Implementing a Survey Feature for Measuring Developer Sentiment

BACHELOR THESIS

Jonas Hemkendreis

Submitted on 25 September 2025

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Thomas Wolter, M.Sc.
Prof. Dr. Dirk Riehle, M.B.A.

Friedrich-Alexander-Universität
Faculty of Engineering

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Erlangen, 25 September 2025

License

This work is licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0), see https://creativecommons.org/licenses/by/4.0/

Erlangen, 25 September 2025

Abstract

Measuring developer productivity is hard. Often times, qualitative human factors such as developer sentiment are neglected in favor of easier to obtain quantitative metrics such as Lines of Code. This thesis advocates for measuring productivity in its multidimensional form, integrating both quantitative and qualitative data. We present the design and implementation of a survey feature intended to measure developer sentiment. Grounded in design science research, we utilize requirements engineering techniques to derive system requirements, evaluate solution alternatives, and implement a viable candidate into a larger productivity measurement ecosystem. A demonstration with three developers confirmed the feature's effectiveness in measuring developer sentiment. As such, this thesis contributes a concrete implementation of a survey feature that serves as a foundation for improving future productivity metrics and research.

Contents

List of Figures

List of Tables

Acronyms

IS	Information System
DSRM	Design Science Research Methodology
SPACE	Satisfaction, Performance, Activity, Communication, and Efficiency
LOC	Lines of Code
SRS	Software Requirements Specification
RE	Requirements Engineering
EARS	Easy Approach to Requirements Syntax
CRUD	Create, Read, Update, and Delete
UR	User Requirement
FR	Functional Requirement
NFR	Non-functional Requirement
UML	Unified Modeling Language
ER	Entity-Relationship
UI	User-Interface
RLS	Row Level Security

1 Introduction

1.1 Motivation

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."

Bill Gates (LINFO, 2005)

The challenge of measuring developer productivity has long been a topic of discussion in computer science and related fields (Forsgren et al., 2021; Jaspan & Sadowski, 2019). A comprehensive metric could provide benefits both in terms of improving engineering outcomes and in supporting developer satisfaction and well-being (Forsgren et al., 2021, p. 1). Yet, traditional approaches that rely on one-dimensional metrics have consistently fallen flat in their attempt to capture the true complexity of productivity (Jaspan & Green, 2025, p. 14). Quantifying such a nuanced concept requires more sophisticated approaches (Jaspan & Sadowski, 2019, p. 15).

One promising direction is the development of multidimensional approaches, such as composite indices, which integrate diverse facets of productivity (Brown et al., 2023, p. 16). These approaches acknowledge that productivity cannot be measured by its outputs alone, but must also take inputs such as human-centered factors into account. Research shows that some of the strongest influences on productivity are non-technical, including motivation, sentiment, and responsibility (Murphy-Hill et al., 2021, pp. 586–589). Owing to the qualitative nature of these factors, they are "best captured with surveys" (Forsgren et al., 2021, p. 7).

This thesis contributes to this research by designing and implementing a survey feature to measure developer sentiment. The survey is implemented as a configurable feature in an existing ecosystem, namely MECOIS, enabling flexible use in different research or organizational contexts. In doing so, the feature provides an additional data source to complement and bolster confidence in existing productivity measurements done by MECOIS.

1.2 Methodological Framework

For research concerned with the development of novel artifacts, literature suggests the application of *design science research* (Hunziker & Blankenagel, 2024, p. 98). Given that the to-be-designed artifact lies in the Information System (IS) domain, the Design Science Research Methodology (DSRM) for IS proposed by Peffers et al. (2007) is considered a suitable framework for this thesis and will be used henceforth. The following section provides a brief overview of the model itself and how it will be applied in this work.

In their paper, Peffers et al. (2007) present a DSRM process model that consists of the six primary activities outlined in Figure 1.1.

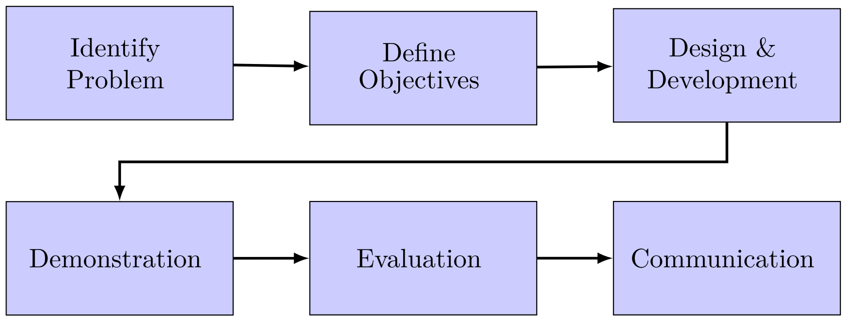


Figure 1.1: The DSRM process model (based on Peffers et al., 2007, p. 54)

Among the four proposed entry points into the research process, this thesis adopts the problem-centered initiation path. This choice is justified as the conceptualization of the artifact stems from the observation of a problem (Peffers et al., 2007, p. 56). Consequently, all six phases of the DSRM are present in this work:

In chapter 2, we identify the problem we intend to solve and motivate a need for a solution. This will be substantiated by analysing previous literature. Subsequently, we infer the objectives of a suitable solution from the identified problems in chapter 3. These objectives will in turn serve as an evaluation and implementation baseline for the solution.

Building on said objectives, chapter 4 deals with transforming these higher-level objectives into concrete, measurable system requirements and the selection among solution alternatives. Once a viable solution has been identified, chapter 5 addresses the implementation aspects of that solution.

The functionality and applicability of the solution are then demonstrated in chapter 6, with its effectiveness assessed in chapter 7 by comparing the achieved results against the previously defined objectives. Finally, chapter 8 summarizes our findings and concludes this paper.

2 Problem Identification

This chapter aims to establish a well-founded definition of developer productivity, examine the multiple dimensions that contribute to its complexity, and finally, justify the use of surveys as a valid and effective instrument for measuring developer sentiment.

2.1 Developer Productivity

2.1.1 Definition and Dimensions

Developer Productivity is often defined in terms of efficiency, inputs and outputs (Meyer et al., 2014, p. 3). In other words, how effective developers can transform development activities into meaningful (software) artifacts over a given time frame. This is a description of a process. Thus, developer productivity in practice has often not been defined by its goals and outcomes, but by the way in which it is *measured*, for example:

- Counting new Lines of Code (LOC) per developer (Walston & Felix, 1977, p. 54–73)
- Deriving function points based on inputs, outputs, inquires, logical files and interfaces (Jones, 1994, p. 100)
- Leveraging cost estimation models such as COCOMO II (Boehm et al., 2000)

Given the inconsistent use of the terms *metric* and *measurement* in software engineering literature, we adopt the definitions by Fenton and Bieman (2014): *Measurement* is the process of quantifying productivity, while a *metric* is the resulting computed value.

Revisiting the earlier examples, it becomes clear that most of them are of quantitative nature. While such measures can be useful, they tend to reduce productivity to simple, log-based metrics and miss the human aspects of software engineering (Meyer et al., 2017, p. 105). As Jaspan and Sadowski (2019) state in their paper *No Single Metric Captures Productivity*, a multifaceted approach is essential to truly grasp the complexity of productivity.

One such approach is to define productivity as a multidimensional concept. A prominent example is the SPACE framework by Forsgren et al. (2021), which defines productivity across five dimensions: Satisfaction, Performance, Activity, Communication, and Efficiency (SPACE).

Each of these dimensions reflect a different aspect of software development work. Unlike traditional measures, SPACE stresses to not only utilize log-based metrics, but also complex qualitative metrics such as productivity perception (Forsgren et al., 2021, p. 14). Both industry and academia have increasingly adopted multidimensional approaches, including initiatives like Google's *EngSat* (D'Angelo et al., 2024) and DORA's four key software delivery metrics (DevOps Research and Assessment (DORA), 2021).

Another example is the MECOIS Productivity Index, developed by the Professorship for Open-Source Software at the University of Erlangen, which constructs a composite productivity index (Wolter & Riehle, 2025). Currently, the index primarily relies on log-based metrics within the Performance and Activity dimensions. This paper proposes extending MECOIS to include qualitatively-assessed metrics and dimensions by, for instance, measuring developer sentiment.

2.1.2 The Dimension of Developer Sentiment

Developer sentiment, also referred to as developer satisfaction or well-being, is a key dimension of productivity (Forsgren et al., 2021, p. 2). For instance, recent research indicates that sentiment and productivity are correlated, with sentiment even serving as a leading indicator of productivity (Murphy-Hill et al., 2021, p. 586). Developer sentiment is characterized by being fundamentally subjective, as each developer experiences productivity through his own perception filter (Meyer et al., 2017, p. 105). As such, sentiment is difficult to measure accurately (Jaspan & Green, 2023, p. 27). While some metrics, such as retention rate, may be easy to capture, others – like flow, focus and friction (Brown et al., 2023, p.16) – require astute observations. According to Jaspan and Green (2025, p. 16) and Forsgren et al. (2021, p. 7), these metrics are best measured with surveys.

2.2 Surveys as a Tool for Measuring Sentiment

Surveys offer an effective approach to measuring qualitative data such as developer sentiment, flow, and well-being, which are otherwise difficult to capture. Within the broader framework of a composite productivity index, surveys enable us to:

- improve confidence in the interpretation of quantitative data;
- gather information quantitative measurements cannot;
- quickly gather feedback on evolving trends; and
- provide additional context to productivity dimensions that where previously only captured via quantitative data (Jaspan & Green, 2025, p. 16).

Having established why surveys are valuable, it is important to understand what makes them effective. Research indicates that surveys should meet the following criteria:

- Surveys should avoid intrusive design patterns (Jaspan & Green, 2025, p. 16).
- The survey infrastructure should be configurable to address the specific needs of different organizations and teams (D'Angelo et al., 2024, p. 21).
- Each survey should have a clear and communicated objective that cannot already be captured by other data sources (D'Angelo et al., 2024, p.23).
- Survey results should be displayed through intuitive, easy-to-understand dashboards (D'Angelo et al., 2024, p. 20).
- Plan for longitudinal surveys to track changes in developer sentiment over time (D'Angelo et al., 2024, p. 19; Fox et al., 2000, p. 6).
- Be aware of survey limitations, particularly regarding the sampling frame and the subjectivity of question wording (Fox et al., 2000, pp. 8, 26).
- Design questionnaires with a mix of open-ended, partially closed-ended and closed-ended questions, the latter including multiple choice, single choice, and Likert-scales (Linåker et al., 2015, pp. 37–43).
- Carefully determine what to measure and communicate accordingly, keeping in mind Goodhart's law: "When a measure becomes a target, it ceases to be a good measure." (Strathern, 1997)

3 Objective Definition

To address the shortcomings of existing productivity measurement approaches in MECOIS research, the goal of this thesis is to implement a survey-based feature for measuring developer sentiment. This feature will be embedded within an existing application and should comply with the following criteria:

Objective 1: The feature must include a configurable survey mechanism tailored to collect and analyze (self-reported) data on developer sentiment, flow, friction, and perceived productivity.

Objective 2: The feature must be adaptable to different organizational or project-specific contexts, allowing administrators to modify survey questions, frequency, and target groups.

Objective 3: The feature's survey design, data collection, and analysis methods must adhere to academic standards and align with current research and best practices in measuring developer sentiment (D'Angelo et al., 2024; Jaspan & Green, 2025).

Objective 4: The implementation should prioritize ease-of-use and avoid intrusive design patterns to ensure high response rates and sustained engagement from developers (Jaspan & Green, 2025, p.16).

Objective 5: The survey feature should integrate seamlessly with existing code at MECOIS and use only non-restrictive open-source licenses (e.g., MIT) to maximize adaptability, reuse, and compliance.

4 Solution Design

In order to produce an artifact that satisfies the objectives set forth in chapter 3, we must first transform these higher-level objectives into implementable requirements. To achieve this, we will employ methods and practices of Requirements Engineering (RE). The outcome of these efforts is a Software Requirements Specification (SRS) sheet, which will serve as the basis for evaluating software alternatives as well as implementing and evaluating the chosen variant. The overall process is outlined in Figure 4.1.

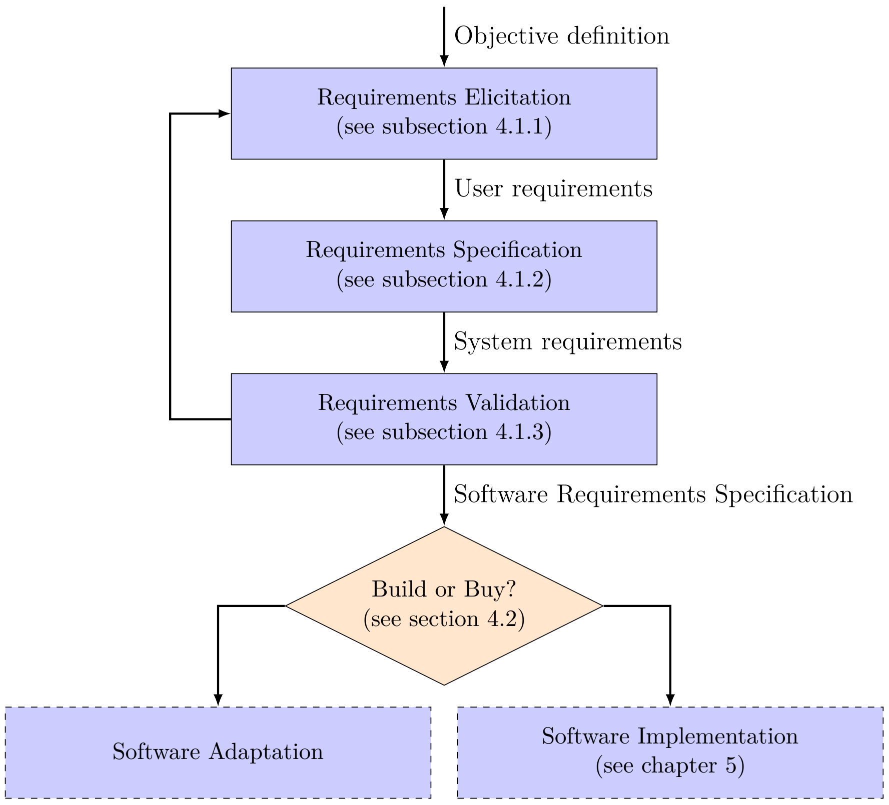


Figure 4.1: The solution design process. Blue boxes represent process activities, arrows denote output artifacts, and dashed boxes highlight activities beyond the scope of solution design.

4.1 Requirements Engineering for the Survey Feature

To create the aforementioned SRS sheet, we shall utilize RE techniques and processes. According to 'IEEE 29148-2018' (2018), RE is well-suited for this task, as it yields requirements that:

- are in reference to a defined system;
- establish a common understanding among stakeholders;
- provide a reference for verifying the produced results;
- can be feasibly implemented; and are therefore
- in-line with our objective definitions from chapter 3.

The process by which we derive these requirements is outlined in Figure 4.1 and will be carried out in the following subsections.

4.1.1 Requirements Elicitation

Requirements elicitation and analysis involve identifying the system domain and capturing the needs of relevant stakeholders (Sommerville, 2018, p. 113). This process was already initiated in chapter 2, in which the main problems were identified, and further refined into objectives in chapter 3. These objectives now serve as input to the elicitation process and are subsequently transformed into well-structured user requirements.

According to Sommerville (2018, p. 102), "user requirements are statements, in natural language [...], of what services the system is expected to provide to system users and the constraints under which it must operate." For the sake of structural consistency, we will narrow this definition such that our syntax only supports statements made using both the Easy Approach to Requirements Syntax (EARS) and MoSCoW pattern:

- Requirements must be expressed using EARS generic requirement syntax or a specialization thereof: `<optional preconditions> <optional trigger>` the `<system name>` shall `<system response>` (Mavin et al., 2009, p. 319). For example, "the feature should allow users to create, edit, inspect and delete surveys" can be formulated as "The survey system shall enable privileged users to Create, Read, Update, and Delete (CRUD) a survey."
- Requirements must be prioritized based on being either *required*, *important*, *desirable*, or *optional* , called *M*ust, *S*hould, *C*ould and *W*on't requirements respectively (Rod Stephens, 2022, p. 57).

Furthermore, to maintain a structured record of requirement origins and their relationships, we adapt a hierarchical ID scheme of the form

$$\text{REQ } \mathrm{X}_1.\mathrm{X}_2.\cdots.\mathrm{X}_N,$$

where REQ $\in$ {User Requirement (UR), Functional Requirement (FR), Non-functional Requirement (NFR)} and $N, \mathrm{X}_i \in \mathbb{N}$. The transformation of objective definitions into user requirements is shown in Table 4.1.

Table 4.1: Overview of user requirements

ID	User Requirement	MoSCoW	Source
UR 1	The survey system shall enable privileged users to CRUD a survey.	Must	Objective 1 Objective 3
UR 1.1	When a privileged user creates or updates a survey, the survey system shall provide meta configuration options.	Must	Objective 2
UR 1.2	When a privileged user creates or updates a survey, the survey system shall provide question configuration options.	Must	Objective 2
UR 2	The survey system shall enable privileged users to CRUD a response.	Must	Objective 1 Objective 3
UR 3	The survey system shall enable privileged users to analyze responses to a survey.	Must	Objective 1 Objective 3
UR 4	The survey system shall verify and validate the user's authentication, privileges, and inputs.	Must	Objective 1
UR 5	The survey system shall conform to usability best practices.	Must	Objective 4
UR 6	The survey system shall comply with data privacy regulations.	Must	Objective 3
UR 7	The survey system's code shall conform to MECOIS contribution guidelines.	Must	Objective 5

4.1.2 Requirements Specification

In the specification phase, the elicited user requirements are translated into more granular functional and non-functional system requirements. Together with system models, these requirements form the SRS, which serves as the basis for evaluating potential solution alternatives. Whereas functional requirements describe what an application should do, non-functional requirements specify a system's quality attributes and constraints (Rod Stephens, 2022, p. 63; Cha et al., 2019, p. 81). Inherently, user requirements primarily fall into the former category, with non-functional requirements often having to be inferred indirectly (Rod Stephens, 2022, p. 63).

Due to the sheer volume of functional and non-functional requirements, the complete listing is included in the SRS only (see section A), concretely in Table A.2 and Table A.3. Here, we only provide a summary of selected system requirements that merit explicit emphasis, shown in Table 4.2.

Table 4.2: Overview of system requirements

ID	System requirement	MoSCoW	Source
FR 1	When a privileged user creates or updates survey metadata, the survey system shall provide the following configuration options: title, description, survey type, owner, availability, target group, languages, anonymous response setting, visibility status, ability to edit responses after submission, and a unique slug.	Must	UR 1.1
FR 2	When a privileged user creates or modifies a survey's questions, the survey system shall allow configuring whether a question is mandatory and selecting question types `short`, `long`, `scala`, `date`, `radio`, and `checkboxes`.	Must	UR 1.2
FR 3	When a privileged user creates or updates a response, the survey system shall present answer forms consistent with the survey's question types and metadata configuration (e.g., anonymity).	Must	UR 2

Table continues on the next page →

Overview of system requirements continued

ID	System requirement	MoSCoW	Source
FR 5	When a user requests surveys from the backend, the survey system shall enforce proper CRUD privileges.	Must	UR 4 UR 6
NFR 1.1	The survey system's frontend shall be implemented using `React`.	Must	UR 7
NFR 1.4	The survey system shall only rely on non-restrictive dependencies (e.g., MIT License).	Must	UR 7
NFR 1.5	The survey system's backend shall use `Supabase` for authentication and relational database storage.	Must	UR 7
NFR 2	The survey system shall handle errors gracefully and notify users in a clear and non-technical way.	Must	UR 5
NFR 3	The survey system shall notify privileged users about surveys in a non-intrusive way.	Should	UR 5

In addition to these system requirements, we also make use of the Unified Modeling Language (UML) defined by the Object Management Group. The purpose of UML is to bridge the gap between natural language specifications and software implementation by providing object-oriented visual modeling tools that support both the design and implementation of software systems (Object Management Group®, 2017, p. 1). To illustrate the system behavior at a high level, Figure 4.2 presents the core survey use cases. This diagram captures the essential interactions between different actors and illustrates which CRUD operations and other functionalities are available to which user roles. At the data layer, Figure 4.3 shows an Entity-Relationship (ER) diagram of the database design, based on the notation defined in Elmasri and Navathe (2016, p. 59). Lastly, from a behavioral perspective, Figure 4.4 shows a sequence diagram of the survey creation logic, covering both the successful publishing process and the alternative error-handling branches. This logic can also be adapted for the response creation logic.

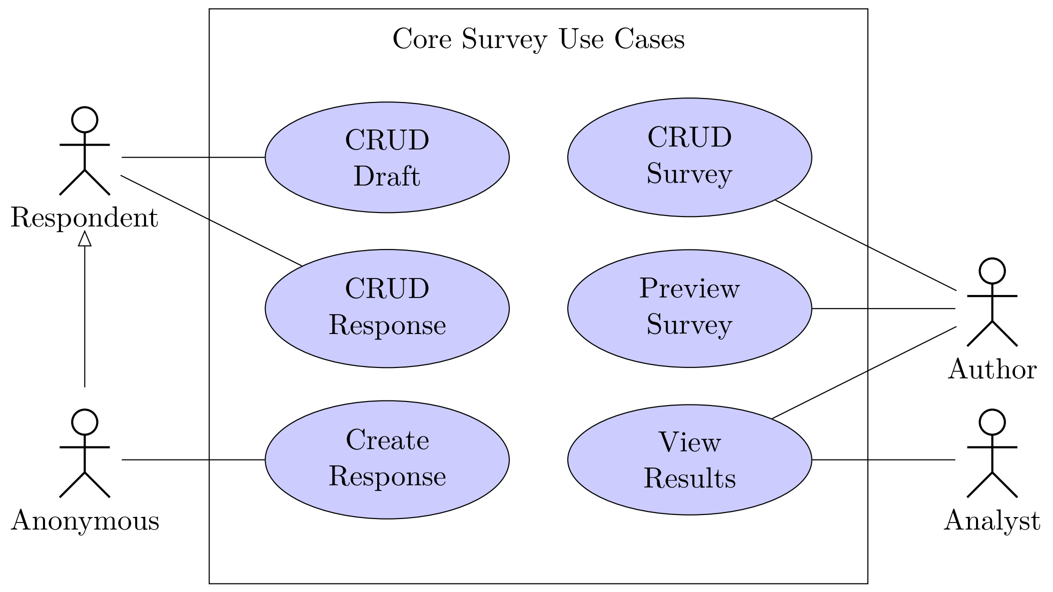


Figure 4.2: A use case diagram of the survey system's core features

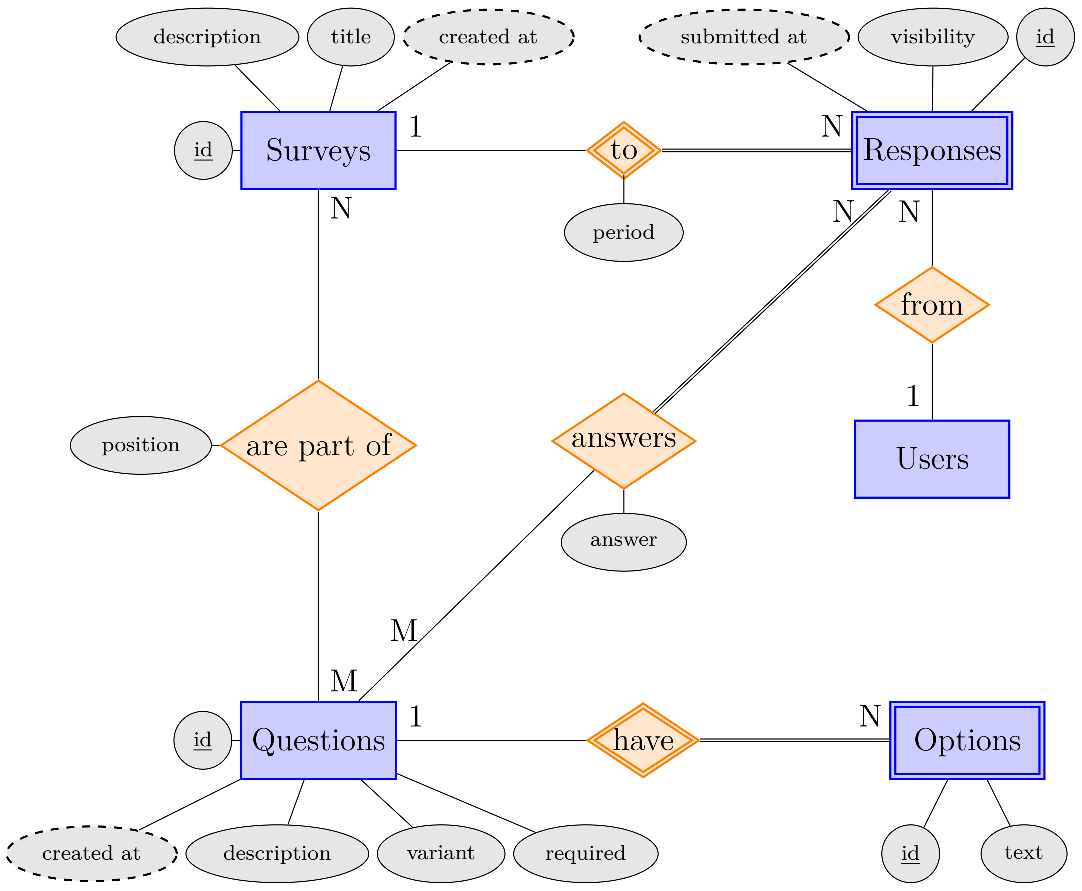


Figure 4.3: An ER-diagram of the survey system's database

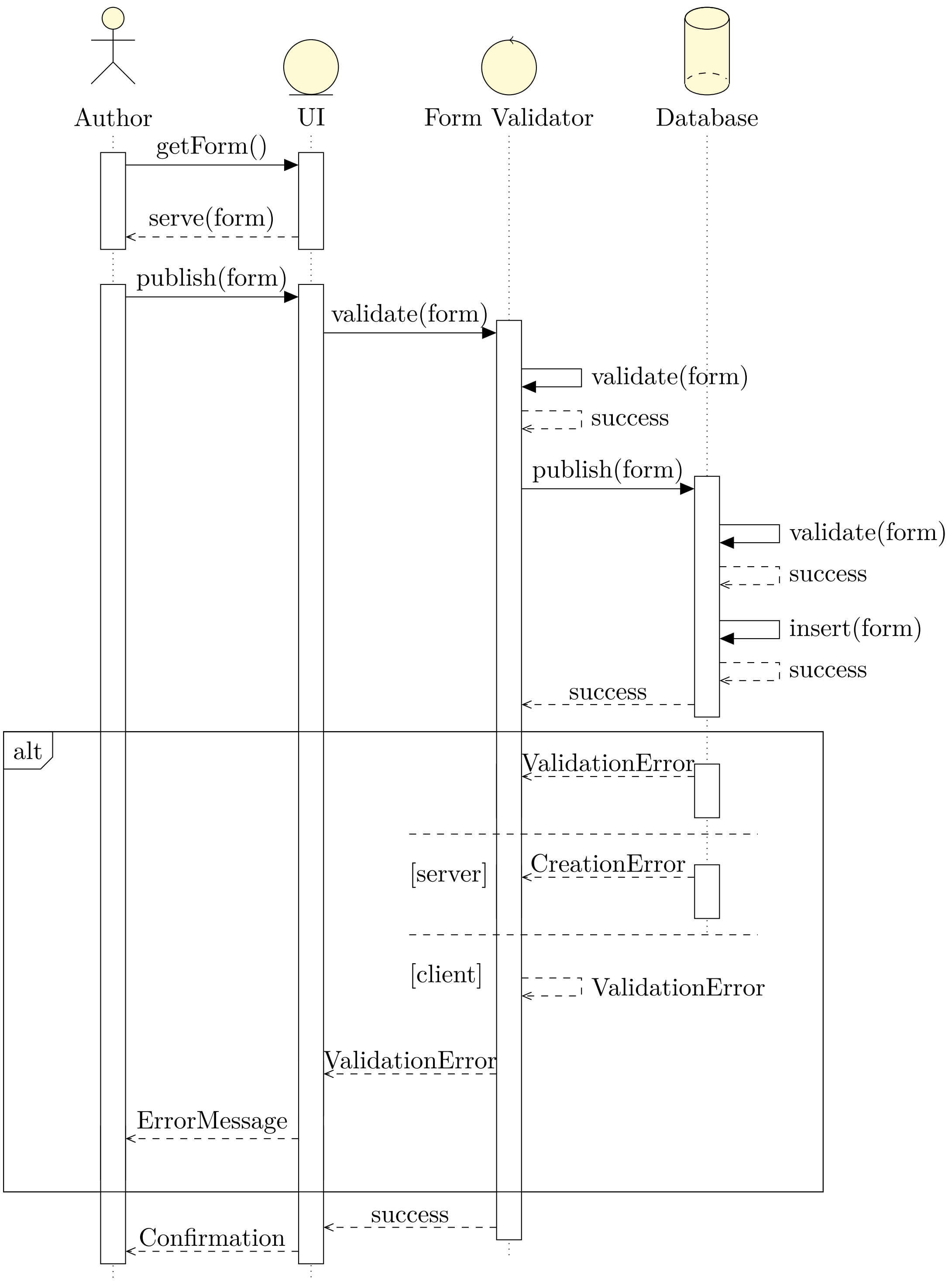
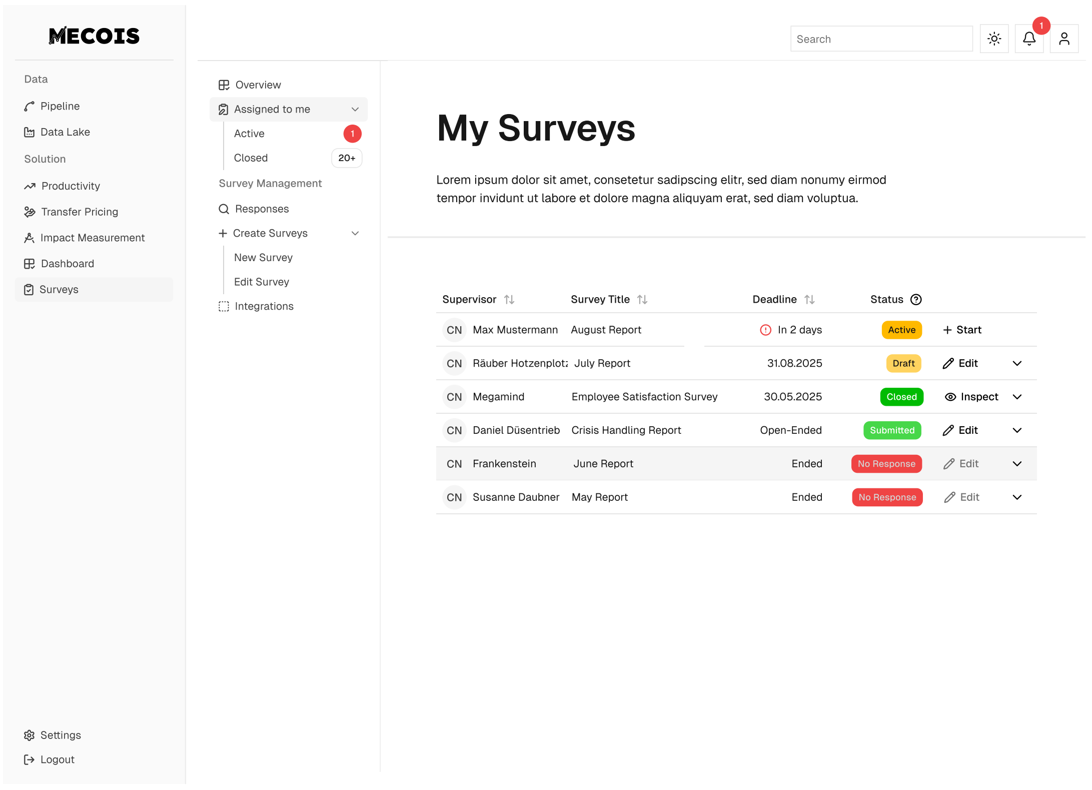


Figure 4.4: A sequence diagram of the survey creation logic. The API entity has been removed for brevity.

4.1.3 Requirements Validation

The requirements validation serves to confirm that the requirements and system models described in the previous section accurately reflect the elicited objectives (Sommerville, 2018, p.129). Upon confirmation, the validated SRS then provides the foundation for the subsequent solution evaluation stage.

As part of this validation, a User-Interface (UI) mockup was created using the prototyping tool Figma by Figma, Inc. (2025). The mockup was developed with NFR 1.2 in mind and therefore utilizes `shadcn/ui` UI components. An example page is shown in Figure 4.5; further illustrations are provided in the SRS (see subsection A.4).



My Surveys

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

Supervisor		Survey Title	Deadline	Status	
CN	Max Mustermann	August Report	In 2 days	Active	+ Start
CN	Räuber Hotzenplotz	July Report	31.08.2025	Draft	Edit
CN	Megamind	Employee Satisfaction Survey	30.05.2025	Closed	Inspect
CN	Daniel Düsentrieb	Crisis Handling Report	Open-Ended	Submitted	Edit
CN	Frankenstein	June Report	Ended	No Response	Edit
CN	Susanne Daubner	May Report	Ended	No Response	Edit

Figure 4.5: UI Mockup of the assigned surveys page

The SRS was reviewed in weekly meetings by two members of MECOIS and was considered suitable for the intended task and original problem. Consequently, we proceed to the *Evaluate Solution Alternatives* stage of the solution design process.

4.2 Evaluate Solution Alternatives

Next, we will investigate which implementation strategies could meet the requirements set forth in the SRS. Before risking to “reinvent the wheel”, it is worthwhile to look at existing off-the-shelf solutions. In practice, multiple solution alternatives exist for building a survey feature for measuring developer sentiment, each with distinct advantages and trade-offs.

Broadly, these alternatives can be grouped into three categories, ranging from low code ownership with high ease of implementation to high code ownership with more challenging implementation: (i) adopting existing third-party survey platforms, (ii) integrating specialized survey libraries, or (iii) developing a solution from scratch. While off-the-shelf or dependency-based options may offer rapid development and reduced initial effort, they can also introduce constraints in terms of flexibility, customization, and long-term ownership of the system.

To evaluate these alternatives, we select a representative candidate from each category and assess how well it satisfies the defined system requirements. For third-party survey platforms, Google Forms by Google LLC (2025) was chosen due to its dominant market position (6Sense Insights, Inc., 2025). As a representative of survey library dependencies, `SurveyJS` by Devsoft Baltic OÜ (2025) was selected based on being open-source, compatible with `React`, and popular enough for almost 5000 stars on GitHub (Github, Inc., 2025). The scoring method used to compare these three alternatives is described as follows:

Given a requirement $r \in \text{REQs}$ and its MoSCoW priority $P(r)$, let

$$W(r) = \begin{cases} 5 & \text{if } P(r) = \text{Must} \\ 3 & \text{if } P(r) = \text{Should} \\ 1 & \text{if } P(r) = \text{Could} \\ 0 & \text{if } P(r) = \text{Won't} \end{cases}$$

denote the weight of r with respect to its priority $P(r)$. Additionally, given an alternative solution $A \in \text{ALTs} = \{\text{Google Forms}, \ \text{SurveyJS}, \ \text{Custom}\}$, let

$$S_A(r) = \begin{cases} 0 & \text{if } A \text{ does not implement } r \\ 1 & \text{if } A \text{ implements } r \\ 2 & \text{if } A \text{ over-satisfies } r \end{cases}$$

denote the score of each alternative solution A per requirement r. Each solution A can then be assigned a satisfiability score

$$\text{SAT}(A) = \sum_{r \in \text{REQs}} S_A(r) \cdot W(r),$$

such that our preferred solution A^* is defined as

$$A^* = \underset{A \in \text{ALTs}}{\arg\max}\ \text{SAT}(A).$$

The calculation of $\text{SAT}(A)$ for all $A \in \text{ALTs}$ is shown in Table 4.3[1].

Table 4.3: Satisfiability score of each solution alternative

REQ r	$W(r)$	Google Forms		SurveyJS		Custom	
		$S(r)$	$S(r)W(r)$	$S(r)$	$S(r)W(r)$	$S(r)$	$S(r)W(r)$
FR 1	5	1	5	1	5	1	5
FR 1.1	5	1	5	1	5	1	5
FR 1.2	5	1	5	1	5	1	5
FR 1.3.2	1	2	2	1	1	0	0
FR 1.4	5	0	0	0	0	1	5
FR 1.5	5	1	5	0	0	1	5
FR 1.6	1	0	0	1	1	1	1
FR 1.6.1	1	0	0	2	2	1	1
FR 1.7	5	1	5	1	5	1	5
FR 1.8	5	1	5	1	5	1	5
FR 1.9	3	1	3	1	3	1	3
FR 1.11	5	1	5	2	10	1	5
FR 1.12	1	1	1	1	1	1	1
FR 1.13	1	1	1	1	1	1	1
FR 2	5	1	5	2	10	1	1
FR 2.1.4	3	1	3	1	3	1	3
FR 2.1.6	3	1	3	1	3	1	3
FR 2.2	5	1	5	1	5	1	5
FR 2.3	1	0	0	2	2	1	1
FR 2.4	1	1	1	1	1	1	1
FR 4	5	1	5	0	0	1	5
NFR 1	5	0	0	1	5	2	10
NFR 1.1	5	0	0	1	5	2	10
NFR 1.2	5	0	0	1	5	2	10
NFR 1.3	5	0	0	1	5	2	10
NFR 1.4	5	0	0	1	5	2	10
SAT			64		93		116

[1]Note that only a subset of system requirements is used in the score calculation, including only the most critical and distinctive ones.

With an SAT score of 116, the custom, self-developed implementation proves to be the most suitable choice. Its advantages are particularly strong in areas such as full code ownership, absence of restrictive dependency licenses, better integration into existing code, and greater customization and extensibility options. Naturally, these benefits come at the cost of higher initial effort and a longer development time.

We decided against the survey library `SurveyJS`, since despite being MIT-licensed, its commercial open-source model was deemed too risky in the long term. Likewise, Google Forms was found unsuitable due to its limited adaptability to our requirements, for instance regarding the specification of survey start and end dates (see FR 1.4) or its integration into the existing code base (see NFR 1.1 – NFR 1.4).

Having established this decision, the next chapter deals with the concrete realization of the survey feature.

5 Implementation

Following the evaluation of solution alternatives in section 4.2, this chapter documents the technical implementation of creating a survey feature for measuring developer sentiment. As a blueprint, the SRS document generated in chapter 4 was used to guide the development efforts. The chapter is structured in a top-down manner: starting with the technology stack, followed by describing core data structures, before showcasing individual components and noteworthy code sections.

5.1 Technology Stack

To begin with, Table 5.1 outlines the key dependencies chosen for the implementation, including their purpose and the rationale behind their selection. As mandated by NFR 1.4, a strict emphasis was placed on using only non-restrictively licensed dependencies. Moreover, efforts were made to minimize the introduction of new dependencies; where unavoidable, preference was given to lightweight, well-established, and actively maintained libraries. Among the new dependencies, one of particular importance is `react-hook-form`, which is used to create and edit survey and response objects through HTML forms. In combination with `yup`, it ensures that all survey and response objects are correctly validated, both at the form field level and when received from the backend. All dependencies are MIT-licensed, with the exception of `Supabase`, which is based on the Apache-2.0 license (Supabase, Inc., 2025).

Table 5.1: The technology stack used to create the survey feature

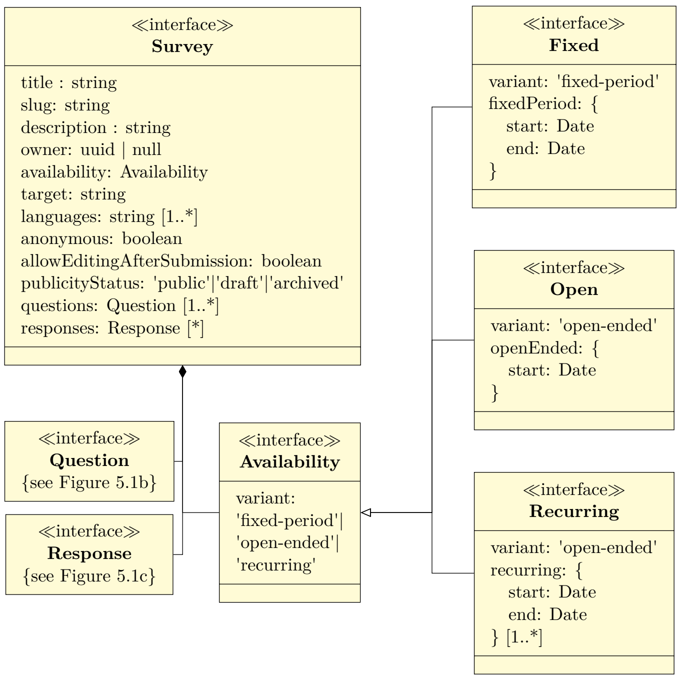
Dependency	**Purpose**	**Rationale**
`React`	Frontend library for creating interactive user interfaces.	Integration with the existing code base, see NFR 1.1.
`react-hook-form`	Form handling and validation (with `yup`) for surveys and responses.	MIT licensed, react compatible form library, see NFR 1.1 and NFR 1.4.
`yup`	Runtime type validation of surveys and responses.	MIT licensed, react compatible runtime validation library, see NFR 1.1 and NFR 1.4.
`Supabase`	Self-hosted backend providing a PostgreSQL database and authentication.	Integration with the existing code base, see NFR 1.1.
`supabase-js`	Javascript client Library for interacting with `Supabase`.	Integration with the existing code base, see NFR 1.1.
`shadcn/ui`	Prebuilt UI components (for survey form elements like buttons, modals, and input fields).	Integration with the existing code base, see NFR 1.2.
`TailWindCSS`	Styling and layout.	Integration with the existing code base, see NFR 1.3.
`pgTap`	Database unit testing.	Recommended tool for testing PostgreSQL databases and part of the `Supabase` software suite, see NFR 6.

5.2 Core Data Structures

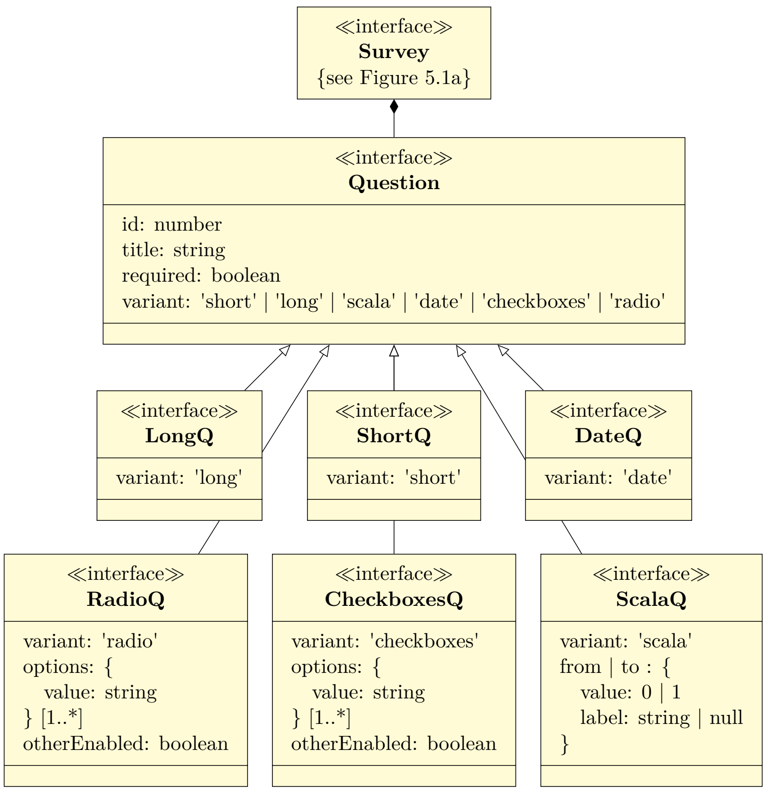
Nearly all newly introduced components depend on either the `Survey` or `Response` type, which form the core structure of a survey and its associated responses. All instantiated objects must adhere to these types, shown in Figure 5.1. A `Survey` object consists of key-value pairs like `title`, `description` and `publicityStatus`, corresponding to functional requirements regarding meta-configuration of surveys (see FR 1.1 – FR 1.9). Additionally, they also store a reference to an array consisting of the survey's questions. As shown in Figure 5.1b, each of these questions is an extension of a discriminatory union type: The question variant must be either `short`, `long`, `scala`, `date`, `checkboxes` or `radio` (see FR 2.1.1 – FR 2.1.6). Depending on the variant, the question object may have more key-value pairs such as an options array for `checkboxes` and `radio`, which stores all available answer options for a multiple choice or single choice question.

Furthermore, a `Survey` object also contains a reference to an array of `Responses` to that survey, with each `Response` object conforming to the type declared in Figure 5.1c. Apart from metadata such as submission period and publicity status, each of these responses also have an array of `answers` towards (at least) the required questions of that survey. The `answer` value type is also dependent on the question variant; for example an `answer` for variant `short` must be of type `string`, whereas a question of variant `date` demands an `answer` of type `Date` (see FR 3.4.1 – FR 3.4.9).

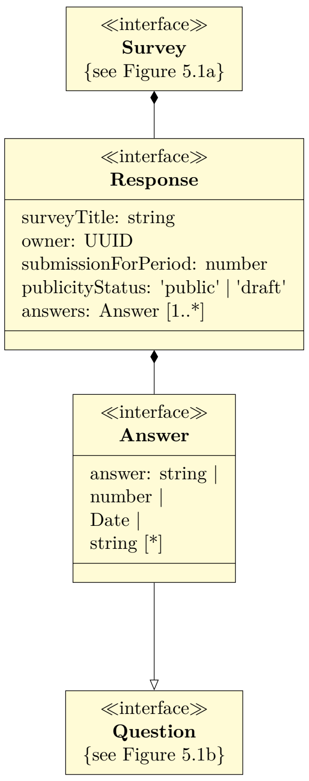
When sent to the backend, Supabase stores `Survey` and `Response` objects in their respective tables. In accordance with best practices, CRUD operations on these tables are policed by Row Level Security (RLS) (The PostgreSQL Global Development Group, 2025). Input validation is done through a mix of PostgreSQL check constraints and RLS policies. To minimize points of contact, write operations are executed via RPC functions, while read operations are conducted through views built on top of the tables. User authentication and access privileges are entirely managed within Supabase, with RLS policies enforcing these permissions.



(a) Survey interface



(b) Question interface



(c) Response interface

Figure 5.1: Class diagrams of (a) survey, (b) question, and (c) response type

5.3 Core Components

We now turn our attention to how these core data objects are employed within application. To do so, we walk through a representative use case, covering all components shown in Figure 5.2.

Suppose we wish to edit a survey previously saved as a draft. After authentication, we navigate to the *Edit Surveys* section of the application. At this point, the `MySurveys` component requests all surveys belonging to the current user from the backend. This request is made through the `useSurveys` hook, which internalizes asynchronous fetching, as well as loading and error states.

Once the surveys are retrieved, the `AssignedSurveys` component presents a table listing all surveys for which the user has edit permissions. We select our draft and click on *Edit Survey*. This triggers a navigation to the route `/surveys/create-surveys/<slug>`, which is managed by the `SurveyCreator` component. The component forwards the survey's unique slug to the `useSurvey` hook, which fetches the corresponding survey from the database. The internal mechanics of the `useSurvey` hook are depicted in Figure 5.3.

Through the UI, we change the survey's `publicityStatus` to `public` and confirm by clicking *Publish Response*. This initiates the `upsertSurvey` routine handled by the `api` component. First, the submitted user form undergoes validation via the `SchemaValidator` component, to ensure it conforms with the `Survey` object. On success, the survey is then sent to the database, which performs the same validation checks again, although this time on the backend-side. Upon successful update, the backend returns the new `Survey` object to the client, prompting a success message to be shown to the user.

As the survey is now public, target users can see it listed in the *Assigned to Me* section, managed by the `AssignedSurveys` component. Finally, they may submit their responses using the `ResponseCreator` component.

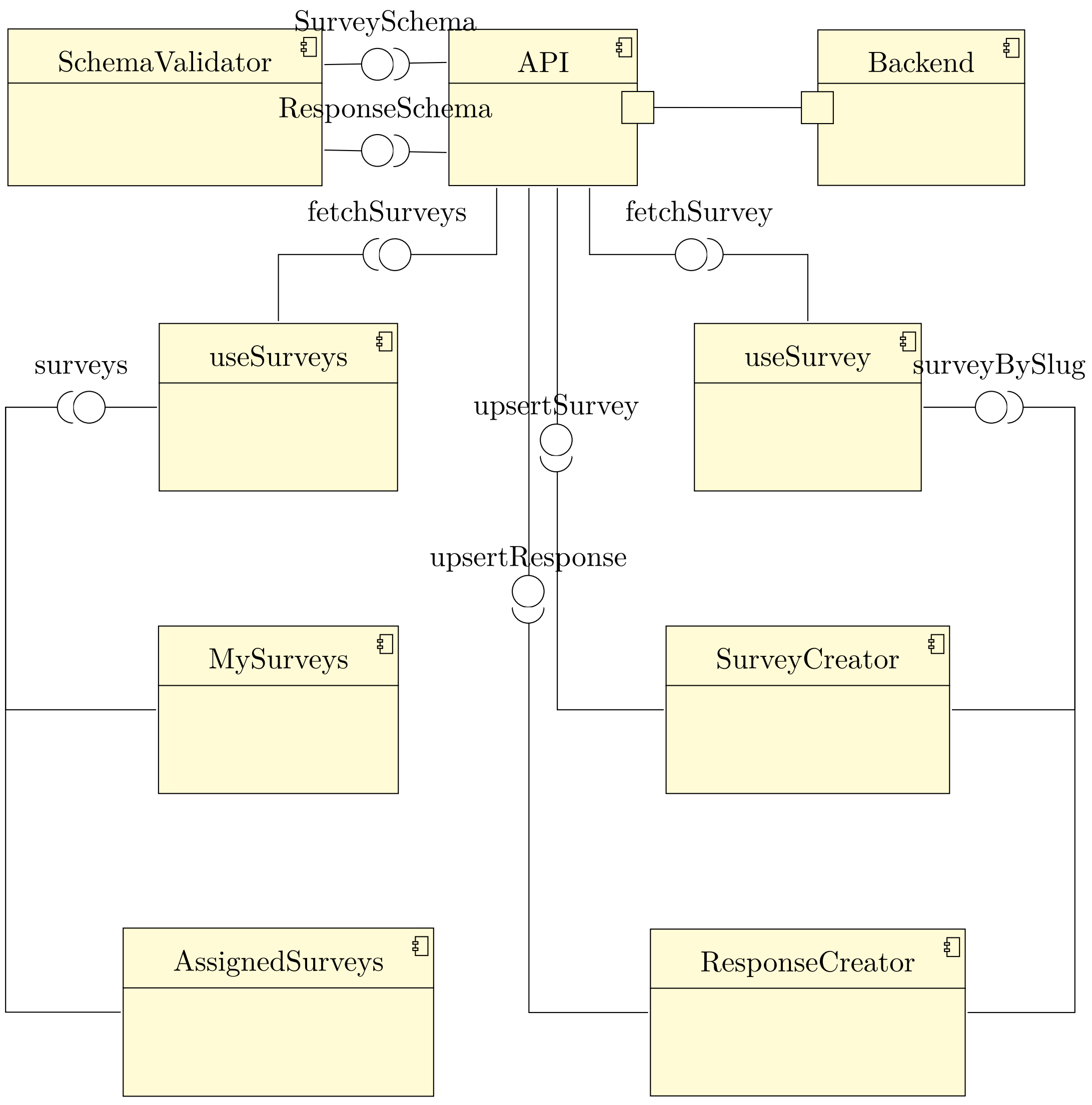


Figure 5.2: Component diagram of the core survey components

```
/**
 * Custom React hook to fetch a survey by its unique slug.
 *
 * @param slug - Unique identifier for the survey.
 * If 'new', returns a default survey object.
 * @returns An object containing:
 *   - `isLoading`: whether the survey is currently being fetched.
 *   - `survey`: the fetched survey object (or default if slug is 'new').
 *   - `error`: any error encountered during the fetch operation, or null
 *
 * This hook handles:
 *   - Loading state management
 *   - Error handling for failed API requests
 *   - Returning a default survey for new entries
 */
export const useSurvey = (slug: string) => {
  const [survey, setSurvey] = useState<SurveySchemaType>(DEFAULT_SURVEY)
  const [isLoading, setIsLoading] = useState(true)
  const [error, setError] = useState<Error | null>(null)

  useEffect(() => {
    const fetchSurvey = async () => { //fetch survey data from the backend.
      try {
        setIsLoading(true)
        const data = // if slug is 'new' return default survey object
          slug === 'new' ? DEFAULT_SURVEY : await getSurveyBySlug(slug)
        setSurvey(data)
      } catch (err) {
        if (err instanceof Error) {
          setError(err)
        } else {
          setError(new Error('Failed to fetch survey'))
        }
      } finally {
        setIsLoading(false)
      }
    }

    fetchSurvey()
  }, [slug])

  return { isLoading, error, survey } // Return hook state to the component
}
```

Figure 5.3: Source code of the useSurvey hook

6 Demonstration

We will demonstrate the survey feature by considering the following – perhaps primary – scenario: Scrum master[1] S has been tasked with routinely measuring developer sentiment and removing any potential productivity blockers. To that end, he decides to employ our survey feature. After logging into the MECOIS platform, he accesses the *create survey* option to design a recurring sentiment survey for developers. The corresponding user interface is shown in Figure 6.2. Using that interface, S configures the survey's metadata as specified in Table 6.1.

Table 6.1: Survey metadata of the exemplary developer sentiment survey

Metadata	**Value**
Title	Developer Sentiment Check-In
Description	We want to hear about your day-to-day developer experience: when you're able to get into flow, where you hit friction, and what's working well. Your input will guide improvements to tools, processes, and team practices that directly affect how you work.
Availability	Recurring survey: Each month, beginning on the first, open for two weeks.
Target Group	Developers
Anonymity	`true`
Visibility	Public

Crucially, he decides to enable anonymous responses instead of collecting user-identifying information. This choice is supported by research showing that developers strongly oppose analytics being used to assess individual productivity (Begel & Zimmermann, 2014, p. 12). Furthermore, maintaining strict privacy safeguards is essential to foster trust and preserve data integrity (Jaspan et al.,

[1]Scrum master as defined in Schwaber and Sutherland (2020, p. 6)

2020, p. 45). Instead of tracking individuals, the survey should pursue a clearly articulated purpose that underscores its relevance and benefits for developers as a team (D'Angelo et al., 2024, p. 23). Accordingly, *S* highlights the survey's intentions through the description field.

Since all metadata entries are valid, the survey system allows *S* to proceed to the next step, that is, creating the actual questionnaire. The corresponding user interface is depicted in Figure 6.3. The questions, in order, are listed in Table 6.2.

Table 6.2: Survey questions of the exemplary developer sentiment survey

Type	Question	Options	Required
Scala	**Q1**: How satisfied are you with your overall experience as a developer in the past month?	0 (Very dissatisfied) – 5 (Very satisfied)	Yes
Scala	**Q2**: How productive do you feel you were in the past month?	0 (Not productive) – 5 (Very productive)	Yes
Scala	**Q23**: In general, how would you rate your ability to get work done quickly and without unnecessary effort?	0 (Very slow/difficult) – 5 (Very fast/easy)	Yes
Single Choice	**Q4**: How often are you able to reach a high level of focus or achieve flow during your development tasks?	– Always – Often – Sometimes – Rarely – Never	Yes
Single Choice	**Q5**: When you do reach a flow state, how long are you typically able to sustain it before being interrupted?	– >2h – 1–2h – 30–60m – 15–30m – <15m	Yes
Multiple Choice	**Q6**: What factors most often support your ability to stay focused?	– Dedicated focus time – Clear goals – Effective tools – Team support – Other	Yes

Table continues on the next page →

List of survey questions continued

Type	**Question**	**Options**	**Required**
Single Choice	**Q7**: In the past month, how often did you encounter friction that slowed down your progress?	– Always – Often – Sometimes – Rarely – Never	Yes
Multiple Choice	**Q8**: What types of friction did you experience?	– Infrastructure – CI/CD bottlenecks – Technical debt – Processes – Other	No
Scala	**Q9**: When you encountered friction, how disruptive did it feel?	0 (Not disruptive) – 5 (Very disruptive)	No
Scala	**Q10**: How satisfied are you with the reliability and speed of your primary development tools and infrastructure?	0 (Very dissatisfied) – 5 (Very satisfied)	Yes
Single Choice	**Q11**: Do you feel you have the resources you need (tools, infrastructure, documentation) to do your job effectively?	– Yes completely – Mostly – Somewhat – Rarely – Not at all	Yes
Short Text	**Q12**: What additional resources, tools, or improvements would help you do your job more effectively?	Free text	No
Long Text	**Q13**: What has most positively impacted your development experience recently?	Free text	No
Long Text	**Q14**: What is the biggest source of friction or frustration you would like to see addressed?	Free text	No

S grounds his questionnaire design in recent research, similar to his approach to survey configuration. The survey is structured into four main sections. The first section collects feedback about the overall developer experience, with a particular emphasis on how each individual *perceives* his satisfaction and productivity (Meyer et al., 2017, p.105). Capturing these self-reported measures is essential, as they cannot be reliably inferred from logs-based quantitative data (D'Angelo et al., 2024, p. 19). The second and third section center on flow and friction, reflecting *S*'s aim to boost the former while reducing the latter. Here, he draws on the definitions of flow and friction provided by Brown et al. (2023). Finally, follow-up questions are intentionally optional and phrased in an open-ended manner to gather richer insights into how developers think about their work and what they feel they need, as recommended by D'Angelo et al. (2024, p. 19).

After configuring the survey, *S* publishes it for the developers to respond. During the survey's active period, they can utilize the UI depicted in Figure 6.4 to respond to the survey.

We have played out this scenario with three developers at MECOIS, and these were the results:

Of the 14 questions, 13 were answered by all participants; only the final optional question received two responses. Excluding free-text questions, the results are visualized in Figure 6.1[2]. From these responses, *S* infers that overall developer sentiment is positive, though perceptions of productivity and flow differ among individuals. Clear goals, effective tools, and uninterrupted focus time emerge as enablers of flow, whereas unreliable infrastructure and deployment issues emerge as major sources of friction, further supported by free text answers to questions 12 and 14. Addressing these blockers would likely yield the greatest improvements in team productivity. Notably, current CI/CD efforts appear to be a recurring pain point, prompting *S* to request additional investment from upper management. On a more positive note, all participants report achieving flow states with relative regularity and sustaining them for longer periods of time. *S* intends to monitor this metric closely and respond promptly to any negative trends.

On a meta-level, we also surveyed the survey itself. Overall, participants found the format effective for capturing developer sentiment. However, one respondent was irritated by differences in optional free text follow-up questions and the ability to give a free text "other" answer in multiple or single choice questions; while another suggested adding the ability to structure a survey into thematic subsections. Building on these suggestions, the next chapter will evaluate the developed artifact by examining how its features and attributes align with the objectives outlined in chapter 3.

[2]The charts in Figure 6.1 are identical to those in the survey analysis UI, but have been condensed here for convenience.

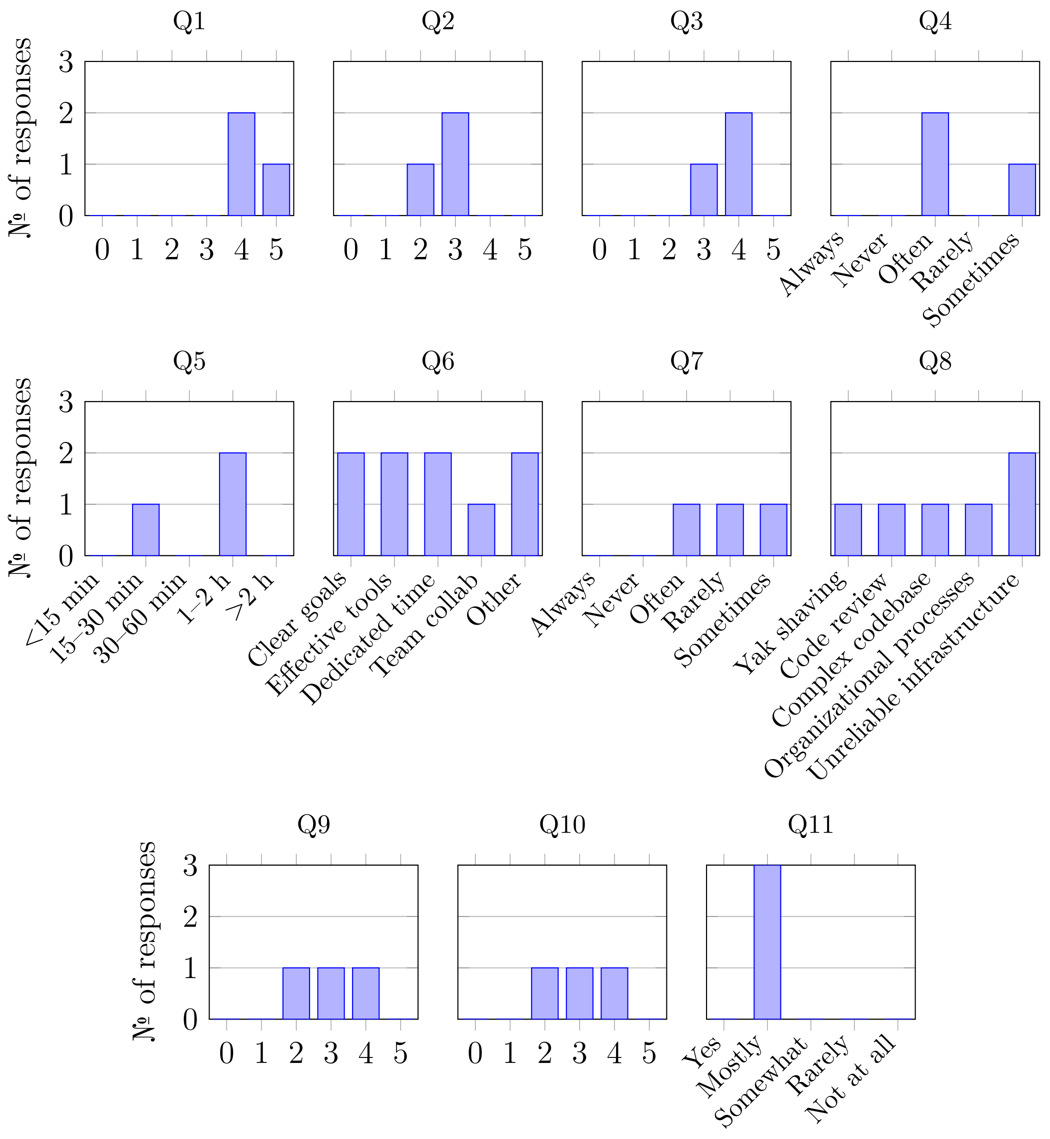
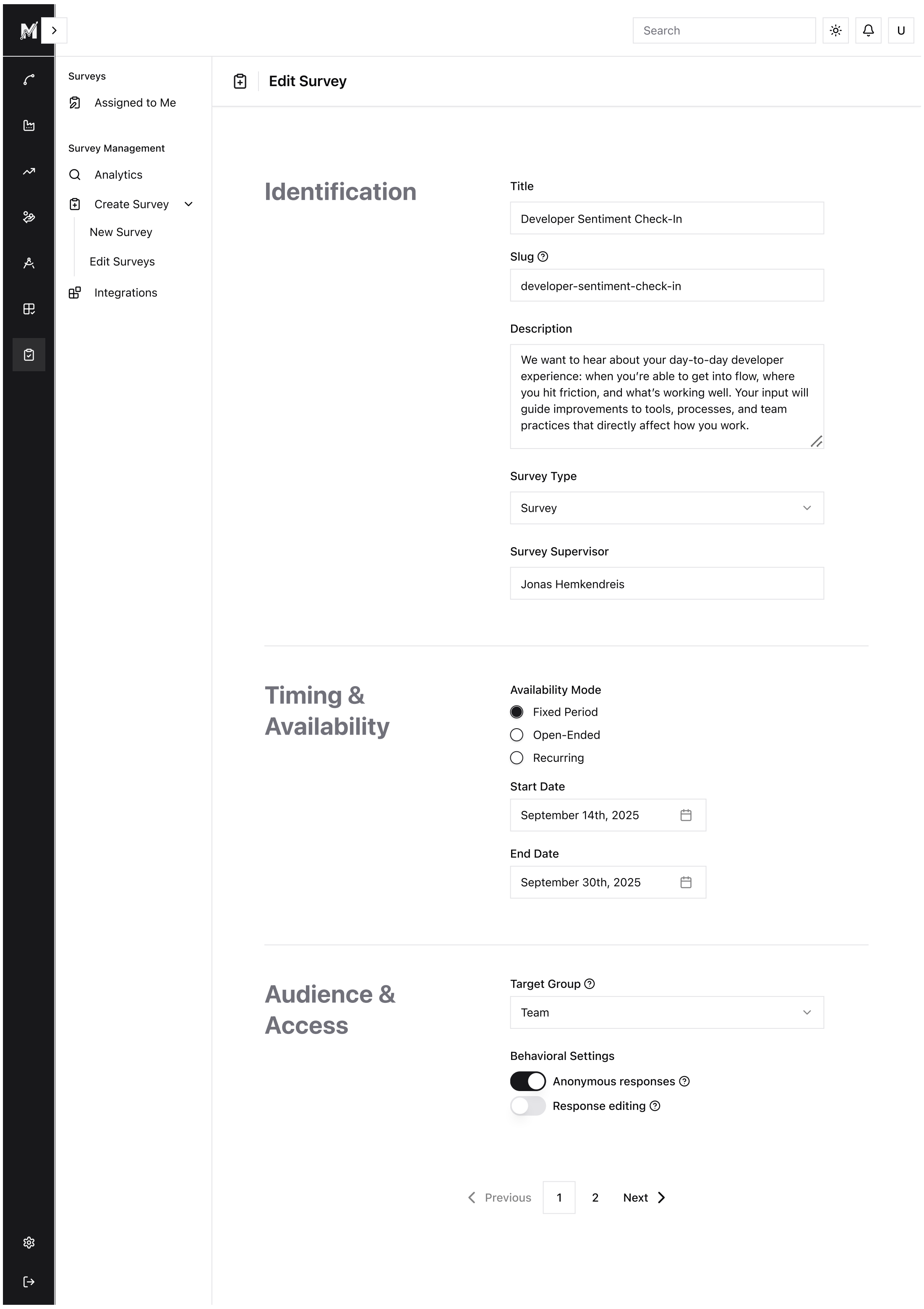


Figure 6.1: Responses to the exemplary developer sentiment survey



Identification

Title: Developer Sentiment Check-In

Slug: developer-sentiment-check-in

Description: We want to hear about your day-to-day developer experience: when you're able to get into flow, where you hit friction, and what's working well. Your input will guide improvements to tools, processes, and team practices that directly affect how you work.

Survey Type: Survey

Survey Supervisor: Jonas Hemkendreis

Timing & Availability

Availability Mode: [✓ Fixed Period] Open-Ended Recurring

Start Date: September 14th, 2025

End Date: September 30th, 2025

Audience & Access

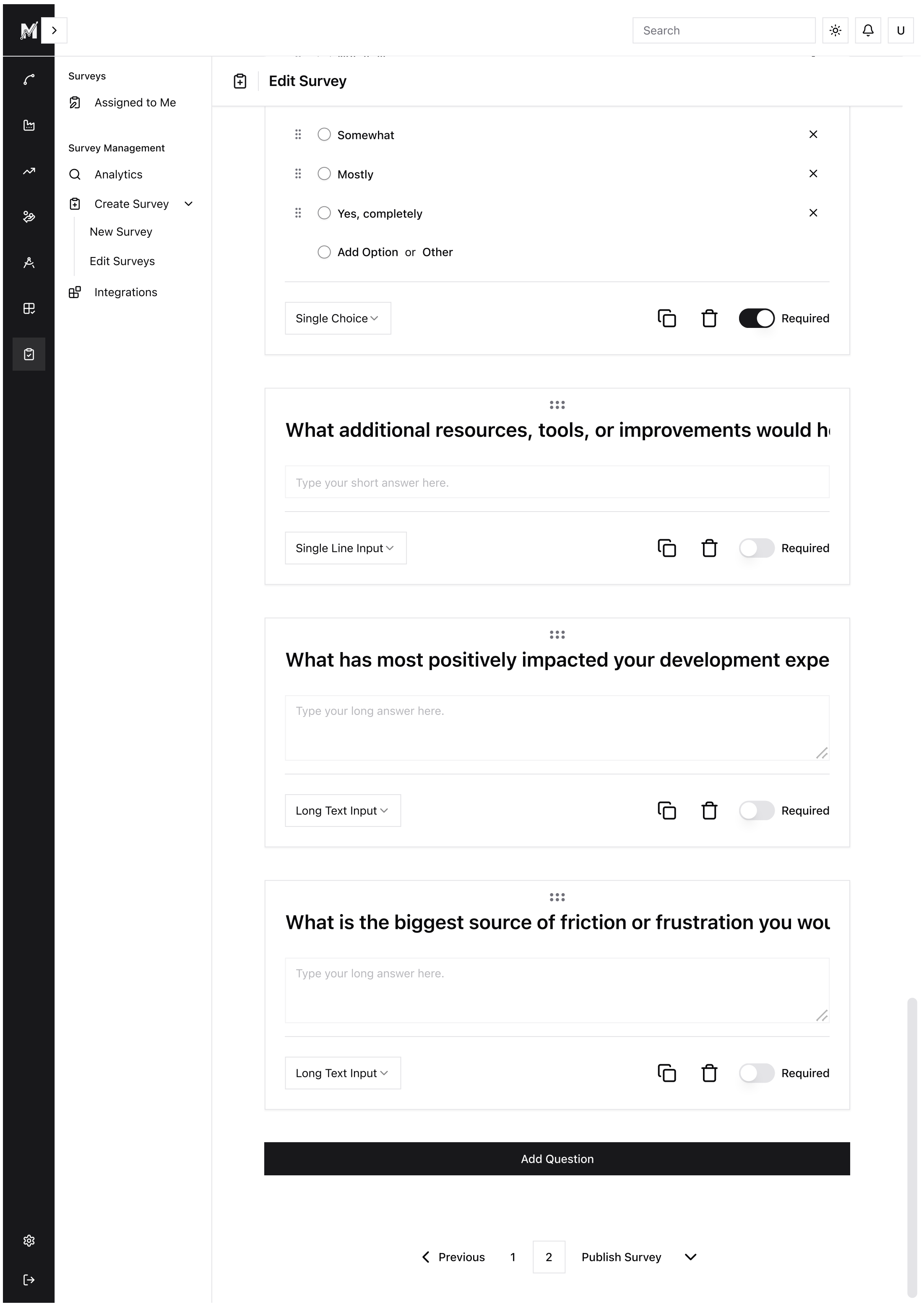
Target Group: Team

Behavioral Settings

Anonymous responses: on

Response editing: off

Figure 6.2: UI for creating and editing survey metadata



Somewhat

Mostly

Yes, completely

Add Option or Other

Single Choice — Required: on

What additional resources, tools, or improvements would h

Type your short answer here.

Single Line Input — Required: off

What has most positively impacted your development expe

Type your long answer here.

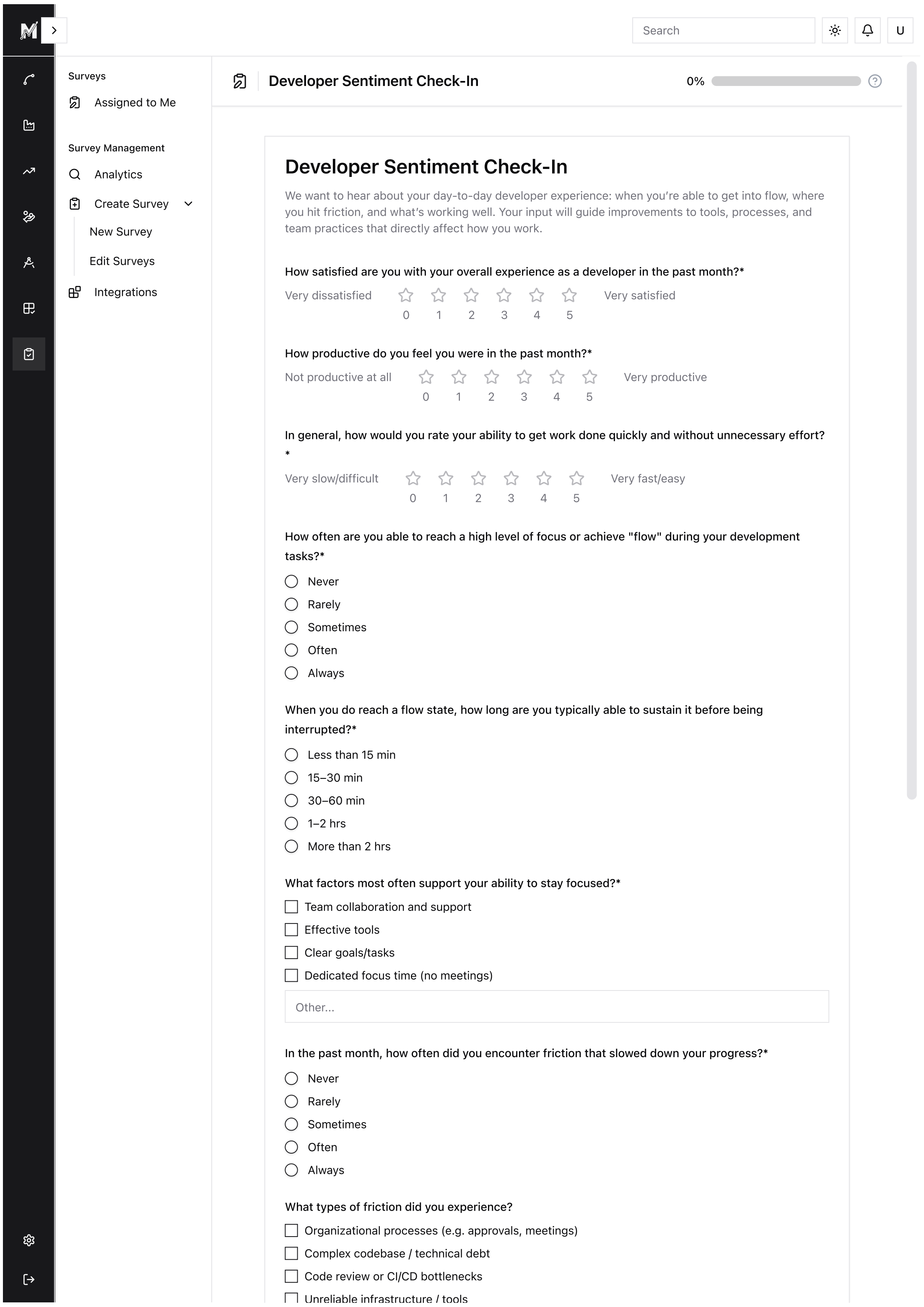
Long Text Input — Required: off

What is the biggest source of friction or frustration you wou

Type your long answer here.

Long Text Input — Required: off

Figure 6.3: UI for creating and editing survey questions



Developer Sentiment Check-In

We want to hear about your day-to-day developer experience: when you're able to get into flow, where you hit friction, and what's working well. Your input will guide improvements to tools, processes, and team practices that directly affect how you work.

How satisfied are you with your overall experience as a developer in the past month?*

Very dissatisfied ☆ ☆ ☆ ☆ ☆ ☆ Very satisfied
0 1 2 3 4 5

How productive do you feel you were in the past month?*

Not productive at all ☆ ☆ ☆ ☆ ☆ ☆ Very productive
0 1 2 3 4 5

In general, how would you rate your ability to get work done quickly and without unnecessary effort?*

Very slow/difficult ☆ ☆ ☆ ☆ ☆ ☆ Very fast/easy
0 1 2 3 4 5

How often are you able to reach a high level of focus or achieve "flow" during your development tasks?*

- ○ Never
- ○ Rarely
- ○ Sometimes
- ○ Often
- ○ Always

When you do reach a flow state, how long are you typically able to sustain it before being interrupted?*

- ○ Less than 15 min
- ○ 15–30 min
- ○ 30–60 min
- ○ 1–2 hrs
- ○ More than 2 hrs

What factors most often support your ability to stay focused?*

- ☐ Team collaboration and support
- ☐ Effective tools
- ☐ Clear goals/tasks
- ☐ Dedicated focus time (no meetings)

Other...

In the past month, how often did you encounter friction that slowed down your progress?*

- ○ Never
- ○ Rarely
- ○ Sometimes
- ○ Often
- ○ Always

What types of friction did you experience?

- ☐ Organizational processes (e.g. approvals, meetings)
- ☐ Complex codebase / technical debt
- ☐ Code review or CI/CD bottlenecks
- ☐ Unreliable infrastructure / tools

Figure 6.4: UI for responding to a survey

7 Evaluation

In accordance with the DSRM process, we assess whether the developed artifact and its demonstration fulfill the objective criteria defined in chapter 3 (Peffers et al., 2007, p. 56). To do so, we will evaluate these objectives based on their associated system requirements derived in subsection 4.1.1. An objective shall be considered fulfilled if and only if all associated *Must* requirements are met.

Out of 84 system requirements, 74 were successfully implemented. This corresponds to 100% of *Must*, 71% of *Should*, and 60% of *Could* requirements. Thus, we can conclude that all high-level objectives were successfully implemented. However, instead of ending our analysis here, we can evaluate the degree to which each objective is satisfied and thereby draw conclusion to the limitations of the current solution. We do so by aggregating requirements to corresponding objectives, as illustrated in Figure 7.1:[1]

It should come as no surprise that Objective 1 and Objective 2 are associated with the largest number of system requirements. These objectives concern the existence of the survey system itself, its configurability, and its ability to cover the use case of measuring developer sentiment. As such, they represent the most general objectives among the five. Consequently, most requirements related to survey and response creation are mapped to these two objectives.

Specifically, of the 68 requirements linked to Objective 1, a total of 60 have been satisfied (43 Must, 13 Should, and 4 Could) by the implemented survey feature. Similarly, Objective 2 comprises of 66 requirements, of which 58 are fulfilled (41 Must, 13 Should, and 4 Could). These include, for example, the ability to configure survey metadata such as availability and anonymity (see FR 1) as well as the ability to configure survey questions types such as Likert scales, single and multiple choice, date, and free text (see FR 2).

In contrast, some Should and Could requirements such as restricting visibility to specific target groups (see FR 1.5.1) remain yet to be implemented. Nevertheless, since all Must requirements are fulfilled and the demonstration shown in chapter 6

[1]Note that one requirement may contribute to multiple objectives.

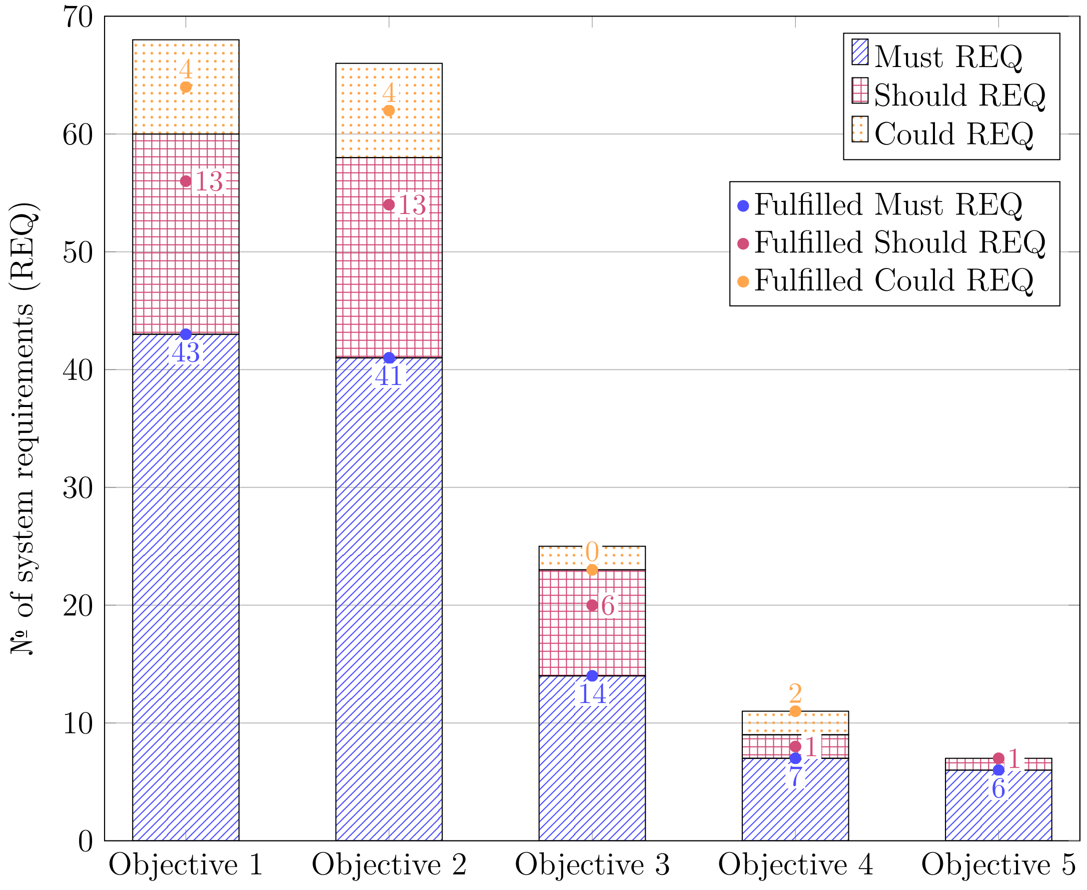


Figure 7.1: № of system requirements per objective

was deemed appropriate for our primary use case, the solution can be regarded as sufficient for Objective 1 and Objective 2.

Objective 3 ensures compliance with academic standards and best practices. Out of 25 requirements, 20 were implemented (14 Must, 6 Should). Noteworthy examples include the decision to let survey creators decide whether a Likert scale should contain a neutral response category (see FR 2.1.2, inspired by Kankaraš & Capecchi, 2025, p.129), as well as setting surveys to be anonymous by default (see FR 1.8.1, inspired by Jaspan & Sadowski, 2019, p.14). While some optional features such as thematic grouping of questions (see FR 2.4) remain unimplemented, all Must requirements were achieved, and the objective is therefore satisfied.

Objective 4 deals with the usability aspect of the survey solution. Out of 11 requirements, 10 were implemented (7 Must, 1 Should, 2 Could). These includes non-functional requirements such as utilizing `shadcn/ui` and `TailwindCSS` (see NFR 1.2 and NFR 1.3), which enable responsive and accessibility-aware design, but also functional requirements such as keyboard navigation and progress bar indicators (see FR 3.1.4 and FR 3.2). Missing features such as non-intrusive reminders via in-app and email notifications (see NFR 3) should be addressed in future iterations. Nonetheless, this objective is considered fulfilled.

Finally, Objective 5 imposes organizational and external requirements on our artifact, as the final solution must integrate with existing code at MECOIS and may only rely on dependencies with non-restrictive licenses. All 7 derived requirements (6 Must, 1 Should) were met, including the use of `React` for the frontend and `Supabase` for the backend (see FR 1.1, FR 1.3). This objective also heavily influenced the decision against adopting `SurveyJS` (see section 4.2), as its commercial open-source model was deemed too risky long-term. Accordingly, this objective is also deemed satisfied.

8 Conclusion

This thesis set out to design and implement a survey feature for measuring developer sentiment. Using Requirements Engineering (RE) techniques, a Software Requirements Specification (SRS) document was developed, which served as the foundation for evaluating alternatives, implementing a solution, and evaluating its functionality.

The resulting artifact successfully met all high-level objectives and was integrated into the MECOIS ecosystem for measuring developer productivity. A demonstration confirmed the solution's ability to capture insights into developer sentiment, thereby addressing key qualitative gaps in traditional log-based productivity measurements.

While the implemented solution is sufficient for its intended purpose, certain non-essential requirements, such as advanced survey structuring and non-intrusive notification features, remain for future iterations.

Ultimately, this work contributes a configurable, research-aligned survey feature to the MECOIS ecosystem; one that strengthens the measurement of productivity by enabling the capture of qualitative metrics such as developer sentiment.

Appendices

A Software Requirement Specification

A.1 User Requirements

Table A.1: List of user requirements

ID	User Requirement	MoSCoW	Source
UR 1	The survey system shall enable privileged users to Create, Read, Update, and Delete (CRUD) a survey.	Must	Objective 1 Objective 3
UR 1.1	When a privileged user creates or updates a survey, the survey system shall provide meta configuration options.	Must	Objective 2
UR 1.2	When a privileged user creates or updates a survey, the survey system shall provide question configuration options.	Must	Objective 2
UR 2	The survey system shall enable privileged users to CRUD a response.	Must	Objective 1 Objective 3
UR 3	The survey system shall enable privileged users to analyze responses to a survey.	Must	Objective 1 Objective 3
UR 4	The survey system shall verify and validate the user's authentication, privileges, and inputs.	Must	Objective 1
UR 5	The survey system shall conform to usability best practices.	Must	Objective 4
UR 6	The survey system shall comply with data privacy regulations.	Must	Objective 3
UR 7	The survey system's code shall conform to MECOIS contribution guidelines.	Must	Objective 5

A.2 System Requirements

Table A.2: List of functional requirements

ID	Functional requirement	MoSCoW	Source
FR 1	When a privileged user creates or updates survey metadata, the survey system shall provide the following configuration options: title, description, survey type, owner, availability, target group, languages, anonymous response setting, visibility status, ability to edit responses after submission, and a unique slug.	Must	UR 1.1
FR 1.1	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option to set a unique survey title and slug.	Must	UR 1.1
FR 1.1.1	While a privileged user creates or edits a survey title, the survey system shall automatically generate a compatible slug.	Could	UR 1.1
FR 1.1.2	While a privileged user creates or edits a survey title or slug, the survey system shall continuously verify that the title and slug are unique.	Must	UR 1.1 UR 4
FR 1.2	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option to set a description for the survey.	Should	UR 1.1
FR 1.3	When a privileged user creates or updates a survey's metadata, the survey system shall automatically assign the user as the owner.	Must	UR 1.1
FR 1.3.1	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option to assign an owner group.	Should	UR 1.1

Table continues on the next page →

List of functional requirements continued

ID	Functional requirement	MoSCoW	Source
FR 1.3.2	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option configure fine-grained permissions (e.g., editorial privileges, analytics-only access) on a per-user basis.	Could	UR 1.1
FR 1.4	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option to set a start and end date.	Must	UR 1.1
FR 1.4.1	When a privileged user wants to sets a survey start and end date, the survey system shall provide three modes: `Fixed-Period` for a survey with fixed start and end date; `Open-Ended` for a survey with only a start date; and `Recurring` for a survey with multiple start and end dates.	Should	UR 1.1
FR 1.4.2	If a survey is configured as `Recurring`, the survey system shall allow the user to specify either the total number of occurrences or an end date, after which no additional survey periods will be created.	Could	UR 1.1
FR 1.4.3	If a survey is configured as `Fixed-Period` or `Recurring`, the survey system shall verify that each end date occurs after its start date.	Must	UR 1.1 UR 4
FR 1.4.4	If a survey is configured as `Recurring`, the survey system shall verify that the start and end dates of open periods to not overlap.	Must	UR 1.1 UR 4

Table continues on the next page →

List of functional requirements continued

ID	Functional requirement	MoSCoW	Source
FR 1.5	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option to set a target group.	Must	UR 1.1
FR 1.5.1	If a survey is configured for a specific target group, the survey system shall restrict visibility to members of that group only.	Should	UR 1.1 UR 4 UR 6
FR 1.6	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option to set the survey's language.	Could	UR 1.1
FR 1.6.1	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option to set multiple languages.	Could	UR 1.1
FR 1.7	When a privileged user creates or updates a survey's metadata, the survey system shall provide the option to set the survey's visibility to `public` or `draft`.	Must	UR 1.1
FR 1.7.1	If visibility is set to `draft`, the survey system shall restrict read access to the survey administrator only.	Should	UR 1.1 UR 4 UR 6
FR 1.8	When a privileged user creates or updates survey metadata, the survey system shall allow specifying whether responses are anonymous.	Must	UR 1.1
FR 1.8.1	If the anonymity setting has not been configured, the survey system shall set the default to anonymous.	Must	UR 1.1 (Jaspan & Green, 2025, p.16)

Table continues on the next page →

List of functional requirements continued

ID	**Functional requirement**	**MoSCoW**	**Source**
FR 1.8.2	If anonymity is enabled, the survey system shall ensure that all responses are stored anonymously.	Must	UR 1.1 UR 6
FR 1.9	When a privileged user creates or updates survey metadata, the survey system shall provide the option to specify whether respondents can edit their responses after submission.	Should	UR 1.1
FR 1.10	When a privileged user tries to delete his survey, the survey system shall reject the promise if and only if there has already been a response to the survey.	Should	UR 1
FR 1.11	While a privileged user creates or edits a survey, the survey system shall validate all inputs.	Must	UR 1 UR 4
FR 1.11.1	When a form field is invalid, the survey system shall display an error message below the field explaining the reason for invalidity.	Must	UR 5
FR 1.12	The survey system shall allow creating or updating a survey using keyboard navigation only.	Could	UR 5
FR 1.13	The survey system shall allow exporting and duplicating surveys for replication or third party integration.	Could	UR 1
FR 1.14	The survey system shall save surveys in a relational database.	Must	UR 1

Table continues on the next page →

List of functional requirements continued

ID	Functional requirement	MoSCoW	Source
FR 2	When a privileged user creates or modifies a survey's questions, the survey system shall allow configuring whether a question is mandatory and selecting question types `short`, `long`, `scala`, `date`, `radio`, and `checkboxes`.	Must	UR 1.2
FR 2.1	When a privileged user creates or updates questions, the survey system shall support configurable question types including Likert scales (`scala`), multiple choice (`checkboxes`), single choice (`radio`), and open-ended questions (`short`, `long`, and `date`).	Must	UR 1.2
FR 2.1.1	When a privileged user selects the question type `scale`, the survey system shall require specifying minimum and maximum values with optional labels.	Must	UR 1.2
FR 2.1.2	When a privileged user selects the question type `scale`, the survey system shall provide the option to include or exclude a neutral response.	Must	UR 1.2 (Kankaraš & Capecchi, 2025, p.129)
FR 2.1.3	When a privileged user selects the question type `single choice`, the survey system shall require specifying at least one response option.	Must	UR 1.2
FR 2.1.4	When a privileged user selects the question type `single choice`, the survey system shall provide the option to enable whether respondents are able to input an alternative answer that is not within the given responses.	Should	UR 1.2

Table continues on the next page →

List of functional requirements continued

ID	Functional requirement	MoSCoW	Source
FR 2.1.5	When a privileged user selects the question type `multiple choice`, the survey system shall require specifying at least one response option.	Must	UR 1.2
FR 2.1.6	When a privileged user selects the question type `multiple choice`, the survey system shall provide the option to enable whether respondents are able to input an alternative answer that is not within the given responses.	Should	UR 1.2
FR 2.2	When a privileged user creates or updates questions, the survey system shall provide the option to mark a question as mandatory or optional.	Must	UR 1.2
FR 2.3	When a privileged user creates or updates questions, the survey system shall allow conditional logic to display questions based on prior responses.	Could	UR 1.2
FR 2.4	When a privileged user creates or updates questions, the survey system shall allow grouping questions into sections and subsections.	Could	UR 1.2
FR 3	When a privileged user creates or updates a response, the survey system shall present answer forms consistent with the survey's question types and metadata configuration (e.g., anonymity).	Must	UR 2
FR 3.1	When a privileged user creates or updates a response, the survey system shall adapt behavior according to the survey's metadata configuration.	Must	UR 2

Table continues on the next page →

List of functional requirements continued

ID	Functional requirement	MoSCoW	Source
FR 3.1.1	If a survey is configured as anonymous, the survey system shall store all responses anonymously.	Must	UR 2 UR 6
FR 3.1.2	If a survey is configured to permit editing responses, the survey system shall allow respondents to edit their response while the survey remains open.	Should	UR 2
FR 3.1.3	If a survey is not anonymous, the survey system shall allow respondents to save responses as drafts.	Should	UR 2
FR 3.1.4	While a respondent completes a survey, the survey system shall indicate progress as the percentage of *required* questions answered.	Could	UR 2 UR 5
FR 3.2	The survey system shall allow creating or updating responses using keyboard navigation only.	Should	UR 5
FR 3.2.1	The survey system shall prevent duplicate responses from the same user.	Must	UR 2 UR 4
FR 3.3	The survey system shall save responses in a relational database.	Must	UR 2
FR 3.4	When a privileged user creates or updates a response, the survey system shall present forms appropriate to each question type.	Must	UR 1.2
FR 3.4.1	If a question is mandatory, the survey system shall mark it with an asterisk.	Must	UR 1.2
FR 3.4.2	If a question type is set to `short` text, the survey system shall provide a text input field.	Must	UR 1.2
FR 3.4.3	If a question type is set to `long` text, the survey system shall provide a text input field.	Must	UR 1.2

Table continues on the next page →

List of functional requirements continued

ID	Functional requirement	MoSCoW	Source
FR 3.4.4	If a question type is set to `scala`, the survey system shall provide a Likert scale input.	Must	UR 1.2
FR 3.4.5	If a question type is set to `date`, the survey system shall provide a date input field.	Must	UR 1.2
FR 3.4.6	If a question type is set to `single choice`, the survey system shall provide radio buttons for the given answer options.	Must	UR 1.2
FR 3.4.7	If a question type is `single choice` and alternative responses are allowed, the survey system shall provide an additional text input field.	Should	UR 1.2
FR 3.4.8	If a question type is `multiple choice`, the survey system shall provide checkboxes for the given answer options.	Must	UR 1.2
FR 3.4.9	If a question type is `multiple choice` and alternative responses are allowed, the survey system shall provide an additional text input field.	Should	UR 1.2
FR 3.5	While a privileged user creates or edits a response, the survey system shall validate all inputs.	Must	UR 1
FR 3.5.1	When an answer is invalid, the survey system shall display an error message below the corresponding field, explaining the reason.	Must	UR 4 UR 5
FR 4	When a privileged user analyzes a survey, the survey system shall present results in an easily digestible format.	Must	UR 3

Table continues on the next page →

List of functional requirements continued

ID	Functional requirement	MoSCoW	Source
FR 4.1	The survey system shall visualize survey results with charts and graphs.	Should	UR 3
FR 4.2	The survey system shall allow exporting survey results to CSV.	Should	UR 3
FR 4.3	The survey system shall allow privileged users to filter responses by team, project, or time range.	Should	UR 3
FR 4.4	The survey system shall allow privileged users to generate summaries.	Should	UR 3
FR 5	When a user requests surveys from the backend, the survey system shall enforce proper CRUD privileges.	Must	UR 4 UR 6
FR 5.1	When a user requests access to respond to a survey, the survey system shall allow access only if the survey is public and the user belongs to the defined target group.	Must	UR 4 UR 6
FR 5.2	When a user requests to update or delete a survey, the survey system shall allow the operation only if the user is the owner and the survey has not received responses.	Must	UR 4 UR 6
FR 5.3	When a user requests access to responses, the survey system shall allow access only if the user is the owner of the responses or the owner of the survey.	Must	UR 4 UR 6
FR 5.4	When a user requests to update a response, the survey system shall allow the operation only if the user has access rights and either the survey permits response editing or the response is in draft status.	Must	UR 4 UR 6

Table A.3: List of non-functional requirements

ID	Non-functional requirement	MoSCoW	Source
NFR 1	The survey system's code shall integrate seamlessly with the existing code base.	Must	UR 7
NFR 1.1	The survey system's frontend shall be implemented using `React`.	Must	UR 7
NFR 1.2	The survey system's frontend shall use `shadcn/ui` as the base component design framework.	Must	UR 5 UR 7
NFR 1.3	The survey system's frontend shall employ `TailwindCSS` for styling.	Must	UR 5 UR 7
NFR 1.4	The survey system shall only rely on non-restrictive dependencies (e.g., MIT License).	Must	UR 7
NFR 1.5	The survey system's backend shall use `Supabase` for authentication and relational database storage.	Must	UR 7
NFR 2	The survey system shall handle errors gracefully and notify users in a clear and non-technical way.	Must	UR 5
NFR 3	The survey system shall notify privileged users about surveys in a non-intrusive way.	Should	UR 5
NFR 4	The survey system shall comply with applicable data privacy regulations (e.g., GDPR).	Must	UR 6
NFR 5	The survey system shall be performant enough to support at least 50 concurrent users.	Must	UR 5
NFR 6	The survey system shall be maintainable, with modular code to facilitate future extensions.	Should	UR 7

Table continues on the next page →

List of non-functional requirements continued

ID	Non-functional requirement	MoSCoW	Source
NFR 7	The survey system shall be reliable, ensuring survey data is not lost in case of unexpected server failure.	Must	Objective 1
NFR 8	The survey system shall be secure, enforcing authentication and authorization consistently across all components.	Must	UR 4

A.3 System Modeling

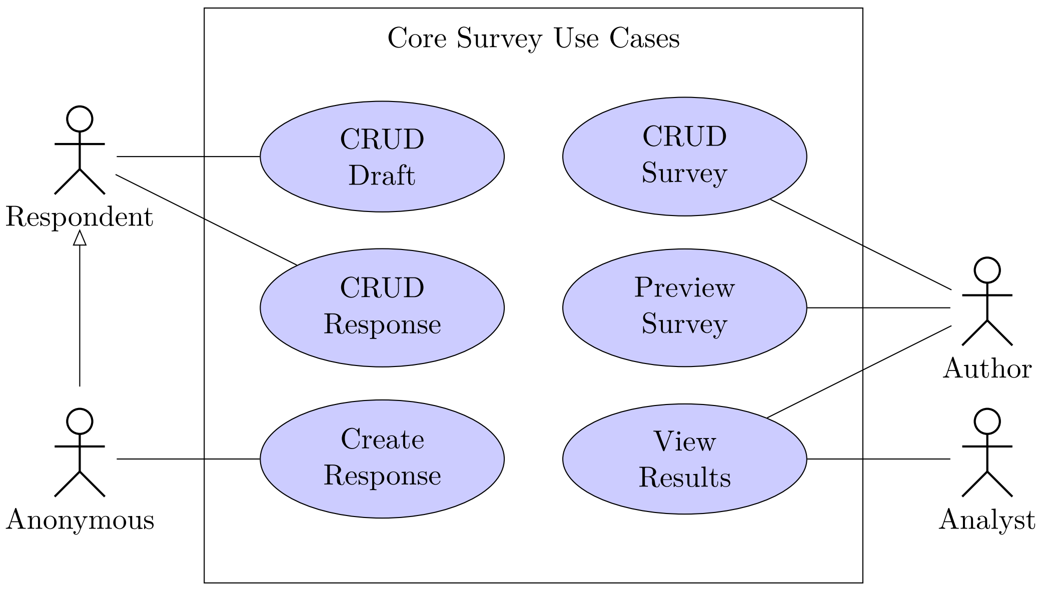


Figure A.1: The survey system's core use cases

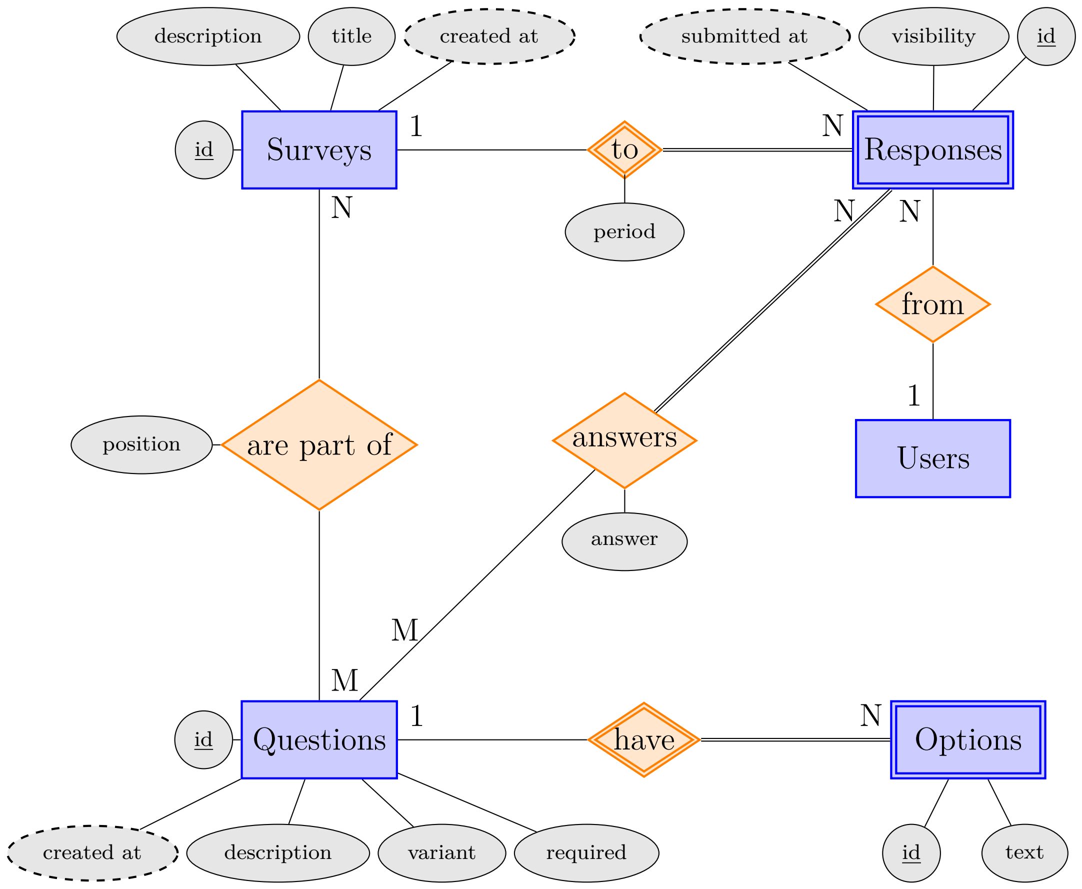


Figure A.2: The survey system's database design

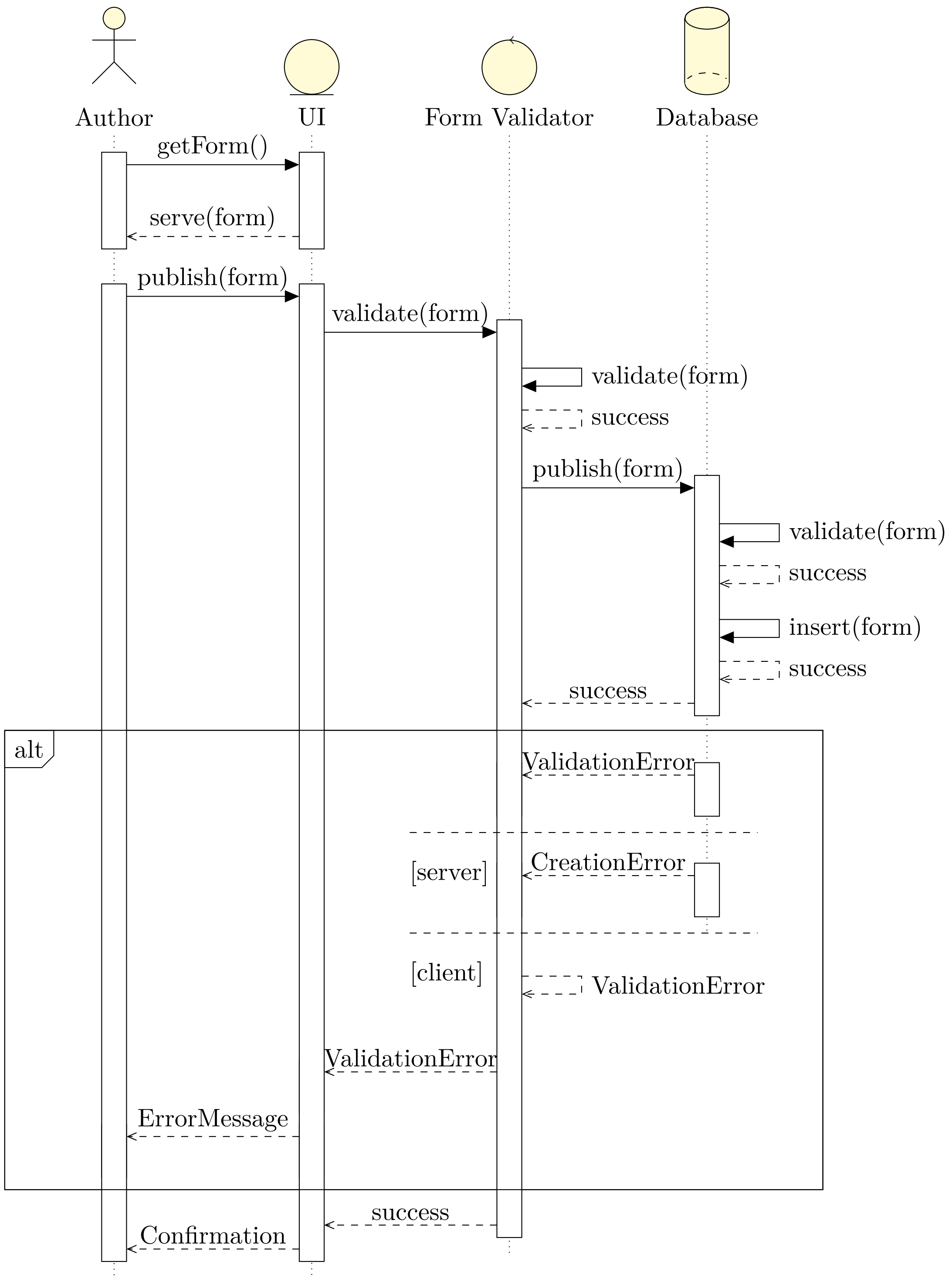
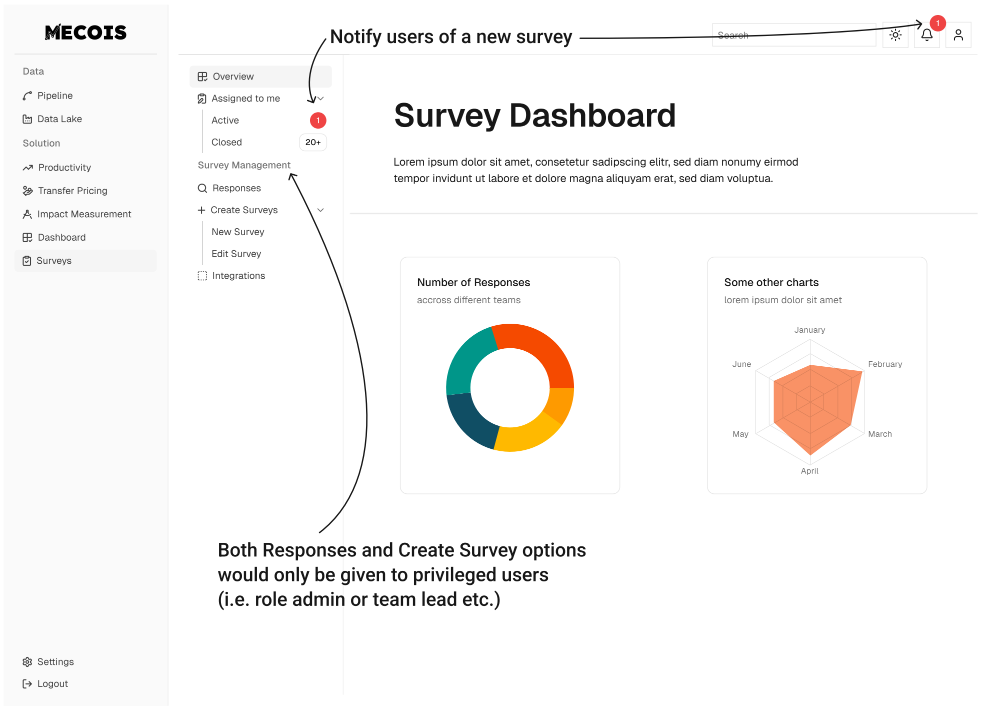


Figure A.3: A sequence diagram of the survey creation logic. The API entity has been removed for brevity.

A.4 User Interface Mockup



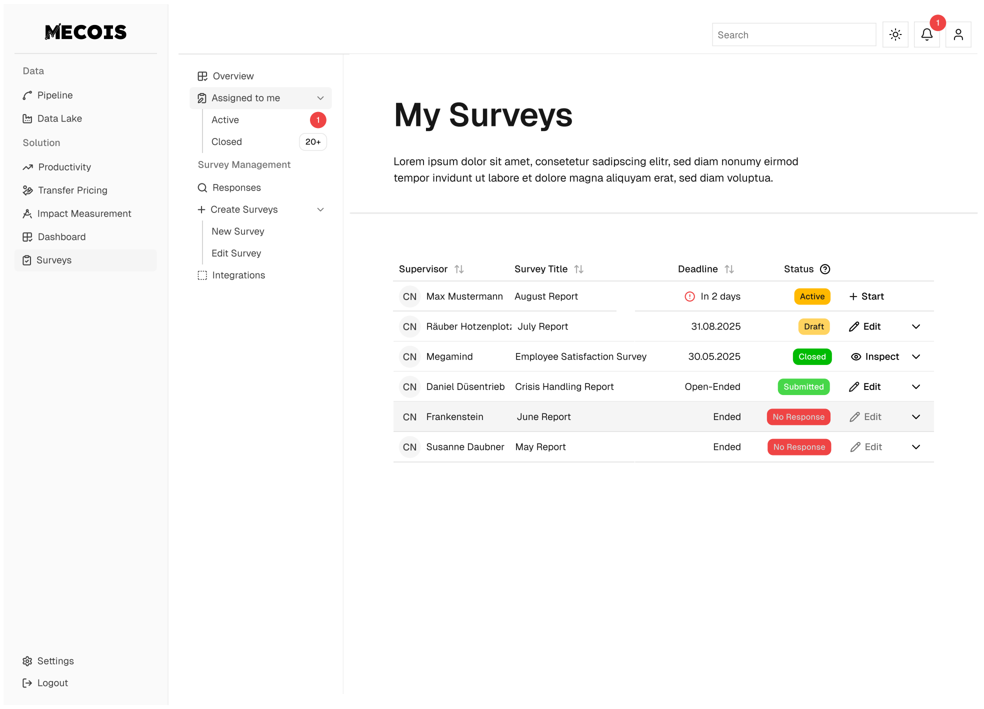
Notify users of a new survey

Survey Dashboard

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

Both Responses and Create Survey options would only be given to privileged users (i.e. role admin or team lead etc.)

(a) Overview page



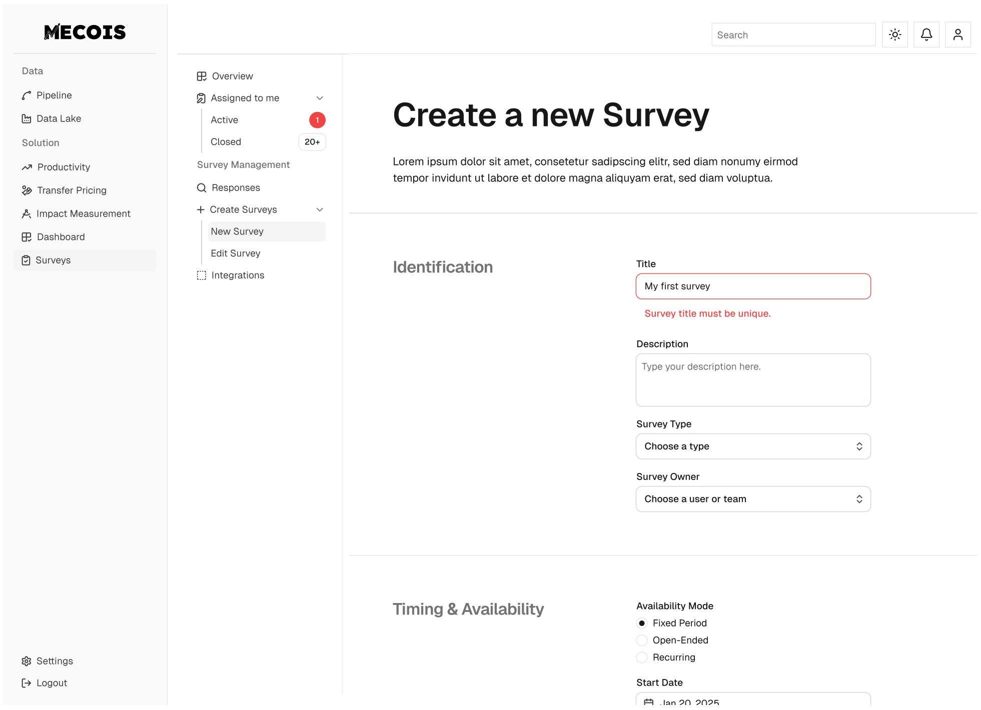
My Surveys

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

	Supervisor	Survey Title	Deadline	Status		
CN	Max Mustermann	August Report	In 2 days	Active	+ Start	
CN	Räuber Hotzenplotz	July Report	31.08.2025	Draft	Edit	⌄
CN	Megamind	Employee Satisfaction Survey	30.05.2025	Closed	Inspect	⌄
CN	Daniel Düsentrieb	Crisis Handling Report	Open-Ended	Submitted	Edit	⌄
CN	Frankenstein	June Report	Ended	No Response	Edit	⌄
CN	Susanne Daubner	May Report	Ended	No Response	Edit	⌄

(b) Assigned surveys page

Figure A.4: UI Mockups of (a) overview and (b) assigned surveys pages



Create a new Survey

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

Identification

Title: My first survey

Survey title must be unique.

Description: Type your description here.

Survey Type: Choose a type

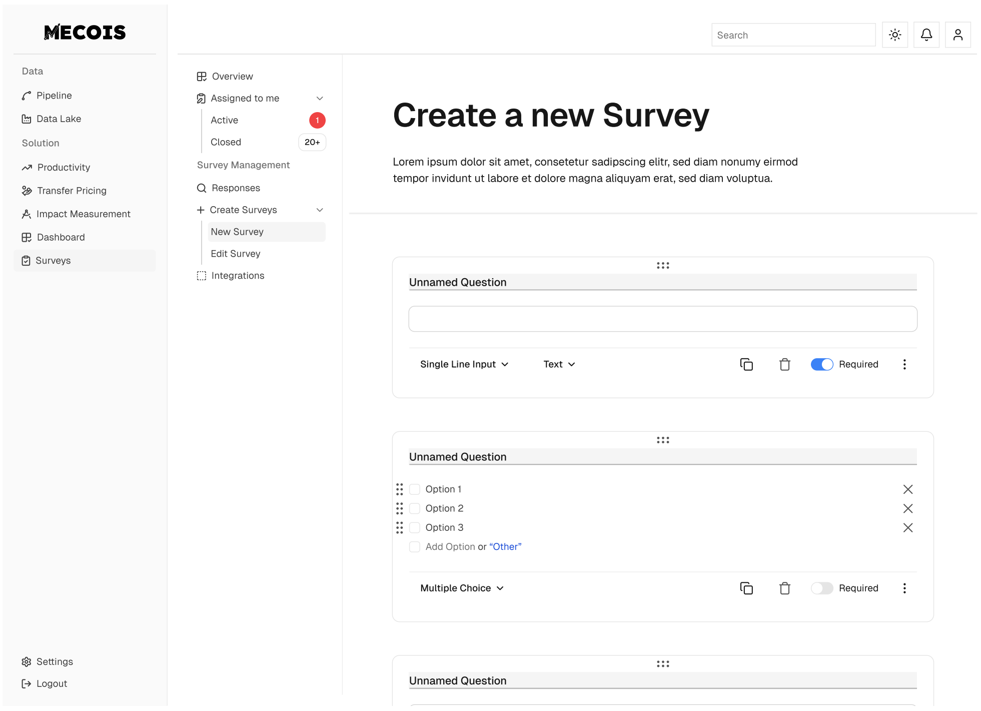
Survey Owner: Choose a user or team

Timing & Availability

Availability Mode
- [✓ Fixed Period]
- Open-Ended
- Recurring

Start Date: Jan 20, 2025

(a) Create survey configuration page



Create a new Survey

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

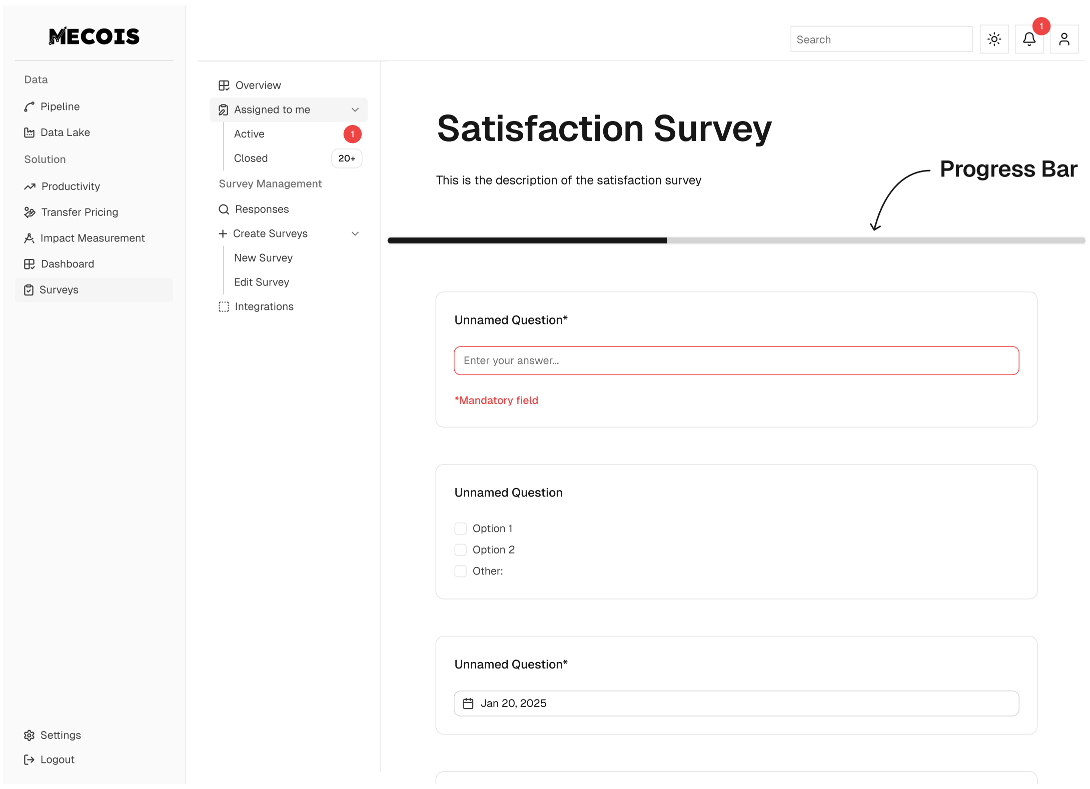
Unnamed Question

Unnamed Question
- Option 1
- Option 2
- Option 3
- Add Option or "Other"

Unnamed Question

(b) Create survey questions page

Figure A.5: UI Mockups of the create survey pages



Satisfaction Survey

This is the description of the satisfaction survey

Unnamed Question*

Enter your answer...

*Mandatory field

Unnamed Question

Option 1

Option 2

Other:

Unnamed Question*

Jan 20, 2025

Figure A.6: UI Mockup of the create response page

References

6Sense Insights, Inc. (2025). *Google forms - market share, competitor insights in survey* [6sense]. Retrieved September 20, 2025, from https://www.6sense.com/tech/survey/google-forms-market-share

Begel, A., & Zimmermann, T. (2014). Analyze this! 145 questions for data scientists in software engineering. *Proceedings of the 36th International Conference on Software Engineering*, 12–23. https://doi.org/10.1145/2568225.2568233

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J., & Steece, B. (2000). *Software cost estimation with COCOMO II* [Google-Books-ID: cRCMQQAACAAJ]. Prentice Hall.

Brown, A., Chang, A., Holtz, B., & D'Angelo, S. (2023). Developer productivity for humans, part 6: Measuring flow, focus, and friction for developers. *IEEE Software*, *40*(6), 16–21. https://doi.org/10.1109/MS.2023.3305718

Cha, S., Taylor, R. N., & Kang, K. (Eds.). (2019). *Handbook of software engineering*. Springer International Publishing. https://doi.org/10.1007/978-3-030-00262-6

D'Angelo, S., Lin, J., Dicker, J., Egelman, C., Hodges, M., Green, C., & Jaspan, C. (2024). Measuring developer experience with a longitudinal survey. *IEEE Software*, *41*(4), 19–24. https://doi.org/10.1109/MS.2024.3386027

DevOps Research and Assessment (DORA). (2021). *DORA | DORA's software delivery metrics: The four keys*. Retrieved September 21, 2025, from https://dora.dev

Devsoft Baltic OÜ. (2025). *SurveyJS - JavaScript libraries for surveys and forms* [SurveyJS]. Retrieved September 18, 2025, from https://surveyjs.io/

Elmasri, R., & Navathe, S. (2016). *Fundamentals of database systems*. Pearson International. Retrieved September 11, 2025, from https://elibrary.pearson.de/book/99.150005/9781292097626

Fenton, N., & Bieman, J. (2014, October 1). *Software metrics: A rigorous and practical approach, third edition* (0th ed.). CRC Press. https://doi.org/10.1201/b17461

Figma, Inc. (2025, September 14). *Figma* [Figma]. Retrieved September 14, 2025, from https://www.figma.com/de-de/

Forsgren, N., Storey, M.-A., Maddila, C., Zimmermann, T., Houck, B., & Butler, J. (2021). The SPACE of developer productivity: There's more to it than you think. *Queue*, *19*(1), Pages 10:20–Pages 10:48. https://doi.org/10.1145/3454122.3454124

Fox, N., Mathers, N., & Hunn, A. (2000, January 1). Surveys and questionnaires.

Github, Inc. (2025). *Surveyjs/survey-library: Free JavaScript form builder library with integration for react, angular, vue, jQuery, and knockout.* Retrieved September 20, 2025, from https://github.com/surveyjs/survey-library

Google LLC. (2025). *Google forms.* Retrieved September 18, 2025, from https://docs.google.com/forms/

Hunziker, S., & Blankenagel, M. (2024). Design science research design. In S. Hunziker & M. Blankenagel (Eds.), *Research design in business and management: A practical guide for students and researchers* (pp. 97–116). Springer Fachmedien. https://doi.org/10.1007/978-3-658-42739-9_6

IEEE 29148-2018. (2018, November). https://doi.org/10.1109/IEEESTD.2018.8559686

Jaspan, C., & Green, C. (2023). A human-centered approach to developer productivity. *IEEE Software*, *40*(1), 23–28. https://doi.org/10.1109/MS.2022.3212165

Jaspan, C., & Green, C. (2025). Measuring productivity: All models are wrong, but some are useful. *IEEE Software*, *42*(2), 13–17. https://doi.org/10.1109/MS.2024.3511735

Jaspan, C., Jorde, M., Egelman, C., Green, C., Holtz, B., Smith, E., Hodges, M., Knight, A., Kammer, L., Dicker, J., Sadowski, C., Lin, J., Cheng, L., Canning, M., & Murphy-Hill, E. (2020). Enabling the study of software development behavior with cross-tool logs. *IEEE Software*, *37*(6), 44–51. https://doi.org/10.1109/MS.2020.3014573

Jaspan, C., & Sadowski, C. (2019, May 8). No single metric captures productivity. https://doi.org/10.1007/978-1-4842-4221-6_2

Jones, C. (1994). Software metrics: Good, bad and missing. *Computer*, *27*(9), 98–100. https://doi.org/10.1109/2.312055

Kankaraš, M., & Capecchi, S. (2025). Neither agree nor disagree: Use and misuse of the neutral response category in likert-type scales. *METRON*, *83*(1), 111–140. https://doi.org/10.1007/s40300-024-00276-5

Linåker, J., Sulaman, S. M., Maiani de Mello, R., & Höst, M. (2015). *Guidelines for conducting surveys in software engineering.* Department of Computer Science, Lund University.

LINFO. (2005). *Best programming quotations* [The linux information project]. Retrieved September 20, 2025, from https://www.linfo.org/q_programming.html

Mavin, A., Wilkinson, P., Harwood, A., & Novak, M. (2009). Easy approach to requirements syntax (EARS). *2009 17th IEEE International Requirements Engineering Conference*, 317–322. https://doi.org/10.1109/RE.2009.9

Meyer, A. N., Fritz, T., Murphy, G. C., & Zimmermann, T. (2014). Software developers' perceptions of productivity. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 19–29. https://doi.org/10.1145/2635868.2635892

Meyer, A. N., Zimmermann, T., & Fritz, T. (2017). Characterizing software developers by perceptions of productivity. *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 105–110. https://doi.org/10.1109/ESEM.2017.17

Murphy-Hill, E., Jaspan, C., Sadowski, C., Shepherd, D., Phillips, M., Winter, C., Knight, A., Smith, E., & Jorde, M. (2021). What predicts software developers' productivity? *IEEE Transactions on Software Engineering*, *47*(3), 582–594. https://doi.org/10.1109/TSE.2019.2900308

Object Management Group®. (2017). Unified modeling language. Retrieved July 28, 2025, from https://www.omg.org/spec/UML/2.5.1/PDF

Peffers, K., Tuunanen, T., Rothenberger, M., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*, 45–77.

Rod Stephens. (2022, November 15). *Beginning software engineering* (2nd ed.). Wiley. https://doi.org/10.1002/9781394320592

Schwaber, K., & Sutherland, J. (2020). Scrum guide. Retrieved September 15, 2025, from https://scrumguides.org/scrum-guide.html

Sommerville, I. (2018). *Software engineering* (K. Pieper & P. Alm, Trans.; 10., aktualisierte Auflage). Pearson.

Strathern, M. (1997). 'improving ratings': Audit in the british university system. *European Review*, *5*(3), 305–321. https://doi.org/10.1002/(SICI)1234-981X(199707)5:3<305::AID-EURO184>3.0.CO;2-4

Supabase, Inc. (2025, September 20). *Supabase on GitHub*. Retrieved September 20, 2025, from https://github.com/supabase/supabase?tab=Apache-2.0-1-ov-file

The PostgreSQL Global Development Group. (2025, August 14). *5.9. row security policies* [PostgreSQL documentation]. Retrieved September 20, 2025, from https://www.postgresql.org/docs/17/ddl-rowsecurity.html

Walston, C. E., & Felix, C. P. (1977). A method of programming measurement and estimation. *IBM Systems Journal*, *16*(1), 54–73. https://doi.org/10.1147/sj.161.0054

Wolter, T., & Riehle, D. (2025). *Creating a composite productivity index* [In submission].