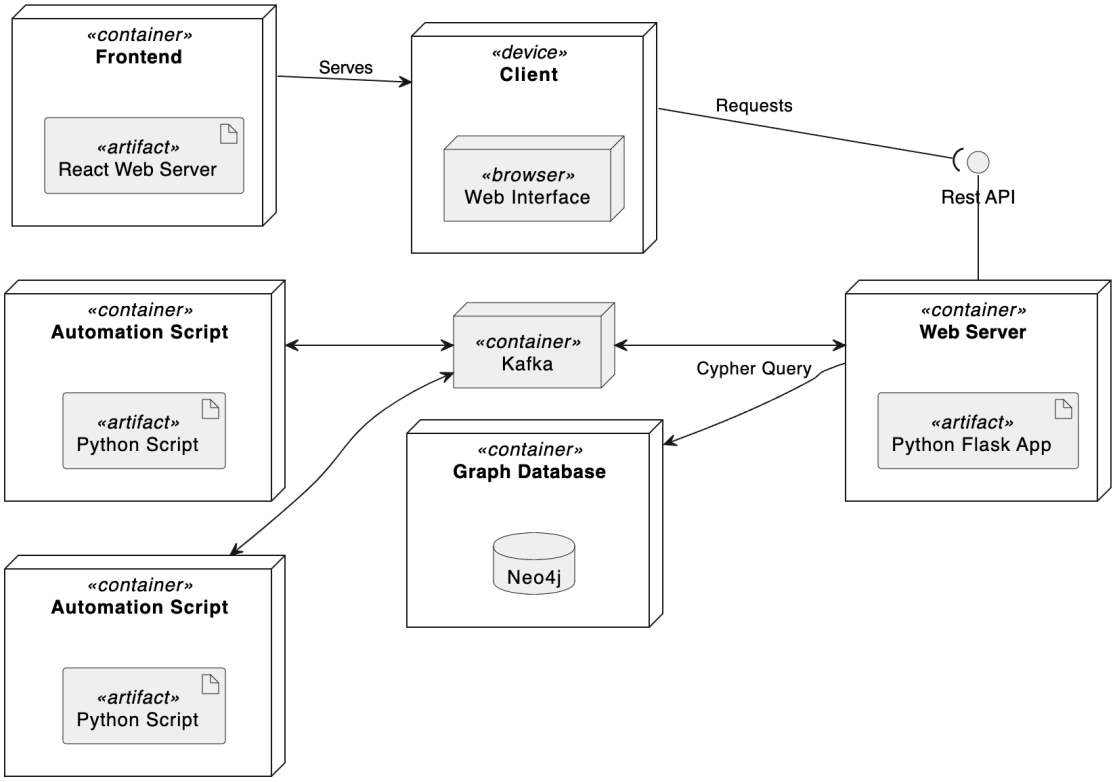


Figure 5.1: Deployment diagram of the system.

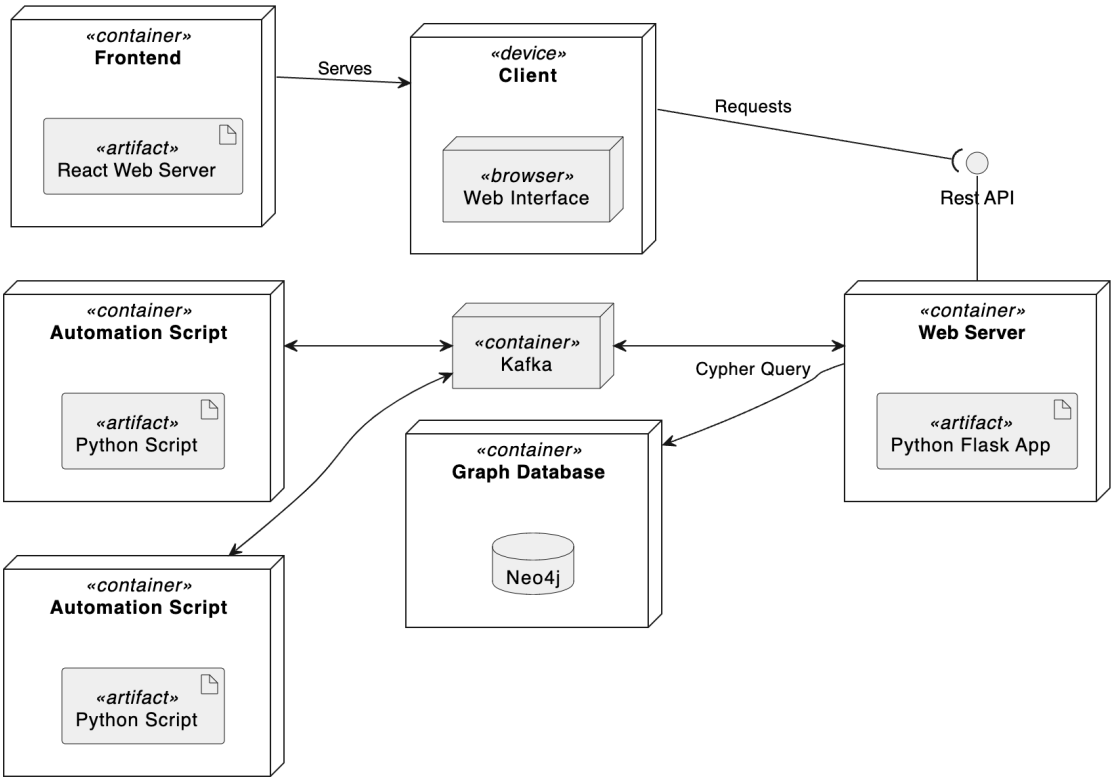


Figure 5.2: Deployment diagram.

5.2 Configuration

5.3 Testing

5.4 Use Cases

5.4.1 GUI

5.5 GUI

5.6 Broker

Chapter 6

Validation

Chapter 7

Conclusions

Bibliography

[CCMO23] Roberta Calegari, Gabriel G. Castañé, Michela Milano, and Barry O'Sullivan. Assessing and enforcing fairness in the AI lifecycle. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 6554–6562. ijcai.org, 2023.

[CMS+25] Giovanni Ciatto, Mattia Matteini, Laura Sartori, Maria Rebrean, Catelijne Muller, Andrea Borghesi, and Roberta Calegari. AI-fairness: the FairBridge approach to practically bridge the gap between socio-legal and technical perspectives. In *Proceedings of the 58th Hawaii International Conference on System Sciences*, page 6499, 2025. (In press).

[IML23] Zahid Irfan, Fergal McCaffery, and Róisín Loughran. Evaluating fairness metrics. In Ludovico Boratto, Stefano Faralli, Mirko Marras, and Giovanni Stilo, editors, *Advances in Bias and Fairness in Information Retrieval - 4th International Workshop, BIAS 2023, Dublin, Ireland, April 2, 2023, Revised Selected Papers*, volume 1840 of *Communications in Computer and Information Science*, pages 31–41. Springer, 2023.

[Mar17] R.C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Robert C. Martin series. Prentice Hall, 2017.

[MT15] Scott Millett and Nick Tune. *Patterns, principles, and practices of domain-driven design*. John Wiley & Sons, 2015.

Acknowledgements

Optional. Max 1 page.