Politecnico di Torino

Software Engineering

Academic year 2024/2025

Master's thesis, July 2025

Optimizing Business Processes with Low-Code Development: From Design to Production

Supervisor:

Tania Cerquitelli

Company supervisor:

Olivier Leyens

Author:

Andrea Scamporrino

Summary

This thesis presents a solution to the efficiency issues experienced by Toyota Motor Europe in its testing procedures, which was slow, error-prone and managed through disconnected Excel files and SharePoint sites. The primary objective was to design and develop a modern, centralized software application to manage the entire life-cycle of these requests, from their creation to their completion. Before development, the application's user interface was carefully designed and prototyped in Figma. This preliminary step was essential to validate the design against user requirements, ensuring the final solution would be both effective and intuitive. This validated design was then implemented using a low-code methodology with the Microsoft Power Platform. The new system features a three-layer architecture: a user interface built with Power Apps, a back-end using SharePoint lists and an automation layer powered by Power Automate to handle tasks like calendar synchronization and user notifications. The project successfully replaced the fragmented and inefficient process with a robust solution that provides a single source of truth for all test data, improving productivity and the overall user experience. This success is demonstrated by the dramatic increase in system performance, with the average time to open a test request dropping from 23.93 seconds in the old Excel system to just 3.18 seconds.

Table of Contents

List of Figures

Source Code

Chapter 1

Introduction

1.1 Project motivations

In the world of large organizations, many essential processes are complicated, slow and misorganized. This results in time waste, delays and reduced productivity. It is from this context that the inspiration of this project came from: finding ways to speed up and streamline a complex business process to make it more accessible to a wider range of people [1].

In this work, I will examine a rather complex and confusing business process of Toyota Motor Europe, and then report my attempt to streamline and automate the flow under investigation to make it faster and more efficient. This goal has been achieved through several stages, which will be described in detail in the following sections. Whenever an improvement is necessary, the initial step involves gaining a comprehensive understanding of the existing system, which is achieved through direct users' interactions observation and through a deep examination of the constituent steps. This enables an analysis of the system's critical issues and strengths, which can then enforce its enhancement. The flow under analysis is how requests for car tests and component tests are handled. In the original flow, requests are created by engineers, scheduled by group leaders, and performed by technicians as follow:

1. The engineers fill out some Excel sheets with all the data needed to execute the task and then upload those files to SharePoint.

2. The group leaders then enter the requests into a calendar created on Excel.

3. Finally, the technicians is able to see the request to be performed in the Excel calendar, execute the work and post the results on Teams.

This method of handling requests can bring to several problems:

- The status of all these requests is not tracked, resulting in confusing situations where engineers do not know whether a test has been scheduled or completed. One could consult the Excel calendar or ask the technician directly; however, both options would require a significant investment of time, either searching for the request among all the others, or finding and contacting the assigned technician.

- Test requests are composed of a large number of details. It is often necessary to retrieve real-time data from disparate data structures and display them in Excel. Furthermore, operations must be performed based on user interactions, which are recorded using macros. These operations significantly slow down the opening of Excel sheets, often resulting in error messages such as "Microsoft Excel has stopped working" [2], [3].

- The system for scheduling requests is not optimized. It is error prone due to the possibility of scheduling the same request on multiple occasions. Furthermore, it is not integrated with the Outlook calendar, which is commonly used for managing other professional obligations.

- A change request by an engineer in the proximity of the scheduled time for the test may result in complications. This is due to the fact that the technician may have an open version of the file that is outdated, which could result in the test being performed incorrectly.

- It is not feasible to analyze the data coming from all the requests. It may, however, be beneficial to aggregate the requests according to the vehicle in question.

- The diversity of test requests introduces a challenge in terms of data management. Different Excel templates are used to facilitate the creation of these requests, but the information they contain may be represented in disparate locations. This makes it more difficult for all users to search or enter information in test requests with different formats.

In consideration of all the issues listed above, it becomes evident that this approach to managing requests for tests not only impedes operational efficiency but also introduces the potential for a significant high error and miscommunication rate. These issues indicate a urgent need for a more robust, integrated, and user-friendly solution to streamline the process, addressing the specific requirements to all the involved stakeholders. By identifying these problems, I have established the basic objectives of the project, namely to design and implement a system that will enhance the functionality and reliability of test request management. The following section, *Project Objectives* 1.2, provides a comprehensive description of the proposed solution and its scope of coverage. This strategic approach guides the response to inefficiencies and the transformation of the process into a seamless and easily accessible framework.

1.2 Project objectives

The main objectives of the project are the design, development and validation of a new system intended to manage the entire life-cycle of test requests for vehicles and their components. The project was carried out as part of a six-month internship at Toyota Motor Europe. The methodology used is based on a low-code approach, to accelerate the process and guarantee greater flexibility.

1.2.1 Functional requirements

This has been achieved through the creation of a customized application using Microsoft Power Apps. The application is specifically tailored to meet the diverse requirements of its users, offering an efficient and flexible solution for managing test requests within the automotive industry. The application developed in this project is designed to meet several key functional requirements:

- *Test request creation*: engineers can efficiently create test requests, either for vehicles or individual components, using intuitive and user-friendly forms.
- *Data integration and pre-filling*: the application is integrated with existing data structures to pre-filled fields in the forms, reducing manual inputs and minimizing the risk of errors.
- *Comprehensive request management*: a centralized list of all test requests is provided, with filtering options based on key criteria such as vehicle type and requester, ensuring easy navigation and management.
- *Detailed request view and operations*: each test request is presented in a standardized, detailed format, enabling users to quickly locate the required informations. Additionally, each request can be edited, deleted or duplicated, facilitating the efficient creation of similar requests.
- *Request clarification*: users can request clarifications directly within the system to resolve ambiguities and obtain additional information.
- *Scheduling and calendar integration*: test requests can be scheduled using a dedicated calendar for every request type. A preliminary section highlights unscheduled requests. The calendar is integrated with Outlook, automatically creating events in both shared and the personal calendars of the engineers and technicians involved.

- *Life-cycle monitoring*: the application automates the tracking of each request's status, ensuring a life-cycle management from the request creation to its fulfillment.

1.2.2 Non-Functional requirements

In addition to the key functional requirements, it is important to set some non-functional requirements. These provide supplementary details to the functional capabilities, defining how the system operates under different conditions. These requirements ensure the application is not only functional but also reliable, efficient, and secure, addressing immediate user needs while accommodating future scalability demands.[4]

Performance

Performance is critical, as the application must deliver a fast and responsive user experience. Users should interact smoothly with forms, data lists, and other features, even if the system scales to handle growing data volumes and user numbers. Scalability is closely linked, ensuring the system can meet increasing demands without sacrificing performance or requiring major redesigns.

Reliability

Reliability and availability are equally vital, guaranteeing minimal downtime and robust functionality. Error detection and recovery mechanisms protect data integrity and foster user trust. Meanwhile, usability is a top priority, with the interface designed for simplicity and ease of navigation. Features like pre-filled forms and confirmation popups minimize errors and improve efficiency. Furthermore, the old system will remain available as a back-up option in case the new system encounters any issues. This ensures that operations can continue using the Excel-based system while any problems with the new application are being resolved, guaranteeing the uninterrupted completion of processes.

Data security

Given the sensitivity of data managed by the system, security takes the center stage. Strong access controls ensure only authorized personnel can perform specific actions, while compliance with data protection standards safeguards sensitive informations. Integration with Microsoft Office 365 services, such as Outlook and Teams, is optimized for seamless functionality, enabling synchronized scheduling and smooth communication between users.

Maintainability

Finally, the application's maintainability and portability ensure its adaptability and longevity. A modular and clear design simplifies updates and future enhancements to meet evolving organizational needs. Cross-device accessibility, including desktops and mobile platforms, further broadens the application's usability and relevance in varied work environments.

1.2.3 Methodology

The methodology selected for the application is the low-code development, using the tools available in the Microsoft Power Platform solution. These tools can be easily integrated within the existing Microsoft environment already in use by the company, providing a familiar environment while maximizing operational effectiveness. A low-code approach accelerates the time to implement an application over coding from scratch. Moreover, the tools are very simple and flexible, which means that adjust or improve the application is easy fitting whatever other processes the business needs to follow.

1.3 Structure of the thesis

In the upcoming chapters, the various stages of project development will be analyzed in detail according to the following structure:

- *Chapter 2* "**Overview of the technologies used**": the chapter will present a comprehensive overview of the technologies employed, with a particular emphasis on the low-code methodology. The core tools of the Power Platform and a design and prototyping tool will be subjected to a detailed examination, with a focus on their distinctive capabilities and functionalities.

- *Chapter 3* "**Application design and architecture**": the following section will address the design phase of the application's graphic interface. This phase will entail an analysis of the users' needs, with the objective of creating a suitable UI and UX. Additionally, this chapter will analyze the architectural choices of the application, with the aim of identifying the optimal alternatives.

- *Chapter 4* "**Application development**": this chapter will outline the principal stages involved in the implementation of the application, beginning with an examination of the configuration and management of data on SharePoint. It will continue with an analysis of the main screens developed in Power Apps and a description of the flows created with Power Automate.

- *Chapter 5* "**Deployment and results**": this chapter will begin by analyzing the aspects of Power Platform environments and the management of security and permissions. It will then cover the phase of releasing the application to users, first as a beta to selected users and later to the entire department. In addition, the results of the chosen approach will be evaluated through the testing of the application and an analysis of its performance.
- *Chapter 6* "**Conclusion**": this chapter presents the final conclusions of the work, discussing the challenges faced and the future potential of the system.

Chapter 2

Overview of the technologies used

This chapter provides a complete overview of the technologies that were used to design, develop and deploy the test request management application. The project is based on a low-code methodology, which allows for fast and flexible development. We will examine the core tools of the Microsoft Power Platform, including Power Apps and Power Automate, which were used to build the application and automate its workflows. We will also discuss SharePoint Online, which serves as the database for the system, and Figma, the tool used for designing and prototyping the user interface before development began. Understanding these tools is essential to appreciate the architectural choices and the implementation process described in the following chapters.

2.1 Low-Code

Low-code is a modern approach to software development that allows the creation of applications through graphical user interfaces and configuration instead of traditional, hand-written computer programming. It represents a fundamental shift in how software is built, moving from complex lines of code to visual, model-driven development. This methodology is built on the principle of abstraction: the low-code platform manages the complex underlying infrastructure, database connections and deployment processes, allowing developers to focus on solving business problems. The core of low-code development lies in its visual environment. Developers typically use drag-and-drop editors to assemble user interfaces from a library of pre-built components, such as forms, charts and buttons. Similarly, the application's logic and workflows are often designed visually by connecting logical blocks, defining rules and modeling data relationships. This visual approach does not just make development faster; it also makes it more accessible to a wider range of individuals. A key characteristic of low-code platforms is the balance they strike between simplicity and power. They are designed to empower not only professional developers but also "citizen developers" [5], business analysts or domain experts who have a deep understanding of the business needs but may have limited coding skills. However, unlike no-code platforms which are often more restrictive, low-code platforms often allow the use of custom components. This means that when a specific or complex functionality is needed, a professional developer can extend the application by writing scripts or code, offering greater flexibility. This approach was selected for this project because it offers several key advantages that directly address the project's objectives.

- *Development speed*: low-code platforms significantly accelerate the time it takes to build and deliver a working application when compared to coding from scratch. This was particularly important for delivering a complete solution within the project's timeframe.
- *Flexibility and agility*: the tools used are very simple and flexible, which means that the application can be easily adjusted or improved to fit new business requirements. This adaptability ensures the system can evolve along with the organization's needs.
- *Seamless integration*: a major benefit of using the Microsoft Power Platform is its strong integration with the existing company IT environment. The platform connects easily with standard tools like Outlook and Microsoft Teams, which was a critical requirement to solve the problems of the old, disconnected system.
- *Accessibility*: by reducing the amount of complex code, low-code development makes it possible for a wider range of people to contribute to building solutions. This aligns with the project's goal of creating a more accessible and user-friendly process for everyone involved.

In summary, the low-code approach provided the right balance of speed, flexibility, and integration needed to successfully replace the old, inefficient system with a modern and robust application.

2.2 Power Platform

The Microsoft Power Platform is a collection of low-code tools designed to work together as one single system. Its main purpose is to allow users to build business applications, automate work processes, and analyze data easily [6]. The platform is composed of five tools, each serving a different purpose:

- *Power BI*, for analyzing and visualizing data;
- *Power Apps*, for building custom applications for internal use;
- *Power Pages*, for building secure, external-facing business websites;
- *Power Automate*, for creating automated workflows;
- *Microsoft Copilot Studio*, for developing custom AI copilots and chatbots.

These tools are not just separate products, they are powerful because they are built on a shared foundation that lets them work together smoothly. The two most important concepts of this foundation are its powerful connectivity model and its structured approach to application management. The following sections will explore these two fundamental concepts in more detail.

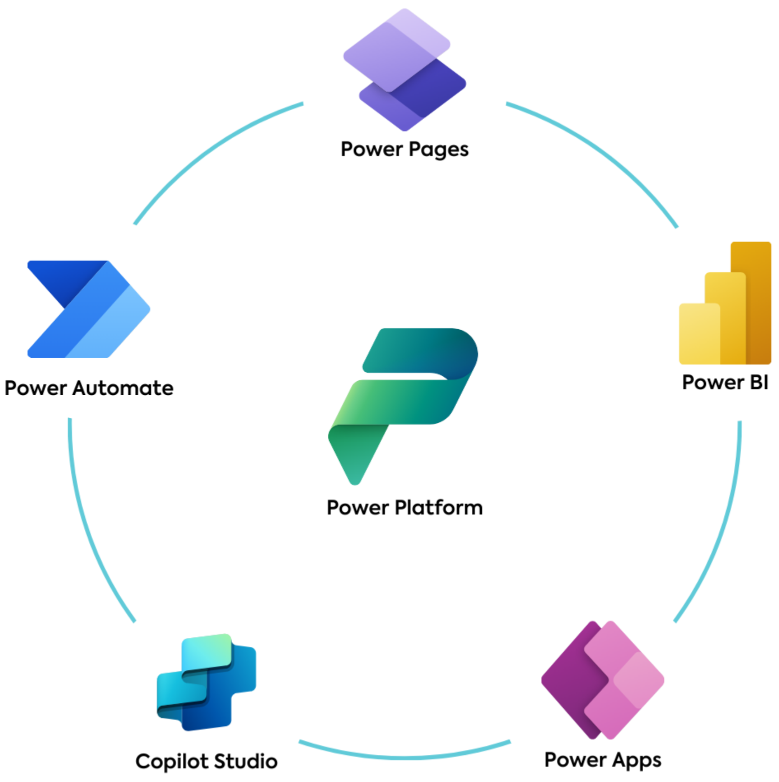


Figure 2.1: Power Platform tools

2.2.1 Connectors

The core of the Power Platform's integration capability lies in its extensive framework of connectors. A connector is essentially a pre-built "bridge" that allows Power Platform services, like Power Apps and Power Automate, to easily and securely communicate with other services and data sources. This simplifies what would traditionally be a complex programming task of writing custom integrations. Instead of writing code to connect to an external service's API, a developer can simply use the corresponding connector. The platform offers hundreds of standard connectors for a wide range of popular services, both inside and outside the Microsoft ecosystem [7]. For this project, the key connectors used were for SharePoint, which acted as the database, and for Outlook and Microsoft Teams, which enabled calendar synchronization and automated user notifications. If a pre-built connector for a specific system does not exist (for example, for a legacy internal database), the platform also allows developers to create their own Custom Connectors. This extensibility ensures that the Power Platform can be integrated with virtually any system, making it a highly flexible solution for enterprise environments. This robust connectivity was fundamental to the success of the project, as it allowed the application to be seamlessly integrated into the existing daily workflows of the employees.

2.2.2 Environments

A fundamental concept for managing applications and data within the Power Platform is the use of environments. An environment is best understood as an isolated container where an organization can securely store, manage, and share its business applications, automated workflows, and data. Each environment acts as a clear boundary, separating resources based on their intended purpose [8]. The primary role of environments is to support a structured Application Lifecycle Management (ALM) process. This allows an organization to create separate spaces for different stages of work. For example, a sandbox environment can be used for development and testing, providing a safe space for developers to experiment with new features and fix bugs without any risk to the live system. A production environment, on the other hand, is where the final, official versions of applications are deployed for their intended daily use by employees. This separation is critical because environments provide strong data isolation. Each environment has its own separate database and storage, meaning an application in a development environment cannot accidentally access or modify the data in the production environment. Furthermore, access to each environment is controlled by specific user roles, creating a strong security boundary that ensures only authorized users can create or change applications.

2.3 Power Apps

Power Apps is a tool that provides a rapid application development environment to build custom apps for business needs [9]. Power Apps offers two primary development models: Canvas Apps and Model-driven Apps [10]. A careful analysis of their capabilities was crucial in selecting the appropriate framework for this project's requirements.

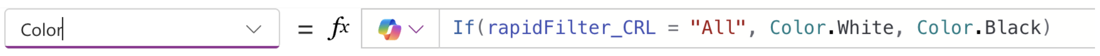
- *Model-driven Apps* are data-centric by design. Their user interface is largely generated automatically based on the structure of the underlying data, typically hosted in Microsoft Dataverse. This model is highly effective for process-driven applications that require standardized forms, views, and business process flows, such as CRM or case management systems [11]. However, it offers limited control over the UI's layout and visual design.

- *Canvas Apps*, in contrast, provide complete creative control. They start from a blank canvas, allowing the developer to design the user interface with precision by dragging and dropping components. This flexibility is essential for creating highly customized user experiences tailored to specific tasks and workflows [12].

For the scope of this thesis, the Canvas App model was unequivocally the more suitable choice. The project requirements demanded a level of UI/UX flexibility that only a Canvas App could provide. This approach ensured that the design of the final application could be precisely aligned with the specific operational workflows of the engineers, technicians and group leaders.

2.3.1 Core components of Canvas Apps

Developing a Canvas App involves assembling and configuring various components on a screen [13]. Each component, from a simple button to a complex data gallery, has a set of properties (e.g., `Color`, `Text`, `OnSelect`) that can be controlled using *Power Fx*, a low-code formula language similar to Microsoft Excel [14]. This formula-based approach allows for the creation of dynamic and responsive user interfaces. For instance, a component's visibility can be determined by a user's role, or its color can change based on the status of a data record, as shown by the `if` condition in the formula bar (Figure 2.2).



```
If(rapidFilter_CRL = "All", Color.White, Color.Black)
```

Figure 2.2: Power Fx formula bar

While there are many components available, the *Gallery* and *Form* controls are fundamental for building data-driven applications and were used extensively in this project.

The Gallery control

The *Gallery* is a component for displaying a set of records from a data source [15]. In this project, it was used to show lists of test requests, vehicle histories, and unscheduled tasks. Its most important property is `Items`, which defines the data to be displayed. This property is not static; it can be dynamically configured with *Power Fx formulas* to filter, sort, and search data in real-time. For example, to display only active requests, the formula would be:

```
Filter(YourDataSource, Status = "Active")
```

Source Code 2.1: Example of filtering records in a Gallery's Items property.

Within a gallery, the `ThisItem` record scope allows access to the data of each individual row, making it easy to display different fields (e.g., `ThisItem.Title`). Furthermore, the `GalleryName.Selected` property identifies the specific item the user has selected, enabling interactive functionalities such as navigating to a detail screen or deleting a record.

The Form control

The *Form control* is the primary tool for creating, viewing, and editing individual data records. It simplifies data management by automatically generating input fields (e.g., text boxes, date pickers) corresponding to the columns in the connected data source [16]. The Form operates in distinct modes:

- *Edit Mode*: used for both creating new records (`NewForm(FormName)`) and modifying existing ones (`EditForm(FormName)`).
- *Display Mode*: used to show a record in a read-only format.

Key functions like `SubmitForm(FormName)` and `ResetForm(FormName)` handle the logic for saving changes or clearing inputs. The Form control also inherits data validation rules from the source, ensuring data integrity by enforcing required fields and data types. This component was essential for building the multi-step interface for submitting and editing test requests.

2.3.2 Data source integration

A core strength of Power Apps is its ability to seamlessly connect to various data sources through a system of connectors. Connecting to a data source is a straightforward process handled within the *Data panel* of the Power Apps Studio. Once connected, the data becomes available to all controls in the application, allowing the display and the manipulation [17].

2.3.3 State management with variables

To create a dynamic and responsive application, it is necessary to manage state and temporary data. Power Apps provides three types of variables for this purpose [18], each with a specific scope and use case:

- *Global variables*: declared with the `Set()` function, they hold values that are accessible across all screens of the application. They are ideal for storing application-wide information, such as the signed-in user's profile or configuration settings.

- *Context variables*: declared with the `UpdateContext()` function, their scope is limited to the screen on which they are created. They are used for managing the state of a single screen, such as controlling the visibility of a pop-up dialog or storing user input before it is saved.

- *Collections*: created and managed with functions like `Collect()`, `Clear()`, and `ClearCollect()`, collections are in-memory tables used to store temporary datasets. They are particularly useful for caching data from a source to improve performance or for gathering user-inputted data before patching it to the backend in a single operation.

Effective use of these variables was fundamental to building the logic of the application, from navigating between screens to implementing complex functionalities like the interactive planners.

2.4 Power Automate

Power Automate is a tool for creating automated workflows. Its core function is to orchestrate and automate repetitive tasks and business processes across a multitude of applications and services without the need for custom code [19]. In the context of this project, Power Automate served as the essential automation and integration layer, operating behind the scenes to connect the Power App with SharePoint, Outlook, and Microsoft Teams, thus ensuring a seamless and efficient workflow.

2.4.1 Flows

The fundamental building block in Power Automate is a flow. Each flow is a sequence of steps designed to perform a specific automated task. Every flow is composed of two primary elements: a trigger and one or more actions.

- *Triggers*: a trigger is the specific event that initiates a workflow. A flow has exactly one trigger. This event can be something that happens within a connected service, such as a new item being created in a SharePoint list, or it can be initiated manually by a user.
- *Actions*: actions are the operations that the flow performs after it has been triggered. A flow can have many actions, each carrying out a specific task.

Power Automate provides different types of flows to accommodate various automation scenarios. Workflows can be triggered manually by a user, which are known as Instant Cloud Flows. In this project, this was the primary method used for user-initiated tasks; for example, flows were started by a user clicking a button within the Power App to schedule a new test request. Another type is the Automated Cloud Flow, which starts automatically in response to a predefined event, such as a new file being created in a SharePoint library. Finally, Scheduled Cloud Flows run at a specific time, on a recurring basis like daily or monthly. This type of flow was used in the project to run a monthly archiving process, which automatically moves old test requests from the active lists to an archive to maintain system performance [20].

2.4.2 Logic and data handling in flows

A workflow in Power Automate can do more than just follow a list of steps; it can also make decisions and repeat tasks. A flow can use simple If/Then logic to perform different actions depending on the data it receives. It is also capable of repeating an action for many items at once by creating a loop [21]. For example, a loop was used in this project to process and copy every file attachment from one test request to another.

2.5 SharePoint

SharePoint is a tool that simplify the management of documents and information within organizations [22]. The platform enables the creation of customized websites that help share content and access structured data from any device, making it an indispensable tool for teamwork and corporate communication. With SharePoint, organizations can create dedicated sites for their work-groups, use document libraries to efficiently store and organize files, and configure customized lists to manage specific tasks such as calendars, projects and reports.

2.5.1 Structure

The SharePoint structure is designed to offer an organized and flexible environment based on sites, which represent the spaces where information is stored. Within sites, different elements such as images, links and text can be added. Navigation between sites and elements is possible by a system of link bars and intuitive menus. In addition to the simple elements mentioned above, it is also possible to add more complex components such as lists and document libraries.

Lists

Lists in SharePoint allow users to store and view data in the same way as a relational database. In fact, lists are organized into rows and columns that represent the elements and properties of them, respectively [23]. This is particularly useful in Power Apps, where lists are often used as data sources to create dynamic applications. This integration allows data to be read, edited, and updated in real time, providing users a productive experience. A key aspect of lists is the ability to customize columns to meet specific business needs. For example, text columns can be created to store descriptions, numeric columns for quantitative values, choice columns for predefined options, and ’lookup’ columns to link data between different lists. In addition, SharePoint offers advanced columns, such as ’person’ columns, which are useful for associating items with specific users. Navigating and interacting with lists is simplified by customizable views that allow filtering, sorting and grouping data to highlight relevant information.

Document Libraries

SharePoint document libraries are one of the most effective and flexible features used to organize files in any organization. It is used to store documents of all types, including large files, with a maximum single file size of 250 GB [24]. This differs from, for example, SharePoint lists, which have greater restrictions on the size of attachments that can be added. These libraries provide a central platform that makes it easier for teams to organize, access and collaborate. They also offer advanced versioning capabilities that allow tracking of changes made to documents and enable reverting to previous versions if necessary. Document libraries also support the use of custom metadata to improve document organization and search. For example, additional columns can be created to categorize files by tags, dates or other relevant information, making it easier to filter and find the content you need.

2.5.2 Security and permissions

Security and permission management in SharePoint are central to ensuring the protection of corporate data and controlled access to information. SharePoint offers a hierarchical and flexible structure, allowing permissions to be assigned at different levels, from sites to lists. This approach provides granular control, allowing access to be personalized to specific needs.The system provides predefined permission levels such as Owners, Members and Visitors, which can be used to quickly and consistently define who can access, edit or view content. These roles can be further customized for particular scenarios. To simplify management, SharePoint uses a permission inheritance system, in which subsections inherit the permissions of the top level. However, this inheritance can be broken to apply unique security configurations to certain elements. Users can be managed individually or through security groups, which aggregate multiple users to simplify administration [25].

2.6 Figma

Figma is a design and prototyping tool used for creating user interfaces and user experiences [26]. Unlike traditional design software installed on a single computer, Figma operates primarily in the cloud, which is the key to its powerful collaborative features. Its main purpose is to provide a single, all-in-one platform for the entire product design lifecycle, from initial mockups to high-fidelity, interactive prototypes. For this thesis, it was the tool chosen to define the complete visual and interactive design of the application before the development phase began in Power Apps.

2.6.1 Core functionalities

Figma integrates several key capabilities into one platform, allowing design teams to work efficiently without needing multiple, disconnected tools. At its core, Figma is a powerful vector graphics editor, which allows designers to create scalable UI elements such as icons, buttons, and entire screen layouts with precision. Advanced features like *Auto Layout* help in creating responsive designs that adapt to different screen sizes, while reusable components ensure a consistent visual language across a large application.

However, Figma's capabilities extend beyond static design by enabling the creation of dynamic, clickable prototypes. Designers can connect different screens and UI elements to simulate user flows and application navigation, providing a realistic preview of the final product long before any code is written. Perhaps Figma's most defining characteristic is its real-time collaboration. Because it is cloud-based, multiple stakeholders, designers, developers, and project managers, can access and work on the same design file simultaneously. This streamlines the review process, allowing team members to leave feedback directly on the designs and see changes as they happen.

2.6.2 Role in design systems

Figma is also a central tool for building and maintaining *Design Systems*. A design system is a comprehensive library of reusable components, style guidelines (such as colors and typography), and best practices. By creating a design system in Figma, an organization can ensure that all its digital products have a consistent look and feel. This not only improves the user experience but also accelerates the development process, as it provides a single source of truth for both designers and developers to follow.

In summary, Figma's combination of advanced design features, interactive prototyping, and powerful collaboration tools makes it an industry-standard choice for modern product design.

Chapter 3

Application design and architecture

3.1 User requirements analysis

The key to designing effective application is understanding the users' needs [27]. In the context of this project, the application was intended to support and streamline internal workflows at Toyota Motor Europe, particularly those related to test request creation, planning, execution, and result reporting within the Powertrain department. Therefore, it was crucial to start the design phase with an in-depth analysis of the current environment and the requirements expressed by the future users of the system.

The first step in this process was to identify the key stakeholders involved in the application's ecosystem. Through preliminary meetings with group leaders, it was clear that the application would need to serve multiple user categories, each with their own roles and functionalities. On one side, there are the technicians, responsible for carrying out scheduled tasks on physical test benches and recording relevant data. On the other, group leaders play a coordinating role, reviewing and assigning test requests, monitoring their progress and ensuring that the workload was distributed efficiently. A third group is composed of engineers responsible for creating detailed test requests, including all relevant parameters and requirements.

During the analysis of user requirements, it also became clear that the application had to fit four different types of testing categories, each with its own operational logic and specific needs. These categories are:

- *Chassis Dynamometer (CDY) tests:* typically conducted to simulate real driving conditions on a stationary vehicle. This category involves assessments such as emissions measurement, fuel consumption analysis, and the evaluation of vehicle performance under varying load and speed profiles. The tests often require precise coordination of multiple systems and instruments to replicate road conditions as accurately as possible.

- *The Workshop category:* refers to testing activities performed on vehicles that are not on a dyno. These include physical inspections, component replacements, electronic diagnostics, and functional validations that are preparatory or complementary to other test campaigns. The activities in this area are more hands-on and involve significant interaction with the vehicle hardware.

- *Hydrogen (*H_2*) tests:* part of the growing effort toward sustainable mobility. These tests focus on hydrogen fuel cell systems, including the evaluation of tank pressures, hydrogen consumption, cell efficiency, and overall system integration. Safety protocols are particularly strict in this category due to the properties of hydrogen as a fuel, and tests are often conducted in specialized environments equipped with dedicated monitoring systems.

- *Engine Bench tests:* involves the analysis of engines mounted on dedicated test rigs. This category includes powertrain calibration, endurance tests, emissions analysis, and performance benchmarking. Unlike chassis tests, engine bench setups isolate the engine from the vehicle, allowing for more controlled and focused evaluations on the engine's behavior under various conditions.

Each of these test environments has its own scheduling logic, data recording requirements and constraints; all these aspects have been taken into account during the design of the application to ensure flexibility, usability, and relevance across different operational contexts.

In order to gather accurate and representative insights from these stakeholders, a user-centered approach was adopted. Rather than relying only on formal questionnaires or top-down specifications, the development process used informal interviews, direct observation sessions and iterative feedback loops [28]. Informal interviews, conducted with people from each role involved, allowed for open dialogue regarding the limitations of the existing tools (a combination of SharePoint lists and spreadsheets) and the several inefficiencies in the current workflow. Observation sessions were equally valuable. By closely observing the daily activities of all key user roles (engineers responsible for test request creation, group leaders in charge of reviewing and planning and technicians executing the tests) it became clear that each role had distinct needs and usage patterns. Engineers required efficient input mechanisms and structured templates to define detailed test parameters. Group leaders needed of quick view of the tests' details and a specific planner tool, while technicians relied on quick access to task lists and intuitive data entry interfaces for results. These differences highlighted the importance of role-specific functionalities, as the application had to adapt to various contexts of use and support each group's operational responsibilities effectively.

Once the functional requirements were gathered, a design and prototyping phase began using Figma [26]. The initial prototypes were created based on the input collected and were then presented on a weekly basis to the group leaders. These review sessions served as an ongoing feedback loop, where stakeholders could suggest changes, identify missing features or refine existing components. This iterative approach ensured that the design remained aligned with actual user needs and could evolve dynamically as new requirements emerged.

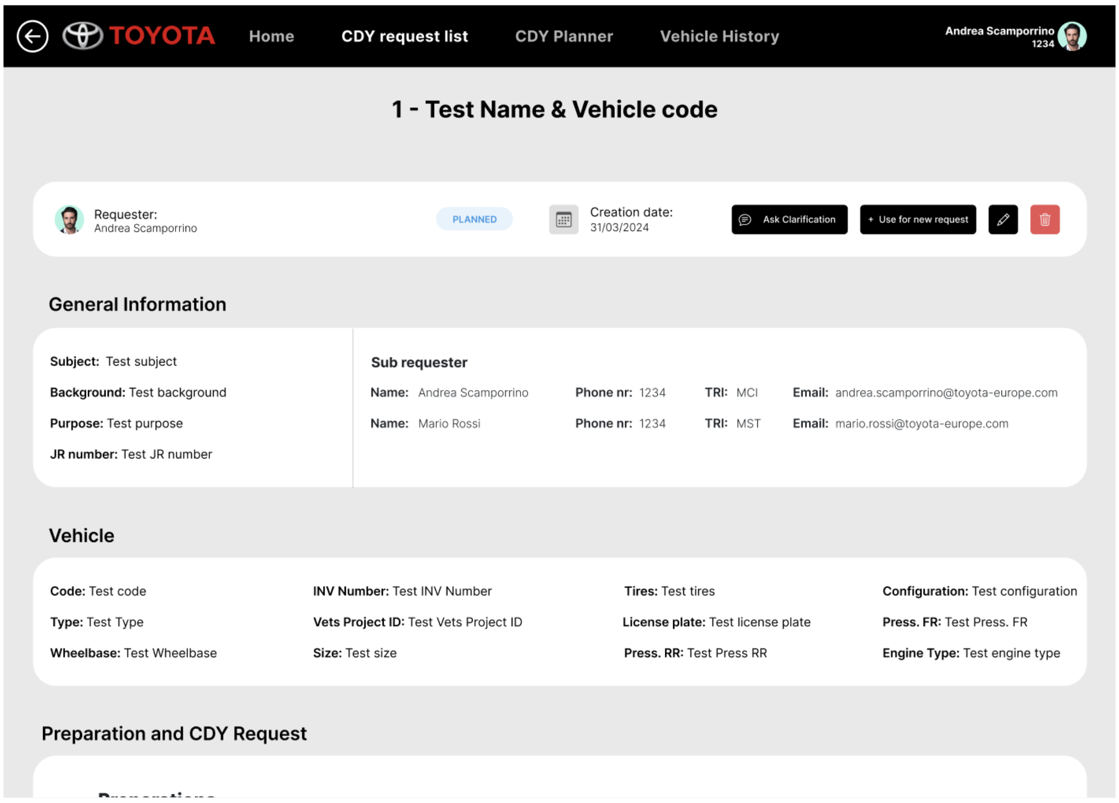
Beyond functional aspects, the analysis also considered non-functional requirements. Given the corporate context, compliance with security policies, such as Microsoft 365 identity management [29] and role based access control was non-negotiable. Performance considerations were equally important, especially due to the reliance on SharePoint as a backend system, which can become slow when handling large datasets [30][31]. For this reason, the application was designed to include filtering mechanisms and background data operations to preserve responsiveness even with growing data volumes.

3.2 UI/UX design approach

Following the thorough analysis of user requirements, the next step was to translate those insights into a coherent and effective user interface (UI) design. The main challenge at this stage was not simply to make the application "usable", but to ensure that every interaction was aligned with the users' real-world workflows, constraints, and expectations. With multiple user roles interacting with the app for different purposes and at different frequencies, the user interface had to provide clarity without rigidity, and flexibility without overcomplication. The design process did not begin with aesthetics or component selection, but with a careful mapping of the user journeys across the core functionalities. These journeys were reconstructed based on the practical scenarios observed during the requirement analysis phase. For instance, the test request creation flow, primarily used by engineers, was structured to reflect the actual steps taken in real operations: selecting a test category, providing detailed parameters and uploading related files and important information. By maintaining this logical and operational continuity, the learning curve for users was significantly reduced.

A key decision was to adopt a progressive disclosure strategy across the interface [32]. Rather than presenting all options and data fields at once, the UI reveals only the most relevant information at each stage of the workflow. For example, when selecting a test category (e.g., engine bench or hydrogen), the interface dynamically loads the corresponding input fields and hides irrelevant ones. This kept the interface clean and prevented cognitive overload, especially for new or infrequent users.

The design also relied on clear visual hierarchies to guide attention. Priority was given to elements that required user action or review. For example, the status of a test request were visually highlighted with color indicators, while optional or less important data were deemphasized through font weight and positioning. As seen in Figure 3.1, the planned state is accentuated and a clearly delineated hierarchy of elements is evident. Furthermore, the use of standardized components, such as modern dropdowns, date pickers, and toggle switches provided by PowerApps, ensured a consistent interaction model across the entire system.



1 - Test Name & Vehicle code

Requester: Andrea Scamporrino

Creation date: 31/03/2024

General Information

Subject: Test subject
Background: Test background
Purpose: Test purpose
JR number: Test JR number

Sub requester

Name	Phone nr	TRI	Email
Andrea Scamporrino	1234	MCI	andrea.scamporrino@toyota-europe.com
Mario Rossi	1234	MST	mario.rossi@toyota-europe.com

Vehicle

Code: Test code	**INV Number:** Test INV Number	**Tires:** Test tires	**Configuration:** Test configuration
Type: Test Type	**Vets Project ID:** Test Vets Project ID	**License plate:** Test license plate	**Press. FR:** Test Press. FR
Wheelbase: Test Wheelbase	**Size:** Test size	**Press. RR:** Test Press RR	**Engine Type:** Test engine type

Preparation and CDY Request

Figure 3.1: Test request details

Another essential principle was error prevention and feedback [33]. Input fields were validated both in real-time and before submission, with inline alerts guiding users to correct mistakes without interrupting the workflow. For instance, if a required field was missing or inconsistent such in the figure 3.2, the interface immediately warned the user without relying exclusively on server-side checks. This not only reduced frustration but also improved data quality across the system.

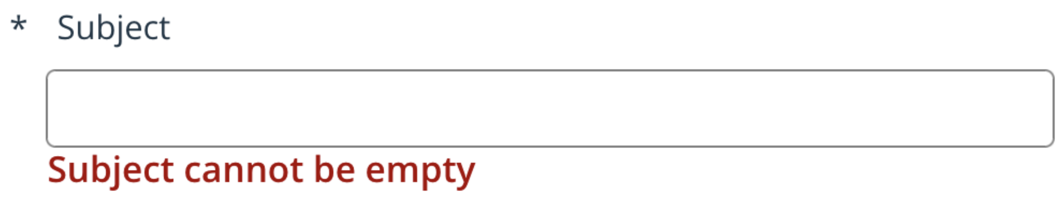


Figure 3.2: Error prevention

Iterative prototyping played a major role in refining the UI. Early versions of each screen were prototyped in Figma and shared with stakeholders during weekly review sessions. This quick prototyping cycle allowed for immediate validation of interface choices and quick adaptation based on real user feedback. Such iterative refinements ensured that the UI was driven by usability rather than assumptions.

Furthermore, the visual design was kept deliberately neutral and professional, in line with internal corporate tools and the expectations of a production-oriented environment. Graphic elements such as icons, button states, and contextual messages were used strategically to reinforce user confidence and support task completion.

In conclusion, the UI/UX approach adopted in this project was neither a superficial afterthought nor a generic overlay. It was a carefully planned response to specific user needs and working conditions, built through direct observation, continuous feedback, and iterative refinement.

3.3 Navigation structure and interaction flow

A fundamental aspect of designing a robust and user-friendly enterprise application lies in crafting a clear and intuitive navigation structure. The test request management application developed in this project adopts a layout and flow that reflect real-world processes and user roles within the organization. This section explores the main navigation paradigm, the logical flow between screens, and the key user interactions that the application supports.

3.3.1 Main screens

The screens presented in the following section are often specific to a single test request category, but the same navigation structure applies consistently across all categories.

Homescreen

When the application is launched, the first screen presented to the user is the *Homescreen* (Figure 3.3). From this screen, the user can select the category they wish to interact with (e.g., to create, schedule, or manage a request).

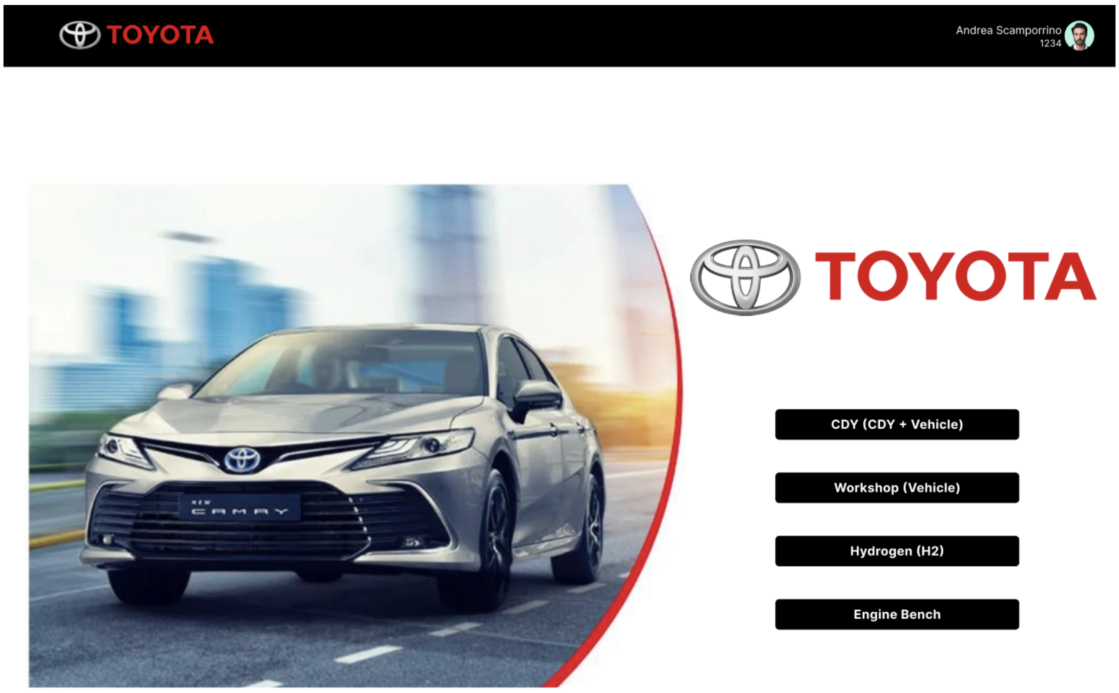


Figure 3.3: Figma design of the *Homescreen*

Once a category has been selected, all subsequent screens include a top navigation bar. This bar allows users to access the different sections within the selected category or return to the previous screen via a back arrow icon.

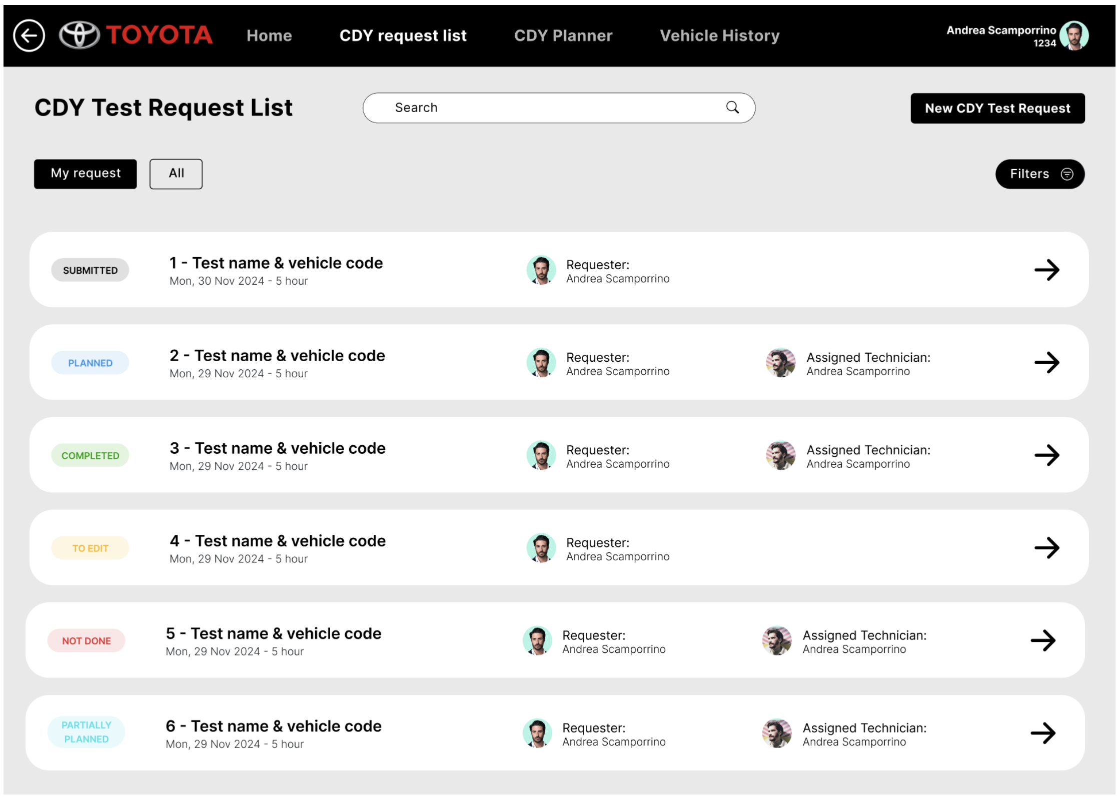
Request list

The *Request list* screen is the first screen shown after a category is selected. It displays all test requests submitted within that category, sorted by creation date. Users can filter the displayed requests using multiple methods:

- A search bar, which allows filtering by title, ID and requester.
- Quick filters that enable the user to view either only their own requests or all requests within the category.
- A button that lets users filter requests by status.

These filters can be applied simultaneously and the resulting list will include only the requests that match all selected criteria.

From this screen, users can also initiate the creation of a new test request by clicking the designated button *"New CDY Test Request"*.



SUBMITTED 1 - Test name & vehicle code
Mon, 30 Nov 2024 - 5 hour
Requester:
Andrea Scamporrino

PLANNED 2 - Test name & vehicle code
Mon, 29 Nov 2024 - 5 hour
Requester:
Andrea Scamporrino
Assigned Technician:
Andrea Scamporrino

COMPLETED 3 - Test name & vehicle code
Mon, 29 Nov 2024 - 5 hour
Requester:
Andrea Scamporrino
Assigned Technician:
Andrea Scamporrino

TO EDIT 4 - Test name & vehicle code
Mon, 29 Nov 2024 - 5 hour
Requester:
Andrea Scamporrino

NOT DONE 5 - Test name & vehicle code
Mon, 29 Nov 2024 - 5 hour
Requester:
Andrea Scamporrino
Assigned Technician:
Andrea Scamporrino

PARTIALLY PLANNED 6 - Test name & vehicle code
Mon, 29 Nov 2024 - 5 hour
Requester:
Andrea Scamporrino
Assigned Technician:
Andrea Scamporrino

Figure 3.4: Figma design of the page *Request List*

Test request details

When a request is selected from the request list, the application navigates to the *Test Request Details* screen (Figure 3.1), which presents all relevant information grouped into cards, each introduced by a descriptive title. The main types of information displayed include:

- Requester information;
- Current status of the request;
- Creation date;
- General details, such as the purpose of the request;
- Vehicle-related information (if the request involves a vehicle);
- Component-specific information;
- Scheduling and task details for technicians;
- Attached files and additional relevant notes.

Depending on the user's role within the organization, a variety of contextual actions are available from the request details screen. These interactions are designed to streamline the workflow and ensure that each stakeholder can perform the necessary operations without navigating away from the current context.

One of the key functionalities provided is the ability to request additional clarification. This feature is particularly useful when the information supplied in the original submission is incomplete, ambiguous, or inconsistent with standard operational procedures. By initiating a clarification request, the user can send a notification directly to the requester, prompting them to review and update the submission accordingly.

To reduce redundancy and improve efficiency, the interface also allows users to replicate an existing request and use it as a template for creating a new one. This is especially beneficial in scenarios where multiple test requests share common parameters, thus minimizing manual data entry and potential errors.

For collaboration and reference purposes, users can copy a direct link to the specific request. This link can be shared via Microsoft Teams or Outlook, ensuring that all relevant stakeholders can access the request details quickly and without confusion.

Users with appropriate permissions (i.e. the original requester or group leaders), have the ability to edit the request after its creation. This is essential in dynamic environments where plans may evolve based on vehicle availability, test prioritization or resource allocation. In cases where a request is no longer relevant or was submitted in error, authorized users can delete the request entirely from the system.

For technicians, once a test activity has been completed, the system provides the option to mark the request as completed and upload related information and files.

Lastly, in some categories, for users in supervisory roles, such as group leaders, the interface includes the ability to formally approve a request. Approval means that the request meets all necessary criteria and can proceed to the planning or execution phase. This approval mechanism is integral to the governance model implemented within the application and ensures that only validated requests are moved forward in the workflow. These role-based interactions contribute to a secure, efficient, and user-friendly experience, supporting both operational needs and compliance requirements across the various departments involved in the testing process.

Planner

The application provides planners with two distinct calendar interfaces, each optimized for the operational characteristics and scheduling requirements of different test categories. These calendar views not only enhance visibility over planned and pending activities but also streamline task allocation according to the unique constraints of each testing environment.
The first calendar view, illustrated in Figure 3.5, adopts a weekly layout and is specifically designed for the following categories: *Workshop*, *Engine Bench*, and *Hydrogen*. This layout aligns with the typical duration of tests in these categories, which often span several days. In this view, each column represents a day of the week, while each row corresponds to a technician. This structure allows planners to assess technician availability across a multi-day horizon and efficiently allocate long-duration tasks. On the right-hand side of the screen, a gallery displays unscheduled activities, making it easy for planners to schedule them into the calendar. Additionally, this section includes a button for adding non-test-related events, such as planned leave, training sessions, or other administrative duties. This integration ensures that all aspects of team scheduling, technical and organizational, are managed within a single interface.

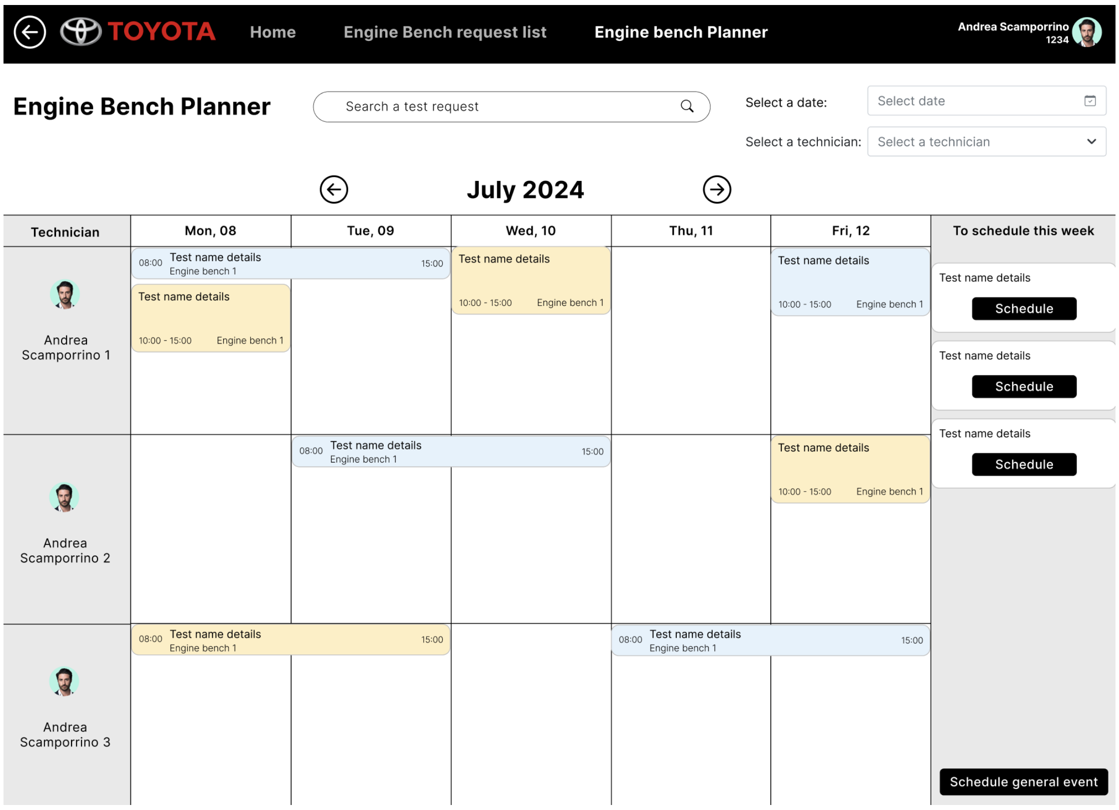


Figure 3.5: Figma design of the *Engine Bench calendar* (Weekly view)

The second calendar view, shown in Figure 3.6, offers a daily layout tailored to the specific needs of the *Chassis Dynamometer (CDY)* category. Unlike the categories using the weekly view, CDY tests are typically shorter in duration, often lasting only a few hours. As such, a day-based view is more suitable, allowing for precise scheduling on an hourly basis. In this configuration, each column corresponds to a test cell, while each row represents half hour of the working day. This setup supports high-resolution planning, enabling the team to maximize the utilization of test infrastructure while avoiding scheduling conflicts.

Both calendar views feature a consistent set of navigation controls: a date picker and arrow buttons placed adjacent to the current date, enabling quick movement between different planning periods. These shared interaction patterns contribute to a cohesive user experience, regardless of the test category being managed.

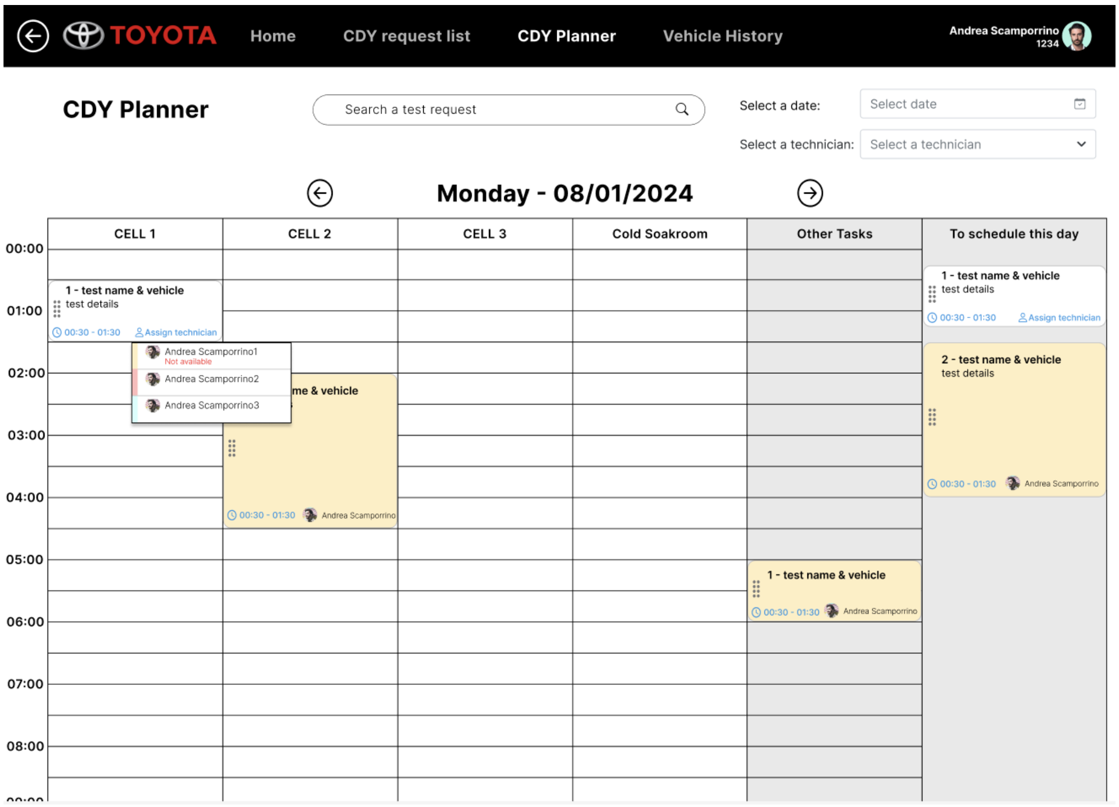


Figure 3.6: Figma design of the *CDY calendar* (Daily view)

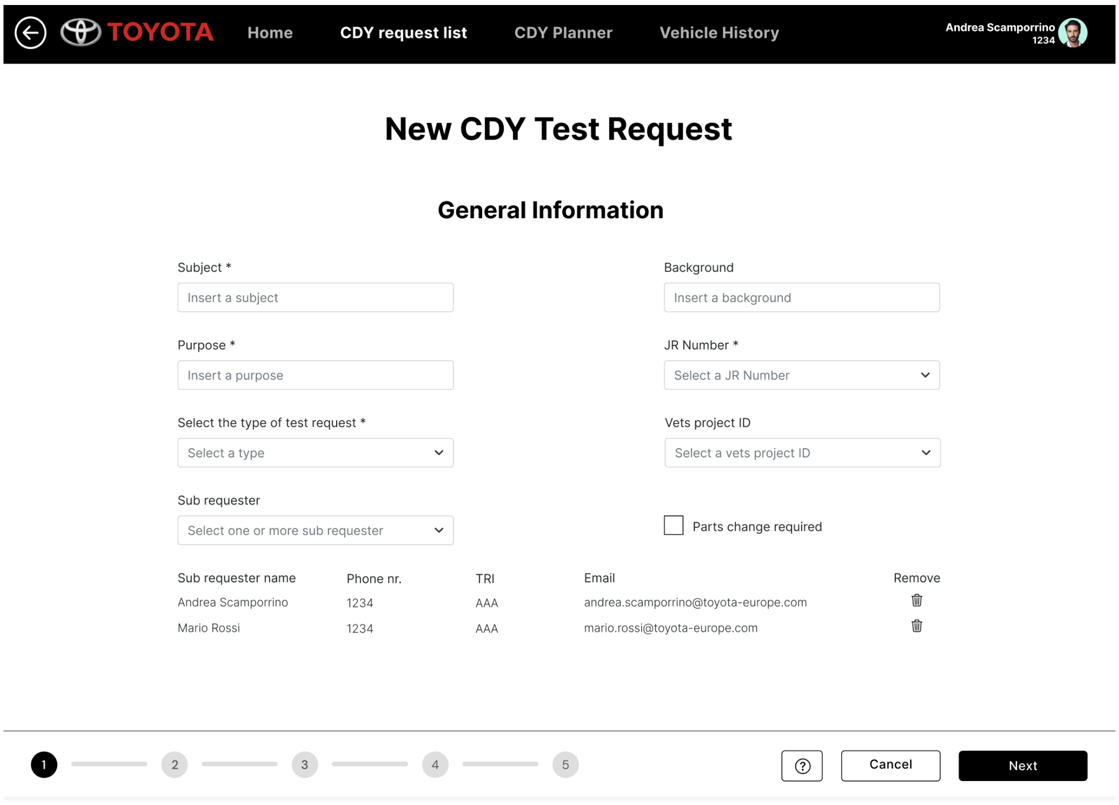
New request

The *New request* section is designed to facilitate the structured creation of a new test request through a user-friendly multi-step form. Rather than presenting a single, overwhelming page, the form is divided into several logically ordered steps, each focused on a specific set of information. These include general details about the test objective, vehicle and component information when needed, scheduling preferences and other categories specific sections. Additionally, users can upload relevant documentation or reference files to support the request. This progressive disclosure approach simplifies data entry and improves the overall user experience, especially in complex test scenarios.

To maintain data integrity and prevent incomplete submissions, mandatory fields are clearly marked with an asterisk (*). Users are not allowed to proceed to the next step until all required inputs in the current section have been provided (Figure 3.2). Moreover, logical validations are applied throughout the form. For example, date fields are checked to ensure that the end date does not precede the start date.

Before the final submission, users are presented with a comprehensive review page that summarizes all previously entered data. This page allows users to verify the accuracy and completeness of their request, make any necessary corrections, and confirm their submission with confidence.

This guided approach ensures consistency across submitted requests, reduces the probability of human error and aligns with best practices in enterprise-level form design.



New CDY Test Request

General Information

Subject *
Insert a subject

Purpose *
Insert a purpose

Select the type of test request *
Select a type

Sub requester
Select one or more sub requester

Background
Insert a background

JR Number *
Select a JR Number

Vets project ID
Select a vets project ID

☐ Parts change required

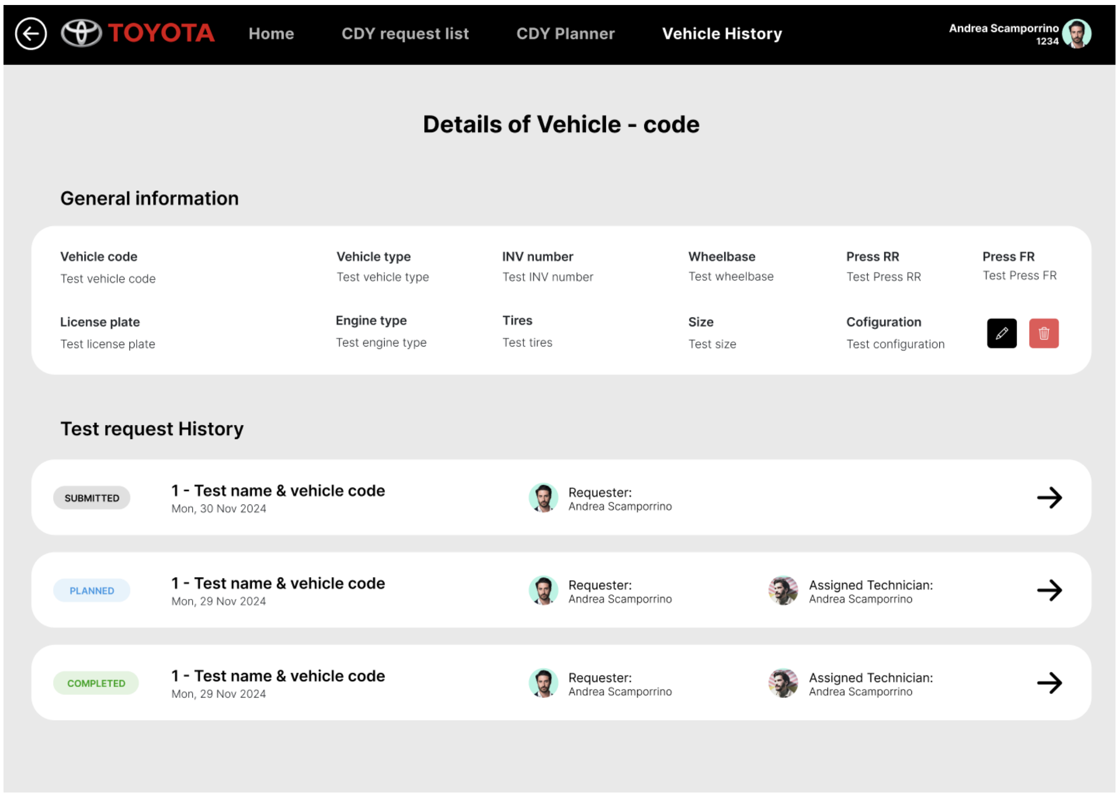
Sub requester name	Phone nr.	TRI	Email	Remove
Andrea Scamporrino	1234	AAA	andrea.scamporrino@toyota-europe.com	🗑
Mario Rossi	1234	AAA	mario.rossi@toyota-europe.com	🗑

Figure 3.7: Figma design of the multi-step form to create a new request

Vehicle history

The *Vehicle history* screen provides an overview of all vehicles that have been added to the application. From this page, users can add new vehicles to the system and apply filters to efficiently search through the existing entries.

By selecting a specific vehicle, users can access a detailed view containing relevant information such as vehicle specifications and identification data. In addition, the screen displays a list of all test requests that have been associated with that particular vehicle. This feature plays a critical role in supporting traceability and informed decision-making. By centralizing the history of each vehicle, the application enables users to track past testing activities, identify recurring issues, and better assess the current condition or development stage of a vehicle. This historical insight is particularly valuable for both technical analysis and strategic planning purposes.



Details of Vehicle - code

General information

Vehicle code Test vehicle code	Vehicle type Test vehicle type	INV number Test INV number	Wheelbase Test wheelbase	Press RR Test Press RR	Press FR Test Press FR
License plate Test license plate	Engine type Test engine type	Tires Test tires	Size Test size	Cofiguration Test configuration	

Test request History

SUBMITTED	1 - Test name & vehicle code Mon, 30 Nov 2024	Requester: Andrea Scamporrino	
PLANNED	1 - Test name & vehicle code Mon, 29 Nov 2024	Requester: Andrea Scamporrino	Assigned Technician: Andrea Scamporrino
COMPLETED	1 - Test name & vehicle code Mon, 29 Nov 2024	Requester: Andrea Scamporrino	Assigned Technician: Andrea Scamporrino

Figure 3.8: Figma design of the vehicle details in the vehicle history section

3.3.2 Request lifecycle: from creation to assignment

One of the most critical flows supported by the application is the creation and processing of a new test request. This process typically involves multiple stakeholders (requester, group leaders and technicians) and progresses through several phases, as illustrated in Figure 3.9.

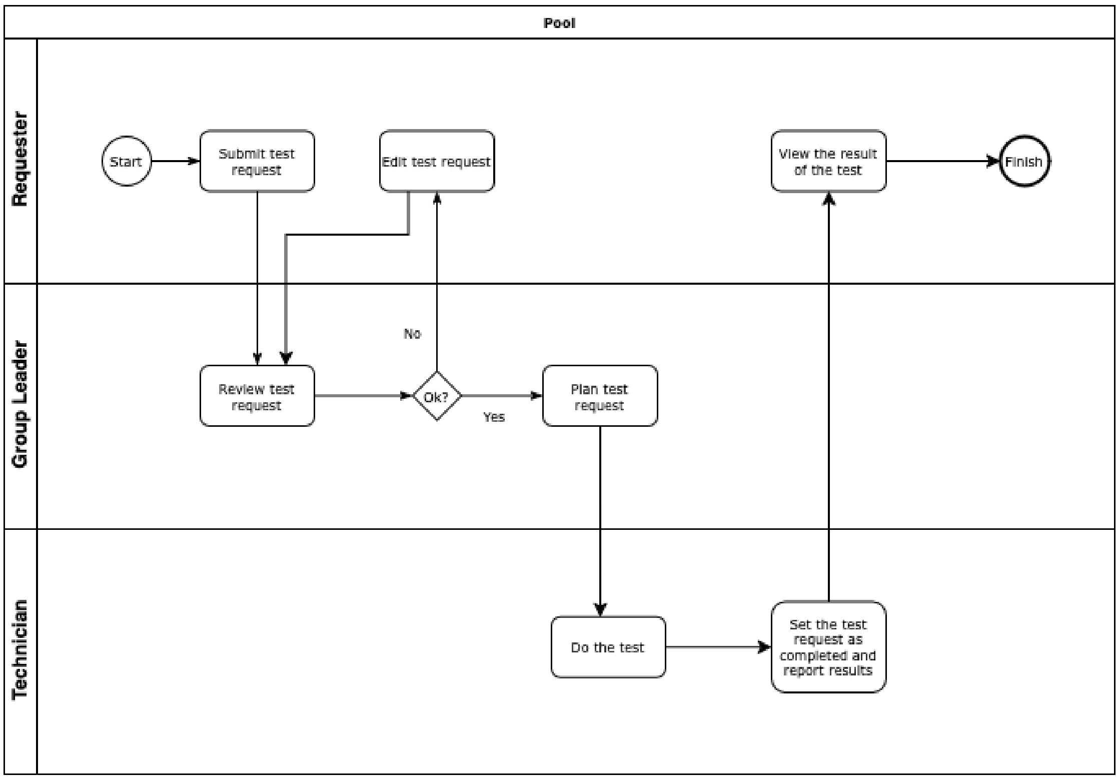


Figure 3.9: Process flowchart illustrating the lifecycle of a test request

1. **Creation phase:** The process begins with the requester, who fills out a structured form accessible from the Request List. This form captures key details such as the subject, objective, technical requirements, and vehicle specifications. *(Refer to the "Submit test request" block in Figure 3.9).*

2. **Review and clarification phase:** Upon submission, the request enters a *Submitted* state and is passed to the group leader. The group leader reviews the information and, if needed, can request clarifications by sending it back to the requester for modification. *(This is represented by the decision point "Ok?" and the associated "Edit test request" loop).*

3. **Planning and assignment phase:** Once the request is complete and approved, the group leader proceeds with planning. Using the calendar interface (Planner), the test is scheduled and assigned to an available technician. Real-time conflict checking ensures that overlapping bookings are avoided. *(See the "Plan test request" block).*

4. **Execution and report:** On the scheduled date, the assigned technician executes the test. After completion, the technician marks the request as *Completed* and uploads the results to the system. *(Illustrated in the steps "Do the test" and "Set the test request as completed and report results").*

5. **Result consultation:** Finally, the requester can access and review the results. *(Refer to "View the result of the test" leading to the "Finish" node).*

This interaction flow mirrors the established business procedures within Toyota, ensuring that the digital tool supports rather than disrupts existing operational practices.

3.4 Application architecture overview

This section provides a comprehensive overview of the application's architecture, including the technological components involved, their interactions, and the data flow between the layers of the system. The architecture is modular and composed of three primary layers:

- The front-end, implemented in PowerApps;
- The back-end, built upon SharePoint Online lists;
- The automation and integration layer, managed through Power Automate.

This layered approach facilitates maintainability, scalability, and adaptability, allowing the application to respond to evolving business requirements while remaining robust and user-centric.

3.4.1 Front-End: PowerApps

The user interface of the application is entirely implemented using Microsoft PowerApps [9]. This platform was chosen for its tight integration with Microsoft 365 services, ease of deployment across the organization, and capability to rapidly develop responsive and secure applications without writing traditional code.

The app utilizes a canvas-based design, where all screens are explicitly configured using controls such as galleries, forms, dropdowns, and modern input components.

PowerApps also enables role-based visibility, allowing different users to view or edit content depending on their permission level. For example, only planners can assign technicians to a request, while requesters can only delete their own submissions. These permission rules are enforced at the application level using logic conditions and contextual user information retrieved via the `User()` function and security-trimmed SharePoint connectors. The PowerApps studio allows direct binding between UI elements and SharePoint data sources, which simplifies data handling and ensures real-time synchronization.

3.4.2 Back-End: SharePoint lists as data layer

The application's data is persisted and managed using SharePoint Online lists [34], which act as the primary data storage layer. This choice was motivated by the seamless compatibility between SharePoint and Power Platform, as well as by the native support for permission management.

Each test category includes specific tables necessary to store data related to various test requests. The main tables under each category are:

- *Test Request*: this table contains key details of test requests, such as ID, Subject, Purpose, Background, Requester, Status, Creation Date, and other relevant information.

- *Roles*: identifies individuals assigned specific roles within each test category, typically including technicians and group leaders.

- *Activity/Tasks/Preparation/Request*: while named differently across categories, these tables all store the individual tasks that compose a test request. Information can be traced to the associated test request through the RequestID field, which references the test request's unique ID.

- *Edit Description*: stores modification records, including the date and description of any edits made to a test request, either on the day of or after the scheduled test, for the technician's reference.

Additionally, two tables are used across all test categories:

- *Vehicle*: contains vehicle records and their associated information.

- *Planner*: stores calendar event data across all test categories.

These lists are not only data repositories but also enforce business constraints through column validation, mandatory fields, and lookups. For example, dropdown fields such as technician is populated through linked lists to avoid data entry errors and ensure referential integrity.

3.4.3 Automation and integration layer: Power Automate

To extend the application's capabilities beyond the boundaries of PowerApps and SharePoint, Power Automate is used to orchestrate workflows, integrate with external services, and perform actions in response to user or system events [19]. These flows serve as the middleware of the application, enabling automation of repetitive tasks and real-time communication across Microsoft 365 services.

The main Power Automate flows integrated into the application are:

- *Outlook calendar synchronization*: multiple flows responsible for creating, updating and deleting calendar events in users' Outlook calendars when a request is scheduled or modified. This ensures that technicians and planners receive up-to-date calendar entries, minimizing the risk of scheduling conflicts and eliminating the need for manual double entry. Additionally, the event includes relevant metadata embedded in the calendar description.

- *Automated Teams chat initiation*: to facilitate real-time communication between stakeholders, another Power Automate flow creates a Teams chat between the requester and the assigned technician. This is implemented using the Microsoft Teams connector, specifically the *Create Chat* and *Send Message* actions. This flow enhances operational efficiency by reducing the friction involved in coordinating tasks. The chat message contains a direct link to the request page in PowerApps, making it easy for users to access all relevant details from within the Teams interface.

- *Archive old items*: the application includes a suite of four Power Automate flows dedicated to long-term data retention and system performance. These flows are triggered monthly and are responsible for identifying and archiving test requests that are more than six months old. Once the criteria are met, the flows move the relevant items from the main operational lists (e.g., CDY Test Requests and Planner) into dedicated archive lists. This process helps to reduce the volume of active data, improving performance within PowerApps. The archive process preserves the structure and metadata of each item enabling traceability and long-term analysis while avoiding any data loss.

3.4.4 Data flow and components interaction

The data flow in the application is orchestrated through direct and asynchronous communication between components. The front-end communicates with the SharePoint lists using the native connectors in PowerApps, enabling two-way binding of forms and galleries. This direct interaction allows users to immediately read or write data to the underlying lists based on user actions (e.g., submitting a form or selecting a record).

Power Automate acts asynchronously, listening to changes in SharePoint lists or triggered by explicit button presses in PowerApps. The flows then invoke actions in Outlook or Teams, or update other SharePoint items accordingly.

The overall architecture can be abstracted in the following flow:

- User submits or modifies data in PowerApps;
- PowerApps writes data to SharePoint List;
- Power Automate flow is triggered;
- Flow interacts with Outlook, Teams and SharePoint List.

This decoupled and event-driven architecture improves the reliability of the application while ensuring scalability. New flows or integrations can be added without modifying the core logic of the PowerApps front-end, adhering to a microservice-inspired design principle within the constraints of the Power Platform.

In summary, the application architecture efficiently exploits Microsoft's cloud ecosystem to deliver a secure, maintainable, and business-aligned solution. The use of SharePoint as a data layer ensures enterprise-grade governance and control, while PowerApps and Power Automate combine to deliver a flexible and responsive user experience integrated with existing Microsoft 365 tools such as Outlook and Teams.

3.4.5 Alternative architectural considerations

During the architecture definition phase, several alternative technologies and design patterns were evaluated to ensure that the chosen solution would balance functionality, scalability, maintainability, and integration within Toyota's existing digital ecosystem.

Microsoft Dataverse

One option considered was the use of Microsoft Dataverse as the primary data storage layer, instead of SharePoint Online. Dataverse offers relational data modeling, support for business rules, and improved scalability for larger datasets [35]. However, this option was excluded due to additional licensing requirements and limited availability within the development environment. SharePoint was already widely adopted within the organization, and its native compatibility with PowerApps and Power Automate made it the most pragmatic choice given the available resources and the expected scale of usage.

Azure SQL database

Another option explored was the use of Azure SQL Database as a backend. This approach would have provided full control over the data schema, complex querying capabilities, and high performance with large volumes of data [36]. However, it would have also introduced significant complexity in terms of security management, integration efforts with Power Platform, and maintenance overhead. Given that the application needed to be deployed and managed by non-developer teams in a no-code/low-code environment, SQL-based solutions were deemed unnecessarily complex for the scope of the project.

Other front-end framework

Finally, the possibility of building a custom front-end using frameworks such as React or Angular was considered. While this approach would allow for full UI flexibility and performance optimization, it would also require a complete custom backend, user authentication integration, and deployment infrastructure, leading to a higher total cost of ownership. Since PowerApps was already supported and aligned with internal IT guidelines, it was ultimately the most effective and sustainable choice.

In conclusion, the architectural choices made were the result of careful evaluation of alternatives, taking into account not only technical merit but also organizational constraints, user needs, and long-term maintainability.

Chapter 4

Application development

This chapter describes the practical implementation of the vehicle test request management system developed during the internship. It focuses on the key phases and components that contributed to the realization of the application, including the definition of the data model on SharePoint, the implementation of core functionalities in Power Apps and the integration of automated workflows through Power Automate. Unlike the previous chapters, which focused on design decisions and architectural planning, the emphasis here is on how these ideas were translated into working software. The goal is to provide a detailed and structured explanation of how each part of the system was built and integrated with other components in the Microsoft Power Platform ecosystem.

4.1 SharePoint configuration

The application's data model is built upon a set of structured SharePoint Online lists [34], which serve as the backend layer for storing and managing all records related to test requests, vehicle data, planning activities, and user roles.

4.1.1 List configuration and column design

Each SharePoint list was configured with carefully selected columns tailored to its specific function in the system. Particular attention was given to:

- Column types (e.g., choice, person, lookup, date/time and single/multiline text);
- Field validations and mandatory constraints to ensure data consistency;
- Lookup columns for linking related data (e.g., linking tasks to test requests via RequestID).

Where applicable, default values and formatting rules were also introduced. For instance, the Status column in the Test Request list is initialized to “Draft” upon creation and then changed to other states such as “Submitted”, "Planned" or “Completed” based on user actions, tracked via Power Apps and Power Automate.

In addition, fields such as *Requester*, *Technician* and *Group Leader* were implemented using the Person column type, allowing integration with the Microsoft 365 user directory. This enabled automatic resolution of user information (name, profile picture, email and department), supported personalized views and filtered content based on the current user.

4.1.2 Cross-List logic and referential integrity

Since SharePoint does not natively support relational databases, maintaining data consistency across multiple lists required careful use of lookup columns and PowerApps logic. For example, the Activity lists reference the parent request through a RequestID lookup. To minimize the risk of broken references or orphan records, deletion operations on parent items, such as test requests, were strictly managed through logic implemented in PowerApps. Depending on the user’s role and permissions, the delete button is either hidden or disabled entirely. When deletion is allowed, the application performs a controlled cascade removal: only the associated records that are no longer needed, such as individual tasks or preparation steps, are deleted automatically. Other related data, such as vehicle entries, are intentionally preserved to maintain data integrity and historical traceability.

4.2 Implementation of main functionalities

This section explains how the main features of the application were built using Power Apps. Each screen will be described in detail, focusing on its key functionalities and how users interact with it. In some parts, code snippets will also be included to show how specific actions or conditions were implemented.

4.2.1 Request list

The *Request list screen* (Figure 3.4) was implemented using a gallery control, which dynamically displays all the requests stored in the corresponding SharePoint list, based on the selected test category. The logic behind this screen focuses on offering real-time filtering, searching and sorting capabilities, ensuring both flexibility and responsiveness for the end user. To support these functionalities, the `Items` property of the gallery was configured with a composite expression that combines multiple filtering layers. The formula used is shown below:

```
Sort(
    Filter(
        Search(
            AddColumns(
                'CDY Test Request',
                IDText, Text(ID), RequesterName, Requester.DisplayName
            ),
            inputSearchBar.Text,
            Title, IDText, RequesterName
        ),
        If(
            rapidFilter = "MyRequest",
            Requester.Email = User().Email,
            true
        ) And If(
            dropdownStatusFilter.SelectedText.Value = "All Status",
            true,
            Status.Value = dropdownStatusFilter.SelectedText.Value
        )
    ),
    Created,
    SortOrder.Descending
)
```

Source Code 4.1: Items property of the gallery which displays the test requests

This expression performs several key operations:

- `AddColumns(...)`: new temporary columns (IDText, RequesterName) are added to enable text-based searching on fields that are not plain text by default (e.g., numeric ID and person type columns) because the *Search* function accept only plain-text.
- `Search(...)`: allows the user to filter the request list based on the text entered in the search bar (inputSearchBar component). The search is performed across multiple fields simultaneously (specifically the title, request ID and requester name) and returns only the requests that match the input criteria.
- `Filter(...)`: two conditional filters are applied:
  - The *"My Requests"* rapid filter shows only the requests where the current user is the requester. This filter is controlled by a context variable called `rapidFilter`, which is updated whenever the user clicks one of the quick filter buttons on the screen. The variable's value determines whether to show only the user's own requests or all requests in the selected category.
  - The status filter lets users show only the requests that match the status selected in the dropdown menu (`dropdownStatusFilter` component).
- `Sort(...)`: finally, the resulting list is sorted by creation date in descending order, ensuring that the most recent requests are shown at the top.

To avoid performance issues related to SharePoint's delegation limits (2000 item threshold) the application was designed to keep the dataset manageable [37]. Each request list is scoped to a specific test category and older records are regularly archived using dedicated Power Automate flows (explained later in the section 4.3.6). As a result, the number of items handled by the gallery remains well below critical limits, ensuring reliable performance and fast load times.

Finally, the screen includes a *"New Request"* button that allows users to start the creation of a new test request. When clicked, it executes the following sequence of actions:

```
ResetForm(GeneralInforationForm);
NewForm(GeneralInforationForm);
Navigate('CDY New Request - General Info', ScreenTransition.Cover)
```

Source Code 4.2: OnSelect property of the "New Test Request" Button

This logic resets the current form, prepares a new blank entry and navigates the user to the appropriate screen for entering general information related to the new request. This ensures a smooth and intuitive experience when submitting new entries.

4.2.2 Test request details

Once a request is selected from the gallery on the *Request List screen*, the following code is executed and the user is redirected to the corresponding detail screen.

```
Navigate(
    'CDY-WS Request Details',
    ScreenTransition.Cover,
    {
        item: AddColumns(
            DropColumns(ThisItem, IDText, RequesterName),
            Source, "CDY"
        )
    }
)
```

Source Code 4.3: Code executed when a request is selected

This formula serves three purposes:

- *DropColumns(...)* removes temporary columns (`IDText` and `RequesterName`) that were previously added for search purposes in the gallery. These columns do not exist in the original SharePoint schema and are excluded to avoid issues when referencing the item.
- *AddColumns(..., Source, "CDY")* adds a new field named Source with the value "CDY" to indicate the category of the request. This is particularly useful when the same screen structure is reused across multiple test categories, allowing for dynamic behavior based on context.
- *Navigate(...)* move the user to the *"CDY-WS Request Details"* screen using a smooth cover animation, passing the processed request record as a context variable called item.

Once received, the item variable is used throughout the screen to retrieve and display not only the request's direct attributes (such as title, status, requester and creation date), but also its related data from other SharePoint lists. These include:

- Associated vehicle information, retrieved by matching the vehicle code stored in the request with the corresponding record in the Vehicle list;
- Lists of test activities which are dynamically loaded from the related sub-tables (such as Tasks or Preparation) by performing a join via the unique `RequestID`.

This relationship is typically implemented using `Filter()` and `LookUp()` functions, as shown in the simplified example below:

```
Filter('Preparation Tasks', RequestID = item.ID)
```

Source Code 4.4: Example of data selection based on item.ID

These joins allow the screen to stay modular and scalable, ensuring that each test request, regardless of its category, displays the correct supporting information without duplicating data across lists.

In addition, dynamic formatting and visibility conditions are applied based on the item status and the role of the current user. For instance:

- The delete button is disabled if the status is "Planned" or "Completed".
- Approval or completion options appear only for group leaders or technicians, respectively.
- Specific sections (e.g., vehicle related fields) are hidden entirely if not applicable to the selected test category.

By centralizing the use of the item variables and leveraging it for cross-list data binding, the screen maintains a single source of truth for all contextual information, improving maintainability and reducing the risk of inconsistency across components.

This screen allows users to view all information associated with the selected request and, depending on their role and the current status of the request, perform a set of related actions (such as editing, copying the link of the request, requesting clarification or marking the request as completed).

Editing functionality

The *Edit* button on the *Test Request Details* screen behaves differently depending on whether the request has already been scheduled or executed. To manage this correctly, the application first checks the earliest scheduled activity associated with the request. If the request has not yet been scheduled or the planned date has not arrived yet, the request can be edited without restrictions. However, if the request is scheduled for today, the system assumes that the test might be in preparation or about to start. In this case, the user is allowed to edit the request, but only after providing a short description explaining what changes they intend to make. This step is used to keep a record of last-minute modifications, so that technicians or planners can be informed. If the request has already been executed, meaning the planned test date is in the past, the *Edit button* is disabled. This prevents users from changing data that has already been used in a real test, ensuring consistency between what was planned and what was actually performed. This behavior guarantees a balance between flexibility and control: users can make changes when necessary, but only in a way that maintains traceability and protects the integrity of the planning process.

Copy link feature

The *copy link feature* allows users to generate a direct URL pointing to the current request. This is useful for quickly sharing the request with other team members through Microsoft Teams or Outlook.

```
Copy("https://apps.powerapps.com/.../?requestId=" & item.ID &
    "&destination=" & item.Source);
UpdateContext({ctxCopiedLink: true})
```

Source Code 4.5: Copy function

The generated link includes the request ID named *requestId* and the test category (CDY, WS, H2, EB) named *destination*, both passed as a query parameter. These parameters allow the application to recognize which request to open when the link is accessed and to load the correct detail screen accordingly. The app reads these values on startup, uses them to fetch the relevant record from SharePoint and display it to the user. Additionally, the `UpdateContext(ctxCopiedLink: true)` function is used to set a local variable, which trigger a brief confirmation message to provide feedback to the user on the success of the operation.

Ask clarification feature

The *ask clarification feature* allows users to request further information about a test request when some of its details are incomplete, unclear or potentially incorrect.

When the user clicks the *Ask Clarification* button, a pop-up dialog appears on the screen. Within this pop-up, the user can type a custom message using a rich text editor and, depending on their role, may be asked to select a technician from a drop-down list.

The logic behind this feature is dynamic and role-dependent:

- If the current user is the test requester, the system requires them to select a technician from a drop-down menu. The message, once submitted, is sent to the selected technician.

- If the current user is a technician, the message is sent directly to the original requester of the test.

- If the user is neither the requester nor a technician (typically a group leader), they can select a technician from the drop-down and the system automatically creates a group chat involving the requester, the technician and the user who initiated the clarification.

This behavior is handled through conditional logic in Power Apps, which determines the correct set of recipients based on the user's identity and role. Once the message and recipients are defined, the system calls a Power Automate flow (Ask clarification) which will be described in detail in Section 4.3.4.

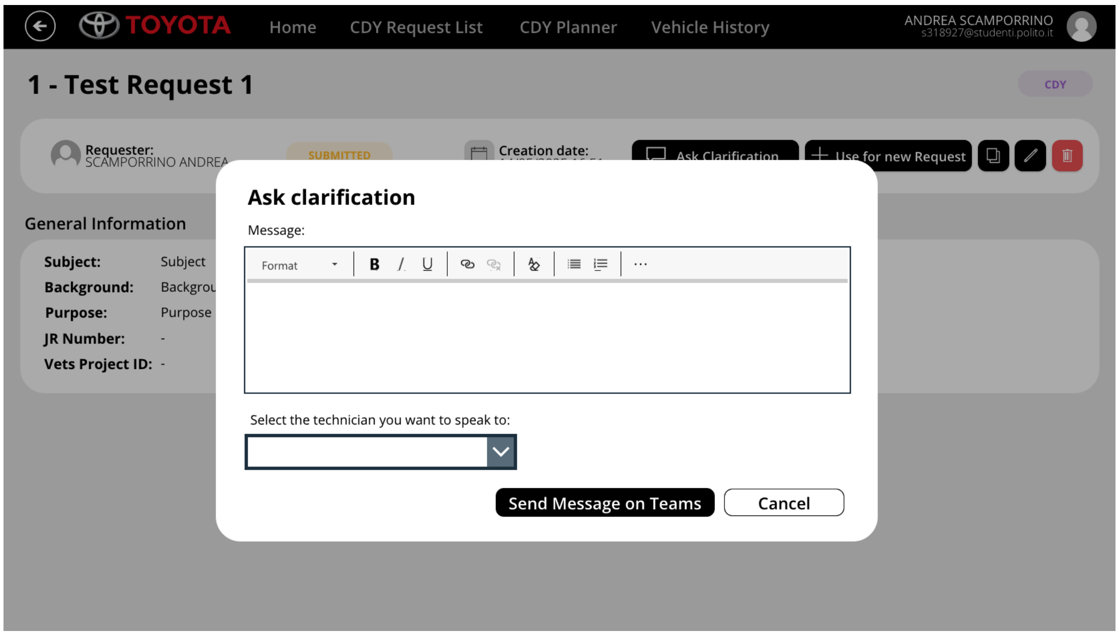


Figure 4.1: Ask clarification pop-up

4.2.3 Planner

The application includes two distinct calendar views, weekly and daily, designed to match the operational logic of different test categories. While the previous chapter focused on the visual structure and purpose of each layout, this section examines the technical implementation of the planning interface in Power Apps, including data handling, interaction logic and calendar synchronization.

Weekly view

The *weekly planner*, designed in the section 3.3.1, used for Workshop, Engine Bench and Hydrogen categories, is structured into three primary sections: the header, the right panel and the central planning area. The code presented here is related specifically to the Engine Bench category, but the same structure applies to the other categories.

The header includes both arrow buttons and a date picker to control the currently displayed week. These inputs update a global variable called `EBPlannerDate`, which is initialized with the current date using the `Today()` function. This variable acts as the reference point for computing the working days of the week, from Monday to Friday. The resulting dates are then stored in a collection named `EBPlannerWeekDay`, as shown in the code 4.6, and are recalculated every time the selected date changes.

```
Set(EBPlannerDate, Today());
ClearCollect(EBPlannerWeekDay,
    { Monday: DateAdd(EBPlannerDate, - (Weekday(EBPlannerDate,
   StartOfWeek.Monday) - 1), TimeUnit.Days),
    Tuesday: DateAdd(EBPlannerDate, - (Weekday(EBPlannerDate,
   StartOfWeek.Monday) - 2), TimeUnit.Days),
    Wednesday: DateAdd(EBPlannerDate, - (Weekday(EBPlannerDate,
   StartOfWeek.Monday) - 3), TimeUnit.Days),
    Thursday: DateAdd(EBPlannerDate, - (Weekday(EBPlannerDate,
   StartOfWeek.Monday) - 4), TimeUnit.Days),
    Friday: DateAdd(EBPlannerDate, - (Weekday(EBPlannerDate,
   StartOfWeek.Monday) - 5), TimeUnit.Days) }
);
```

Source Code 4.6: Startup function for set the days of the planner

The function works by first identifying the weekday index of the selected date using `Weekday(EBPlannerDate, StartOfWeek.Monday)`, where Monday is treated as the first day of the week (i.e., index 1). Then, for each day from Monday to Friday, a specific number of days is subtracted from `EBPlannerDate` to compute the correct date.

For example, if `EBPlannerDate` falls on a Wednesday (which would return an index of 3), the calculation for Monday would be:

```
DateAdd(EBPlannerDate, - (3 - 1), TimeUnit.Days)
= DateAdd(EBPlannerDate, -2, TimeUnit.Days)
```

Source Code 4.7: Example of weekly date calculation logic for Monday

This correctly adjusts the date back to the start of the week. The same logic is applied to the other weekdays by simply changing the offset in the formula.

The right panel displays a gallery containing unscheduled test request activities. Users can select an activity and open a form to assign dates and technicians. The planned activities are stored in a local collection called colEBPlanTestRequest, which holds relevant fields such as start/end date, assigned technician and the flags `isNew`, `isEdited` and `isDeleted`. These flags are updated through user interactions to track changes and support synchronization logic. When the user confirms the schedule:

- The test request status is updated using a `Patch` statement.
- The Outlook calendar is updated via dedicated Power Automate flows: create event(4.2), update event(4.4), or delete event(4.3).
- The SharePoint list Planner is updated to reflect the latest planning data.

This mechanism allows the app to manage both SharePoint based visibility and Outlook based coordination from a single user interface.

In the central section, each technician is represented by a row. For each weekday (Monday to Friday), a separate gallery displays the activities assigned to that technician on that day. Events are shown with visual markers and an edit icon is available next to each entry. When the user clicks the edit icon, the system distinguishes between test requests and other types of events (e.g., leave, training). If it's a test request, the corresponding form is opened with context variables such as `ctxRequest` set; if it's another type of event, the variable `ctxOtherActivityID` is used instead. This branching logic allows the system to use a shared interface while adapting behavior based on the selected event type.

Daily view

The *daily planner*, used specifically for the Chassis Dynamometer (CDY) category, is designed to handle short and frequent test sessions. Just like the weekly view, the top section (header) of the planner allows users to change the date using a date picker. The selected date is stored in a variable called `CDYPlannerDate`. Additionally, the header includes a zoom control, which lets users adjust how much space each time slot takes up on the screen. When the zoom level changes, a variable called `ctxCDYPlannerZoom` is updated. This variable controls how test sessions are displayed visually, both in terms of their height and position.

The central section of the daily view is split into two areas:

- Time Slot Column (left side): this part shows the timeline of the day, split into fixed intervals of 15 minutes.
- Cell Grid (right side): each column represents a test cell (a physical test station) and each row corresponds to a specific time slot. This layout allows very precise scheduling.

In each cell, there is a gallery that shows all the test requests scheduled for that cell on the selected day. Every request is represented by a container (a visual box). The vertical position (Y) and height of the container are calculated automatically, based on the start time of the request, the total duration of the request and the current zoom level selected by the user. For example, the formula used to determine the vertical position is:

```
DateDiff(DateAdd(DateValue(ThisItem.PlannedStartTime), 0,
    TimeUnit.Days), ThisItem.PlannedStartTime,  TimeUnit.Minutes)*
    (ctxCDYPlannerZoom / 15) - ThisItem.RowNumber
```

Source Code 4.8: Vertical position of a task in the CDY planner

This formula ensures that each request appears at the right height in the planner, exactly matching its starting time. The duration of the request determines the height of the container and the zoom level affects how compressed or expanded the planner looks.

4.2.4 New request: multi-step form

The functionality for creating a new test request has been implemented as a sequence of separate screens, each dedicated to collecting a specific group of information. This mirrors the design structure previously discussed, where the request form is divided into multiple logical steps to guide the user through the data entry process. Navigation between screens is handled through “Next” and “Back” buttons, which allow users to move forward and backward freely during the request creation process. Behind each of these steps, a set of validations and data-saving operations ensures that inputs are correct and progressively stored. Each screen includes input fields such as text boxes, drop-downs and date selectors. Some of these fields are marked as mandatory and cannot be left empty. To enforce this, a validation mechanism is triggered before moving to the next screen. If a required field is blank or invalid, an error message is shown using `UpdateContext()`, preventing the user from continuing until the issue is resolved. For example, the subject field is validated using a condition like the following:

```
If(
    IsBlank(inputSubject.Text),
    UpdateContext({ SubjectError: "Subject cannot be empty" }),
    UpdateContext({ SubjectError: Validate( ... ) })
)
```

Source Code 4.9: Example of the subject field validation

This logic displays a clear error if the subject is missing or does not meet the expected format, helping users to correct the input in real time.

At the end of the first screen, once the initial general information is filled, the app creates a new SharePoint record using `Patch()` and stores it in a global variable. This variable stores the newly created SharePoint record and is used in the following screens to:

- Retrieve the existing values;
- Update the record with additional information;
- Ensure that all parts of the form are referring to the same request.

This mechanism also supports the Edit functionality described earlier in the Test Request Details section. When the user clicks the Edit button from the request details screen, the application simply sets the global variable with the values of the selected request. This allows the form screens to be opened with all fields already filled in, enabling seamless editing without additional complexity. At each step, the data filled from the user are saved in Sharepoint, using the `Patch()` or the `SubmitForm()` function. This progressive saving approach reduces the risk of data loss and allows the form to be easily extended with additional steps if needed in the future [38]. It also supports more robust error handling and improves the responsiveness of the interface, since only a portion of the data is processed at each step. All newly created requests are marked with the status “Draft”, which allows users to save their progress and return to complete the request later. That also enables the system to recognize incomplete submissions and differentiate them from finalized ones. At the end of the multi-step form, users are presented with a summary screen where all the previously entered data are displayed in a structured format. This allows the user to review the request before confirming. A Submit button is then provided to finalize the process: when clicked, it updates the SharePoint record by changing its status from “Draft” to “Submitted”.

4.3 Power Automate flows

This section provides a detailed overview of all the Power Automate flows implemented as part of the test request management system. These flows play a key role in automating communications, synchronizing data with external services such as Outlook and Teams, and enforcing business rules across the platform. Each flow will be described individually, focusing on its purpose, input parameters, logic, and integration with other application components. The objective is to give a clear and technical understanding of how automation was used to enhance the overall functionality and efficiency of the system.

4.3.1 Create event flow

The *Create event flow* is responsible for creating calendar events in Outlook when a test request is scheduled in the application. This automation ensures that both the requester and the assigned technician receive a calendar invitation, helping to improve coordination and avoid scheduling conflicts. The flow is triggered directly from Power Apps, whenever a new planning is confirmed through the calendar interface. It receives as input several details about the test, such as the title, the scheduled start and end date-time, the calendar name, the request ID and the emails of both the requester and the technician.

Once triggered, the flow performs the following steps:

1. Formats the start and end times by combining the selected date and time into a full timestamp in ISO format, required by Outlook.

2. Retrieves all available Outlook calendars using the “Get calendars” action.

3. Creates the event in the correct calendar by checking if the calendar name matches the one received from Power Apps.
   If so, the flow creates a new event that includes:

   - The subject of the test (which includes the request ID and title);
   - The scheduled date and time;
   - A short message body, including a direct link to the request in Power Apps;
   - The technician and requester added as required attendees.

4. Updates the related SharePoint item to save the ID of the event just created in Outlook. This is important to enable future modifications or deletions of the same event.

To manage different test categories (such as CDY, H2, WS, EB), the flow uses a *Switch block* that updates the correct SharePoint list depending on the category passed in the input. This allows the same flow to work across multiple planners while maintaining data consistency.

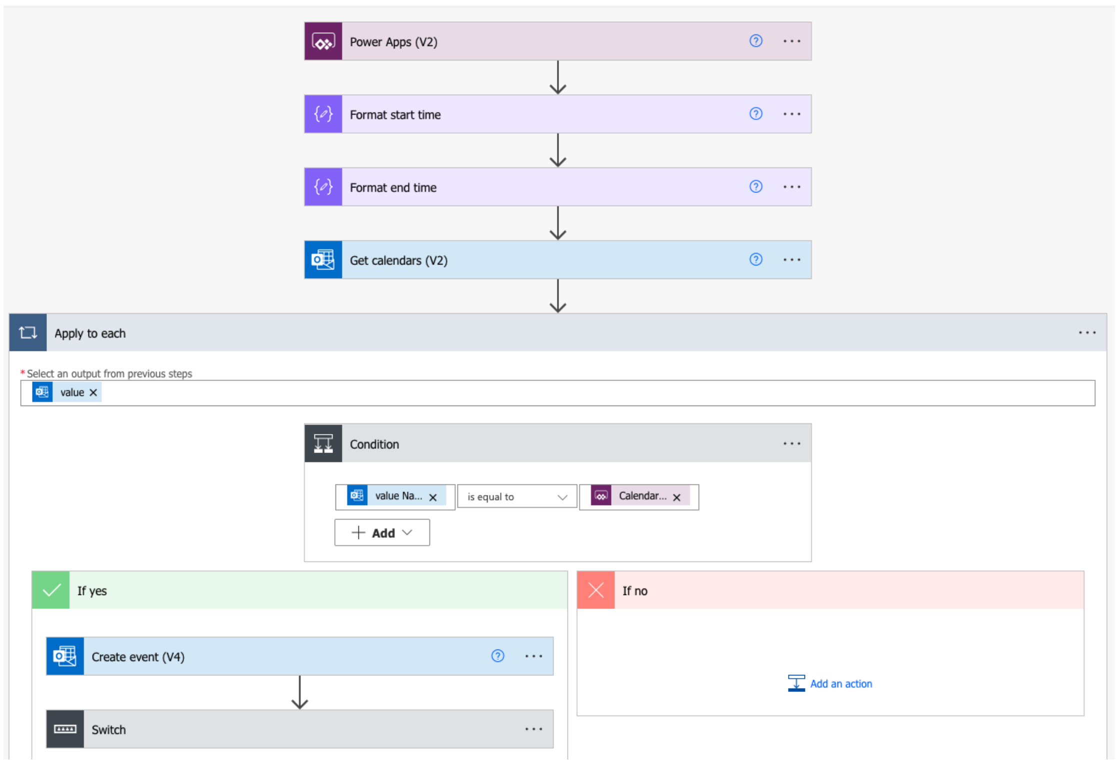


Figure 4.2: Create event flow

4.3.2 Delete event flow

The *Delete event flow* is designed to remove a previously scheduled Outlook calendar event when a test request is canceled or rescheduled in the application. This ensures that outdated or invalid appointments do not remain visible in users' calendars, avoiding confusion or duplication. Like the *Create event flow*, this automation is also triggered from Power Apps and receives three key inputs: the name of the calendar from which the event should be deleted, the ID of the Outlook event to be removed and the SharePoint item ID used to update the corresponding request record. Once started, the flow performs the following steps:

1. Retrieves all available calendars using the "Get calendars" action.
2. Loops through each calendar, checking whether its name matches the one received from Power Apps.
3. Deletes the event from the matching calendar using the Outlook event ID provided.
4. After deletion, the flow uses a Switch block to identify the category of the planner and updates the corresponding record in the appropriate SharePoint

list. In particular, the field that stores the Outlook EventID is set to empty, indicating that the event has been removed.

This logic ensures that the system remains in sync: when a scheduled request is deleted both Outlook and SharePoint are updated to reflect the change.

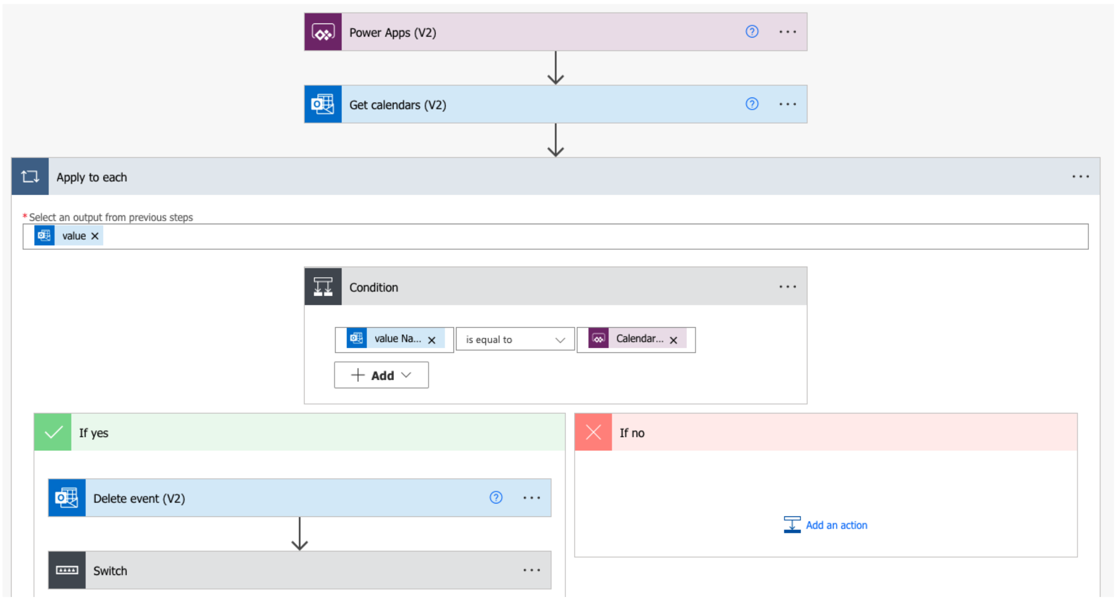


Figure 4.3: Delete event flow

4.3.3 Update event flow

The *Update event flow* is used when a test request has already been scheduled, but some of its details, such as the date, the time or the assigned participants, need to be changed. Instead of deleting the original calendar event and creating a new one from scratch, this flow updates the existing Outlook event to reflect the latest information provided by the user in Power Apps. The flow is triggered directly from the application when a user edits an already planned test request. It receives the following information from Power Apps:

- the calendar name where the event is located;
- the event ID of the Outlook appointment to be modified;
- the title and the ID of the request;
- the new start and end date-time;
- the email addresses of the technician and the requester;

- a deep link to the request within Power Apps.

Once triggered, the flow follows these steps:

1. formats the start and end times by combining the selected date and time into a full timestamp in ISO format, required by Outlook;
2. retrieves all available Outlook calendars using the "Get calendars" action;
3. update the event in the correct calendar by checking if the calendar name matches the one received from Power Apps.

This flow ensures that scheduling changes are immediately reflected in users' calendars, improving coordination while avoiding duplicate or outdated events. It is especially useful in dynamic testing environments where dates or resources may change frequently.

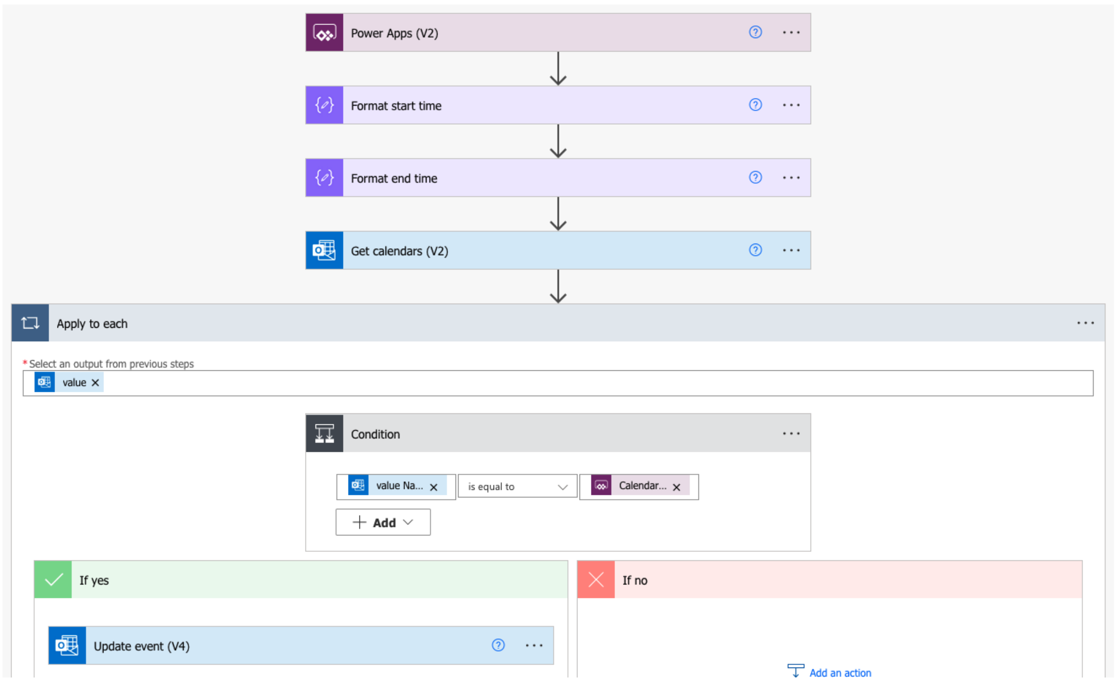


Figure 4.4: Update event flow

4.3.4 Ask clarification flow

The *Ask clarification flow* is used when a user wants to request additional information about a test request directly from within the application. Instead of relying on external emails or manual communication, this flow creates an automatic Microsoft Teams chat between the relevant users and posts a customized message containing a link to the request. When triggered from Power Apps, the flow receives the following inputs:

- the request title and the ID;
- the message written by the user;
- the recipients' email addresses (calculated in Power Apps);
- the category of the request, used to build the deep link.

The flow begins by creating a new Teams chat using the list of email addresses passed from the app. This is achieved through the "Create Chat" action from the Microsoft Teams connector. Depending on the number of recipients (one-to-one or group chat), Teams automatically creates the appropriate conversation thread. Next, two messages are sent within the newly created chat:

1. A system-generated message from the flow itself, which briefly introduces the context and includes a deep link to the related request in Power Apps. This ensures that recipients can directly open the relevant screen and review the test request without having to search manually.
2. The user's message, sent on behalf of the individual who initiated the clarification request. This message preserves the original formatting and content provided in the rich text editor within the app.

The usage of two separate messages allows the flow to clearly distinguish between the structural context (system message) and the actual clarification content (user input), improving readability for recipients.

4.3.5 Duplicate attachment flow

The *Duplicate attachment flow* is used to copy all attachments from one SharePoint item to another. The flow is triggered directly from Power Apps and receives three input values:

- the source item ID (the original request);
- the destination item ID (the new or duplicated request);

- the name of the SharePoint list containing the items.

Once started, the flow performs the following steps:

1. get all attachments from the source item using its ID and the list name;
2. for each attachment found, the flow retrieves its content (the actual file);
3. it then adds the attachment to the destination item, keeping the same file name and content.

This logic is implemented using a loop that repeats the download and upload process for each file. By doing this, all files from the original request are transferred to the new request automatically, without the user needing to re-upload anything manually. The *Duplicate attachment flow* simplifies the process of reusing test request and ensures that duplicated requests are complete and consistent. It is especially helpful when similar test requests need to be submitted, saving time and reducing the risk of missing important information.

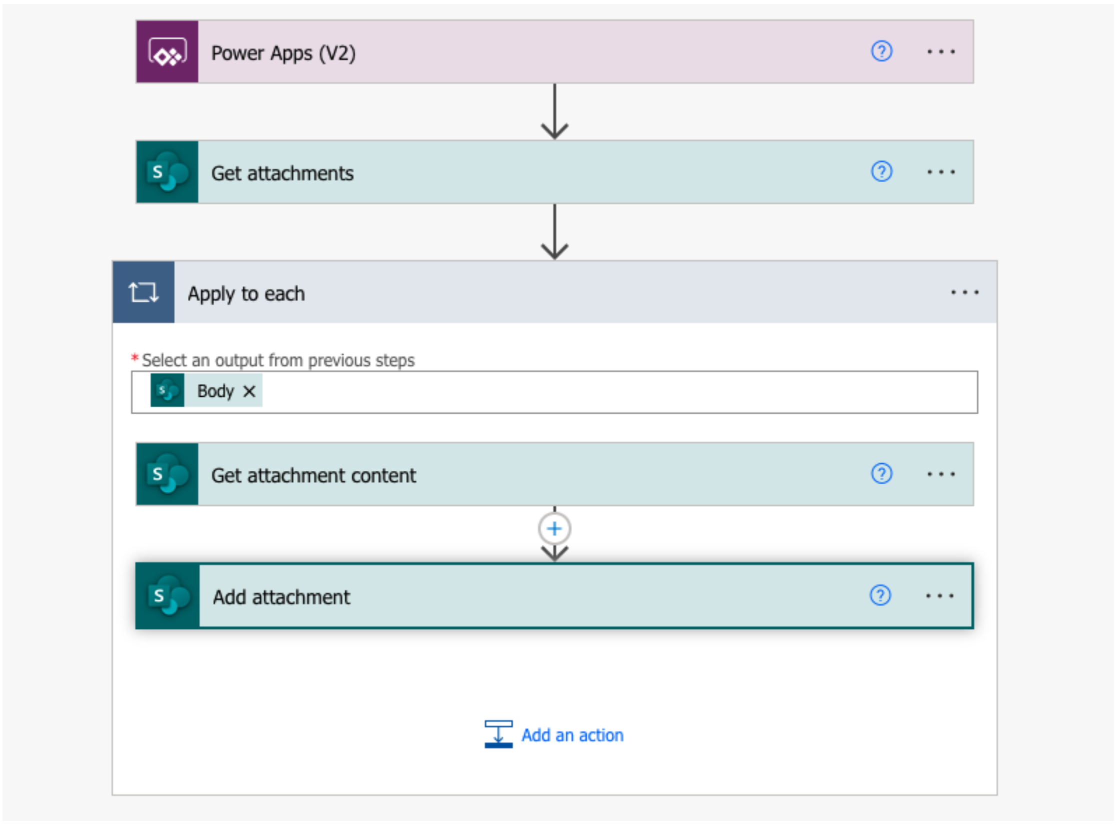


Figure 4.5: Duplicate attachment flow

4.3.6 Archive flows

To keep the system efficient and prevent the SharePoint lists from growing indefinitely, a set of *Power Automate flows* was implemented to periodically archive old test requests. These flows are category-specific, meaning that each test category has its own dedicated flow that handles its data structure. Each of these flows follows the same logic and is triggered automatically once a month. The process begins by identifying all requests that were created more than six months ago. This is done by subtracting six months from the current date and filtering the items accordingly. Once the outdated requests are identified, the flow performs the following steps for each one:

1. Create a copy of the request in a dedicated archive list. All the main fields and metadata are preserved.
2. Copy associated sub-items such as tasks, planning events, edit descriptions, or hardware changes. These items are duplicated and linked to the new archived request using its ID.
3. Delete the original items from the active SharePoint lists to avoid duplication and reduce the volume of active data.

By following this structure, the archive flows ensure that historical data is safely preserved in a separate list while keeping the active workspace clean and responsive. This approach also helps prevent hitting delegation or record count limits in Power Apps, as the live data is limited to only recent and relevant records. These flows run in the background and require no user intervention, making the archiving process fully automated.

Chapter 5

Deployment and results

5.1 Environment setup

During the development of the application, a multi-environment strategy was adopted to ensure safe testing, smooth deployment and minimal disruption to existing processes. The Microsoft Power Platform supports the use of multiple environments, such as Development and Production, to enable the separation of concerns and to establish a controlled pipeline for application lifecycle management. The environments used for the project are:

1. *TME-Codriver-Dev*: the development environment, where all new features, screens, flows and data structures were initially created and tested.

2. *TME-Codriver-Prod*: the production environment, intended for final deployment and real user interaction.

The development environment provided a controlled and isolated space where new features could be implemented and tested. Those were first implemented and then reviewed here through regular feedback sessions with group leaders. Once a stable version was achieved, the application was packaged and exported to be deployed into the production environment. This separation ensured that the final users were not affected by incomplete features or potentially unstable updates, allowing for a more agile and secure release process. The deployment process relied on the Power Platform Solutions feature, which enables developers to group application components, including Power Apps, SharePoint connections and Power Automate flows, into reusable and portable packages [39]. The application was exported from the development environment as a managed solution and imported into production.

5.2 Security and permissions

To make sure that the application is secure and accessible only to the right users, a layered permission model was adopted using the tools available in the Microsoft ecosystem. The goal was to ensure that only people who are directly involved in the testing process could access the app and interact with its data.

5.2.1 Power Apps access

Access to the application begins with the Power Apps sharing process. Only users who are explicitly added to the app in the TME-Codriver-Prod environment are able to open and use it. The sharing is managed through the Power Apps interface by selecting the app and adding users manually through their email addresses.

5.2.2 SharePoint site access

In addition to the application itself, users must also have the correct permissions to access the SharePoint site that stores all test-related data. This site acts as the main database for the application, containing lists such as Test Requests, Planner, Vehicle and others that support the full test lifecycle. Access to the SharePoint site is controlled through its permission settings, where users are assigned specific roles that define what they can see and modify. There are three main roles that can be assigned:

- *Group leader*: users in this group have permission to view and edit all the SharePoint lists, including the Roles lists (explained later in the section 5.2.4). This is important because group leaders are responsible for managing the team structure inside the application and must be able to assign or update user roles when needed.
- *Member*: members can interact with almost all data; they can create, read, and edit items in most lists, such as Test Requests, Planner and Vehicle. However, they cannot access or modify the Roles lists, ensuring that role assignments remain under the control of group leaders only.
- *Owner*: this role is reserved for administrators. Owners have full control over the entire site and its configuration. They can manage user permissions, site settings, and all underlying data structures. This role should be assigned with caution, as it allows unrestricted access.

Each user must be added to the SharePoint site with the correct role according to their responsibilities. This multi-level access model ensures data security and enforces role separation, reducing the risk of accidental changes or unauthorized access. It also aligns with the in-app logic, where features are shown or hidden based on a user's role as defined in SharePoint.

5.2.3 Calendar permissions

Users who need to schedule activities (typically group leaders) also need access to Outlook calendars. These calendars are used by the Power Automate flows to create and update test events. Without the correct permissions, group leaders would not be able to schedule tests or see existing events. Permissions to the calendars are managed directly from Outlook by setting the appropriate access rights.

5.2.4 Role management in the application

Inside the app, access to specific features is also controlled based on the user's assigned role. Roles are stored in dedicated SharePoint lists:

- *CDY roles*;
- *Workshop roles*;
- *EB roles*;
- *H2 roles.*

These lists define which users are group leaders and technicians for each test category. Group leaders can approve and plan requests, while technicians can mark them as completed. Only members of the "Test Request Group Leader" group in SharePoint can update these role lists. Once a user's role is added or removed, the changes are reflected immediately in the app interface.

5.3 Release strategy

The release of the Test Request Application was done in several steps to make sure everything worked well and users could learn to use the new tool without problems. After setting up the development and production environments (as explained in Section 5.1), the application was first tested by a small group of people before being shared with everyone. The first release was a beta version, shared only with a few group leaders and technicians. These users already knew how the test request process was working, so they were the best people to give useful feedback. A Microsoft Teams channel was created to help with communication during this phase. Inside the channel, there were different sections for reporting bugs or problems, suggesting new features and sharing updates or announcements. This setup helped to improve the application based on real feedback from the users. The group leaders used the app in real situations, checked if it was easy to use and gave suggestions about what to improve. Thanks to their help, several issues were fixed and small changes were made to better fit their needs. Once the beta phase was complete and the app was stable, it was shared with the entire department, which included about 150 people. To help everyone learn how to use the new system, three training sessions were organized. All the three sessions were the same, so that users could choose the one that best fit their schedule. During those sessions, users were shown how to:

- create new test requests;
- visualize, edit, delete and ask clarification of a submitted test request;
- schedule activities;
- mark requests as completed and upload results.

This step-by-step release allowed group leaders, who had already used the app during the beta phase, to help their teammates when needed. Thanks to this gradual approach, the app was well accepted by the team and quickly became part of their daily work.

5.4 Performance evaluation

After the full deployment of the application, it was important to evaluate its performance from both a technical and user perspective. This section provides an overview of how well the system performed under real usage conditions and whether it met expectations in terms of speed, stability, and user experience. The data presented in this section was collected approximately one week after the application was released to all users. This timeframe allowed the system to be used in a realistic context and provided a meaningful snapshot of its initial performance. While these early insights are valuable, it is important to note that a more comprehensive assessment would require data collected over a longer period. Unfortunately, extended monitoring was not possible in this case, as access to the system and its analytics tools ended with the conclusion of the internship. Therefore, the evaluation focuses on short-term performance based on the information available at that time.

5.4.1 Daily active users

One of the first indicators of adoption and engagement is the number of users who access the application each day. This metric helps evaluate whether the application is being used regularly and by how many individuals. Figure 5.1 shows the number of unique daily users over a one-week period. The usage data was collected through Power Apps Analytics [40] and highlights that, on average, around 41 different users accessed the application per day. It is also important to note that on November 16 and 17, usage dropped to zero. These days correspond to Saturday and Sunday, when the employees are not working. This confirms that the application is mainly used during working days as expected, and that it has already become an essential tool in the daily workflow of the team.

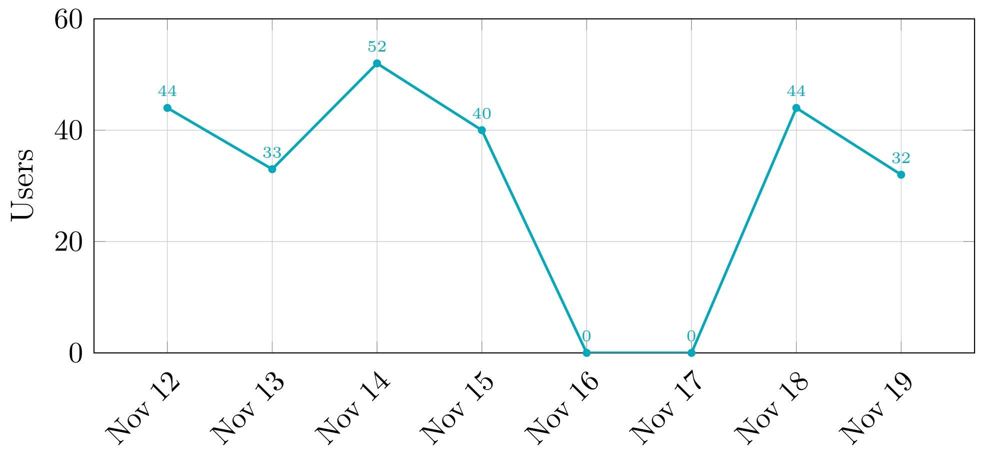


Figure 5.1: Daily active users

5.4.2 Load times and responsiveness

One of the major pain points of the previous Excel-based system was its slow loadings, especially when opening test request files with many formulas and macros. On average, opening a single Excel-based test request required around 23.93 seconds. In contrast, with the new Power Apps solution, users can access and open a test request in approximately 3.18 seconds, marking a substantial improvement in speed and usability. The following chart (Figure 5.2) illustrates the performance gain between the two systems:

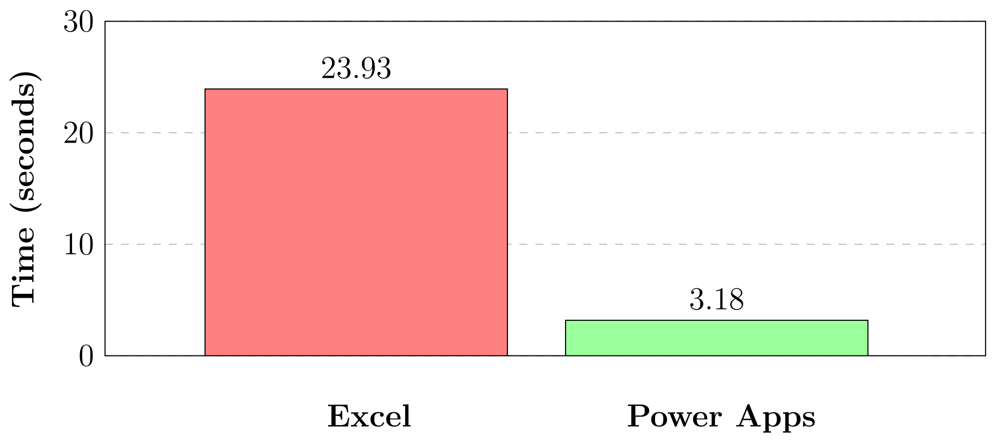


Figure 5.2: Average seconds to open a test request

In real usage, the Power Apps-based application showed good performance across all standard actions, such as:

- loading test requests from SharePoint lists;
- navigating between screens;
- filtering and searching inside galleries;
- submitting or editing test request forms.

Since the application separates test requests by category (e.g., CDY, Workshop, H2, Engine Bench), the number of records loaded per screen remains limited, reducing the risk of performance drops. In addition, old requests are regularly archived using automated Power Automate flows, which helps keep SharePoint lists light and manageable.

Another relevant metrics collected was the "Time to First Screen", measured through the Power Apps Analytics dashboard [40]. This metric captures the time it takes from the moment a user clicks on the app icon until the first screen becomes interactive. In Figure 5.3, the average "Time to First Screen" is shown for several consecutive days, split across three lines that represent different percentiles:

- The red line corresponds to the 95th percentile. This value shows the slowest loading experiences among the top 5% of users. Peaks in this line indicate that a small group of users experienced significantly longer wait times on certain days, possibly due to slower network conditions or older devices.
- The black line represents the 75th percentile, meaning that 75% of users experienced a loading time equal to or faster than the value shown. It gives a good indication of upper-bound performance for most users.
- The teal line (cyan/green) shows the 50th percentile, or the median value. This is the most typical experience, half of the users had load times below this threshold.

The overall trend observed in the first week of deployment showed that the majority of users experienced a consistent and fast startup experience. The median time remained close to 2 seconds, while even the 95th percentile rarely exceeded 8 seconds. This confirms that the application delivers a responsive and smooth experience for nearly all users. Tracking this metric over time helps identify if there are performance regressions and ensures that any issues affecting loading times are quickly detected and addressed.

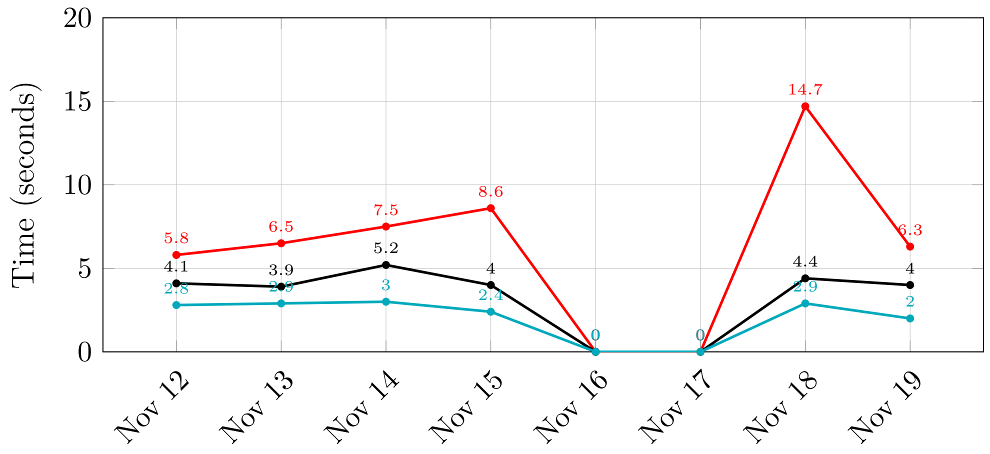


Figure 5.3: Time to first screen

5.4.3 Device usage

Usage data from Power Apps Analytics [40] shows that the application is accessed primarily from desktop computers. Approximately 95% of user sessions occur on PCs or laptops, while only 5% originate from mobile devices. This distribution confirms that the application is mainly used in office or lab environments where users operate from their workstations. This outcome was anticipated and directly guided the design of the user interface, which was tailored specifically for desktop use. The layout, interactions, and component sizes were all optimized for larger screens and mouse-based navigation, ensuring a smooth experience for users working from their desks.

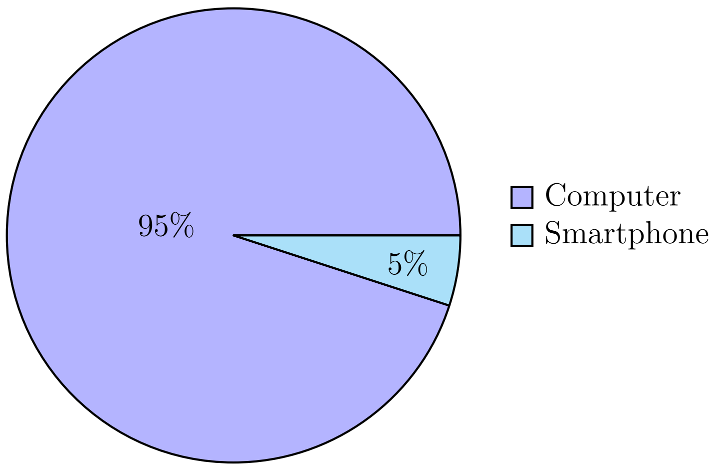


Figure 5.4: Device usage distribution for the application

Chapter 6

Conclusion and outlook

This final chapter summarizes the work accomplished in this thesis and discusses its main outcomes. It also looks to the future, suggesting potential improvements and new directions for the developed system.

6.1 Conclusion

This project started with a clear challenge: to improve a complex and inefficient business process at Toyota Motor Europe. The old system for managing test requests, which relied on many different Excel files and SharePoint sites, was slow, disorganized and prone to errors. This inefficiency wasted time and reduced productivity for engineers, group leaders and technicians. The main goal of this thesis was to design and build a modern and centralized software application to solve these problems. Using a low-code development approach with the Microsoft Power Platform, a new integrated system to manage the entire life-cycle of test requests was successfully created. The project followed a user-centered design methodology, starting with a detailed analysis of the needs of all engineers, group leaders and technicians to ensure the final application was practical and effective. The system architecture consists of a front-end built with Power Apps, a structured back-end using SharePoint Online lists and an automation layer powered by Power Automate to handle tasks like calendar integration and notifications. The results after launching the application showed significant improvements. The most impressive quantitative result was the dramatic reduction in the time needed to open a test request, which dropped from an average of 23.93 seconds with the old Excel system to just 3.18 seconds with the new Power Apps solution. The application was also quickly adopted by the team, with an average of 45 unique users accessing it daily in the first week. Qualitatively, the new system established a single, reliable source for all test data, giving everyone a real-time view of the status of every request.

Automated features, such as integration with the Outlook calendar and notifications via Microsoft Teams, have improved communication and reduced manual work. This project has demonstrated that a low-code platform can be effectively used to develop and deploy a robust enterprise-level application that solves critical business problems. The new system has replaced an inefficient and fragmented process, leading to increased productivity and a better user experience for everyone involved.

6.2 Future work and outlook

The application developed in this thesis provides a strong and stable foundation, but there are several opportunities for future enhancements that could deliver even more value. One key area for future work is the development of advanced analytics dashboards. Since all test-related data is now centralized and structured, it is possible to build dashboards to monitor Key Performance Indicators (KPIs). For example, managers could track metrics like test bench utilization, the average time to complete a request or the number of tests performed per vehicle. These insights could help optimize resource allocation and identify bottlenecks in the workflow, leading to better-informed management decisions. Furthermore, the application can be continuously developed and maintained, with the addition of new features and bug fixes. By continuing to follow the user-centered approach adopted during this project, it will be possible to further refine the application, ensuring it remains aligned with the team's evolving needs and continues to improve their experience. Finally, the modular design of the application and the flexibility of the low-code platform mean that the solution is not limited to its current scope. The system could be scaled and adapted for use by other departments within Toyota that follow similar request-management processes. By creating a standardized template from the current application, the core functionalities could be quickly replicated and customized for different teams, promoting operational consistency and efficiency across the wider organization. In summary, this project has not only solved an immediate business need but has also created a scalable platform that is ready for future enhancements, positioning it to meet the evolving needs of the department and the organization as a whole.

Bibliography

[1] Sameer Kumar and Ralph Harms. «Improving business processes for increased operational efficiency: a case study». In: *Journal of Manufacturing Technology Management* 15 (2004), pp. 662–674. URL: https://api.semanticscholar.org/CorpusID:110241935 (cit. on p. 1).

[2] Sajjadur Rahman, Kelly Avery Mack, Mangesh Bendre, Ruilin Zhang, Karrie Karahalios, and Aditya G. Parameswaran. «Benchmarking Spreadsheet Systems». In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020). URL: https://api.semanticscholar.org/CorpusID:208781474 (cit. on p. 2).

[3] Microsoft. *Excel specifications and limits*. Accessed: December 2024. URL: https://support.microsoft.com/en-us/office/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3?ui=en-us&rs=en-us&ad=us (cit. on p. 2).

[4] Steve Tockey. «How to Engineer Software: A Model-Based Approach». In: Wiley-IEEE Computer Society Pr, 2019. Chap. 4 (Functional and Nonfunctional Requirements) (cit. on p. 4).

[5] Inc. Gartner. *Gartner IT Glossary: Citizen Developer*. Accessed: July 2025. 2021. URL: https://www.gartner.com/en/information-technology/glossary/citizen-developer (cit. on p. 8).

[6] Microsoft. *Introduction to Microsoft Power Platform for developers*. Accessed: November 2024. URL: https://learn.microsoft.com/en-us/power-platform/developer/get-started (cit. on p. 8).

[7] Microsoft. *Connectors overview*. Accessed: July 2025. URL: https://learn.microsoft.com/en-us/connectors/overview (cit. on p. 10).

[8] Microsoft. *Power Platform environments overview*. Accessed: June 2025. URL: https://learn.microsoft.com/en-us/power-platform/admin/environments-overview?tabs=new (cit. on p. 10).

[9] Microsoft. *What is Power Apps?* Accessed: December 2024. URL: `https://learn.microsoft.com/en-us/power-apps/powerapps-overview` (cit. on pp. 11, 32).

[10] Microsoft. *Overview of creating apps in Power Apps.* Accessed: December 2024. URL: `https://learn.microsoft.com/en-us/power-apps/maker/` (cit. on p. 11).

[11] Microsoft. *What are model-driven apps in Power Apps?* Accessed: December 2024. URL: `https://learn.microsoft.com/en-us/power-apps/maker/model-driven-apps/model-driven-app-overview` (cit. on p. 11).

[12] Microsoft. *What are canvas apps?* Accessed: December 2024. URL: `https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/getting-started` (cit. on p. 11).

[13] Microsoft. *Create a blank canvas app from scratch.* Accessed: December 2024. URL: `https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/create-blank-app` (cit. on p. 11).

[14] Microsoft. *Formula reference for Power Apps.* Accessed: July 2025. URL: `https://learn.microsoft.com/en-us/power-platform/power-fx/formula-reference-canvas-apps` (cit. on p. 11).

[15] Microsoft. *Gallery control in Power Apps.* Accessed: July 2025. URL: `https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/controls/control-gallery` (cit. on p. 12).

[16] Microsoft. *Understand canvas-app forms.* Accessed: July 2025. URL: `https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/working-with-forms` (cit. on p. 12).

[17] Microsoft. *Understand data sources for canvas apps.* Accessed: July 2025. URL: `https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/working-with-data-sources` (cit. on p. 13).

[18] Microsoft. *Understand variables in canvas apps.* Accessed: July 2025. URL: `https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/working-with-variables` (cit. on p. 13).

[19] Microsoft. *What is Power Automate?* Accessed: May 2025. URL: `https://learn.microsoft.com/en-us/power-automate/flow-types` (cit. on pp. 13, 34).

[20] Microsoft. *Overview of cloud flows.* Accessed: July 2025. URL: `https://learn.microsoft.com/en-us/power-automate/overview-cloud` (cit. on p. 14).

[21] Microsoft. *Add a condition to a cloud flow in Power Automate.* Accessed: July 2025. URL: `https://learn.microsoft.com/en-us/power-automate/add-condition#add-a-condition` (cit. on p. 14).

[22] ShareGate. *SharePoint intranet development: Empowering team collaboration.* Accessed: July 2025. 2023. URL: `https://sharegate.com/blog/benefits-sharepoint-intranet` (cit. on p. 14).

[23] Microsoft. *Introduction to lists.* Accessed: July 2025. URL: `https://support.microsoft.com/en-us/office/introduction-to-lists-0a1c3ace-def0-44af-b225-cfa8d92c52d7` (cit. on p. 15).

[24] Microsoft. *What is a document library?* Accessed: July 2025. URL: `https://support.microsoft.com/en-us/office/what-is-a-document-library-3b5976dd-65cf-4c9e-bf5a-713c10ca2872` (cit. on p. 15).

[25] Microsoft. *Customize permissions for a SharePoint list or library.* Accessed: July 2025. URL: `https://support.microsoft.com/en-us/office/customize-permissions-for-a-sharepoint-list-or-library-02d770f3-59eb-4910-a608-5f84cc297782` (cit. on p. 16).

[26] Figma. *What is Figma?* Accessed: November 2024. URL: `https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma` (cit. on pp. 16, 20).

[27] Don Norman. «The Design of Everyday Things: Revised and Expanded Edition». In: MIT Press, 2014. Chap. 1.3 (Human-Centered Design) (cit. on p. 18).

[28] Martin Maguire. «Methods to support human-centred design». In: International Journal of Human-Computer Studies, 2001. Chap. Volume 55, Issue 4. URL: `https://www.sciencedirect.com/science/article/pii/S1071581901905038` (cit. on p. 19).

[29] Microsoft. *What is identity and access management (IAM)?* Accessed: May 2025. URL: `https://www.microsoft.com/en/security/business/security-101/what-is-identity-access-management-iam` (cit. on p. 20).

[30] Microsoft. *SharePoint limits.* Accessed: May 2025. URL: `https://learn.microsoft.com/en-us/office365/servicedescriptions/sharepoint-online-service-description/sharepoint-online-limits` (cit. on p. 20).

[31] Microsoft. *Use the SharePoint Site performance page.* Accessed: May 2025. URL: `https://support.microsoft.com/en-us/office/use-the-sharepoint-site-performance-page-38a1f782-2e73-4ec8-b55e-827611bc3632` (cit. on p. 20).

[32] Jakob Nielsen. «Progressive Disclosure». In: *Nielsen Norman Group* (2006). URL: `https://www.nngroup.com/articles/progressive-disclosure/` (cit. on p. 21).

[33] Jakob Nielsen. «10 Usability Heuristics for User Interface Design». In: *Nielsen Norman Group* (2024). URL: `https://www.nngroup.com/articles/ten-usability-heuristics/` (cit. on p. 22).

[34] Microsoft. *Introduction to lists.* Accessed: May 2025. URL: `https://support.microsoft.com/en-us/office/introduction-to-lists-0a1c3ace-def0-44af-b225-cfa8d92c52d7` (cit. on pp. 33, 37).

[35] Microsoft. *What is Microsoft Dataverse?* Accessed: April 2025. URL: `https://learn.microsoft.com/en-us/power-apps/maker/data-platform/data-platform-intro` (cit. on p. 36).

[36] Microsoft. *What is the Azure SQL Database service?* Accessed: July 2025. URL: `https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql` (cit. on p. 36).

[37] Microsoft. *Query limitations: Delegation and query limits.* Accessed: June 2025. URL: `https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/delegation-overview` (cit. on p. 40).

[38] Luke Wroblewsky. «Web Form Design: Filling in the Blanks». In: Rosenfeld Media, 2008. Chap. 8, 11. URL: `https://www.lukew.com/resources/web_form_design.asp` (cit. on p. 48).

[39] Microsoft. *Solutions in Power Apps overview.* Accessed: May 2025. URL: `https://learn.microsoft.com/en-us/power-apps/maker/data-platform/solutions-overview` (cit. on p. 56).

[40] Microsoft. *Admin Analytics for Power Apps.* Accessed: May 2025. URL: `https://learn.microsoft.com/en-us/power-platform/admin/analytics-powerapps?tabs=new` (cit. on pp. 60, 61, 63).