Parking Garage

David Albrecht, TEKO Bern

May 2023

Contents

List of Figures

List of Tables

1 Project summery

2 Project initialization

2.1 Initial situation

The company ParkinTown currently manages five multi-storey parking lots at different locations on behalf of several customers. The IT system used for this purpose was developed by a former employee who carries out the necessary maintenance work on a contract basis. The system is getting on in years and will have to be replaced in the near future in view of the company's planned expansion plans. The managing director of ParkinTown has evaluated several IT solutions for the management of parking garages, but considers them all too complicated. He prefers a simple solution tailored to ParkinTown's needs. He is aware of the risks involved in developing it himself and therefore, before placing the order for the development of the entire application, he wants to have a prototype developed that simulates the operation of the parking garages. Based on the information in this document, a specification for the prototype must first be created. The information in this document must be checked for completeness and supplemented if necessary. Then a prototype of the IT system must be built on the basis of the specifications.

2.2 Situation analysis

The current software is already old and becoming harder to maintain. ParkingTown does not have the necessary development capabilities and is therefore searching for a external company that handles the development, maintenance and hosting of the new Software.

2.2.1 Product situation

The parking garage manager from ParkingTown is a tool that supports the companies employee in their daily tasks involving the management of parking garages on behalf of their clients.

The new Software should be tailored to the processes of ParkingTown and support the employees in their daily tasks.

ParkinTown is a company that manages multiple parking lots for various clients. Currently, the company uses an IT system that was developed by a former employee and requires maintenance work to keep it functioning. However, the system is outdated and needs to be replaced to accommodate the company's future expansion plans.

The managing director of ParkinTown has evaluated several IT solutions but has found them to be too complicated. Therefore, he desires a simple solution that is tailored to the company's specific needs. However, he is aware of the risks involved in developing an IT system in-house and wants to have a prototype developed to simulate the operation of the parking garages before placing an order for the development of the entire application.

The prototype must be developed based on a comprehensive specification that needs to be created. The specification must take into account all the requirements and operational needs of the parking garages managed by ParkinTown. It must also consider the different locations and customers of the company.

The prototype should be a functional system that simulates the operation of the parking garages. It should have all the necessary features and functionalities to manage the parking garages effectively. Additionally, it should be user-friendly and easy to use by the company's employees and customers.

The success of the prototype will depend on how well it meets the requirements of the company and its customers. It must be able to handle the volume of parking transactions effectively, and it should also be reliable and secure.

Overall, the development of a prototype for the IT system used in managing the parking garages is a critical step towards the success of ParkinTown's expansion plans. It will help the company to identify any gaps in the current system and ensure that the new system meets all the requirements of the company and its customers.

2.2.2 Competitive situation

In the parking garage management industry, there are several IT solutions available that offer similar services to ParkinTown. Some of ParkinTown's competitors include:

1. TIBA Parking Systems - TIBA Parking Systems is a global leader in parking solutions. They offer a comprehensive suite of parking solutions that include hardware, software, and services. TIBA's parking management system is highly customizable and can be tailored to meet the specific needs of each client. (tiba-parking, 2023)
2. Skidata - Skidata is another major player in the parking garage management industry. They offer a range of parking solutions, including software, hardware, and services. Skidata's solutions are designed to be highly efficient and can handle high volumes of parking transactions. (skidata, 2023)
3. Parkmobile - Parkmobile is a mobile parking solution that allows customers to pay for parking using their mobile devices. The company's solution is highly convenient for customers and can be integrated with other parking management systems. (parkmobile, 2023)
4. FlashParking - FlashParking is a cloud-based parking management solution that offers a range of features, including real-time parking availability, revenue management, and customer insights. FlashParking's solution is designed to be highly scalable and can be used by parking garages of all sizes. (flashparking, 2023)

Compared to these competitors, ParkinTown's current IT system is outdated and lacks some of the advanced features offered by these companies. However, ParkinTown's managing director believes that a simple and tailored solution is better suited to the company's needs.

Therefore, the success of ParkinTown's new IT system will depend on how well it meets the specific needs of the company and its customers. ParkinTown may have an advantage over its competitors in terms of the personalized service it can offer to its customers. Additionally, if the new system is user-friendly and easy to use, it may help ParkinTown differentiate itself from its competitors and attract more customers.

2.2.3 Sales situation

2.3 General conditions

ParkinTown requires a new parking garage management tool.

2.3.1 Process-related general conditions

1. Project Management: A project manager should be appointed to oversee the development of the new IT system for ParkinTown. The project manager should be responsible for ensuring that the project is completed on time, within budget, and to the required quality standards.
2. Stakeholder Management: All stakeholders, including the managing director of ParkinTown and the customers should be involved in the development process. Their input should be sought throughout the project to ensure that their needs are met.
3. Requirements Engineering: All requirements for the new IT system should be gathered and documented in detail before the development process begins. This will help ensure that the end product meets the desired functionality.
4. Design: A detailed design specification should be created based on the requirements gathered in the previous step. This design should include all aspects of the system, including user interfaces, database design, and system architecture.
5. Quality Assurance: A quality assurance process should be put in place to ensure that the new IT system is of high quality and meets the required standards. This should include testing and validation of the system to ensure that it works as intended.
6. Documentation: All aspects of the development process, including design decisions, testing results, and system specifications, should be documented. This will allow for future maintenance and updates to be carried out more easily.
7. Training and Support: Training and support should be provided to all users of the new IT system to ensure that they are able to use it effectively. This should include user manuals and training sessions.
8. Data Privacy and Security: The new IT system should be designed with data privacy and security in mind. Measures should be put in place to ensure that customer data is protected and that the system is secure from external threats. This may include encryption of sensitive data and regular security updates.

2.3.2 Product-related general conditions

Table 1: Product-related general conditions

ID	Trace From	Description
GCR-101	Requirements	The IT system must allow for flexible configuration of the parking garages, including the number of floors and parking spaces per floor. The system should also enable customization of parking rates and tariffs, which may vary depending on the time of day and day of the week.
GCR-102	Requirements	Each parking garage should have one entrance and one exit barrier. The system must be able to generate digital parking tickets for occasional users, which include the date and time of entry as well as the assigned parking space. Permanent tenants should enter a unique personal code at the entrance barrier, and their tickets should be validated before exit. All entry and exit times must be logged for billing and evaluation purposes.
GCR-103	Requirements	The IT system must support two categories of customers, casual users and permanent tenants. Permanent tenants should be assigned a fixed parking space and pay a monthly rent. If the rent is not paid on time, the customer should be blocked until payment is made. The system should also ensure a balanced distribution of parking spaces for occasional users.

ID	Trace From	Description
GCR-104	Requirements	The application should offer a graphical representation of the different floors per parking garage, showing the free and occupied parking spaces as well as those of the permanent tenants. The system should be able to generate reports and evaluations per parking garage and customer category over a defined time period at any time. The turnover of each parking lot should be calculated and displayed for a given month and annually, with individual months shown separately.
GCR-105	Requirements	The amount owed by occasional users should be calculated based on the length of stay and the applicable parking tariff. The system should automatically switch to a daily flat rate for stays longer than 24 hours, and the full amount should be charged for the elapsed days. After payment, a digital exit ticket should be issued to the customer.

2.4 Delimitations

1. Geographic scope: The system will be designed to meet the parking management needs of a single municipality or town, rather than multiple locations or regions.
2. Language and cultural context: The system will be developed in a specific language and cultural context, which may not be applicable or easily adaptable to other contexts or languages.
3. Hardware and software limitations: The system will be designed to operate on specific hardware and software platforms, and may not be compatible with other systems or technologies.
4. Timeframe: The project will focus on the development and implementation of the IT system, rather than the long-term maintenance or evolution of the system over time.
5. Budget constraints: The system will be designed to meet specific budgetary constraints, which may limit the scope or functionality of the system.
6. Parking garage configurations: The system will be designed to support a specific range of parking garage configurations, and may not be easily adaptable to other

configurations or designs.

7. User categories: The system will be designed to support two specific categories of users (casual users and permanent tenants), and may not be applicable to other categories of users.
8. Parking rates and tariffs: The system will be designed to support a specific range of parking rates and tariffs, which may not be applicable or adaptable to other contexts or markets.
9. Algorithm for parking space allocation: The system will use a specific algorithm to allocate parking spaces for occasional users, which may not be easily adaptable to other algorithms or methods of allocation.

2.5 Stakeholder analysis

2.5.1 Stakeholders

Table 2: List of Stakeholders and their abbreviation

ID	Stakeholder
CUPT	Customers (Casual Users and Permanent Tenants)
PGO	Parking Garage Owners
IDSA	IT Developers and System Administrators
MAR	Municipal Authorities and Regulators
FPSP	Financial Institutions and Payment Service Providers

2.6 Project planning

The Parking Garage project aims to develop a software system that automates the management of parking garages. The project requires a comprehensive approach that involves planning, analysis, design, implementation, verification, and documentation. The Gantt chart outlines the key activities involved in the project.

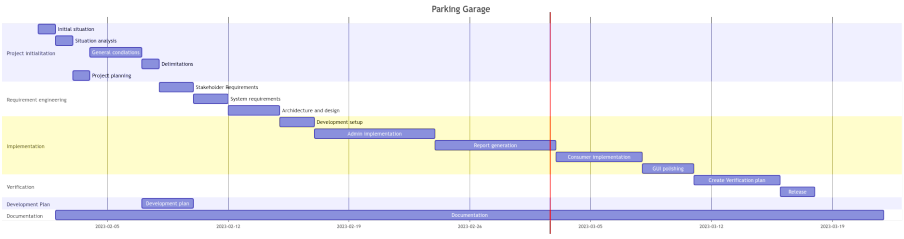

Figure 1: Gantt

2.6.1 Project Initiation

The project initiation phase involves defining the scope and objectives of the project, identifying stakeholders, and creating a project plan. The activities involved in this

phase are:

1. Initial Situation (1 day): This activity involves setting up the project, defining the project team, and identifying the project's objectives.
2. Situation Analysis (1 day): This activity involves analyzing the current parking system in place, including identifying the strengths and weaknesses of the current system.
3. General Conditions (3 days): This activity involves defining the constraints, assumptions, and dependencies of the project.
4. Delimitations (1 day): This activity involves setting the boundaries of the project, defining what is included and excluded.
5. Project Planning (1 day): This activity involves creating a detailed project plan, including timelines, resources, and budgets.

2.6.2 Requirement Engineering

The requirement engineering phase involves gathering, analyzing, and documenting the requirements for the software system. The activities involved in this phase are:

1. Stakeholder Requirements (2 days): This activity involves gathering the requirements of the stakeholders, including the customers, users, and other interested parties.
2. System Requirements (2 days): This activity involves translating the stakeholder requirements into system requirements.
3. Architecture and Design (3 days): This activity involves designing the architecture of the system, including defining the components, interfaces, and data flows.

2.6.3 Implementation

The implementation phase involves developing and testing the software system. The activities involved in this phase are:

1. Development Setup (2 days): This activity involves setting up the development environment, including installing the necessary software and hardware.
2. Admin Implementation (7 days): This activity involves developing the administrative features of the system, including user management and database management.
3. Report Generation (7 days): This activity involves developing the reporting features of the system, including generating reports on occupancy, revenue, and other metrics.
4. Consumer Implementation (5 days): This activity involves developing the consumer-facing features of the system, including user interfaces and payment processing.
5. GUI Polishing (3 days): This activity involves refining the user interface design to improve the user experience.

2.6.4 Verification

The verification phase involves testing the software system to ensure that it meets the requirements and is free of defects. The activities involved in this phase are:

1. Create Verification Plan (5 days): This activity involves creating a plan for testing the system, including defining the test cases, test data, and test procedures.
2. Release (2 days): This activity involves releasing the system to the stakeholders after it has been tested and approved.

2.6.5 Development Plan

The development plan phase involves creating a plan for the long-term development and maintenance of the software system. The activities involved in this phase are:

1. Development Plan (3 days): This activity involves creating a plan for the ongoing development of the system, including identifying future features and improvements.

2.6.6 Documentation

The documentation phase involves technical documentation, and other materials to support the use and maintenance of the software system. The activities involved in this phase are:

1. Documentation (49 days): This activity involves creating user manuals, technical documentation, and other materials to support the use and maintenance of the software system.

2.6.7 Conclusion

The Parking Garage project requires a comprehensive approach that involves planning, analysis, design, implementation, verification, and documentation. The Gantt chart provides a detailed overview of the key activities involved in each phase of the project.

3 Development plan

4 Stakeholder requirements

Table 3: List of Stakeholders and their abbreviation

ID	Stakeholder	Description
CUPT	Customers (Casual Users and Permanent Tenants)	They are the primary stakeholders who will use the parking system and benefit from its features. Casual users will expect a hassle-free parking experience with transparent billing and adequate parking space availability, while permanent tenants will expect a personalized parking solution with timely billing and automated payment options.
PGO	Parking Garage Owners	They are the ones who will own and operate the parking garages. They will expect a reliable IT system that can cater to the diverse needs of their customers and offer timely billing and revenue generation. They may also expect a user-friendly interface to manage parking space allocation and rates.
IDSA	IT Developers and System Administrators	They are responsible for developing and maintaining the parking system. They will expect a clear understanding of the stakeholders' requirements, access to the necessary resources, and a flexible system architecture to accommodate future changes in the parking industry.
MAR	Municipal Authorities and Regulators	They are responsible for ensuring compliance with local regulations, safety standards, and environmental norms. They will expect the parking system to follow the applicable rules and regulations, maintain accurate records of parking transactions, and provide transparent billing information to customers and regulators.
FPSP	Financial Institutions and Payment Service Providers	They are responsible for processing payments and ensuring the security of financial transactions. They will expect the parking system to integrate with their payment processing platforms, offer secure payment options, and maintain accurate records of financial transactions.

5 System requirements

5.1 Functional requirements

Table 4: Functional requirements

ID	Trace from	Description
R101	PGO, IDSA	The IT system must allow for the configuration of parking garages individually, with the number of floors and parking spaces per floor being freely definable.
R102	PGO, IDSA	The number of parking spaces per floor may vary and can be set independently of other floors.
R103	PGO, IDSA	Each parking garage must have exactly one entrance and one exit barrier.
R104	CUPT, PGO	There are two categories of customers for each parking garage - Casual users and permanent tenants.
R105	CUPT, PGO	Permanent tenants are assigned a fixed parking space and pay a monthly rent for it.
R106	CUPT, PGO	If a permanent tenant's rent is not paid on the 15th of each month, their access is blocked until payment is made.
R107	CUPT	Occasional users can generate a digital parking ticket by pressing a button on the entrance barrier, which contains at least the date and time of entry, floor number, and assigned parking space number.
R108	CUPT	Permanent tenants must enter a unique personal code at the entrance barrier.
R109	CUPT, PGO	The date and time of entry are registered for both permanent tenants and occasional users for internal billing purposes.
R110	CUPT	The parking ticket for occasional users is validated before exit.
R111	CUPT	The date and time of exit are recorded.
R112	CUPT	The amount owed is calculated for occasional users based on the length of stay and the parking tariff.
R113	CUPT	When leaving the parking garage, either the unique personal code (permanent tenant) or the ticket number (occasional user) is read or entered at the exit barrier.
R114	CUPT, PGO, MAR	All entry and exit times are logged for both permanent tenants and occasional users.
R115	PGO, IDSA	The system must be able to generate an evaluation per parking garage and user category over a defined time period at any time.
R116	PGO, IDSA	Parking rates must be flexible and vary depending on the time of day.

ID	Trace from	Description
R117	PGO, IDSA	A separate tariff applies for weekends and public holidays.
R118	PGO, IDSA	The parking tariff calculation is made on a quarter-hour basis, with the tariff at the beginning of the respective quarter-hour applying for the entire quarter-hour.
R119	CUPT	Any quarter of an hour that has elapsed will be charged in full.
R120	CUPT	If the parking time is longer than 24 hours, the billing will automatically switch to daily flat rate, charging the full amount for the days that have elapsed.
R121	CUPT	A graphical representation of the different floors per parking garage must be available on the application, showing the free and occupied parking spaces, as well as those of the permanent tenants.
R122	CUPT, IDSA	The allocation of parking spaces for occasional users must be based on an algorithm, which aims at a balanced distribution.
R123	PGO	The application must be able to calculate and display the turnover of each parking lot (divided by customer category) for a given month at any time.
R124	PGO	The application must also be able to calculate and display the annual turnover, showing the individual months separately.

5.2 Non-Functional requirements

Table 5: Non-Functional requirements

ID	Trace from	Description
NFR101	CUPT, PGO, IDSA	The system must be easy to use and navigate for both occasional users and permanent tenants.
NFR102	CUPT, PGO, IDSA	The system must be secure, ensuring that personal data is protected.
NFR103	CUPT, PGO, IDSA	The system must be reliable and available at all times, with minimal downtime for maintenance.
NFR104	CUPT, PGO, IDSA	The application must have a responsive design and work on multiple devices and platforms.
NFR105	CUPT, PGO, IDSA	The system must be scalable and able to accommodate future growth.
NFR106	CUPT, PGO, IDSA, MAR, FPSP	The IT system must be compliant with relevant laws and regulations, including data privacy and protection regulations.

ID	Trace from	Description
NFR107	CUPT, PGO, IDSA	The application must have a user-friendly interface with clear instructions on how to use the system and its features.
NFR108	IDSA	The IT system must be compatible with existing software and hardware systems used by the company.
NFR109	IDSA	The system must have a backup and recovery plan in case of system failure or data loss.

6 System architecture and design

The new parking garage software for ParkingTown is a web based sveltekit application. (svelte, 2023a) The Application wil be delivered as a state of the art OCI compliant container image. (OCI, 2023) Database access is handled by the ORM (wikipedia, 2023a) prisma (prisma, 2023)

prisma allows a wide variety of SQL based database providers.

6.1 Svelte / Sveltekit

6.1.1 What is Svelte

Svelte is a front-end JavaScript framework that allows developers to build web applications using reactive components. It was created by Rich Harris in 2016 and has since gained popularity due to its unique approach to building web applications. Unlike other front-end frameworks that rely on a virtual DOM, Svelte compiles your code into highly optimized and efficient JavaScript code at build time.

In Svelte, components are defined using a syntax similar to HTML and CSS, making it easy for developers to create reusable and reactive UI elements. Svelte's compiler then generates efficient code that updates the DOM directly, resulting in fast and smooth user experiences.

Svelte provides several built-in features that make it easier for developers to build web applications quickly. These features include:

- Reactivity: Svelte provides a reactive system that allows components to update automatically when data changes.
- Scoped CSS: Svelte allows developers to define CSS styles that are scoped to a specific component, reducing the risk of CSS conflicts.
- Animations: Svelte provides an easy-to-use API for adding animations to your components.
- Event handling: Svelte provides an intuitive syntax for handling events and user interactions.

(wikipedia, 2023c)

6.1.2 What is SvelteKit

SvelteKit is a high-performance web application framework built on top of Svelte. It was released in 2021 and provides several built-in features that make it easier to build server-rendered web applications. SvelteKit is designed to create highly optimized web applications with minimal overhead.

SvelteKit provides several key features that make it a compelling choice for web application development:

- Server-side rendering: SvelteKit allows developers to render components on the server, reducing the initial load time of your web application.
- Automatic code splitting: SvelteKit automatically splits your code into smaller chunks, reducing the amount of code that needs to be downloaded by the client.
- File-based routing: SvelteKit allows you to define routes for your web application using a file-based routing system, making it easier to organize your code and routes.
- API routes: SvelteKit provides an easy-to-use API for defining server-side routes and interacting with databases and other services.
- Integration with other tools: SvelteKit integrates with popular tools like TypeScript, GraphQL, and Tailwind CSS, making it easier to build complex web applications.

SvelteKit builds on the strengths of Svelte and provides a powerful and efficient framework for building server-rendered web applications.

(svelte, 2023b)

6.1.3 Why Sveltekit instead of other fullstack javascript frameworks

One of the main reasons sveltekit was chosen for this project is, that i wanted to learn this technology for a long time. This project was a perfect learning opportunity to get hands on experience with a new up and coming javascript framework.

According to the biggest javascript survey, Svelte / Sveltekit is amongst the most loved frameworks. (StateOfJS, 2023)

Some of the reasons why sveltekit is so loved are:

1. **Performance** One of the primary benefits of using SvelteKit is its performance. SvelteKit is designed to create highly optimized web applications with minimal overhead. It does this by using a unique feature called the “Svelte Compiler” that compiles your code during the build process, resulting in highly optimized and lightweight code.

2. **Easy to learn and use** SvelteKit is relatively easy to learn and use, especially if you are already familiar with modern JavaScript frameworks like React or Vue. SvelteKit provides an intuitive and straightforward syntax that makes it easy to write and understand code.

3. **Built-in features and integrations** SvelteKit comes with several built-in features and integrations, making it easier to build web applications quickly. It includes features such as server-side rendering, automatic code splitting, file-based routing, and more. Additionally, it integrates with popular tools like Tailwind CSS, TypeScript, and GraphQL.
4. **Flexibility** SvelteKit is highly flexible and customizable, making it an excellent choice for a wide range of projects. Whether you are building a simple blog or a complex e-commerce website, SvelteKit can adapt to your needs and requirements.
5. **Great community support** SvelteKit has a great community that is continually improving and contributing to the framework. The community provides excellent documentation, resources, and support to help you get started and overcome any challenges you may encounter.

In conclusion, SvelteKit is an excellent choice for web development projects, providing high performance, ease of use, built-in features and integrations, flexibility, and great community support.

6.2 Containers

Containers are a method of software virtualization that allows you to package an application and all its dependencies into a single lightweight package, which can be deployed and run consistently across different computing environments.

6.2.1 What are Containers

To understand containers, it's helpful to first understand traditional virtualization, which involves running multiple operating systems on a single physical machine. With virtualization, each operating system has its own set of resources (like memory and CPU), and the hypervisor (a layer of software that sits between the operating system and the physical hardware) manages the allocation of those resources.

Containers, on the other hand, are a form of operating-system-level virtualization, which means they share the same kernel as the host operating system. This makes them much more lightweight and portable than traditional virtual machines, since they don't require a separate operating system to be installed and managed for each container.

(docker, 2023b)

Instead, containers use a combination of file system isolation and process isolation to provide a separate environment for each application. Each container has its own file system that is isolated from the host file system, and each container runs in its own process space, separate from other containers on the same host.

Containers can be created from a base image (like an operating system or a programming language runtime) and customized with additional software and configurations

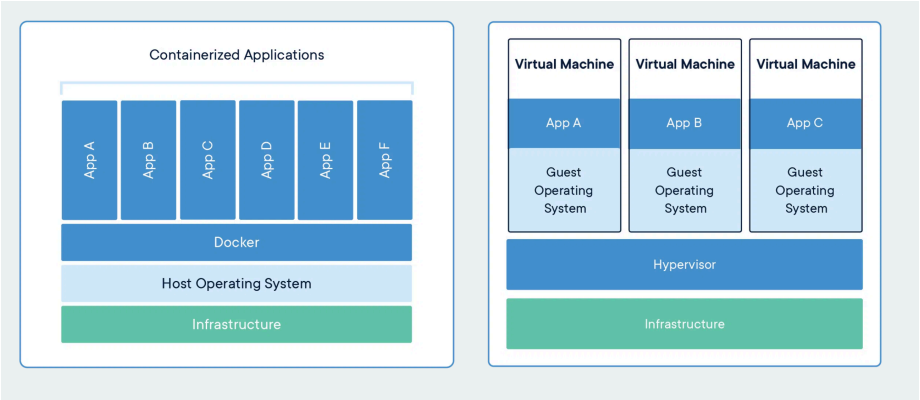

Figure 2: Comparing Docker to VM‘s

specific to your application. Once the container is built, it can be deployed to any host that supports the container runtime (like Docker or Kubernetes).

Some of the benefits of using containers include:

- **Portability:** Containers can be run on any host that supports the container runtime, making them a great option for deploying applications across different environments (like development, staging, and production).
- **Consistency:** Since each container contains all the dependencies required to run the application, you can be sure that the application will run consistently across different hosts.
- **Scalability:** Containers can be easily scaled up or down to meet changing demand, since they are lightweight and don’t require a lot of resources.
- **Isolation:** Since each container runs in its own process space, containers provide a degree of isolation between different applications, reducing the risk of conflicts or security vulnerabilities.

(wikipedia, 2023b)

6.2.2 OCI Containers

The OCI is a collaborative project hosted by the Linux Foundation, with the goal of creating open standards for container formats and runtimes. The OCI was founded in 2015 by Docker and other industry leaders, in response to a growing need for interoperability and standardization in the container ecosystem.

The OCI is focused on two main specifications: the Image Format Specification and the Runtime Specification. These specifications define how containers should be packaged, distributed, and run, and are designed to be vendor-neutral and interoperable.

The Image Format Specification defines a standard format for container images, which is used to package an application and its dependencies into a single distributable artifact.

This specification includes details like how images should be structured, how layers should be defined and represented, and how metadata should be included.

The Runtime Specification defines a standard interface between the container image and the host operating system, which allows the container to be run in a consistent and portable way across different environments. This specification includes details like how the container should be started and stopped, how it should interact with the host filesystem and network, and how resources like CPU and memory should be managed.

Both specifications are designed to be flexible and extensible, allowing for innovation and customization while maintaining interoperability. The OCI also includes tools and utilities for working with container images and runtimes, like the OCI Image Tool and the OCI Runtime Tool.

One of the main goals of the OCI is to foster collaboration and interoperability in the container ecosystem. By creating open standards that are supported by a broad community of vendors and developers, the OCI aims to reduce fragmentation and increase adoption of containers as a standard way to package, distribute, and run applications.

(Foundation, 2023)

6.2.3 Docker

Docker is a containerization platform that allows you to package an application and its dependencies into a single portable unit called a container. Docker is built on top of the Linux kernel's containerization features, like cgroups and namespaces, and provides a simple and user-friendly interface for working with containers.

Docker containers are OCI compliant and therefore can run on any Container runtime that supports OCI Images like Kubernetes or Openshift.

Docker is the facto standard for building and running containers on a development machine. Docker is available for Linux (native) or as a desktop application called Docker Desktop.

Docker Desktop is a desktop application for Windows and macOS that provides an easy-to-use interface for working with containers. It includes all the tools and services needed to develop, build, and deploy containerized applications on a local machine.

(docker, 2023a)

6.2.3.1 When to use Docker

Docker in present is the most widely used container runtime / platform. The Docker Platform is easy to use and comes with a feature rich GUI. Docker is mostly used in development setups. Docker was one of the first implementations of linux cgroup containers under windows and contributed with this massively to the success of containerization.

With the majority of containers and container orchestration tools like kubernetes rising in popularity docker is no longer the industry leader for building, packaging and running production grade containers.

Some of the reasons are:

- **Kubernetes:** kubernetes drooped the docker runtime support in kubernetes version v1.24 (kubernetes, 2023a) (Docker containers still run on kubernetes since they are OCI compliant)
- **Develope close to production:** With most container running in a Kubernetes Cluster developers started using local kubernetes clusters like mininkube (kubernetes, 2023b) or rancher desktop (SUSE, 2023) as development environments. This enables the developers to be as close to the production infrastructure as possible.
- **Docker in Docker:** In order to build docker images from inside docker (for example CI pipeline) the docker demon is required to run as root. This is a potential security risk for the toolchain and its recommended to use different tools like kaniko (Google, 2023) to build your production container images.

6.3 Prisma

Prisma is an open-source tool that simplifies database access for developers. It provides a type-safe and scalable ORM (Object-Relational Mapping) layer that enables developers to interact with their database in a more intuitive and efficient manner.

With Prisma, developers can define their data models in a declarative schema language called Prisma Schema. This schema defines the structure of the database and the relationships between tables. Prisma Schema also supports various data types, such as strings, numbers, booleans, and timestamps.

Prisma generates a set of CRUD (Create, Read, Update, Delete) operations based on the defined schema. These operations are generated as TypeScript or JavaScript functions that can be called directly from the application code. This removes the need to write SQL queries manually, and also makes the codebase more maintainable and readable.

Prisma is designed to work with multiple database engines such as MySQL, PostgreSQL, and MongoDB and other SQL based databases. With the abstraction provided by Prisma switching between different databases is achievable without having to rewrite their application code.

Prisma also provides advanced features such as data validation, query optimization, and transaction management. These features ensure that the data stored in the database is consistent and secure.

In addition to its core features, Prisma also has a growing ecosystem of plugins and integrations that make it even more powerful. For example, Prisma Migrate enables developers to manage database schema changes in a version-controlled manner, while Prisma Client allows for real-time data synchronization between the client and server.

(prisma, 2023)

6.3.1 ORM

An ORM (Object-Relational Mapping) is a technique used to map data between an application's object-oriented programming language and a relational database. In other words, an ORM acts as a bridge between the application and the database, providing a layer of abstraction that makes it easier for developers to interact with the database.

The basic idea behind an ORM is to represent database tables as classes in the programming language used by the application. Each row in the table is then represented as an instance of the class, and the columns in the table are represented as properties of the class. This allows developers to interact with the database using familiar object-oriented programming concepts, such as objects, methods, and properties.

One of the primary benefits of using an ORM is that it can simplify the process of interacting with the database. Rather than writing SQL queries directly, developers can use the ORM to perform CRUD (creating, reading, updating, and deleting) database operations. These operations are typically performed using methods provided by the ORM, rather than by writing SQL queries manually.

Another benefit of using an ORM is that it can make the application code more maintainable and readable. By abstracting away the details of how the data is stored in the database, the ORM can make the application code more focused on the business logic of the application, rather than on the details of how the data is stored.

(wikipedia, 2023a)

6.3.2 PostgreSQL

PostgreSQL is an open-source object-relational database management system (ORDBMS) that is designed to store, manage, and manipulate large amounts of data.

PostgreSQL is known for its ability to handle complex transactions and data structures, making it ideal for use in applications that require high levels of reliability and concurrency. It uses a technique called multi-version concurrency control (MVCC) to allow multiple users to access the database simultaneously without conflicts. This technique ensures that data consistency is maintained even when multiple users are making changes to the database concurrently. MVCC allows PostgreSQL to handle complex transactions and large amounts of data with ease, making it suitable for a wide range of applications.

One of the main advantages of PostgreSQL is its extensibility. It provides support for user-defined functions, triggers, and custom data types, allowing developers to extend the database to meet their specific needs.

PostgreSQL's support for advanced indexing and querying is another key strength. It provides a variety of indexing options, including B-tree, hash, GiST, and SP-GiST, as well as support for full-text search and geospatial queries. PostgreSQL's indexing and querying capabilities are highly optimized, allowing for efficient retrieval and analysis of large amounts of data. This makes it suitable for use in applications that require complex querying and data analysis, such as business intelligence and data warehousing.

PostgreSQL is also known for its focus on data security and reliability. It provides robust authentication and encryption mechanisms, as well as support for backup and recovery operations. PostgreSQL's focus on data security and reliability makes it a popular choice for applications that require high levels of data security and availability, such as healthcare systems and government agencies.

These are some of the reasons why PostgreSQL is used as a database, running inside a container.

(Postgres, 2023)

6.4 Context diagram

6.5 Sequence diagram

6.6 Class diagram

6.7 ERD

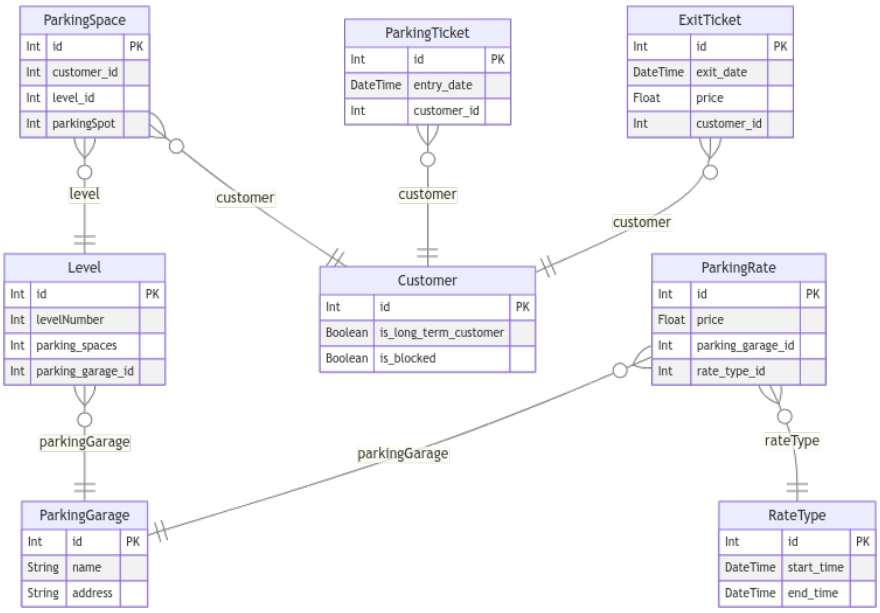

Figure 3: ERD

7 Implementation

7.1 Documentation workflow

In the Company i work we create SaMD (Software as Medical Device). Software that classifies as medical device needs to be developed with specific regulations in mind and requires to follow specific process. Present we use Microsoft Word to write the required documentation but this raises a few challenges.

1. Word is not as easy to generate from within a pipeline (For example openAPI specification)
2. Word is not as easy to Lint inside a pipeline
3. Including Code or Project specific references in good looking format is not possible
4. Word is slow and the formatting is tedious

With this Project i wanted to try if its possible to write large project documentation using Markdown and export the written documentation in a professional looking PDF. In order to achieve this the following software is used:

1. **Pandoc:** used to convert the markdown into LaTex format and print a PDF (Pandoc, 2023)
2. **LaTeX:** Pandoc rquires LaTeX to be installed in order to generate the PDF (LaTeX, 2023)

7.1.1 Folder Structure

In order to generate an academic looking paper the following folder structure is used

```
latex-pdf
 |assets
 | -citation-style.csl
 |-build
 | -output_print.pdf
 |-content
 | -images
 | -bibliography.bib
 | -index.md
 |-layouts
 | -print.tex
 |-Makefile
 |-build.sh
```

- **assets:** Used for third party styles, in this case how to style the citations
- **build:** The output directory where the PDF will be generated to
- **content** Holds the full markdown file, all used figures and the bibliography
- **layouts:** Pandoc can convert the markdown into different output formats, in the layout folder could the different formats specify some individual layouts

7.1.2 Buildscript

The script “build.sh” is handling the generation of the PDF file.

```
#!/bin/bash
CONTENTDIR="content"
BUILDDIR="build"
FILENAME="index"
ASSETSDIR="assets"

download_csl() {
    mkdir "${ASSETSDIR}" -p
    wget -O "${ASSETSDIR}/citation-style.csl" \
        "https://raw.githubusercontent.com/citation-style-language/styles/master/harvard-ang
}

pdf_print() {
    mkdir "${BUILDDIR}"
    echo "Creating pdf-print output"
    pandoc "${CONTENTDIR}/${FILENAME}.md" \
        --resource-path="${CONTENTDIR}" \
        --citeproc \
        --csl="${ASSETSDIR}/citation-style.csl" \
        --from="markdown+tex_math_single_backslash+tex_math_dollars+raw_tex" \
        --to="latex" \
        --output="${BUILDDIR}/output_print.pdf" \
        --pdf-engine="xelatex" \
        --include-in-header="layouts/print.tex"
}
# Example: `./build.sh pdf_print`
$*
```

The Script defines four Variables:

1. CONTENTDIR: the path to the directory that holds all figures and the markdown file
2. BUILDDIR: the path to the output directory of this script
3. FILENAME: the filename of the markdown file
4. ASSETSDIR: the path to the asset directory that holds the csl

To run the script run the following command from the latex-pdf directory:

```
chmod +x build.sh #allow the script to be executed
./build.sh pdf_print #run the script
```

7.2 Development environment

7.3 Algorithm

8 Verification and validation

9 Conclusion

10 Glossary

References

docker, 2023a. *Docker*. [online] Available at: <https://www.docker.com/> [Accessed 2 March 2023].

docker, 2023b. *What are containers*. [online] Available at: <https://www.docker.com/resources/what-container/> [Accessed 4 March 2023].

flashparking, 2023. *Why flashparking*. [online] Available at: <https://www.flashparking.com/why-flashparking/> [Accessed 2 March 2023].

Foundation, T.L., 2023. *Open container initiative*. [online] Available at: <https://opencontainers.org> [Accessed 4 March 2023].

Google, 2023. *Kaniko*. [online] Available at: <https://github.com/GoogleContainerTools/kaniko> [Accessed 4 March 2023].

kubernetes, 2023a. *Docker runtime*. [online] Available at: <https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/#so-why-the-confusion-and-what-is-everyone-freaking-out-about> [Accessed 4 March 2023].

kubernetes, 2023b. *Minikube*. [online] Available at: <https://minikube.sigs.k8s.io/docs/> [Accessed 4 March 2023].

LaTeX, 2023. *Latex*. [online] Available at: <https://www.latex-project.org/> [Accessed 4 March 2023].

OCI, 2023. *Open container initiative*. [online] Available at: <https://github.com/opencontainers/image-spec> [Accessed 2 March 2023].

Pandoc, 2023. *Pandoc*. [online] Available at: <https://pandoc.org/> [Accessed 4 March 2023].

parkmobile, 2023. *Parkmobile parking solutions*. [online] Available at: <https://parkmobile.io/parking-solutions/> [Accessed 2 March 2023].

Postgres, 2023. *Postgres*. [online] Available at: <https://www.postgresql.org/> [Accessed 4 March 2023].

prisma, 2023. *Prisma*. [online] Available at: <https://www.prisma.io/> [Accessed 2 March 2023].

skidata, 2023. *Skidata parking solution*. [online] Available at: <https://www.skidata.com/de-ch/loesungen/parken-mobilitaet/parkleitsystem-digitale-beschilderung> [Accessed 2 March 2023].

StateOfJS, 2023. *State of js*. [online] Available at: <https://2022.stateofjs.com/en-US/libraries/#tier_list> [Accessed 4 March 2023].

SUSE, 2023. *Rancher desktop*. [online] Available at: <https://rancherdesktop.io/>

[Accessed 4 March 2023].
svelte, 2023a. *Svelte*. [online] Available at: <https://svelte.dev/> [Accessed 2 March 2023].
svelte, 2023b. *Svelte kit*. [online] Available at: <https://kit.svelte.dev/docs/introduction> [Accessed 4 March 2023].
tibaparking, 2023. *About tibaparking*. [online] Available at: <https://tibaparking.com> [Accessed 2 March 2023].
wikipedia, 2023a. *Object relational mapping*. [online] Available at: <https://en.wikipedia.org/wiki/Object-relational_mapping> [Accessed 2 March 2023].
wikipedia, 2023b. *Operating system level virtualization*. [online] Available at: <https://en.wikipedia.org/wiki/OS-level_virtualization> [Accessed 4 March 2023].
wikipedia, 2023c. *Svelte*. [online] Available at: <https://en.wikipedia.org/wiki/Svelte> [Accessed 4 March 2023].