PartnerOS

Module 11 Design Project

Made by Group 2

Jelle Veldmaat	(s2570238)
Andre Andringa	(s2618184)
Cristian Zubcu	(s2558440)
Maouheb Bessi	(s2297205)
Marc Souvannasouck	(s2554224)

Supervisor: Vadim Zaytsev

Table of Contents

1 Introduction

The proposal of this project came from our client Jisse, a partner manager in the rapidly evolving SaaS (Software as a Service) industry. Jisse identified significant gaps in existing partner management solutions, which often fail to effectively support partner managers in their daily tasks. Recognizing the need for a comprehensive tool tailored to this specific domain, Jisse proposed the development of PartnerOS.

PartnerOS is a SaaS product engineered to address the shortcomings of current partner management systems by incorporating advanced features such as partner ROI measurement, efficient partner recruitment, and comparisons to ideal partner profiles. KPI extraction serves as the cornerstone of PartnerOS, enabling partner managers to monitor and evaluate key performance metrics with ease.

Leveraging Jisse's expertise and insights into the SaaS landscape, our team collaborated closely with him to bring the vision of PartnerOS to life. This innovative platform promises to revolutionize partner management, providing a seamless and user-friendly solution for professionals in the SaaS sector.

1.1 Definitions and terms

To guarantee a shared understanding of various terms, we have compiled a list of definitions. These will provide clarity on the meanings of these words for the subsequent report and throughout the entire project.

Definitions:
Client - Jisse, domain expert, project supplier (in the report the term *Client* and the name *Jisse* are used interchangeably).

Customer - A company that is a customer of our client.

Partner - A business partner of our customer. This can be any type of partner including but not limited to: sales partners, integration partners, affiliate partners.

End-customer - A company that is a customer of the partner and so, an indirect customer of the clients customer.

Partner Manager - The direct end user of our product, a person that manages a certain amount of partners for a customer. Employed by the customer. Using the software from the client as a tool.

KPI - Key Performance Indicators, in our case ideal partner profile comparison.

Journey - All steps in the relationship between a customer and their partners.

Board - A specific part of a journey represented by a kanban board. This board represents a part of a phase for example: recruitment, on-boarding, growing and retainment .

Phase - A specific set of tasks part of a board.

Task - A specific task that is part of a phase.

2 Domain analysis

2.1 Domain introduction

SaaS partnerships refer to collaborations between Software-as-a-Service (SaaS) providers and other businesses or organizations. SaaS is a cloud-based software delivery model in which software applications are hosted by a third-party provider and made available to customers over the internet.

SaaS partnerships can take many forms, such as integration partnerships, reseller partnerships, referral partnerships, and co-marketing partnerships. These partnerships are aimed at delivering more value to customers by combining the strengths of two or more SaaS providers.

SaaS partnerships are becoming increasingly important in the tech industry as businesses look to increase value to their customers by offering extensive solutions that are easy to use and integrate with existing systems.

2.2 Domain expert insights

These insights are from Jisse, our client, regarding the domain, other products available on the market, and the gap that PartnerOS should fill.

Most PRM's / Partner Relationship Management platforms in the market started as tools to make channel sales efficient and scalable, leading to self-service partner portals for partners to find all necessary resources and training to be able to sell the vendor's solution. This means that every vendor selling through partnerships, is offering a partner portal solution on their own.

One of the main indicators for a team managing sales partnerships is partner engagement data. If your partners are not engaged with your company/product, then it is really likely that they won't mention you in the next sales opportunity. Or even worse, mentioning your competitor.

In the current SaaS landscape, a lot of tools can be integrated with one another. Meaning that it is now possible to capture and combine activity data for every interaction that a partner has with your company. For example, if they send an email to your support team, it'll land in the support tool. If they send an email to the sales team, it will land in their CRM etc. All of this activity data can be combined via the API and used as input for a partner's health score to measure the relationship with your partner. This does not exist within the current PRM tools and will be implemented in PartnerOS in the future.

Where PartnerOS currently differentiates is measuring ROI. According to the research of Jisse Plaggenborg in summer 2022, all partnership teams have trouble with calculating ROI on partnership. In the current 'ecosystem' approach there are many different types of partners engaging in partnerships with each other. Ranging from integrations (SaaS software connecting to other software), affiliates, sales, services, system integrators, software development partners and/or BI dashboard development partners.

PartnerOS offers a solution where partners move through the partner lifecycle, managed by a partner manager while actively collecting interaction data such as time spent on meetings, tasks and calls. Combining this with output data such as the amount of leads generated and/or deals won we can calculate the ROI for sales. In the future, input data such as marketing costs, wage of the partnership professional per hour, upsells, renewals, churn prevention metrics etc. can all be added to the ROI overview.

For partnership teams this is crucial in order to understand which partners they'll need to invest their time in, what the output of their work is, and to get budget allocations based on true ROI.

2.3 Conclusion

In closing, the SaaS partnership domain is continuously evolving, with businesses seeking to enhance the value they provide to their customers through various partnership types. The current PRM tools available in the market focus on facilitating partner engagement, but there is a gap in the measurement of ROI for partnerships. PartnerOS aims to fill this gap by offering a solution that tracks and calculates ROI throughout the partner lifecycle, enabling partnership teams to optimize their efforts and allocate resources effectively. This approach will contribute to the efficient management of partnerships and help teams make data-driven decisions for their organizations.

3 Requirements

The foundation of any successful project lies in the meticulous gathering and analysis of its requirements. For PartnerOS, we embarked on a comprehensive process of collecting all essential information by conducting stakeholder interviews and thoroughly examining existing systems and documentation together with our client. This commitment to understand the problem directly from individuals working in the domain is crucial. As it enables us to construct a software product that is not simply suitable for the needs of the target user, but it’s a standing testament for our promise of delivering a truly custom tailored solution. In this chapter, we will explore the insights obtained from the requirement gathering process, which serve as a blueprint for our final software product.

3.1 Initial general requirements

To gather a general understanding on what the intended use of PartnerOS would be, we initially conducted stakeholder interviews with our client Jisse. These meetings helped define the main goal of the platform, which is to manage, measure and grow SaaS partner success (source 1).

Firstly, an efficient way of managing a partner’s journey through the system had to be implemented. Jisse suggested adopting a kanban board style design. To be precise, each partner phase, recruiting, onboarding and retaining would be its own board. Each partner in the system would essentially represent a sticky note, which can be placed around by the user anywhere on the board. Based on the position of each Partner on the board, a user could easily tell in which stage the respective partner is. By clicking each Partner, the user would be able to access its tasks, therefore allowing for easy task management. Similar software such as HubSpot adopts the same design, as it is more efficient than other alternatives.

Secondly, a way to measure a partner’s success needed to be crafted. To solve this issue, PartnerOS would be required to track two things: time spent in each phase of the system and each partner's characteristics, such as end-customer base, ROI, etc. To solve the former issue, an automatic time and working hours tracker should be implemented. In this way, the system would record each time a partner is added/moved or deleted within the system. Ideally, the time spent on each task should be recorded as well. To address the latter problem, the system should track a partner's characteristics, and automatically assign scores based on how close these characteristics are to ideal values.

Lastly, to grow the success of each partner, the system should allow partner managers to view analytics on each partner, therefore allowing them to make accurate business decisions. Our client suggested implementing a Give vs Get overview, which would easily display how much the customer gains or loses with each partner.

When talking about the system itself, another crucial general requirement is that the platform should be scalable and allow for integration with other systems. This would make it easy for customers that already have a considerable partner base and want to switch to a different system.

3.2 Requirement selection

After obtaining a general overview of what PartnerOS should represent, we proceeded to select and define specific requirements for the system. This section presents the chosen functional and non-functional requirements, which together provide a comprehensive picture of the desired system capabilities.

3.2.1 Functional requirements

To begin with, we focused on defining the functional requirements, which outline the specific features and capabilities of PartnerOS. Firstly, for the **workflow management** three requirements were constructed:

R1: The system should implement kanban boards for each partner type and each phase of the partner journey, allowing users to manage and track the progress of partners.

R2: The system should enable customization of kanban boards to reflect the unique needs and processes of different partner types, ensuring that the boards are relevant and applicable to each specific partner.

R3: The system should allow users to add phases and tasks to kanban boards, providing a visual representation of the partner journey and facilitating progress tracking.

Next, three more requirements were added to address **task management** for each partner.

R4: The system should define standard tasks for each phase of the partner journey, ensuring a consistent approach across different partners.

R5: The system should allow for customization of tasks for individual partners based on specific requirements, enabling users to tailor the partner journey to meet unique needs.

R6: The system should enable the addition of extra tasks for a single partner or phase as needed, providing flexibility in managing the partner journey.

With the six requirements defined above, it becomes more clear how the system will handle the kanban-style workflow mentioned in Section 3.1. Moreover, a clever way of assigning tasks to a Partner was found, based on their current journey phase, while also allowing users to add extra customized tasks. Next, three requirement were added to define the mechanism for **time and effort tracking**:

R7: The system should measure time and working hours for each phase of the partner journey, capturing start and end times for each task to accurately track efforts.

R8: The system should implement standardized time estimates for common tasks (e.g., meetings: 30 min, emails: 5 *min), providing a consistent basis for time tracking across tasks.*

R9: The system should automatically measure time spent on each task, ensuring accurate tracking of efforts and enabling users to monitor the progress of the partner journey.

Requirements 7 through 9 describe the system recording start and end times of tasks, as well as having a list of standardized times to easily track time of activities. The system should also be able to track the time of activities automatically.

In order to show the analytics collected from all of the tracked data on our platform and allow users to make informed business decisions, the system also requires a way to manage the partners themselves, as opposed to their journey or their tasks. This includes having access to partner pages where company characteristics are shown, as well as showing scores to quantify each partner. For **partner management** we have three requirements:

R10: The system should create partner pages with company and contact information, allowing for easy viewing, editing, and storage of partner data.

R11: The system should track partner statistics, such as time spent on each task or project phase, and visualize this data on the company page, providing insights into partner performance.

R12: The system should monitor the current phase of the partner journey and display the partner's ideal profile score, enabling users to assess and manage partner progress effectively.

Our system would make use of "ideal profiles", which represent partner profiles with maximized characteristics for the clients gain, which are used as a comparison to generate partner scores. Therefore, for the **ideal partner profiles** we added the following requirements:

R13: The system should allow users to create custom profiles to compare partners, taking into account the unique characteristics and requirements of different partner types, allowing for more accurate assessments.

R14: The system should extract the ideal partner profile based on the success of previous partners, enabling users to identify key attributes that contribute to successful partnerships.

R15: The system should quantify partner performance using a scoring system, allowing for comparisons and evaluations of partners to facilitate data-driven decision-making.

Lastly, we have 2 additional requirements. Specifically, the R16 addresses the issue of **scalability** and R17 addresses the issue of **integration** with another system, HubSpot. These are the last 2 functional requirements:

R16: The system should allow easy onboarding of new customers by allowing them to use the same database template with the press of a button.

R17: The system should allow users to import HubSpot accounts to PartnerOS, *ensuring seamless synchronization of partner data.*

In total, we have defined 17 functional requirements that cover various aspects of PartnerOS, from workflow management and task assignment to performance measurement and integration with external systems.

3.2.2 Non-functional requirements

After defining the functional requirements, we turned our attention to the non-functional requirements. These requirements are essential to ensuring a positive user experience and meeting performance expectations. We have divided these non-functional requirements into three categories: performance, usability, and flexibility. For **performance**:
R18: The system should ensure that the infrastructure can handle an increasing number of users and partners without compromising performance.

R19: The system should provide a dedicated analytics page to visualize time and effort data, enabling data-driven decision making.

For **usability**:
R20: The system should have an intuitive and user-friendly interface for easy navigation and efficient use of the platform.

And lastly, for **flexibility**:
R21: The system should cater to the specific needs of different partner types and ensure a streamlined and efficient process for all stakeholders involved.

By incorporating both functional and non-functional requirements, we have created a comprehensive framework for the design and development of PartnerOS, ensuring that the final product meets the clients expectations and effectively addresses the challenges associated with partner management.

3.3 Requirement prioritization

Given that there are 21 requirements to be considered, it is prudent to prioritize them for effective project management.

To prioritize the identified requirements for PartnerOS, we utilized the MoSCoW method. In this approach, all requirements are divided into four distinct categories. The most crucial requirements are classified as Must-haves, which are essential for implementation. The next category, Should-haves, consists of tasks that are important for the final product but not strictly necessary. Following that, Could-haves are nice-to-have features that do not impede the system's functionality if left unimplemented. Lastly, the Will-not-haves are requirements that are not scheduled for implementation during the current development phase. These may be considered for future implementation, but a new list of requirements and an updated prioritization would be necessary.

For our project, we established the requirements' prioritization through two meetings with our client. The initial meeting aimed to lay the groundwork for the priority levels. After the meeting, our team drafted a proposal for the prioritization. In the subsequent meeting, we reviewed this proposal with the client and made minor adjustments based on the feedback provided by Jisse. We then finalized the prioritized list of requirements, as presented below in the form of a table, where each requirement is indicated by its number.

	Requirement number
Must *(have)*	R1, R2, R4, R5, R7, R10, R15, R16, R18, R19, R20, R21
Should	R3, R6, R8, R11, R12, R13
Could	
Will not	R9, R14, R17

Table 1: MoSCoW prioritization table

First, the Will-not requirements are addressed. We found R9 to be difficult to complete as it is hard to automatically measure timing of activities if they are not done directly on PartnerOS. This is due to the fact that activities such as online and phone calls, sending emails and holding online video conferences can be done on countless platforms. Therefore, it would require a considerable amount of work to integrate so many possibilities into PartnerOS. R17 has been marked as Will-not for the same reasoning. A different problem arises with R14, which states that the system should extract ideal partner profiles based on the success of previous partners. We found that this problem is very complex, as it involves taking a large amount of characteristics into account. It was decided that at this stage, this requirement is not needed.

Moving onto the Should-have category, which contains features that are important but not crucial, we added R3 and R6 which covers the ability for users to add extra phases and tasks for the partner's journey. R8 addresses the issue of standardized time estimates, which are not essential but would make the process of collecting analytics much easier. R11 and R12 address monitoring a partner's statistics and displaying partner scores, which aid the user in making informed decisions, but are still not essential for the use of the system. R13 was deemed not essential as it involves allowing users to create custom profiles to compare other partners, however, the comparing partners feature is not essential for the system to work.

It is also important to note that we have no Could-have requirements. This is due to the fact that we did not have enough time to spend on features which would not serve any greater purpose.

The rest of the requirements in the Must category, are all essential to the system and cannot be omitted.

3.4 Conclusion

In short, the thorough requirement gathering process enabled us to identify and prioritize essential features for the PartnerOS platform. By engaging with the client, we have devised a comprehensive list of functional and non-functional requirements. The MoSCoW prioritization method further refined this list, ensuring that the most crucial aspects of the system are addressed. As a result, we have established a solid foundation upon which to design and develop a tailor-made solution that meets the needs of our client and target users.

4 Technology and Tool research

Embarking on such a substantial project demands a more thorough understanding of the most suitable existing technologies and tools to ensure the achievement of optimal effectiveness and efficacy. Using the right resources from the beginning, can pave the way for a successful and seamless development project journey. In this chapter, we delve into the vast technological pool of tools, in order to identify and evaluate the best options to facilitate the attainment of the project's goals. By equipping our team with the most fitting tools, we can transform challenges into opportunities, therefore, enhancing the overall project experience.

4.1 User Interface

The client Jisse provided the front-end team with a brief sketch of the desired user interface for PartnerOS, see appendix A. This gave the team a great deal of creative freedom, and the progress was presented to the client weekly, discussing design choices and coming to final decisions. Utilizing the initial sketch and feedback from the client, the team managed to create an initial version of the platform. A few weeks later, Jisse provided the team with a more detailed design of the pages in Figma, which greatly helped the team in aligning the platform with the client's preferences.

4.2 Back-end

The database is anticipated to be the backbone of this project. In order to develop the database effectively, it is crucial to research and evaluate various tools. This chapter will discuss the possible tools, their benefits, and drawbacks, as well as the rationale behind the final back-end decisions.

4.2.1 Database Design

Two promising tools for database design are: dbdiagram.io and Lucidchart. Dbdiagram.io is a popular tool for designing databases as it provides a combination of SQL code for creating the database and a database diagram. The diagram allows for easy visualization and verification of the correctness of the database, with clear representations of tables and their relationships. Additionally, the tool updates the diagram and SQL code in real-time, facilitating seamless collaboration. Considering these features, dbdiagram.io is a strong candidate for our project. On the other hand, Lucidchart is an option that offers drag-and-drop functionality and collaboration features. However, it lacks the real-time SQL code generation feature found in dbdiagram.io. Although Lucidchart is a powerful diagramming tool, the absence of this feature may hinder the efficiency of our back-end development process.

4.2.2 Database Deployment and API Provision

When tackling the issue of database deployment and API provisioning, two considered tools were Supabase and Firebase. Supabase is a free and open-source alternative to Firebase that handles database deployment and provides API endpoints for the front-end. The platform eliminates the need for programming a server, which can save valuable time and resources. Furthermore, a basic version of the project using Supabase has already been initiated, with some rudimentary requests functioning. This makes Supabase an attractive option for our database deployment. Moving onto the second option, Firebase, a well-known Google product, offers a comprehensive suite of features, including real-time database management, authentication, and cloud functions. While Firebase is a powerful and widely-used platform, its proprietary nature and potential cost implications may not align with our project's goals. Another option that was considered is building our own server and database. This custom backend would offer increased versatility. However, due to the significant investment of time and effort required, and the short timeframe for the project, we decided that using a third-party backend service would be the most efficient option. Building our own server would require us to handle tasks such as maintenance, security, and scalability, which could distract us from our core goal of delivering a high-quality product which satisfies the requirements as stated in the requirements section. By using a third-party service, we could focus on developing the frontend and other important features while leaving the backend infrastructure to experts in the field.

After careful consideration of the aforementioned tools, we propose using dbdiagram.io for database design and Supabase for deployment and API provision. The real-time collaboration and SQL code generation features of dbdiagram.io will streamline our back-end development process. Supabase's open-source nature, coupled with the initial progress already made using the platform, make it an ideal choice for our project.

4.3 Collaboration

In addition to the technology used for developing the product, it is essential to select tools that facilitate effective collaboration among team members. During our research, we focused on three main categories: communication, task management, and planning and note-taking. In this section, we will discuss the potential options for each category and the rationale behind our choices.

4.3.1 Communication

For communication, we considered platforms like Discord, WhatsApp, and Slack. Discord and WhatsApp were already familiar to most team members, and their ease of use made them strong contenders. Moreover, our client was also comfortable using WhatsApp as the main communication channel. Slack, on the other hand, is a popular tool designed specifically for team collaboration. Although Slack offers integrations and a rich feature set, the familiarity and simplicity of Discord and WhatsApp outweighed the additional capabilities.

4.3.2 Task management

When it comes to task management, tools like Trello, Asana, and ClickUp were explored. Trello is a widely-used tool that allows for easy organization and visualization of tasks. With its labeling and progress tracking features, Trello enables effective task management for both front- and back-end teams. Asana and ClickUp, while offering more advanced features, may have a steeper learning curve and might not align with our project requirements. Thus, Trello emerged as the preferred option for keeping track of tasks.

4.3.3 Planning and note-taking

Lastly, for planning and note-taking, we assessed various options, including Google Drive, Microsoft OneDrive, and Notion. Google Drive, with its integrated suite of tools like Google Docs and Google Slides, allows for real-time collaboration and efficient sharing of documents and meeting notes. The familiarity of Google Drive among team members made it a convenient choice. Microsoft OneDrive, while offering similar features, might not provide the same level of seamless collaboration. Notion, though versatile, could be more complex than necessary for our project's scope. Therefore, Google Drive was deemed the most suitable choice for planning and note-taking.

4.4 Conclusion

In conclusion, this chapter explored various tools and technologies to find the best options for our project. We decided on dbdiagram.io for database design and Supabase for deployment and API provision. For collaboration, we chose WhatsApp for communication, Trello for task management, and Google Drive for planning and note-taking. These choices aim to enhance our team's efficiency and collaboration, ultimately leading to a successful and well-executed project.

5 Internal project management

Utilizing the latest technologies and tools tailored to the project requirements is crucial for any software project. However, the real driving force behind a project's successful development lies in a well-crafted team strategy. In this chapter, we delve into the intricacies of our project management strategies, highlighting the significance of team roles and dynamics, as well as the integration of appropriate tools. From effective communication channels to setting realistic internal deadlines and managing potential risks, we emphasize the importance of harmonizing our team's efforts, ultimately ensuring a seamless transformation from concept to completion.

5.1 Team Strategy

In the initial development phase, our team adopted the SCRUM methodology, a popular Agile framework known for its flexibility and adaptability. Given that our client consistently introduced changes and refinements to the system's design, this approach allowed us to remain responsive throughout the fluctuating project requirements. In our project specifically, we opted for one-week Sprints to ensure a fast feedback loop and facilitate continuous improvement.

Each Sprint began with a client SCRUM meeting, where we identified the tasks requiring completion as a team. Throughout the Sprint, we held biweekly stand-up meetings to discuss progress, address any encountered roadblocks, and share information with all team members. Given that each member had a distinct area of focus, the stand-up meetings were essential to ensure the delivery of a high-quality product.

To foster a sense of shared responsibility within the team, we implemented a strategy of rotating the Scrum Master role among team members. This approach ensured that everyone had the opportunity to serve as Scrum Master at least once. Moreover, by switching the role around, the team benefited from diverse perspectives, leading to a more inclusive and well-rounded decision-making process.

5.2 Team organization and roles

Our team's organization and roles were structured to ensure an effective division of labor, with the work being split into two main areas: Front-end and Back-end development.

5.2.1 Front-end team

The Front-end team consisted of Jelle, Cristi and Maouheb. The team worked collaboratively, with tasks dynamically assigned based on the project's requirements, as well as individual strengths. This flexible approach makes sure that the team adapts quickly to changes, which was established to be a team requirement in Section 5.1.

5.2.2 Back-end team

Andre and Marc focused on the database aspect of the project. Andre primarily dealt with developing database functions for a more seamless integration with API calls, while Marc worked on designing and structuring the database to ensure utmost efficiency in data storage and retrieval.

This team organization and roles distribution facilitated a smooth development process, enabling each team member to contribute their expertise and effectively collaborate to achieve the project's goals.

5.3 Risk management

Risk management played a crucial role in our project's success. To address risks, we have implemented a systematic process to identify, assess and manage potential risks throughout the project's lifecycle. This section will discuss key risks encountered and the strategies adopted to address them.

5.3.1 Risk management process

The risk management process consists of four simple steps:

1. Identify risks
2. Assess risks
3. Plan response and mitigation
4. Monitor and review

The first step of identifying the risks was done through having regular brainstorming sessions where each team member, and our client Jisse, have the chance to speak up on any risks they might think of. This way we ensure that as a team we can anticipate and prepare for a wide range of issues.

Next, after a team member speaks up about a certain risk, we collectively assess the likelihood and potential project impact the risk might have. This process allows the team to prioritize risks and allocate time resources suitably.

For each identified risk, we then developed ways of minimizing its likelihood, but we also crafted response plans in case the risk actually occurs.

Lastly, we make sure to continuously monitor and report each one of these risks, as well as the mitigation strategies.

5.3.2 Key risks and mitigation strategies

Throughout the development phase of the project we encountered 4 big risks. In this section each one of these risks is stated, explained and a mitigation strategy is included:

Misalignment of expectations: Being Technical Computer Science students collaborating with an external client from the business domain, we recognized the risk of potential misunderstandings or misaligned expectations. To tackle this challenge, we ensured open and consistent communication with the client, organized regular progress meetings, as mentioned in Section 5.1. Therefore, we supplied our client with comprehensive documentation, thus maintaining alignment on project goals, requirements, and deliverables.

Skill gaps: Our team members possessed diverse technical backgrounds, which might have resulted in gaps in knowledge or expertise. To counteract this risk, we split the work according to preference and heavily promoted knowledge sharing within the team. This in turn created opportunities for team members to learn from each other. Furthermore, we consulted our supervisor and client when necessary for additional guidance.

Scope creep: Considering the nature of our project, we were vulnerable to scope creep, since new requirements or features frequently arose during development. To address this risk, we employed the Agile SCRUM methodology, as mentioned in Section 5.1. This method enabled us to prioritize tasks effectively and maintain flexibility, while still adhering to the project timeline.

Team member availability: As university students, we understood that our team members might face availability constraints due to other academic commitments or even personal circumstances. To alleviate this risk, we set up transparent communication channels, as mentioned in Section 4.3 and established clear expectations regarding work hours, project tasks, and deadlines, which allowed us to manage our time and resources efficiently.

5.4 Conclusion

All in all, this chapter discussed the importance of effective project management strategies for the success of our software project. By adopting the SCRUM methodology, organizing our team into specialized roles, and implementing a systematic risk management process, we aimed to ensure a smooth development process and deliver a high-quality product. The focus on communication, flexibility, and collaboration helped us to adapt to changes and achieve our project goals.

6 System architecture

In this chapter, a comprehensive overview of the system architecture of the project is provided. The following sections will firstly explore the global design, focusing on how the different components of the system interact with each other. Afterwards, we will dive into the specifics regarding the user interface design, highlighting its essential aspects and how it caters to the user experience. Subsequently, we will discuss the back-end design, examining its considerable role in supporting all system functionalities. Lastly, we will address the integration design, outlining techniques and strategies employed to seamlessly combine front-end and back-end components. By the end of this chapter, readers will have a clear understanding of the overall structure and organization of our system, as well as the rationale behind our design choices.

6.1 Global design

6.1.1 Overview

PartnerOS is a partner relationship management system designed to help businesses manage their partner networks, distribute their products, and monitor their sales activities (R1-R17). The system is built on a client-server architecture and uses web technologies to provide a modern and user-friendly interface (R20). To get a general overview of what different entities want from the partnerOS system we have made a Business Diagram, figure D.3.

6.1.2 High-level System Architecture

PartnerOS is a web-based system that uses a client-server architecture, implementing requirements R1-R17. The client-side of the system is a single-page application (SPA) built using Vue, which communicates with the server-side of the system through a RESTful API. The server-side of the system is built using Vue.js, and it interacts with a PostgreSQL database to store and retrieve data.

PartnerOS uses Supabase, which is a cloud-based open-source database as a service, to store and retrieve data. This design choice supports the system's ability to handle an increasing number of users and partners without compromising performance (R18).

6.1.3 Objectives

The objectives as described by Figure D.2 in Appendix D are as follows:

- Efficiently manage and store data associated with partners, end customers, and their attributes.
- Facilitate collaboration between team members through task management and assignment
- Support decision-making and reporting through a well-organized and easily accessible data structure

6.1.4 Key Components

PostgreSQL Database: The core of the system, containing all relevant data structures and relationships.

User Interface (UI): The front-end interface allows users to interact with the system.

Back-end: The server-side logic that handles user requests, processes data, and communicates with the database.

Integration: The connections and interactions between different components of the system, ensuring seamless functionality.

6.2 User interface design

6.2.1 Overview

PartnerOS features a modern and user-friendly interface that provides easy access to essential features and functions, fulfilling the usability requirement R20. Screenshots of the user interface are included in Appendix C and are references throughout this section. Users navigate on the screen by selecting pages on the Navigation Panel. The whole screen is showcased in Figure C.1, while the Navigation Panel is highlighted in Figure C.2. The Navigation Panel also contains a Settings dropdown. On this page users can do additional actions, which will be mentioned later. A screenshot of the Settings dropdown is shown in Figure C.3.

6.2.2 Design Principles and Objectives

Simplicity: The interface is designed to be simple and intuitive, with a clean and modern design.

Consistency: The interface maintains consistency across all pages and features, providing a familiar experience for users.

Accessibility: The interface is designed to be accessible to all users, including those with disabilities.

6.2.3 Key Features and Functions

Logging In: The system allows users to log in with the correct credentials. The login screen is shown in Figure C.4 in Appendix C.

Kanban Boards: The system implements kanban boards for each partner type and phase of the partner (R1), allowing users to manage and track the progress of partners. These boards can be seen in Figure C.5. Users interact with the board by dragging and dropping the partner tiles across the board. A board can also be edited using the Edit Board functionality, shown in Figure C.6. This feature is present in the Settings Dropdown.

Task Management: The system defines standard tasks for each phase of the partner journey (R4) and allows for customization of tasks for individual partners (R5) and addition of extra tasks as needed (R6). A screenshot of this functionality is shown in Figure C.7.

Partner Management: The system creates partner pages (R10), tracks partner statistics (R11), and monitors the current phase of the partner journey (R12). The partner page is shown in Figure C.8. This figure displays 3 separate components: Company Characteristics, Ideal Partner Score and the Give vs Get component. Our system also has a Partners page where a whole overview of all partners can be seen. This is highlighted in Figure C.9.

Time and Effort Tracking: The system measures time and working hours for each phase of the partner journey (R7) and implements standardized time estimates (R8). These time estimates are shown on Give vs Get cards from the Partner page. A screenshot of the Give vs Get element in detail is shown in Figure C.10.

Ideal Partner Profiles: The system enables users to create custom ideal profiles (R13), and quantify partner performance using a scoring system (R15). Adding an ideal partner profile is shown in Figure C.11. The partner score component is highlighted in Figure C.12.

Importing Partners: The system enables users to upload CSV files containing partners. This feature is shown in Figure C.13.

6.3 Backend Design

6.3.1 Overview

The Back-end Design chapter focuses on the server-side architecture and services that power PartnerOS, ensuring efficient data processing, storage, and retrieval as described in Figure 1, Appendix D. It highlights the use of Supabase and database functions to create an extensible and integrable system.

6.3.2 Key Components

Supabase Integration: PartnerOS leverages Supabase, an open-source alternative to Firebase, for real-time data synchronization, user authentication, and API management. Supabase simplifies the back-end development by providing a set of tools and services that work seamlessly with the PostgreSQL database.

RESTful API: PartnerOS utilizes Supabase's auto-generated API endpoints, which provide a standardized interface for the user interface and external systems to interact with the back-end services and the PostgreSQL database.

Database Functions: PartnerOS relies on PostgreSQL database functions to encapsulate complex business logic and calculations, such as partner scoring, task assignment, and ROI calculation. For examples of how these functions are implemented see Figures D.4-6 in the appendix. By implementing these functions in the database layer, PartnerOS ensures optimal performance, maintainability, and extensibility.

Event Triggers and Webhooks: PartnerOS uses Supabase's event triggers and webhooks to respond to changes in the database and execute specific actions, such as sending notifications or synchronizing data with external systems. This approach enables seamless integration with third-party platforms and facilitates the development of custom extensions.

Scalability and Performance: PartnerOS is designed to handle a growing number of users, partners, and end-customers. By leveraging Supabase's scalable infrastructure and optimizing the database schema and functions, the system can maintain high performance and reliability even as the data volume increases.

Security and Authentication: PartnerOS employs Supabase's built-in authentication and role-based access control features to ensure the security and privacy of user data. By using secure tokens and API keys, the system can restrict access to specific resources and functions based on user roles and permissions.

6.4 Integration design

6.4.1 Overview

In Section 3.3, R17 is marked as Will not in the MoSCoW prioritization table, showcased in Table 1. The following chapter presents the Integration design of PartnerOS, had we committed to implementing R17. Therefore the chapter specifically focuses on the specific techniques and approaches employed by PartnerOS to facilitate seamless communication between the PRM software and external systems, such as HubSpot. This chapter will detail the integration mechanisms that make it easier for users to connect PartnerOS with their existing tools and platforms, as well as discuss the support for CSV imports for partner data.

6.4.2 Authentication and Authorization

PartnerOS uses OAuth 2.0 to provide secure and streamlined authentication and authorization for third-party integrations, including HubSpot. (R17) Furthermore, Role-Based Access Control (RBAC) is employed to manage permissions for API-based integrations, allowing users to define roles and assign them to individual API keys for fine-tuned access control.

6.4.3 Data Mapping and Transformation

PartnerOS offers a flexible data mapping system that enables users to define how data from external systems should be mapped to internal data structures, ensuring seamless data exchange and synchronization. The built-in data transformation engine allows users to manipulate and convert data formats as needed during the integration process, maintaining data compatibility between different systems.

6.4.4 Integration Templates and CSV Imports

Integration templates simplify the process of connecting PartnerOS with popular third-party services like HubSpot, allowing users to quickly establish connections with their existing tools. Additionally, PartnerOS supports importing partner data using CSV files, streamlining the process of onboarding new partners or migrating data from other systems.

6.4.5 Monitoring and Maintenance

PartnerOS offers built-in monitoring tools that allow users to track the performance and health of their integrations in real-time. These tools provide insights and alerts, enabling users to identify and address potential issues promptly. Regular updates, patches, and improvements are provided for SDKs, libraries, and templates to ensure that integrations remain stable and secure. Users also have access to support resources and troubleshooting guides to help maintain and optimize their integrations with HubSpot and other systems over time.

7 Development process

In this chapter, we delve into the development process of the PartnerOS project, offering a comprehensive insight into the various stages and critical decisions that shaped the final product. We begin by discussing the key design decisions made by the team, highlighting their implications and potential alternatives. Next, we provide a weekly activity breakdown, illustrating the progression of the project and the contributions of each team member. The objective of this chapter is to present a transparent view of our project's journey, elucidating how the team reached crucial decisions and the specific roles and contributions of each individual.

7.1 Key design decisions

In this section, we will explore the pivotal design decisions that played a crucial role in the development of PartnerOS. We will outline the rationale behind these choices, discuss their implications, and consider potential alternatives. The design decisions will be categorized into three main areas: prototyping, database and backend, and front-end. By examining these decisions, we aim to provide a clear understanding of the thought process and the factors that influenced our team's approach to the project.

7.1.1 Prototyping design decisions

For the database design and prototyping we used dbdiagram.io a tool where we could visually design our tables and connect them. Afterwards we could directly import this structure into our database provider of choice.

For the UI design our client made some designs in Figma. However, the client started changing and adding Figma designs half way through the project. So, ofcourse, we were not able to change the design every time. We talked about this with the client and could conclude that changing up the UI may seem easy for someone that does not have any experience implementing a UI. We showed him how much time it would take to change up the UI again and we concluded that it was not worth the time.

7.1.2 Database design decisions

For the database, we needed to make some major decisions before starting. First of all, we needed to determine the type of database that would fit our needs. Furthermore, we needed to choose how we would access the data in the database and how it would be connected to the frontend. These decisions all influenced the hosting platform of the database. Initially, the project used Supabase for the backend, but there was quite a bit of discussion on whether we should keep using it or switch to something else. This is why we looked at different parts of the backend.

For the type of database, we decided to use a relational database. We made this decision for a plethora of reasons. Firstly, a relational database is designed for scalability, which is important for this project according to R18. Secondly, the data in the database is divided into multiple tables that are connected through the use of primary and foreign keys. This allows for great customizability, which was also needed based on requirements R2, R3, and R5. Lastly, relational databases are very flexible in querying and analyzing data. With SQL, we can easily retrieve data based on specific criteria, perform complex joins across tables, and aggregate data to generate reports and statistics. This was important based on the project's requirements R9 and R11, which call for advanced reporting capabilities, such as calculating partner scores and the time spent based on standard times for a task. Overall, the decision to use a relational database was based on several factors, including scalability, customizability, and flexibility.

Furthermore, we have chosen to use stored procedures to access the data. This was a controversial decision in our group because Jelle had already started with an initial proof of concept using Supabase and its functionality to query the data. His main argument was that it was easier and faster to implement. However, others argued that using stored procedures was a better course of action. Arguments for using stored procedures were that they were faster and that storing precompiled SQL on the server is more efficient and can reduce network traffic. Besides this, using stored procedures also improved reusability. This last point has proven to be very beneficial as we ran into more and more issues with using Supabase as a backend host. Therefore, if this project is continued, it would be smart to convert the backend to be self-hosted, and the use of stored procedures aids in an easier transition to this other backend.

The connection from the frontend to the database is done through Supabase. This was not a unanimous decision, however. The other possible option that was considered is building the backend ourselves using a SQL database. This option has a major advantage in terms of customizability. However, it would also cost more time because it would require us to build an authentication system. Concluding, for this project, Supabase was more suited because of the time constraints. However, if the given project is continued, it would be recommended to build a server.

After researching the best fit for the different aspects of the database, we concluded that the use of Supabase was the right fit for this project. However, the accessing of the data will be done in a different manner compared to the initial proof of concept done by Jelle. In the end, we have decided to use stored procedures for the data access, mainly because of their efficiency and reusability.

7.1.3 Front-end design decisions

PartnerOS has chosen to implement its frontend using VueJS, which is a popular JavaScript framework for building user interfaces. The decision to use VueJS was based on its ability to bind data easily between the template modules and scripts, thus enabling interaction with different components allowing for a seamless re-render of the interface upon receiving new data. This has helped PartnerOS to create a highly responsive and dynamic user interface that enhances the overall user experience.

In addition, VueJS has provided support for the Quasar framework, which has proven to be a valuable asset for the development team. Quasar is a material design component library that provides developers with pre-built components, such as forms, buttons, and sliders, that are customizable and easy to use. This has significantly reduced the time and effort required to build custom components from scratch, freeing up valuable time that can be allocated to more important aspects of the project.

Moreover, Quasar has also contributed to the aesthetic interface of the application by providing good looking components that can be customized to fit the specific design of the project.

After conducting thorough research and practical experimentation, the front-end team reconsidered their initial plan of using Tailwind in combination with Quasar. The team discovered that Quasar already provided most of the functionality that Tailwind offered and using both resulted in several conflicts. As a result, the team decided to abandon the use of Tailwind altogether.

The front-end team faced a challenging decision during the project's development: switching from JavaScript to TypeScript, several weeks after starting the project. As the codebase grew rapidly, JavaScript became increasingly messy and complex, resulting in hard-to-read and hard-to-debug code. Although the team initially chose JavaScript due to familiarity with the language, it proved to be a bad decision since object creation and type annotations were difficult to manage in JavaScript. The conversion to TypeScript, which took a few days and caused some file breakage, was successful. The team believes this was the correct decision and would be beneficial if the project continued professionally in the future.

Overall, the combination of VueJS and Quasar has proven to be an effective and efficient solution for the frontend development of PartnerOS. It has enabled the team to build a user-friendly and visually appealing interface that is both easy to use and functional.

7.2 Weekly activity

As outlined in Section 5.1 of this report, our team has implemented the Scrum methodology. With our one-week sprints and regular SCRUM meetings, tracking the progress of each team member's tasks has been a straightforward process. In order to enhance readability and provide a detailed account of our progress, we have included Table B.1 in Appendix B. This table presents all the Sprints, the corresponding SCRUM Master, as well as the time period for each sprint.

Additionally, we have maintained a thorough record of each member's responsibilities per sprint in Table B.2, also located in Appendix B. Keeping track of individual tasks and contributions is crucial for the development process, as it allows us to monitor progress, identify potential bottlenecks, and ensure that each team member is accountable for their work. By meticulously documenting our team's efforts, we have fostered a collaborative environment that promotes transparency, efficiency, and ultimately, the successful completion of our project.

During the initial stages, Sprint 1 and 2, our team began slowly by setting up tools and implementing the primary functions for the database. In the front-end, initial pages were created but were not yet connected to the database. The connection between the front-end and back-end had not been established at this point. As we progressed into Sprint 2 and 3, the front-end and back-end were integrated, and additional requirements, such as the give vs get card and partner score display, were added. Improvements in navigation and routing on the website were also made during this period.

Sprint 4 and 5 focused more on refining the current implementation and fixing bugs. In these sprints, the team dedicated their efforts to optimizing the system and ensuring that all features were functioning as intended. Finally, Sprint 6 and 7 were dedicated to further bug fixing and preparing the product for deployment to the customer. The team worked diligently to ensure that the final product was polished and ready for use, meeting the high standards we had set for ourselves throughout the development process.

7.3 Conclusion

To sum up, this chapter provided an in-depth look into the development process of PartnerOS, discussing the critical design decisions encountered by the team, their implications and alternatives. The team's adoption of VueJS, Quasar, and a relational database proved to be effective choices that contributed to the success of the project. Furthermore, the transparency and accountability fostered through our SCRUM methodology, along with the detailed records of tasks and progress, facilitated a well-organized and efficient development process.

8 Testing

Testing is a critical component of software development. It plays a key role in ensuring that the end product meets the requirements and expectations of our client. In the following section, we will discuss the testing plan and results for this project. For the testing plan we rely on meetings with our client, manual testing and user-testing. The goal of this plan is for the client to be involved in the development process. This ensured that the client and team were on the same pages regarding the requirements and miscommunications can be resolved early on. The results of the testing are then reviewed, and the issues that arose were addressed. Overall the testing helped improve our software, and identify issues with the project.

8.1 Testing plan

The testing plan for this project will be a combination of meetings with the client, manual testing, and user-testing to ensure the final product meets the requirements and provides a satisfactory user experience. The agile nature of this project means that the requirements may change during development, but we will start by establishing the general requirements as outlined in Section 3.2. After creating the initial design, we will consult with our client and incorporate their feedback.

Upon implementing the features, we will begin functionality testing to ensure all interactive elements, such as buttons and forms, are working correctly. We will also test all links and the login function. Compatibility testing will be conducted to ensure the website functions well on different screen aspect ratios, although mobile compatibility is not a requirement for this project.

Following the manual testing phase, we will invite volunteers to participate in usability testing. This will help identify any errors or inconsistencies that may be difficult to detect when working closely with the product. By engaging external individuals, we aim to gain fresh perspectives and benefit from their unbiased feedback.

Lastly, we will present the product to the client for further testing. The client has agreed to conduct their own tests, and any issues found will be addressed by our team. This process will be completed in multiple increments to ensure a final product with as few flaws as possible.

This testing plan primarily relies on our testing efforts and the client's input. Our testing will ensure that the product is free of major flaws and functions broadly, while the client's testing will confirm that all requirements are met and potentially refine minor issues. Please refer to Appendix E, Table E.1 for a detailed list of test use cases.

8.2 Testing results

In the following section the results of the testing will be explained. We will start with the functionality testing results, continue with usability testing and end with the testing conducted by our client.

Functionality testing resulted in numerous errors, however these errors were overall minor and could be resolved relatively quickly. These tests made some errors in the backend evident, which were then resolved by the backend team. Besides this it also found multiple faults in the frontend which then were fixed as well. No bugs were found which could not be resolved relatively quickly.

The usability tests conducted by asking volunteers to use the product resulted in some great feedback. The test-subjects commented on some consistency issues between the different colors and buttons. These consistency issues were simple to solve. One of the volunteers also tried to enter malicious inputs. SQL injection was tried, however because of the stored procedures this did not pose a risk. Furthermore he entered the number of people working for a company to be less than 0, this ofcourse should not be possible. However in the briefing by our client he stated that the product will only be used by trained individuals, therefore it is not a big issue that someone can enter negative numbers for attributes. This issue would also be relatively hard to resolve because some company attributes could contain negative numbers. So to fix the problem the database would need to be changed, and the frontend would need the functionality to specify if an attribute can be negative. Therefore it was chosen to mention the issue in this report but not resolve it. This can be done when the project is continued on a later date.

Client testing has provided us with a lot of feedback throughout the project, including bug reports as well as an opportunity for Jisse to validate the requirements and clarify them if needed. We were able to successfully address all issues reported by the client, which included some significant changes such as adding a co-sales pipeline board for end-customers, ensuring that partner journeys always consist of the same boards regardless of partner type, and making sure that standard tasks and sections differ depending on partner type and board.

Overall the testing has provided us with a large quantity of usable feedback. We have tried to mend all the problems that came to light during the testing phase. Most bugs are fixed successfully, but some bigger shortcomings of our project are not feasible to fix in the scope of this project and will have to be attended to at a later date.

8.3 Conclusion

To conclude, our testing plan, which included functionality testing, usability testing, and client testing, played a crucial role in ensuring the quality of the PartnerOS project. By addressing bugs and incorporating valuable feedback, we have significantly improved the product and ensured it meets client requirements. Although some limitations remain, our comprehensive testing approach has contributed to delivering a robust and reliable software solution.

9 The final product

Presenting PartnerOS, the advanced partner management system tailored for SaaS companies. PartnerOS enables businesses to efficiently streamline their partner management processes, effectively monitor partner performance, and effortlessly recruit new partners. The system is designed to enhance human capabilities, empowering employees to focus on the less repetitive and more impactful aspects of their jobs. Through its real-time analytics and reporting capabilities, PartnerOS equips businesses with the essential insights required for making data-driven decisions regarding their partner network.

9.1 Final features

An important feature of our product was that it would be very customizable. This added a lot of difficulty in the design and implementation. In the following section we will describe the features in the final product. These descriptions will be done in the I-form.

First of all, with the press of a button, I will get a fresh copy of the template database for my new instance of the product. This makes the product very scalable.

I am able to log into the web-app with my own email and password (Figure C.1). Since it is my first time logging into the product I can set my avatar and add other users as necessary. Since I already have all my existing partners in Excel I can import them into the product as a CSV (Figure C.2). All partner attributes will be automatically added to the database even if the column does not exist, for example the field of the amount of end-customer a partner has. Also, any partner type present in the CSV file will be imported into the product.

Now I am able to create an Ideal Partner Profile (IPP) based on the attributes imported (for example amount of end-customers) and connect an ideal value and a weight to it (Figure C.3). The value and weight for each attribute will determine the final ideal partner score. This of course can be done for each partner type. Also, I can view all the partners in the partner overview with their score and sort them based on the IPP score (Figure C.4).

There are 3 journeys (boards) for each partner type and partner phase: Recruit, Onboard and Retain (Figure C.5). Each journey (board) can be fully customized within the product (Figure C.6). With different sections and tasks. Each task has a custom task type, for example an online meeting or going out for a coffee. Each task has a different standardized time of completion. Now, I, as a partner manager can complete a task for a section in the board and add a comment to it for that particular partner. I can also add a private task only for that particular partner (Figure C.7).

I can add a custom partner and add it to the Recruit phase for that partner type. There I complete section tasks and drag and drop him through the sections (Figure C.5). Time is measured on how long a partner is in a section. This can be used to see the average time to recruit a partner and the time for a particular partner.

To extract partner Return On Investment (ROI) we need a way to find a way to measure the Return part. Here the end-customer comes in. On the Co-sales board I can add end-customers, add a deal value to them and connect them to a partner that manages them. Here I can again drag and drop them across the sections (Figure C.8). Here the difference is that there is a won and a lost section in the end to indicate a lost or won end-customer.

On the partner profile I can now see the give vs get. The man hour spent on a partner (give) versus the total end-customers provided, the won-lost rate and the total value provided by a partner (Figure C.9). Together with the Ideal Partner Score and how it was composed (Figure C.10). These things are the most important things of our product for the client.

9.2 Excluded features

Unfortunately we were not able to implement all requirements. This was expected due to the complicated nature of this project and the changing requirements. Because of proper prioritization in this project we have managed to implement all crucial requirements.

One of the requirements that we did not manage to implement is the integration with the Hubspot API according to R17. This requirement was relatively important to our client, but it was outside of the scope of this project to implement. This integration is quite complex and we could not integrate HubSpot until we had the MVP of partnerOS working. We are content with the decision of making R17 a won't have on the MoSCoW scale. Making this decision early on enabled us to communicate with our client clearly and not disappoint him at the end.

The extracting of ideal partner scores based on the existing partners is another requirement that is not fulfilled in the current project. According to R14 we should automatically calculate an ideal partner profile based on a list of partners with their give vs get scores. For the client this feature was nice to have, but it was not critical. Therefore the project group has decided together with the client to prioritize this as a won't have requirement.

Lastly, another requirement that is not implemented is the automatic measurement of the time spent on a task. This requirement would be very useful to have, since it would allow for more accurate give versus get information, and a better ROI prediction. The problem is that for this to be done, our project needs integration with numerous different tools. For example, if we want to measure the time spent on writing an email, integrations with gmail are needed, but also with outlook and other mailing services. This would require a lot of work, and also for every different task the time spent would be measured differently using different tools. This would cascade when new task types are added. Therefore we have decided to rate this one as a won’t have requirement.

9.3 Limitations

In the following section, we will outline the parts of our project that are implemented but need some extra attention when the project is continued.

The main limitation of the current project is that it is not as polished as we would like it to be. This is due to the high number of requirements that needed to be implemented to even have an MVP. In the end, we did not have enough time or resources to make the product as perfect as we wanted it to be. Some flaws that were found during testing could not be resolved in the current development cycle.

Another limitation is the manual labor needed for duplicating the project for new customers. We have partially fixed this issue by writing a script that automatically creates a duplicate supabase project with the same database structure and methods. However, this step still requires customers to manually create an empty supabase project first. One of the reasons why we were not able to solve this limitation is because of the decision to use supabase as our backend host. On the contrary, a custom backend would have enabled fully automated duplication of the database for new customers. But this option would have required more development time. That is why, given the limited time span of this project, we chose to use supabase, which brings with it the limitation of requiring manual labor for new customers.

The last limitation is also caused by the use of supabase. As mentioned earlier, every client has their own supabase project. However, the premium fee for supabase is relatively high compared to hosting a custom backend. This cost results in a higher monthly fee for our customers. To mitigate this limitation, the same solution can be used as for the previous limitation, which is switching from supabase to a custom backend.

In conclusion, while our project was successful according to us, our client, and the requirements, it falls short of the level of polish we were aiming for. Time constraints and limited resources prevented us from resolving all the flaws found during testing. Additionally, while we have partially addressed the issue of manual labor needed for duplicating the project for new customers with a script, the requirement for customers to create an empty supabase project manually remains. The decision to use supabase as our backend host also limits us in terms of monthly costs for our customers. However, we recognize that the choice of supabase was necessary given the limited timeframe of this project. To overcome these limitations, it should be considered to switch to a custom backend in future development cycles, which would provide more flexibility in terms of cost and automation.

10 Reflection on the process

As the development team, we take pride in the successful completion of the development cycle and the resulting product. However, this journey has not been without its challenges and critical decision-making. Upon reflecting on our experiences, we recognize that there are aspects of the design process we would approach differently in future projects. This chapter delves into the lessons learned, the challenges faced, and the potential improvements we have identified, providing valuable insights for the continuous evolution of our team and the development process as a whole.

10.1 Requirement negotiations

During the initial meetings with our client, multiple requirements were discussed, with an emphasis on high levels of customization and a few non-negotiable aspects. This created difficulties during the design and implementation phases. Considering the time constraints and the client's goal to create a Minimal Viable Product, we should have negotiated more regarding the complexity of the project requirements. A reduction in customizability would have enabled us to allocate more time to other areas, such as the product's aesthetics.

10.2 Backend scalability

One of the early dilemmas was choosing between building our own server with custom endpoints and authentication system or utilizing Supabase API with their pre-existing endpoints and database. Ultimately, we went with Supabase, as it saved us from having to develop everything from scratch in the back-end. In hindsight, we should have built our own server due to our scaling design choice. Our goal was to have a unique database for each new client, as the entire database is customizable. We wanted to be able to scale this effortlessly, such as creating replicas of existing database templates and running them in separate docker containers. Unfortunately, we realized this too late, and it was too time-consuming to switch to a custom-built server. If we plan to continue this project in the future, we will need our server to handle these operations. Supabase is convenient, but it also has limitations in terms of control. While Supabase offers the ability to create distinct projects through its API, configuring the templates to match our requirements is not feasible. On top of that, using this approach becomes costly since Supabase charges for each project created. Therefore, utilizing Supabase for our project would not be a viable option due to the limitations and high expenses it imposes.

10.3 Future improvements

In this section we will talk about the enhancements that we recommend for future development efforts, whether undertaken by our team or other developers. First and foremost, it is crucial to implement a custom back-end and host the database separately, as opposed to the current setup. This change would improve the overall performance and security of the system.

Secondly, it is essential to strengthen the input sanitation in the front-end. Although the product is intended for use by trained professionals, human error is inevitable. Therefore, it is prudent to implement more robust input validation mechanisms to minimize the occurrence of input-related errors.

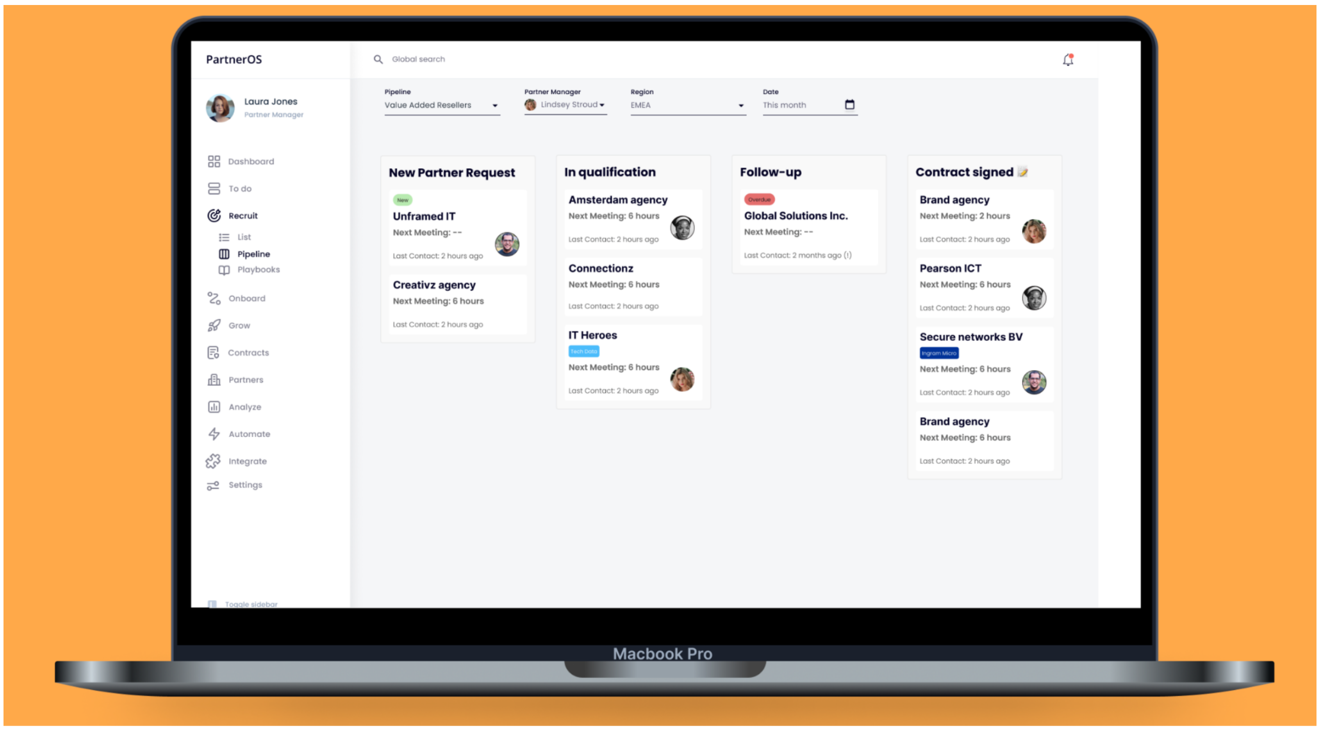
Additionally, we recommend refining the design and user experience of the board and section editing pages. For example, allowing users to rename section and task names, as well as reordering sections, would enhance the usability of the platform.

Lastly, we suggest revisiting the unimplemented requirements outlined in Section 3.3 of this report. Integrating the system with popular platforms like Hubspot is one such requirement. Such an integration could expand the user base and attract more customers to the platform.

Addressing the above recommendations at this point in time will significantly help with the future development of extra features. Thereby, improving the functionality, usability and overall appeal of PartnerOS in the future.

11 Appendix

Appendix A



Pipeline: Value Added Resellers
Partner Manager: Lindsey Stroud
Region: EMEA
Date: This month

New Partner Request

New
Unframed IT
Next Meeting: --
Last Contact: 2 hours ago

Creativz agency
Next Meeting: 6 hours
Last Contact: 2 hours ago

In qualification

Amsterdam agency
Next Meeting: 6 hours
Last Contact: 2 hours ago

Connectionz
Next Meeting: 6 hours
Last Contact: 2 hours ago

IT Heroes
Tech Data
Next Meeting: 6 hours
Last Contact: 2 hours ago

Follow-up

Overdue
Global Solutions Inc.
Next Meeting: --
Last Contact: 2 months ago (!)

Contract signed

Brand agency
Next Meeting: 2 hours
Last Contact: 2 hours ago

Pearson ICT
Next Meeting: 6 hours
Last Contact: 2 hours ago

Secure networks BV
Ingram Micro
Next Meeting: 6 hours
Last Contact: 2 hours ago

Brand agency
Next Meeting: 6 hours
Last Contact: 2 hours ago

Figure A.1 Initial sketch from our client Jisse

Appendix B

Sprint Number	SCRUM Master	Period
Sprint 0	Jelle	1/03/2023 - 07/03/2023
Sprint 1	André	7/03/2023 - 10/03/2023
Sprint 2	Maouheb	13/03/2023 - 17/03/2023
Sprint 3	Cristian	20/03/2023 - 24/03/2023
Sprint 4	Marc	27/03/2023 - 31/03/2023
Sprint 5	Jelle	03/04/2023 - 06/04/2023
Sprint 6	Andre	11/04/2023 - 14/04/2023
Sprint 7	Maouheb	17/04/2023 - 21/04/2023

Table B.1: Sprint periods and assigned Master

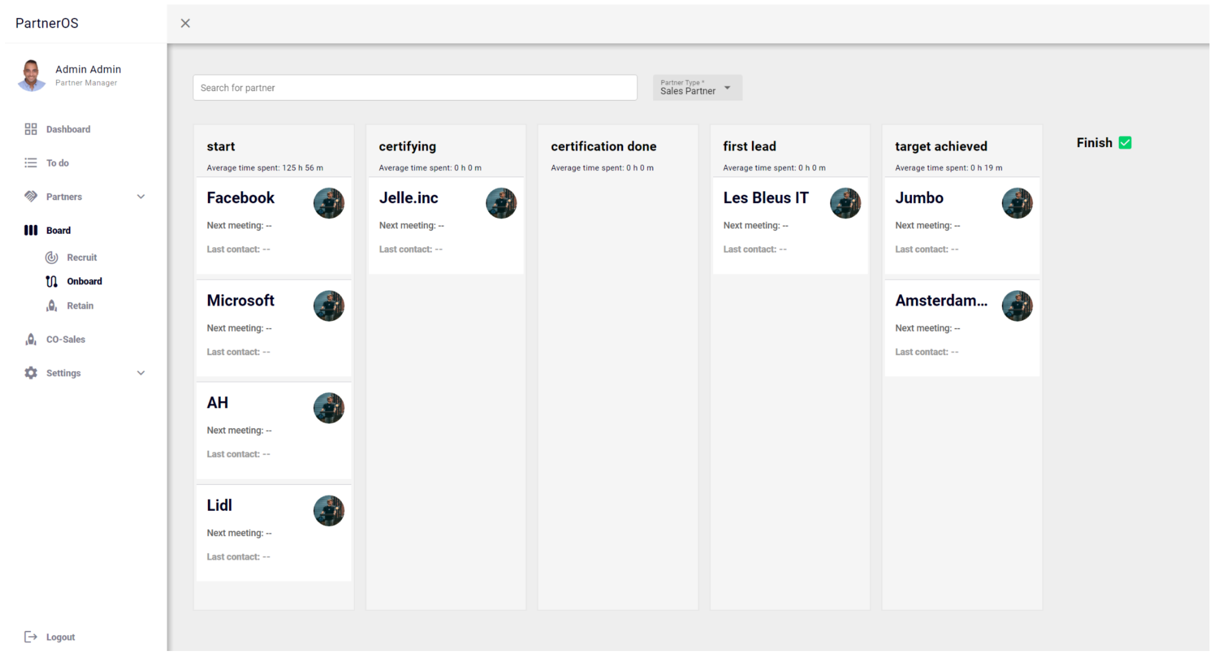
Sprint Nr	Team Member	Responsability
0	Jelle	Design Ideal Partner Score with weights
	André	Take meeting notes and write SQL functions
	Maouheb	Setup project and implement tailwind in project
	Cristian	log-in and register page design
	Marc	Design the initial database
1	Jelle	Report work, implement script to insert dummy data
	André	Impleget methods for journey, board, section, partner
	Maouheb	Match front-end design to prototypes
	Cristian	Implement drag-and-drop for partners between sections
	Marc	Import database schema into Supabase
2	Jelle	Show partner data on front-end, continue drag-and-drop work
	André	Get partner data, implement method to move partner to section
	Maouheb	Improve front-end design, implement settings page
	Cristian	Implement partner score display
	Marc	Input methods for journey, board, section, partner
3	Jelle	Implement adding a partner, getting task types
	André	Implement adding ideal partners and calculating partner scores
	Maouheb	Improve website layout, navigation, routing, settings and authentication
	Cristian	Implement give-vs-get card
	Marc	Update board API calls, implement showing partners on board
4	Jelle	Implement the Grow board

	André	Figure how to convert a board into JSON format
	Maouheb	Fix drag-and-drop bugs, improve boards generally.
	Cristian	Implement Partner task list
	Marc	Improve database and modify partner table
5	Jelle	Create the UI and function that translate the csv data to json to send it to the backend
	André	Make sql functions for import partners from csv, get next deadline and previous contact times.
	Maouheb	Transform project from JS to TS along with creating models. Implement settings page along with avatar upload and profile creation upon signup.
	Cristian	Display next and last contact, fix layout of partner page and bugs
	Marc	Research how to create new instances for every partner and import schema.
6	Jelle	Adding a new instance of the database and a connected branch so that our client can see the product any time and add his own data
	André	Removing add attribute and adding it to the creation of partners and ideal partners, making sure the query accepts new attributes and automatically creates them.
	Maouheb	Implement Give vs Get statistics. Improve scalability of the components. Perform code reviews.
	Cristian	Implementing the Partner overview page. Fix API calls on front
	Marc	Create a python script that duplicates the database
7	Jelle	Testing and fixing bugs emerged from the new database instance of the product and bugs found in the feedback from Jisse
	André	Help fix bugs with the new database, and other bugs that emerged with the client-testing, start writing reports.
	Maouheb	Deploy the website to the main.partneros.io and demo.partneros.io domain with a demo version running on the demo branch and a main version running from the master branch.
	Cristian	Improve layout and design of task list and partner overview

	Marc	Improve the script and also check how identical are the databases

Table B.2: Individual responsibilities per Sprint

Appendix C



start
Average time spent: 125 h 56 m

Facebook
Next meeting: --
Last contact: --

Microsoft
Next meeting: --
Last contact: --

AH
Next meeting: --
Last contact: --

Lidl
Next meeting: --
Last contact: --

certifying
Average time spent: 0 h 0 m

Jelle.inc
Next meeting: --
Last contact: --

certification done
Average time spent: 0 h 0 m

first lead
Average time spent: 0 h 0 m

Les Bleus IT
Next meeting: --
Last contact: --

target achieved
Average time spent: 0 h 19 m

Jumbo
Next meeting: --
Last contact: --

Amsterdam...
Next meeting: --
Last contact: --

Figure C.1: PartnerOS main screen

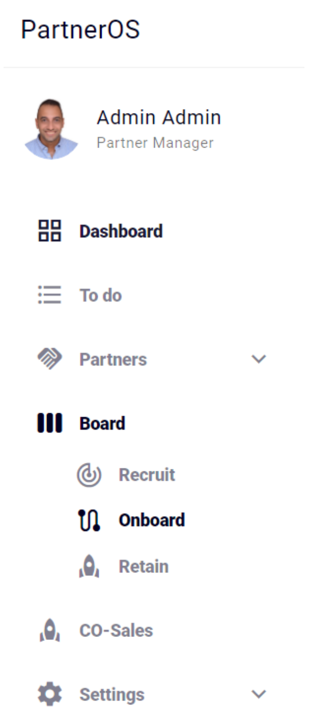


Figure C.2: Navigation Panel

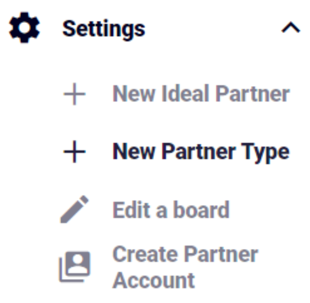


Figure C.3: Settings Dropdown

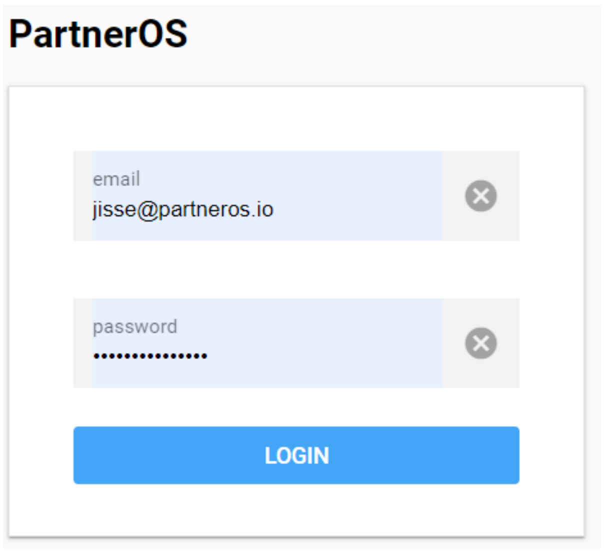
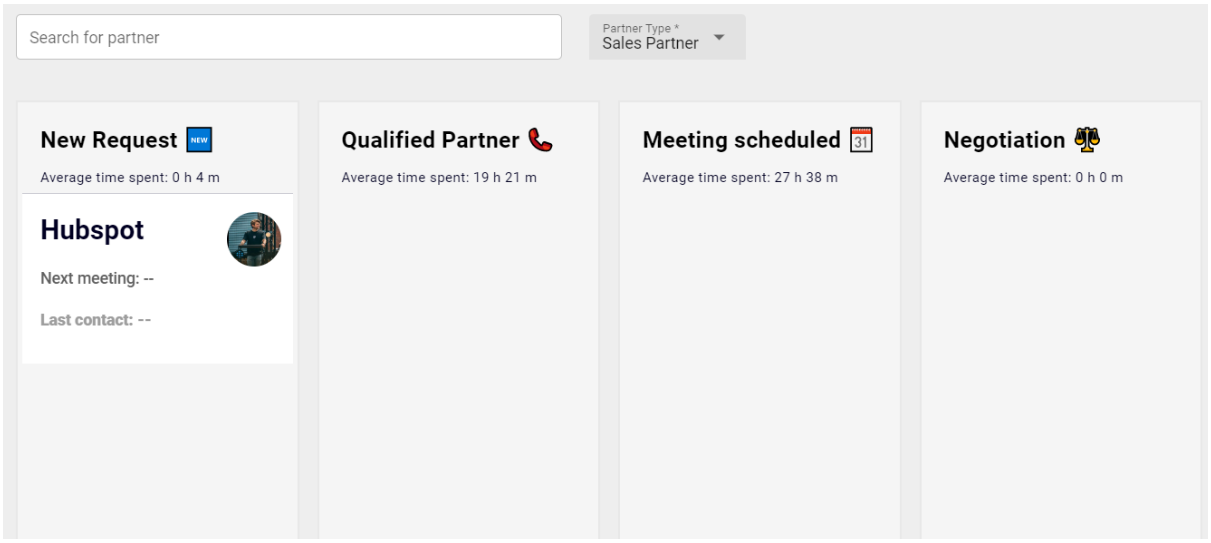


Figure C.4: Login component


Search for partner

Partner Type *
Sales Partner

New Request
Average time spent: 0 h 4 m

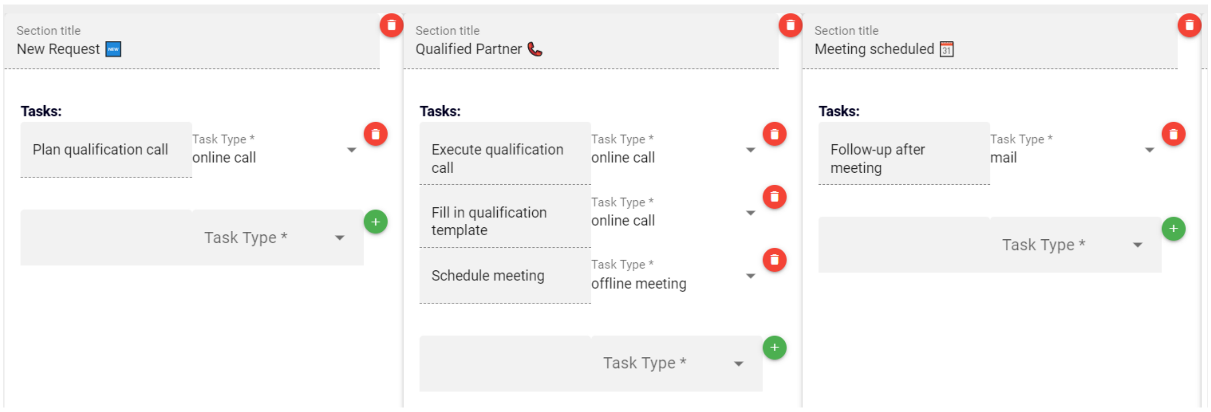
Hubspot
Next meeting: --
Last contact: --

Qualified Partner
Average time spent: 19 h 21 m

Meeting scheduled
Average time spent: 27 h 38 m

Negotiation
Average time spent: 0 h 0 m

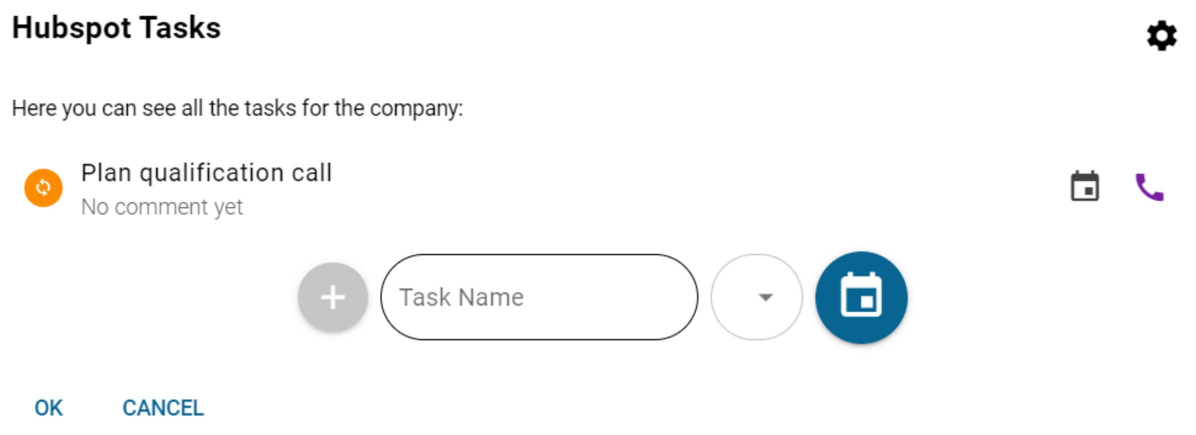
Figure C.5: Board overview component


Section title: New Request
Tasks:
Plan qualification call — Task Type *: online call
Task Type *

Section title: Qualified Partner
Tasks:
Execute qualification call — Task Type *: online call
Fill in qualification template — Task Type *: online call
Schedule meeting — Task Type *: offline meeting
Task Type *

Section title: Meeting scheduled
Tasks:
Follow-up after meeting — Task Type *: mail
Task Type *

Figure C.6: Edit board component


Hubspot Tasks

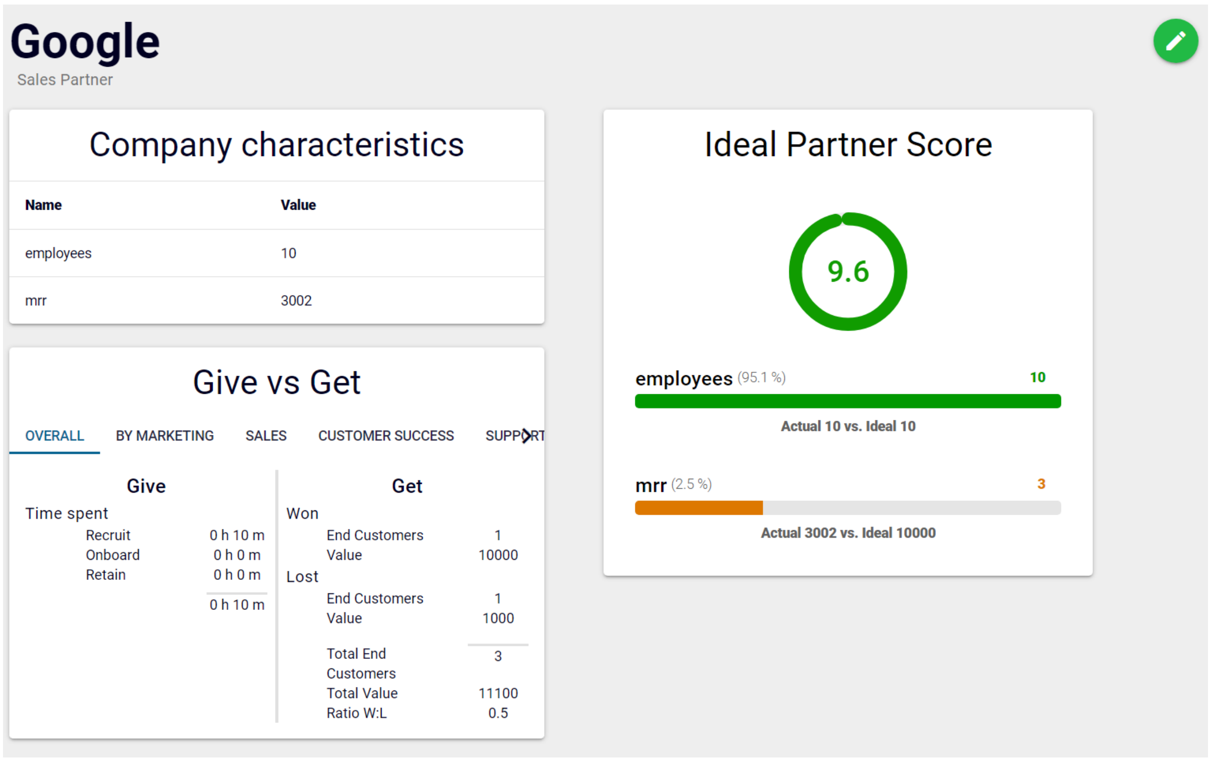
Here you can see all the tasks for the company:

Plan qualification call
No comment yet

Task Name

OK CANCEL

Figure C.7: Task list and adding a task component



Google
Sales Partner

Company characteristics

Name	Value
employees	10
mrr	3002

Give vs Get

Give			Get		
Time spent			Won		
	Recruit	0 h 10 m		End Customers	1
	Onboard	0 h 0 m		Value	10000
	Retain	0 h 0 m	Lost		
		0 h 10 m		End Customers	1
				Value	1000
				Total End Customers	3
				Total Value	11100
				Ratio W:L	0.5

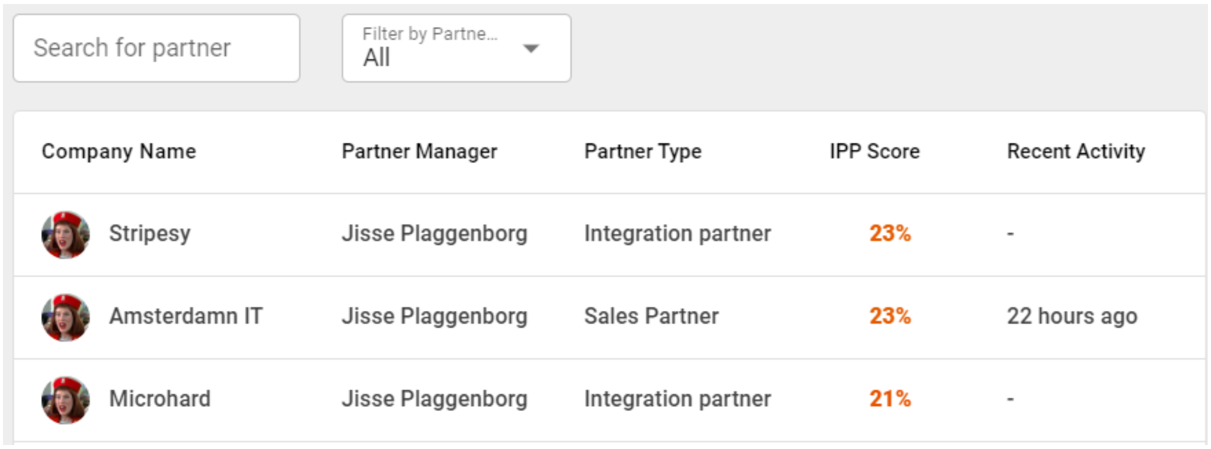
Ideal Partner Score

9.6

employees (95.1 %) 10
Actual 10 vs. Ideal 10

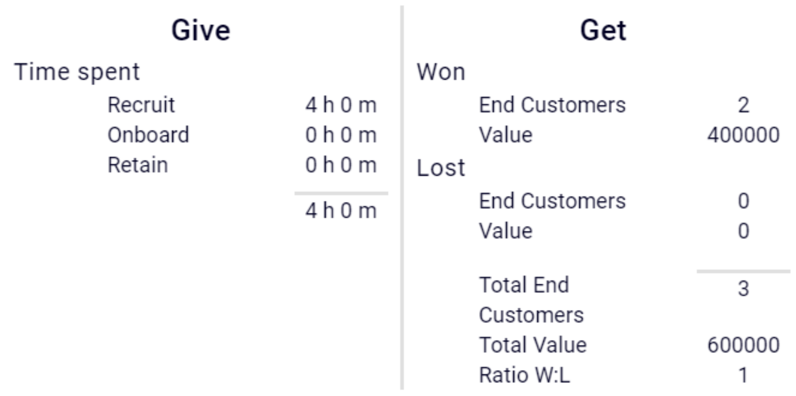
mrr (2.5 %) 3
Actual 3002 vs. Ideal 10000

Figure C.8: Partner Page


Search for partner
Filter by Partne... All

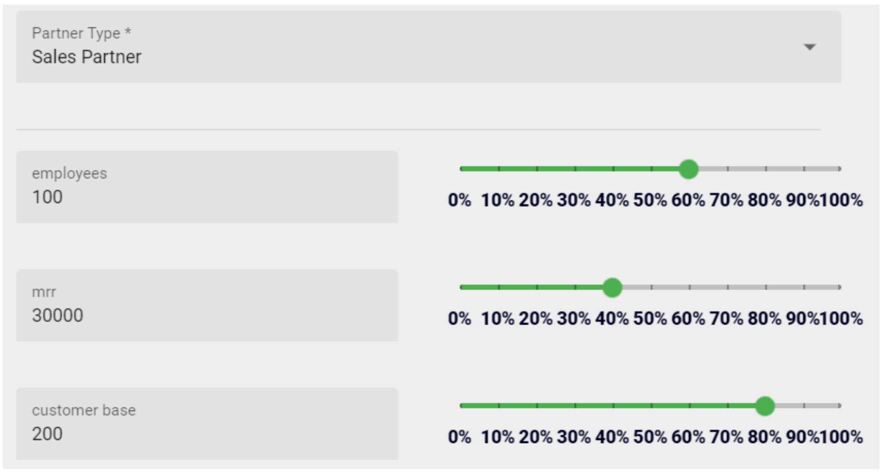
Company Name	Partner Manager	Partner Type	IPP Score	Recent Activity
Stripesy	Jisse Plaggenborg	Integration partner	23%	-
Amsterdamn IT	Jisse Plaggenborg	Sales Partner	23%	22 hours ago
Microhard	Jisse Plaggenborg	Integration partner	21%	-

Figure C.9: Partner overview component



Give			Get		
Time spent			Won		
	Recruit	4 h 0 m		End Customers	2
	Onboard	0 h 0 m		Value	400000
	Retain	0 h 0 m	Lost		
		4 h 0 m		End Customers	0
				Value	0
				Total End Customers	3
				Total Value	600000
				Ratio W:L	1

Figure C.10: Give versus Get component


Partner Type *
Sales Partner

employees
100

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

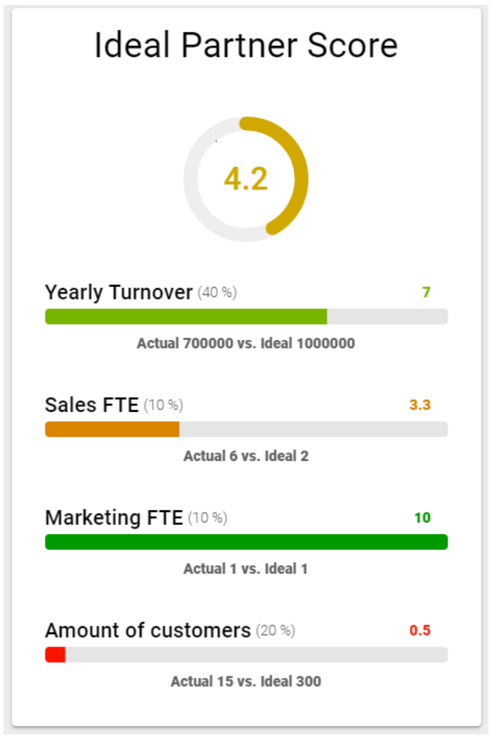
mrr
30000

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

customer base
200

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

Figure C.11: Add ideal partner component


Ideal Partner Score

4.2

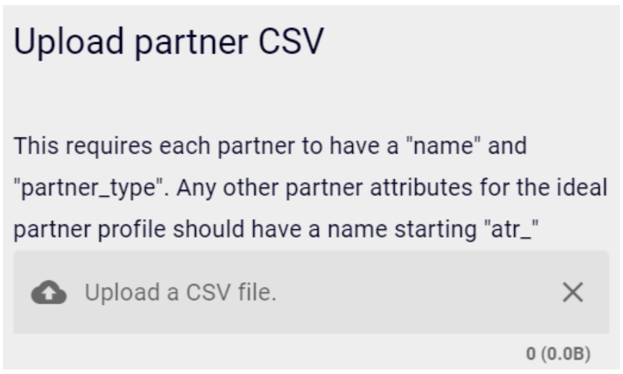
Yearly Turnover (40 %) 7
Actual 700000 vs. Ideal 1000000

Sales FTE (10 %) 3.3
Actual 6 vs. Ideal 2

Marketing FTE (10 %) 10
Actual 1 vs. Ideal 1

Amount of customers (20 %) 0.5
Actual 15 vs. Ideal 300

Figure C.12: Ideal Partner Score component


Upload partner CSV

This requires each partner to have a "name" and "partner_type". Any other partner attributes for the ideal partner profile should have a name starting "atr_"

Upload a CSV file.

0 (0.0B)

Figure C.13: Upload CSV file component

Appendix D

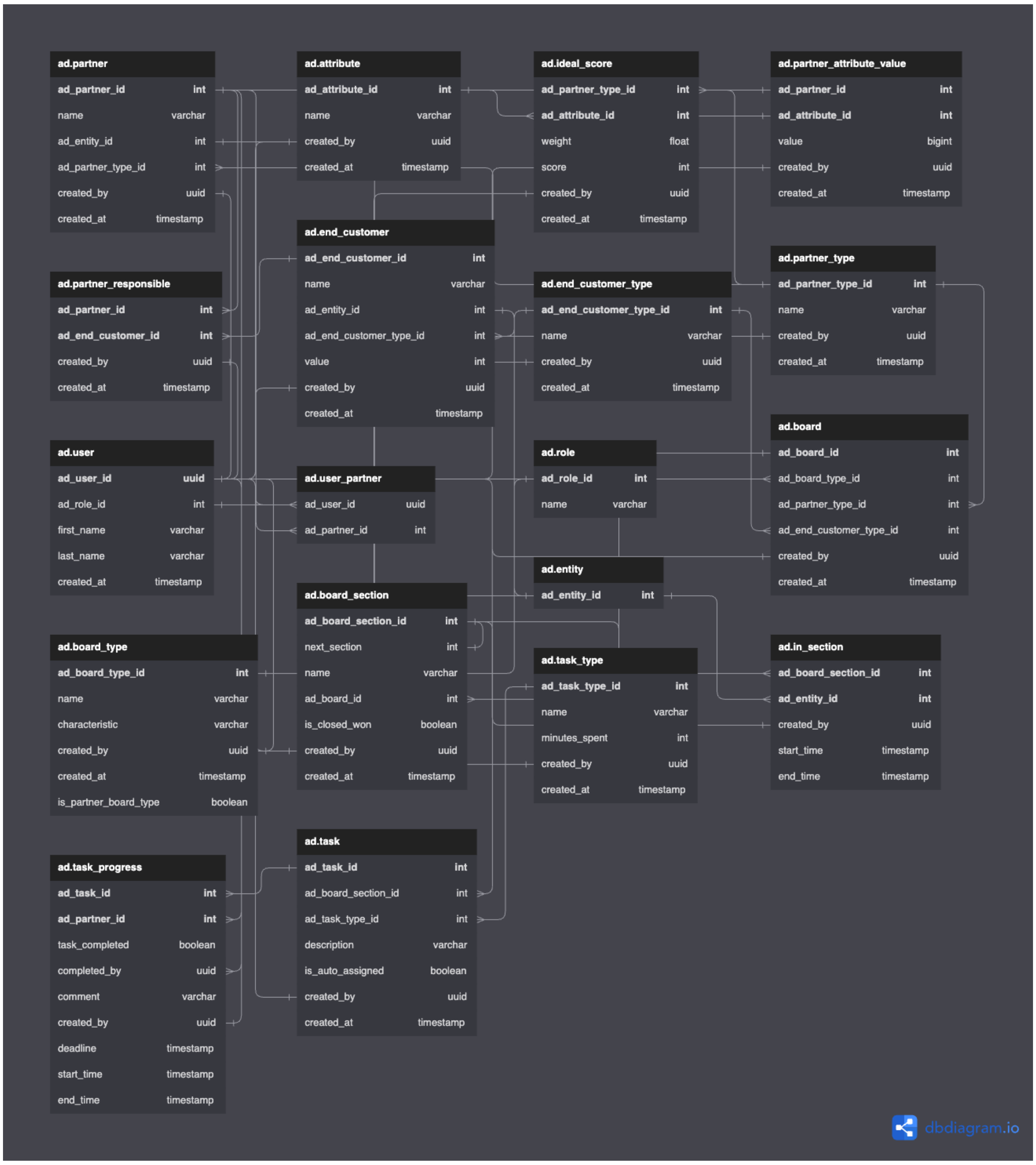


Figure D.1: Final Database Diagram

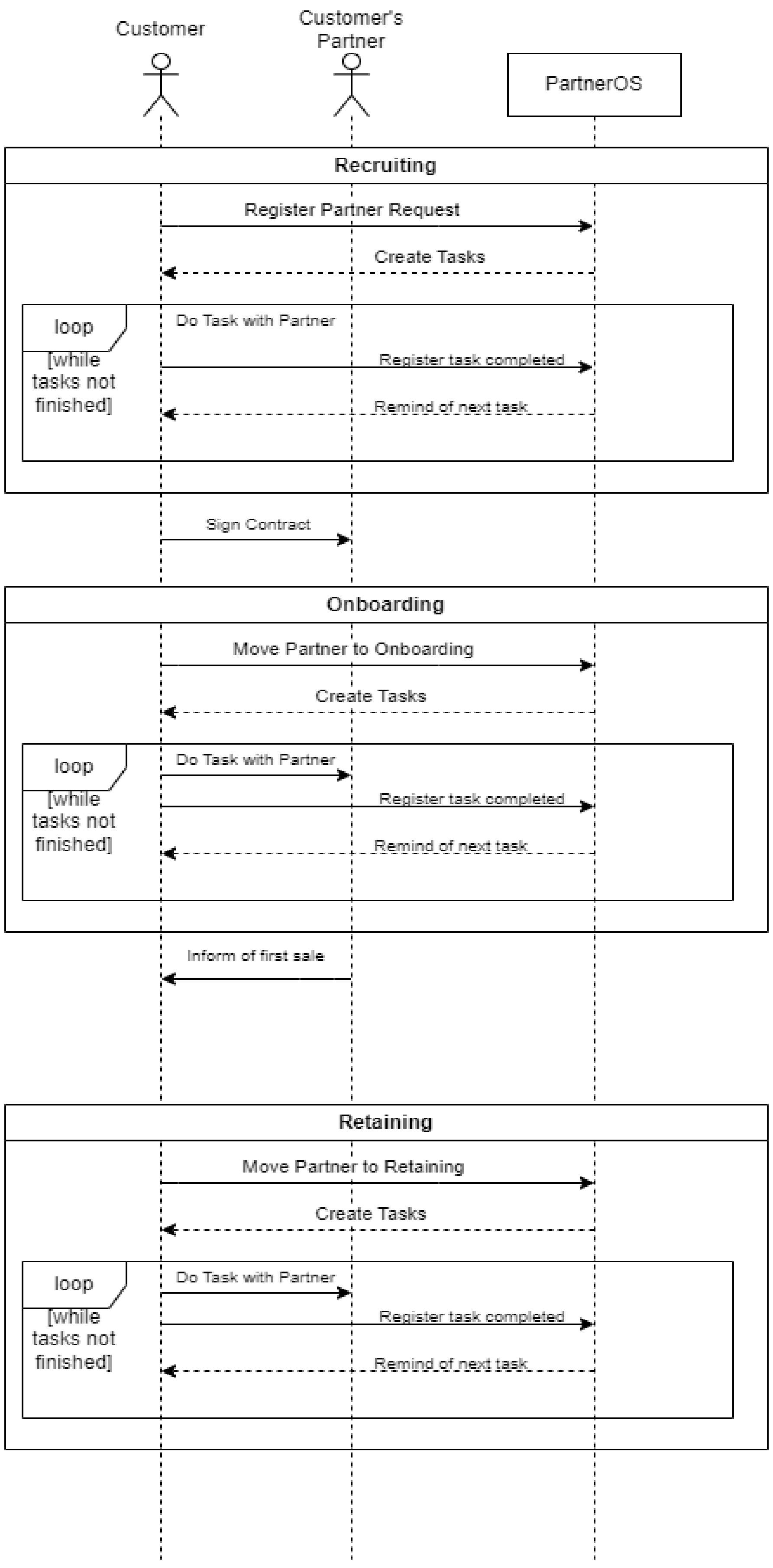


Figure D.2: PartnerOS Sequence Diagram

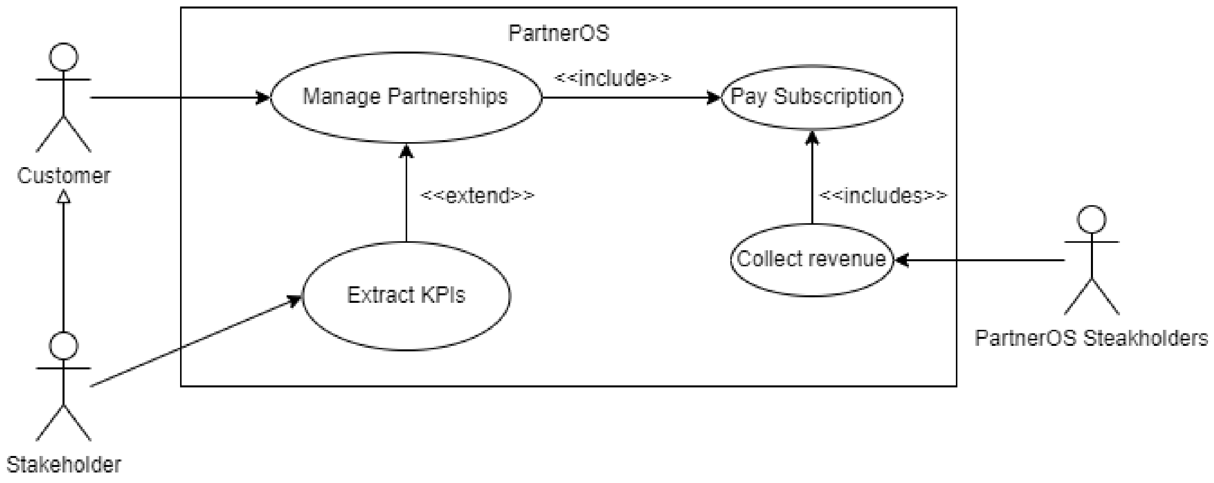
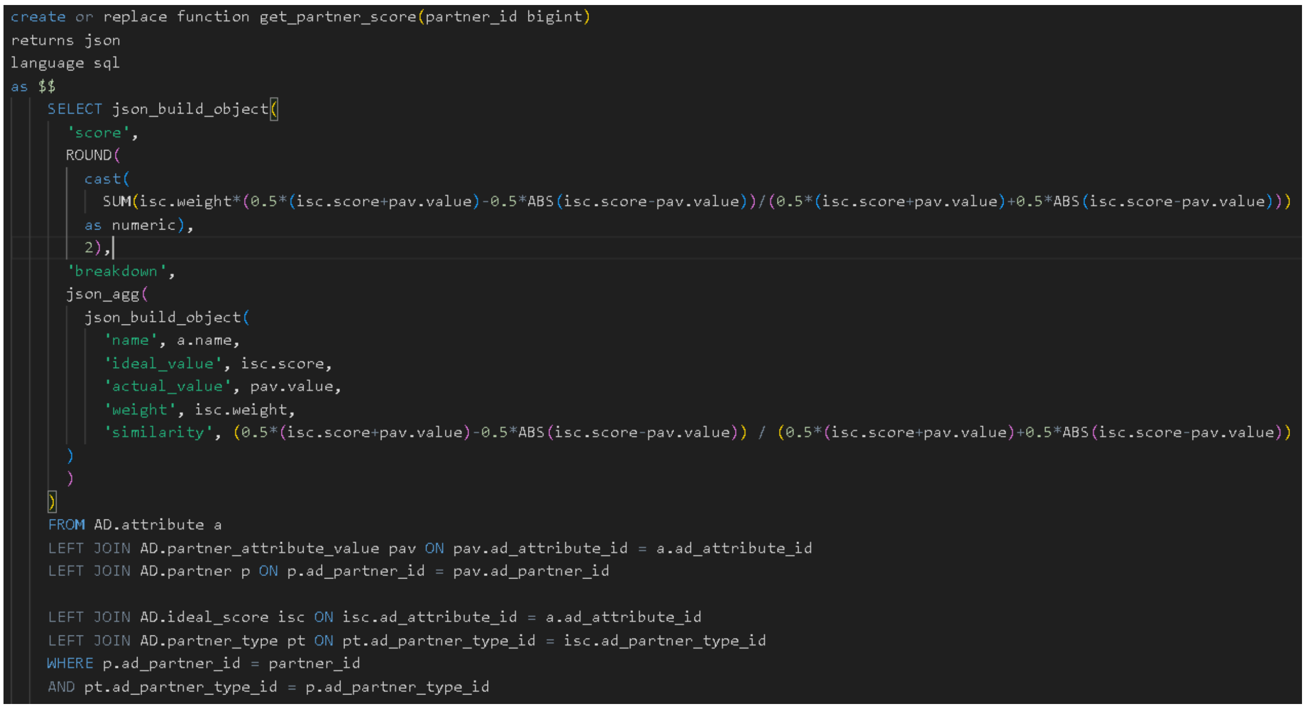
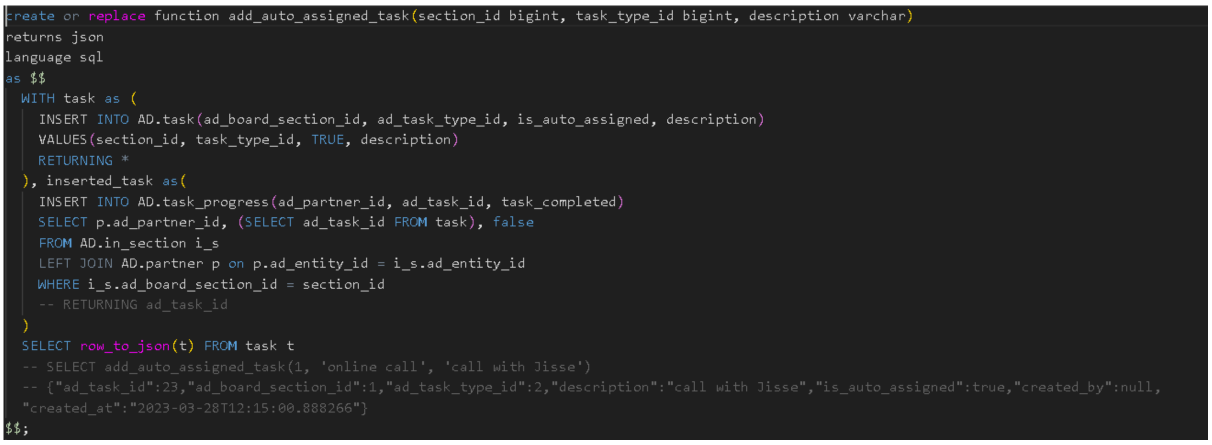


Figure D.3: PartnerOS Business Diagram



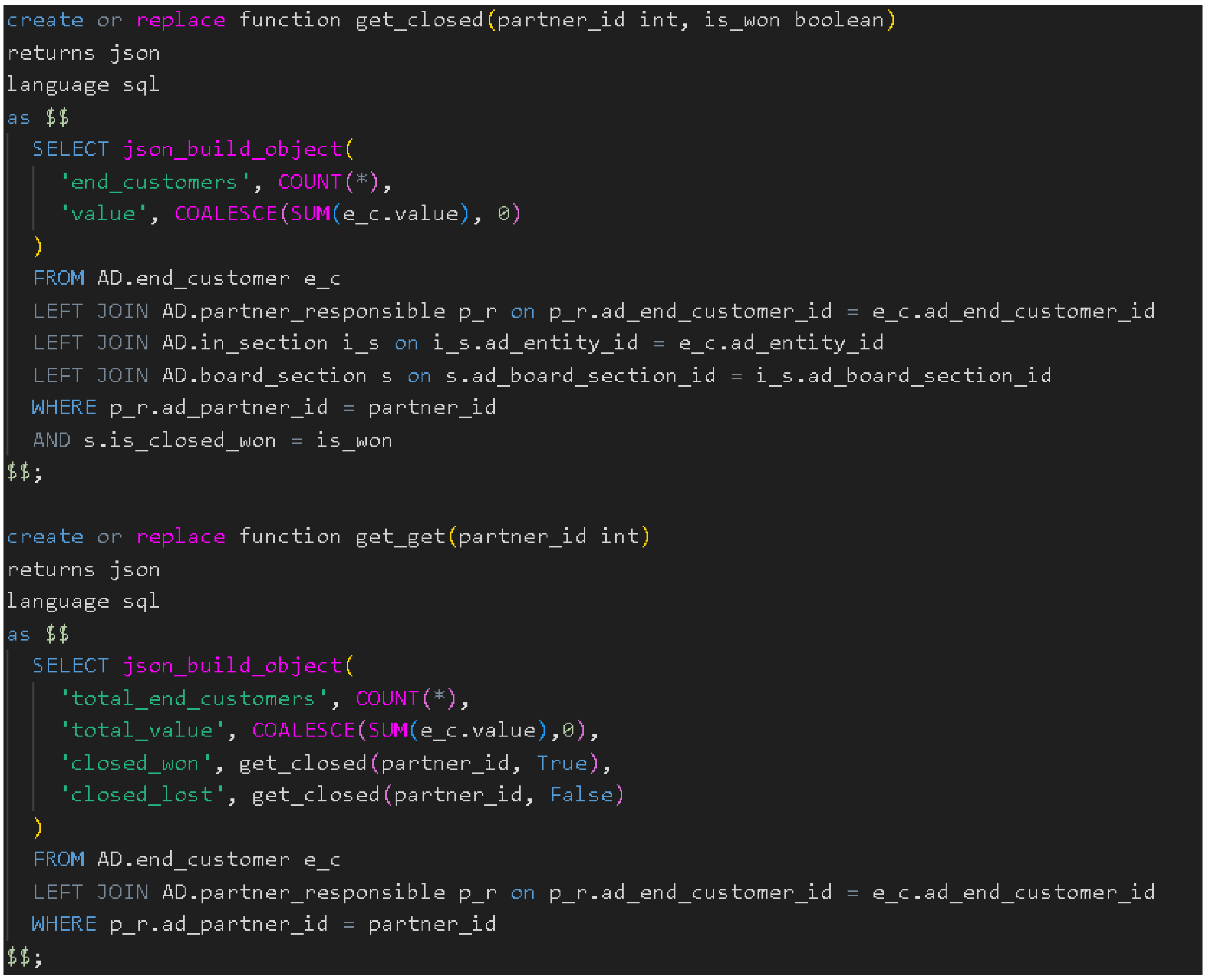
```
create or replace function get_partner_score(partner_id bigint)
returns json
language sql
as $$
    SELECT json_build_object(
      'score',
      ROUND(
        cast(
          SUM(isc.weight*(0.5*(isc.score+pav.value)-0.5*ABS(isc.score-pav.value))/(0.5*(isc.score+pav.value)+0.5*ABS(isc.score-pav.value)))
        as numeric),
        2),
      'breakdown',
      json_agg(
        json_build_object(
          'name', a.name,
          'ideal_value', isc.score,
          'actual_value', pav.value,
          'weight', isc.weight,
          'similarity', (0.5*(isc.score+pav.value)-0.5*ABS(isc.score-pav.value)) / (0.5*(isc.score+pav.value)+0.5*ABS(isc.score-pav.value))
      )
      )
    )
    FROM AD.attribute a
    LEFT JOIN AD.partner_attribute_value pav ON pav.ad_attribute_id = a.ad_attribute_id
    LEFT JOIN AD.partner p ON p.ad_partner_id = pav.ad_partner_id

    LEFT JOIN AD.ideal_score isc ON isc.ad_attribute_id = a.ad_attribute_id
    LEFT JOIN AD.partner_type pt ON pt.ad_partner_type_id = isc.ad_partner_type_id
    WHERE p.ad_partner_id = partner_id
    AND pt.ad_partner_type_id = p.ad_partner_type_id
```

Figure D.4: SQL function for calculating partner scores



```
create or replace function add_auto_assigned_task(section_id bigint, task_type_id bigint, description varchar)
returns json
language sql
as $$
  WITH task as (
    INSERT INTO AD.task(ad_board_section_id, ad_task_type_id, is_auto_assigned, description)
    VALUES(section_id, task_type_id, TRUE, description)
    RETURNING *
  ), inserted_task as(
    INSERT INTO AD.task_progress(ad_partner_id, ad_task_id, task_completed)
    SELECT p.ad_partner_id, (SELECT ad_task_id FROM task), false
    FROM AD.in_section i_s
    LEFT JOIN AD.partner p on p.ad_entity_id = i_s.ad_entity_id
    WHERE i_s.ad_board_section_id = section_id
    -- RETURNING ad_task_id
  )
  SELECT row_to_json(t) FROM task t
  -- SELECT add_auto_assigned_task(1, 'online call', 'call with Jisse')
  -- {"ad_task_id":23,"ad_board_section_id":1,"ad_task_type_id":2,"description":"call with Jisse","is_auto_assigned":true,"created_by":null,
  "created_at":"2023-03-28T12:15:00.888266"}
$$;
```

Figure D.5: SQL function for adding auto assigned tasks



```sql
create or replace function get_closed(partner_id int, is_won boolean)
returns json
language sql
as $$
  SELECT json_build_object(
    'end_customers', COUNT(*),
    'value', COALESCE(SUM(e_c.value), 0)
  )
  FROM AD.end_customer e_c
  LEFT JOIN AD.partner_responsible p_r on p_r.ad_end_customer_id = e_c.ad_end_customer_id
  LEFT JOIN AD.in_section i_s on i_s.ad_entity_id = e_c.ad_entity_id
  LEFT JOIN AD.board_section s on s.ad_board_section_id = i_s.ad_board_section_id
  WHERE p_r.ad_partner_id = partner_id
  AND s.is_closed_won = is_won
$$;

create or replace function get_get(partner_id int)
returns json
language sql
as $$
  SELECT json_build_object(
    'total_end_customers', COUNT(*),
    'total_value', COALESCE(SUM(e_c.value),0),
    'closed_won', get_closed(partner_id, True),
    'closed_lost', get_closed(partner_id, False)
  )
  FROM AD.end_customer e_c
  LEFT JOIN AD.partner_responsible p_r on p_r.ad_end_customer_id = e_c.ad_end_customer_id
  WHERE p_r.ad_partner_id = partner_id
$$;
```

Figure D.6: SQL function for the get of a partner.

Appendix E

Test #	Tested functionality	Test description	Test steps	Expected result	Pass/ fail
1	Login	Verify that a user can log in with valid credentials.	- Enter a valid username and password. - Click the "Login" button.	The user is successfully logged in and redirected to the dashboard.	Pass
2	Invalid Login	Verify that a user cannot log in with invalid credentials.	- Enter an invalid username and/or password. - Click the "Login" button.	An error message is displayed, indicating that the login attempt was unsuccessful.	Pass
3	Add Partner	Verify that a user can add a new partner.	- Navigate to Partners, then select + New Partner - Fill in the required partner details. - Click the Submit button	Notification that a new Partner was submitted is shown. New Partner should be visible in the Partners page.	Pass
4	Add Ideal Partner	Verify that a user can add a new ideal partner.	- Navigate to Settings, then select + New Ideal Partner - Fill in the required Ideal Partner details. - Click the Submit button	Notification that a new Ideal Partner was submitted is shown. The Partner Type should now reflect in the respective Partner's score.	Pass
5	Add Partner Type	Verify that a user can add a new partner type.	- Navigate to Settings, then select + New Partner Type - Fill in the Partner Type name - Click the + button	Notification that a new Ideal Partner Type was submitted is shown. The Partner Type should now be visible in any Partner Type selection forms.	Pass
6	Task Creation	Verify that a user can create a new task for a partner.	- Navigate to a Partners Task List by clicking on a respective Partner on the Board - Click the Add Task button - Fill in required details - Click on the green plus button	The input form for adding a task disappears and the task is visibly added on the list	Pass
7	Edit board	Verify that a user can edit a board.	- Navigate to Settings, then select Edit a Board - Choose the recruit Board Type to be edited - Add a new default task and click the green plus button	The changes should be reflected when navigating to the Recruit Board	Pass

			- Remove a section by clicking the red icon		
8	Task Compl etion	Verify that a user can mark a task as completed.	- Navigate to a Partners Task List by clicking on a respective Partner on the Board - Click on the yellow round icon at the beginning of a task	The round icon should now become green. In the Board the last activity on the respective Partner's card should say "just now"	Pass

Table E.1: Test cases for PartnerOS