**Thy Khuu**

**House Services Management System**

**Thesis**

**CENTRIA UNIVERSITY OF APPLIED SCIENCES**

**Bachelor of Engineering, Information Technology**

**December 2022**

**ABSTRACT**

|  |  |  |
| --- | --- | --- |
| Centria University  of Applied Sciences | Date  December 2022 | Author  Thy Khuu |
| Degree programme  Bachelor of Engineering, Information Technology | | |
| Name of thesis  House Services Management System | | |
| Centria supervisor  First name Last name | | Pages  7 + 2 |
| Instructor representing commissioning institution or company  First name Last name | | |
| Since the Covid epidemic becomes a global crisis and many countries in the world are facing with hard situation about moving away, work and study on campus. The domestic migration case has been detected that many regions become a red warning due to the critical spreading of disease. People tend to look for a peaceful province and area where the Covid situation has less become risk to protect themselves and their family. As the sequence of problem, the migration to safe region from risk region that makes a pressure under rental housing service at these green regions.  There are efforts from landlord in cities where a huge number of migrations is massively moving in, are trying to expand their rental housing service by upgrading building infrastructure and constructing new apartment to meet the significant increase of people looking for housing service temporarily. Dealing with the big statistics every day is truly challenged to the service while the need of renting house service is suddenly going on increased.  Effective solution for the demand of service management is important to meet the urgent needs. Therefore, the research is aims at exploring the design of application to manage the housing services based on the current situation and the wish of landlord about optimal ways for their service manage-ment. The research not only introduces about the necessary of efficient management system, but also involves the system design based on the techniques and subjects of software engineering. | | |
|  | | |
| Key words  House Management, Homecare, Software Architecture, System Design, Modelling, User Interface. | | |

**CONCEPT DEFINITIONS**

**UML**

Unified Modelling Language

**UI**

User Interface

**UX**

User Experience

**IOS**

Iphone Operating System

**ABSTRACT**

**CONCEPT DEFINITIONS**

**CONTENTS**

[1 INTRODUCTION 1](#_Toc124101266)

[2 Literature Overview 2](#_Toc124101267)

[2.1 Objective 2](#_Toc124101268)

[2.2 Scope of the thesis 2](#_Toc124101269)

[2.3 Structure of the thesis 3](#_Toc124101270)

[3 Requirements analysis 4](#_Toc124101271)

[3.1 Research 4](#_Toc124101272)

[3.2 Designing 4](#_Toc124101273)

[3.3 Development 5](#_Toc124101274)

[3.3.1 Front-end development 6](#_Toc124101275)

[3.3.2 Back-end development 6](#_Toc124101276)

[3.4 Testing 6](#_Toc124101277)

[4 Tools and technologies for building web application 8](#_Toc124101278)

[4.1 Deno 8](#_Toc124101279)

[4.2 VSCode 8](#_Toc124101280)

[4.3 Elephant PostgreSQL 9](#_Toc124101281)

[4.4 Google Chrome 9](#_Toc124101282)

[4.5 Docker and Docker Compose 10](#_Toc124101283)

[4.6 Heroku Deployment 14](#_Toc124101284)

[5 Web development components 15](#_Toc124101285)

[5.1 Monolithic Architecture 15](#_Toc124101286)

[5.2 HyperText Makeup Language 18](#_Toc124101287)

[5.3 Cascading Style Sheets 18](#_Toc124101288)

[5.4 Javascript Primer 19](#_Toc124101289)

[5.5 Working with Containers 19](#_Toc124101290)

[5.6 Managing Dependencies 20](#_Toc124101291)

[5.7 Routes, Request Methods and View templates 22](#_Toc124101292)

[5.8 Working with Form and Data 25](#_Toc124101293)

[5.9 Cookies and Sessions 26](#_Toc124101294)

[5.10 Authentication and Authorization 28](#_Toc124101295)

[5.11 Data Validation 29](#_Toc124101296)

[5.12 Web Security Basics 31](#_Toc124101297)

[6 Implementation Process of web application 33](#_Toc124101298)

[6.1 Usecase Analysis 33](#_Toc124101299)

[6.2 Designing Database 33](#_Toc124101300)

[6.3 Almost a Walking Skeleton 33](#_Toc124101301)

[6.4 Adding and Listing Chores 33](#_Toc124101302)

[6.5 Claiming Chores 34](#_Toc124101303)

[6.6 Completing Chores 34](#_Toc124101304)

[6.7 Registration and Authentication 34](#_Toc124101305)

[6.8 User Chorecoins 34](#_Toc124101306)

[6.9 Statistics 34](#_Toc124101307)

[6.10 Validation 34](#_Toc124101308)

[6.11 Styles 34](#_Toc124101309)

[6.12 Deployment 34](#_Toc124101310)

[7 CONCLUSION 36](#_Toc124101311)

**REFERENCES 11**

**APPENDICES**

**FIGURES**

FIGURE 1 Relationships between components of the docker 4

FIGURE 2 The execution of a system using Docker 4

**PICTURES**

PICTURE 1. Motorcycling in Finland´s Winter 5

PICTURE 2. Heading 5

**TABLES**

TABLE 1. Age distribution of the respondents 5

TABLE 2. Heading 5

# [INTRODUCTION](#_INTRODUCTION)

Management system of a service become necessary towards business and organization because of the workflows and operation of data statistic to keep tracking activities. Nowadays, the field of Information Technology is being developed significantly. Thus, computer has been widely used in office, factory, school, etc... to solve mathematic problem and save human costs. The effectiveness of using computer is undeniable, helping to minimize the inefficient manual working stages. Thus, the access to management systems via a computer has become the top goal of organizations and businesses for their information management purposes. A housing service management system is one of information systems was built to address the needs of management in the house. The system will help businesses handle work accurately, quickly, save costs and hu-man resources, on the other hand, also promote the image of the home to friends inside and out-side the province, domestically and internationally.

# Literature Overview

House and homecare management system is one of business sectors that uses the progression of technologies. The progresses of system occur in sequences including computerized apartment reservation and house resource management. This research represents the advantages of effective system and the process of system design based on the knowledge of software engineering. The topic is one of the popular applies in the service management due to the increasing demand for the number of tourists, the number of services served, the commercial competition and the cost savings. Operating costs and more importantly, the correct and effective operation in management is the key to determining the success of the rental house service. The research is the most effective way to test the knowledge that student learned from school, the most effective approach to reality and more importantly the fundamentals for further research and development about information system in future.

## Objective

The objective of this thesis is to explore the possibilities and design a small, portable, and robust Lab

PC for use with prototyping (hardware and software) within Sasken Finland. The PC is to use a RPi as

its base and utilise HAT’s where possible to expand it as needed. The RPi Lab PC is not meant to be a

replacement for the day-to day working laptop but is rather an addition to the tools that the engineers

can use within their projects. A Lab PC based upon a RPi is hoped to bring greater portability, usability, customisations, and control over the various devices required and used in prototyping, whist also being cost affective.

## Scope of the thesis

The main scope of this thesis will be in the designing of such a PC, focusing in particular in the hardware and software requirements. It will look at and assess a selection of the various hardware available to best meet the needs of the project. A basic proof-of-concept prototype of a RPi Lab PC will be made if possible, incorporating the main basic required features as available. The design will be assessed to establish if it would be a viable option for use within Sasken. Issues on connection to the network drives, security and the possibility of custom-made casing will not be addressed within the scope of this thesis.

## Structure of the thesis

The thesis will firstly establish the requirements and determine the essential features required for the

Lab PC. This involves looking at what the current uses and methods are and gathering requirements

from the users, paying attention to the hardware, as well as software that will be required. Secondly the

thesis will then look more in depth at all the different and possible hardware available to fulfil the requirements. It will look at the suitability of the hardware for its use and compare options where appropriate. It will consider any potential conflicts between hardware and take this into account when designing. It will then continue to look at the software available starting with the operating system and continuing to the software currently used and its suitability and availability for use with the RPi. Lastly it will look at the possible ways in which the RPi could be used within Sasken.

# Requirements analysis

## Research

Due to the nature of the project much of the research information will be taken from technical specifications for the hardware or software, other technical publications, forums, hardware reviews and articles. Discussion with manufacturers on product suitability and use will be used where necessary. Informal meetings and questionnaires with key engineers will also be used to gather requirement information.

## Designing

The concept of web designing has not been consistent since it arises. Depending on the times technologies and programming architecture its definition well changes accordingly. In the older times web pages were developer through static HTML whose functionalities and logics were hard codded in the HTML page. At this point web designing was defined as building the backend functionality of a web page. [10, p. 2]

At this point the previous definition for web designing would be ambiguous. Hence a better definition needs to be made because the advancement of web application development process and development tools. A suitable definition for web designing at this point of time can be the making of the user interface looks and the detailed artistic feels of a website so that it would reflect the sense of the purpose it was built for. [11, p.203]

To design effective and attractive websites one has to master both the technical aspects and the artistic implementation of these technicalities on to a certain website. To make websites reach the current standards websites have to work effectively and smoothly with a database. The data sent and received to and from the database should be displayed in a format that is intuitive good looking and attractive. So that users can feel comfortable with the website. Therefore, the best websites are constructed by fulfilling both the technical requirements and the artistic feels. [8, p. 2-3]

In this thesis project designing the UI looks and feels was a little out of the norm. This is because the design of the web application was based on the former desktop application. There was no UI/UX design protocols or mockups or user experience cases. This is mainly because the old desktop application has been used for the past ten or more years by similar users, the UI looks, and feels are familiar to users. Completely changing the UI design would make target users feel uncomfortable and the web application might seem strange too. For this reason, it was decided that the new web application to look similar to the old web application so that users could find it easy to use.

To achieve the needed UI for the web application, several tools and technologies have been brought to use. Bootstrap integrated with HTML and CSS was used to develop the frontend framework. Since the old application includes lots of grids in most parts using normal tables would be ineffective and time consuming. Therefore, a grid system framework that suits the application well was purchased. This grid system is called Kendo UI. All the tables in the old application are replaced by Kendo grids. which looks better and made the programming a lot easier. Detailed explanation of the Kendo UI framework will be provided in the upcoming chapters.

## Development

The next important step in website development after the design is giving the components of the design actual purpose and functionality. This step has several parts in it. The basic part is structuring the web application. As mentioned above this application is structured to be a single page application. After structuring and building a skeleton or a frame for the web application in a suitable order giving functionality for all UI components follows. Finally, after everything is in place making data on the UI change every time there is an update in the server-side data in Realtime. This part is a basic and important part since the rest of the components are developed on top of it.

As mentioned in previous chapters this thesis project is a single page web application. To make this application, the JavaScript framework used is AngularJS. The main reason for this framework to be chosen was the authors personal interest and prior experience on this area. AngularJS by its nature a simple to use framework supporting CRUD operations. since this project itself requires CRUD operations angular has worked nicely with it.

### Front-end development

Angular JS or angular 1 is the first version of the angular families. It is a JavaScript based framework that was developed in 2009. It provides an MVC (model view controller), also addressed as MVW (model view whatever) development architecture. It's components as scope, controllers, rotoscope and so on enable two-way data binding in development. Which is quite helpful in changing data in both the UI and the script. Development is simplified and modularised in angular JS mainly because of the ability to reuse codes in a precise and clean way. This is due to Angular JS has a introduced the concept of  
directives. Directives made it possible to write a more elegant and divided code. [7, p. 69-77]

### Back-end development

Throughout the development process there are several functionalities requested by the application. Amongst them was real-time data update. In this project most of the data coming from the database are changing constantly. Also, those changes are mandatory for the clients in making the decisions and provide the service. In the old desktop application clients had to request updates whenever a latest data was needed. Furthermore, in some parts the application was forced to fetch updates from the database in a given range of time. The same approach could have been used in the web application. Since this would give the web servers a lot to work, an alternative and better approach has to be used. This approach was using SignalR to handle the Realtime updates and data flow.

SignalR is an open-source library that takes care of real-time data flow between web servers and Clients. The SignalR makes things a lot easier since it uses WebSocket to establish a persistent connection in the beginning. The connection is very useful in monitoring data flow. This connection management lets the server to recognize connected users and decide which data shall be accessed or be hidden to which customer. This improves the security of the application in general. [12]

## Testing

Web Testing, or website testing is checking your web application or website for potential bugs before its made live and is accessible to general public. Web Testing checks for functionality, usability, security, compatibility, performance of the web application or website. During this stage issues such as that of web application security, the functioning of the site, its access to handicapped as well as regular users and its ability to handle traffic is checked.

# Tools and technologies for building web application

Technologies and tools that helps to build progressive web application will be mentioned

and discussed in this section.

## Deno

On May 13, 2018, Ryan Dahl - the creator of Node.js released a Runtime Enviroment for Javascript called Deno, which is supposed to fix all the problems of Node.js. Deno is a new Runtime Environment for Javascript and TypeScript, using the V8 Engine and the Rust programming language.

Deno has some highlights such as:

Secure by default.

Modules cannot be accessed without being enabled.

TypeScript support.

The standardized modular system is guaranteed to work with Deno.

Send only one file.

Deno is made for browser compatibility. Technically, when using ES modules, Deno won't need to use webpack so it will always be cross-browser compatible, even with older browsers like Internet Explorer. You can use import and you don't need NPM to install all packages. Looks like GoLang, you can import from URL. Using TypeScript is very easy in Deno without any configuration files. However, you can also write in Javascript without any hassle.

<https://deno.land/manual@v1.29.1/introduction>

## VSCode

Visual Studio Code is an application that allows editing and editing of code to support in the process of building and designing websites quickly. Visual Studio Code is also abbreviated as VS Code. This editor works smoothly on platforms like Windows, macOS, Linux. Moreover, VS Code also offers compatibility with mid-range computer devices that can still be used easily. Visual Studio Code supports a variety of Debug functions, comes with Git, has Syntax Highlighting. Especially intelligent code completion, Snippets, and source code improvement capabilities. Thanks to the customization feature, Visual Studio Code also allows developers to change Themes, keyboard shortcuts, and a variety of other options. Although this Code editor is relatively lightweight, it includes powerful features. Although newly released, VSCode is one of the most powerful and popular Code Editors for programmers. Thanks to support for many popular programming languages, full integration of features and extensibility, VSCode becomes extremely familiar to any programmer.

<https://code.visualstudio.com/docs>

## Elephant PostgreSQL

PostgreSQL is the most advanced free and open source object-relational database management system (RDBMS) available today. High scalability and compliance with technical standards. It is designed to handle a wide range of workloads, from personal computers to data warehouses or Web services with many concurrent users. PostgreSQL is developed by PostgreSQL Global Development Group.

First Released: 08/07/1996

PostgreSQL is flexible and can be run on many different platforms such as Mac OS X, Solaris and Windows.

PostgreSQL is free and open-source software so PostgreSQL can be used, modified, and disseminated by anyone for any purpose.

PostgreSQL is highly stable. PostgreSQL was the first database management system to implement multiple version control (MVCC).

PostgreSQL integrates many great features to help developers build apps that respond to complex functions, query quickly, and securely maintain integrity and reliability. To be more reliable, Postgresql provides various security, authentication, and disaster recovery options. PostgreSQL has proven to be highly scalable both in terms of the amount of data and the number of concurrent users.

<https://www.elephantsql.com/docs/index.html>

## Google Chrome

Google Chrome is a  free web browser developed by Google, used to access websites on the Internet. As of April 2019, it is the most widely used web browser worldwide, accounting for 60% of the web browser market share. Google Chrome is also a cross-platform browser with full versions that work on computers, smart mobile devices and different operating systems. According to Statista, Google Chrome for Android is the most used version, accounting for nearly 31% of the global web browser market share in 2019.

The browser is fast, secure and easy to use Perhaps the biggest draw to Google Chrome is its reputation for performance. Web pages can be opened and loaded extremely quickly – even when browsing through pages with heavy graphics, ads or video content. The simple interface is easy to use even for beginners and the updates are automatically rolled out regularly to ensure a safer browsing experience.

<https://www.google.com/chrome/?brand=YTUH&gclid=Cj0KCQiA5NSdBhDfARIsALzs2ECbu4lU0JhO-6uXuSjHlP_6tqyTfyK6VCXxDyr76pq0QfY0E7ZDZwEaAnntEALw_wcB&gclsrc=aw.ds>

## Docker and Docker Compose

Docker is a platform to provide an easier way to build, deploy, and run applications using containers (on virtualization platforms). Originally written in Python, now switched to Golang.

Containers allow developers to package an application with all the necessary parts, such as libraries and other dependencies, and package it all out as a single package.

That way, thanks to the container, the application will run on any other Linux machine regardless of any customization settings the machine may have that are different from the one used to write the code.

In a way, Docker is quite similar to a virtual machine. But why is Docker growing and popularizing so quickly? Here are the causes:

Ease of application: Docker is easy for everyone to use from programmers, sys admins... it takes advantage of containers to build and test quickly. Can package the application on their laptop and run on public cloud, private cloud… The mantra is “Build once, run anywhere”.

Speed: Docker containers are so light and fast, you can create and run docker containers in seconds.

Run environment and extensibility: You can break down the functionality of your application into individual containers. The Database example runs on one container and the Redis cache can run on another while the Node.js application runs on another. With Docker, it is very easy to link containers together to form an application, making it easy to scale and update components independently of each other.

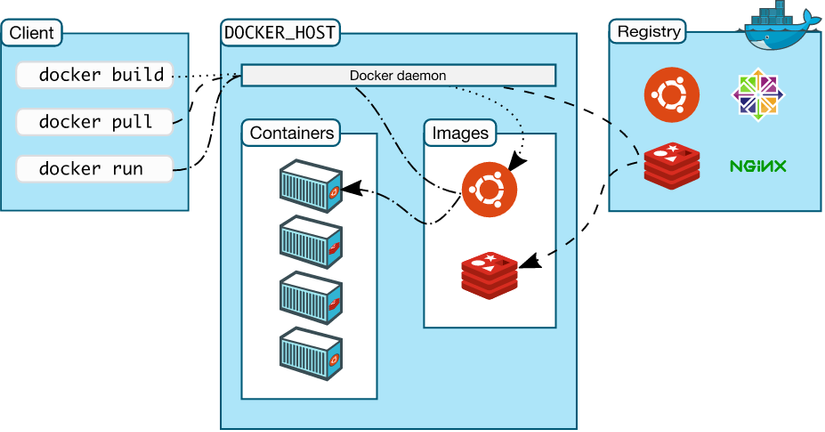


FIGURE 1 Relationships between components of the docker

* Docker Engine: is the main component of Docker, as a tool to package applications
* Docker Hub: is a “github for docker images”. On DockerHub there are thousands of public images created by the community allowing you to easily find the images you need. And just pull back and use with some config you want.
* Images: is a template for creating a container. Usually the image will be based on an existing image with additional customizations. For example, you build an image based on a sample Centos image available to run Nginx and customize and configure your web application to run. You can build your own images or use shared images from the Docker Hub community. An image will be built based on the instructions of the Dockerfile.
* Container: is an instance of an image. You can create, start, stop, move or delete containers based on Docker API or Docker CLI.
* Docker Client: is a tool to help users communicate with Docker host. Docker Daemon: listens to requests from Docker Client to manage objects such as Containers, Image, Network and Volumes through REST API.
* Docker Daemons also communicate with each other to manage Docker Services.
* Dockerfile: A file that contains instructions for building an image.
* Volumes: is the piece of data created when the container is initialized.

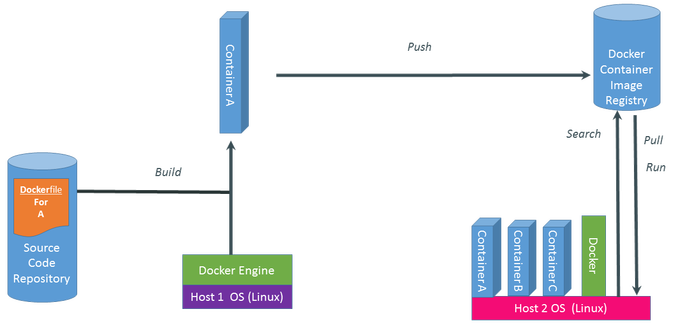
<https://docs.docker.com/desktop/>

Docker compose is a tool used to define and run multi-containers for Docker applications. With compose you use the YAML file to configure the services for your application. Then use com-mand to create and run from those configs. Using is also quite simple with only three steps:

• Declare app's environment in Dockerfile.

• Declare the services needed to run the application in the docker-compose.yml file.

• Run docker-compose up to start and run the app.



FUGURE 2 The execution of a system using Docker

Build - First create a dockerfile, in this dockerfile is our code. This Dockerfile will be built on a computer that has Docker Engine installed. After the build we will get the Container, in this Container contains the application with our library.

Push - After getting the Container, we push this Container to the cloud and save it there.

Pull, Run - If another computer wants to use our Container, it must be forced to pull this container to the machine, of course this machine must also install Docker Engine. Then execute this Run Container.

Config in Dockerfile. If you don't know about the commands to configure Dockerfile. Don't worry watch part 1: here

Graphical user interface, text, application, email

Description automatically generated

Config services cần start và run trong file **docker-compose.yml**

Graphical user interface, text

Description automatically generated with medium confidence

* version: indicates the docker-compose version used.
* services: set the services(containers) you want to install and run.
* image: Specifies the image used during the creation of the container.
* build: used to create containers.
* ports: Set ports to run at the host machine and in the container. restart: automatically launched when the container is shut down.

<https://docs.docker.com/compose/>

## Heroku Deployment

It is a cloud platform that allows developers to build, deploy, manage and scale applications (PaaS - Platform as a service). It is very flexible and easy to use, providing the simplest way to bring the product to the user. It helps developers focus on product development without worrying about running servers or hardware… Heroku runs applications in dynos – it's a virtual machine that can scale up and down depending on the size of the application. Free work will only be limited to 550 hours per month. If you want to increase it to 1000 hours, you need to install a payment method on it. However, to test an idea or a small website, that is more than enough to get results.

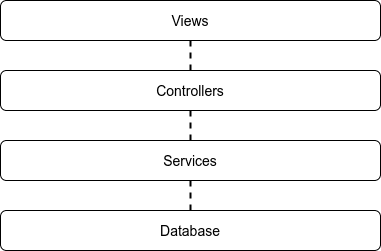
# Web development components

## Monolithic Architecture

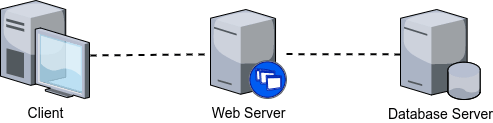
Here, we discuss briefly monolithic architecture. The term monolith refers to "an organized whole that acts as a single unified powerful or influential force" ([Merriam-Webster](https://www.merriam-webster.com/dictionary/monolith)), and the term **monolithic architecture** refers to an application that is a single coherent unit where all the code is in the same directory hierarchy and where the application is deployed as a single unit.

Applications that use monolithic architecture may contain a user interface (view templates, javascript, images, ...), a server-side application responsible for handling requests, and the functionality related to handling database requests. The database can be separate from the application, but the database could be also included in the application. The key is that the server-side functionality is built as a single unit, and the functionality provided by the application resides in the same codebase.

As an example, the [Task Management Application from Application Example I](https://fitech101.aalto.fi/web-software-development/17-application-example-i/1-overview/) has monolithic architecture. The application follows a layered architecture, where the code is divided into logical entities, and a multitier architecture, where the functionality of the application is divided over the client, the server-side application, and the database. The application code is in the same directory hierarchy, and the application is deployed as a single unit.

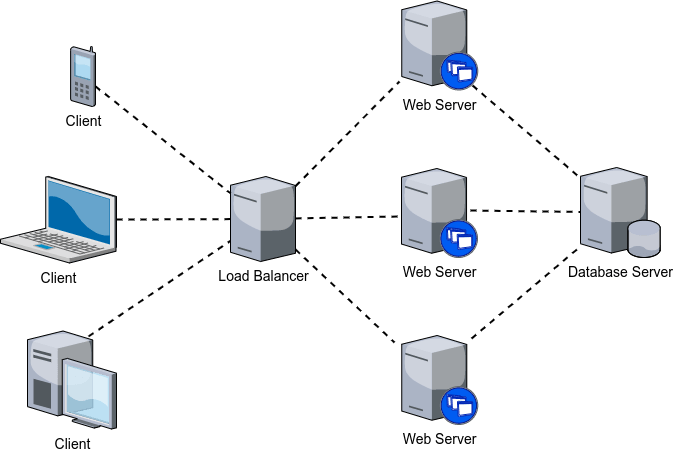


Layered architecture with four layers: views, controllers, services, database.



Web applications that have a client, a web server, and a database, use 3-tier architecture.

When new functionality is added to an application with monolithic architecture, the application as a whole must be deployed again when the functionality is put to production. Similarly, if one would wish to increase the throughput or performance of such an application, one approach would be increase the amount of servers and add a load balancer that would distribute the incoming requests to the web servers. The key here is that each new web server would run the whole application; that is, it would not matter which server the client would be forwarded to by the load balancer as all of them behave the same way.

Increasing the amount of servers and adding a load balancer that is responsible for distributing the incoming requests to the web servers (4 tier architecture).

Monolithic applications can be tested in multiple ways. One can test individual components or functions, one can test the HTTP interfaces of the application, and one can -- naturally -- do manual testing. One can also mock functions, interfaces, etc. Monolithic applications are straightforward to test and deploy using [continuous integration](https://fitech101.aalto.fi/web-software-development/29-continuous-integration-and-deployment/1-continuous-integration/) and [continuous deployment](https://fitech101.aalto.fi/web-software-development/29-continuous-integration-and-deployment/2-continuous-deployment/) as the application can be deployed as a whole.

Similarly, software developers can also typically launch the application as a whole on their computers, likely (hopefully!) adjusting the database configuration so that a development database is used instead of a production database.

However, as the size of the application grows, maintaining an understandable internal structure of the codebase becomes harder. In our examples, we have included all service functionality in a folder called services and all controller functionality within a folder called controllers that resides within a folder routes. If there are tens of controllers and dozens of services, and the developer does not know the application well enough, it can be tempting to simply mimic the existing code and add a new controller instead of studying how the application works and e.g. and refactoring the application codebase to improve it.

With the increasing size of the application, the need to deploy the whole application after every change can also be somewhat cumbersome. For example, if we would adjust the title of the task application or add styles to it, we would have to deploy the whole application, including the server-side functionality, even if those would not have been touched at all. The larger the application is, the longer it also takes to start up after deployment, which can result in poor user experience if the deployment process or timeline is not well thought out.

Increased size of an application also tends to lead to increased commitment to an application and the technology used by the application. The [sunk cost fallacy](https://en.wikipedia.org/wiki/Sunk_cost) stipulates that people in general have a greater tendency to continue investing time and other resources into an effort once an investment to that effort has already been made. With the relatively rapid development of technologies and frameworks, investment into an application may lead to a situation where resources will be redirected to that application also in the future.

Also, considering the scalability; while it is easy to add new servers that host the application, it is possible that only small parts of the application are those that really need to scale, while the majority of the application requires rather little resources.

To summarize, a monolithic architecture is an architectural pattern where the application is developed and deployed as a whole. The benefits of monolithic architecture include easiness in developing, deploying, and scaling. However, as the size of an application grows, it can become harder to maintain and develop, the deployment times can become longer, and one may become dependent on the used technologies.

<https://microservices.io/patterns/monolithic.html>

## HyperText Makeup Language

HTML stands for Hypertext Markup Language (roughly translated as Hypertext Markup Language). HTML is used to create and structure elements in a web page or application, divide paragraphs, headings, titles, blockquotes, etc., and HTML is not a programming language.

An HTML document is formed by HTML elements (HTML Elements) specified by pairs of tags (tags and attributes). These tag pairs are enclosed by a curly brace (e.g. <html>) and are usually declared as a pair, consisting of an opening tag and a closing tag. For example, we can create a paragraph by placing text inside the opening and closing tags <p> and </p> :

<p>This is how you add paragraphs in HTML.</p>

But some special tags don't have a closing tag and the declared data will be in attributes (eg the <img> tag).

The father of HTML is Tim Berners-Lee, also the founder of the World Wide Web and president of the World Wide Web Consortium (W3C - the organization that sets standards on the Internet). HTML settings and structures are operated and developed by the World Wide Web Consortium (W3C). You can check the latest status of this language at any time on the W3C's website.

<https://developer.mozilla.org/en-US/docs/Web/HTML>

## Cascading Style Sheets

CSS stands for Cascading Style Sheets, it is a language used to find and reformat elements generated by markup languages (HTML). In short, the language of styling the website. You can simply understand that, if HTML plays the role of formatting elements on the website such as creating paragraphs, headings, tables, etc., then CSS will help us to add styles to the elements. That HTML like changing layout, page color, changing text color, font, changing structure... CSS was developed by the W3C (World Wide Web Consortium) in 1996, because HTML was not designed to be tagged to help format web pages. The way CSS works is that it will search based on selections, the selection can be the name of an HTML tag, the name of an ID, class or many other types. Then it will apply the properties to be changed to that selection. The correlation between HTML and CSS is very close. HTML is the markup language (the foundation of the site) and CSS defines the style (all that makes up the website interface), they are inseparable.

<https://developer.mozilla.org/en-US/docs/Web/CSS>

## Javascript Primer

JavaScript is a programming language used by developers to create interactive websites. From refreshing social media feeds to displaying animations and interactive maps, JavaScript capabilities can improve a website's user experience. As a client-side scripting language, JavaScript is one of the core technologies of the World Wide Web. For example, when browsing the internet, whenever you see an image carousel, a click-to-display drop-down menu, or dynamically changing element colors on a web page, that's when you see the effects of JavaScript. JavaScript is gradually becoming known as a browser-side technology to make web applications more flexible. Using JavaScript, browsers can respond to user interactions and change the layout of content on a web page.

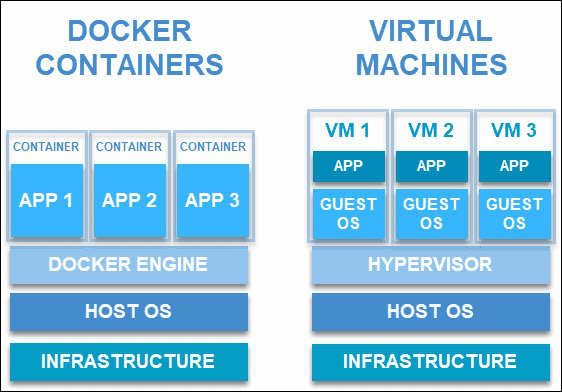
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

## Working with Containers

Containerization refers to packaging an application so that it comes with the necessary operating system libraries and dependencies needed to run the application. An application within a container can be run practically anywhere, given that there exists support for the containerization software. One of the most commonly used software for containerization is [Docker](https://www.docker.com/).

Applications will run on containers and share the same OS. The container will contain all the components to create an environment that allows the application to run on it. For that reason, containers have a number of advantages such as: versatile, lightweight, portable, easy to upgrade, replace, and extensible.

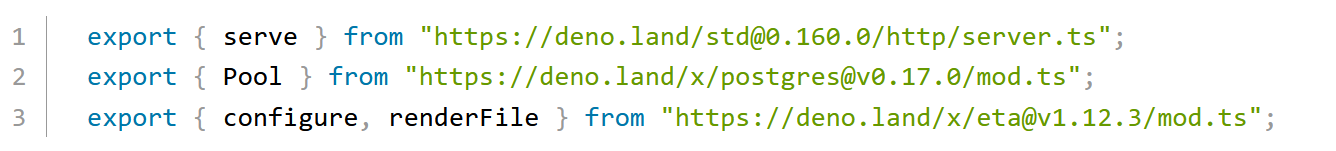
Here, we briefly visit how a Deno project that uses a database could be developed with the help of Docker. In the following subsections, we set up multiple containers: one that will run the Deno application, one that will run the database that the application uses, and one that will be responsible for database migrations.



## Managing Dependencies

As we increasingly work with applications that have multiple files with logically separated content, we wish to make sure that the dependencies (i.e. external libraries) of the project are the same across the files of the project, and that updating dependency versions is straightforward. The way how this is done in Deno is through introducing a separate centralized file for importing dependencies from external sources, which are then exported into use within the project. In Deno, the file is called deps.js.

Presently, as our projects use the function serve from Deno, configure and renderFile from Eta, and Pool from the PostgreSQL database library, we would include these into the deps.js that would then export them into the of the project. In practice, the file deps.js file would look as follows.



The deps.js file would be placed to the root of the project into the same folder with app.js.

Graphical user interface, application

Description automatically generated

Now, for example, the database.js would import Pool from deps.js instead of the actual library address. This would look as follows.

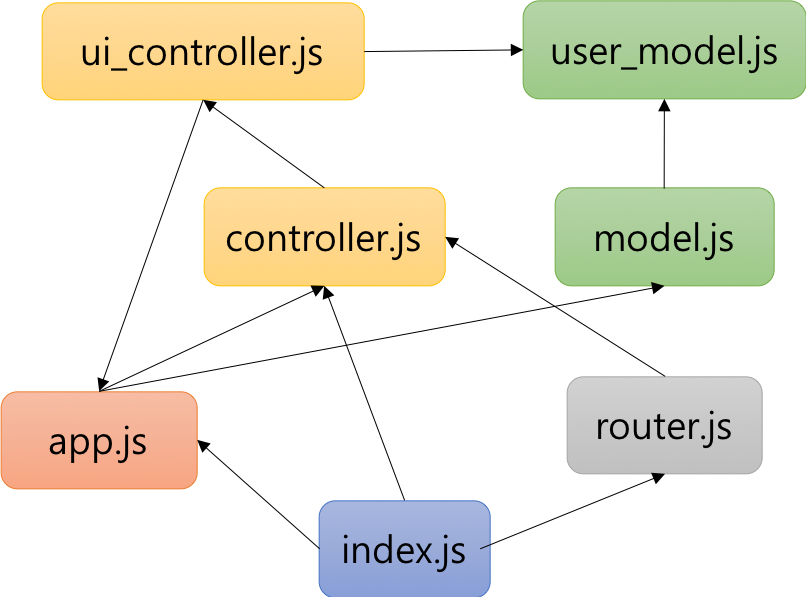
Graphical user interface, text, application, email

Description automatically generated

Similar adjustments would be made to all other files that depend on external libraries. As another

Text

Description automatically generated



FUGURE 3 The execution of a system using Docker

## Routes, Request Methods and View templates

When building web applications with vanilla Deno, we used the term path to identify the path (part of an address or URI, e.g. /tasks/1) to which the request had been made. The path was where the request was made was, in part, used to determine the code that should be executed for the specific request.

When working with oak, we can extract the path (or pathname) from the url within the request. This information could be used to direct the requests to correct functionality. As such behavior is often needed, however, oak provides routing middleware that helps mapping paths to functions.

The term router refers to a part of the framework that is responsible for determining which function should be called for which path and which request method. A router directs (or routes) requests to correct functions. A route, then, can be seen as a mapping between a path and a request method to a function that is responsible for handling the specific request

Routes are created using the Router class from oak. In the following example, we import both the application and the Router class from oak. We create an instance of the application and an instance of the router. Then, we define a function called greet that uses the response object from the context to set the response to 'Hello world!'.

After this, we add the function greet to the router. When calling router.get('/', greet), we state that whenever a GET request is made to the path /, the request should be routed to the function greet. This is followed by a call app.use(router.routes()), where we ask the application to use the routes (or route) that we just defined.

Finally, we ask the application to start listening for requests.

Graphical user interface, text, application

Description automatically generated

Adding multiple routes is straightforward. For every route that we wish that the application handles, we create a mapping from a path to a function using the router. In the example below, we have defined two paths that the application listens to. When a request is made to the root path of the application, the response will contain the string 'Hello world!', while when a request is made to the path '/another', the application response will contain the string 'Another path!'.

Graphical user interface, text, application

Description automatically generated

We previously mentioned that the router is used to map paths and methods to functions. So far, we have used the get-method of the router; this leads to mapping HTTP GET requests. The router also has other methods that correspond to the remaining HTTP request methods. The example below shows the use of get, post, and delete methods of the router, which correspond to the HTTP methods with the same names.

In the example below, we named the functions that handle the requests get, post, and del -- we use del instead of delete, as delete is a reserved word in JavaScript.

Text

Description automatically generated

We would create a folder called routes and within that a folder called controllers. In the folder controllers, we would have a files that export functionality used for handling individual requests (like the functions get, post, and del above. Then, in the folder routes, we would have a file called routes.js that would import the files from the folder controllers, and map the functionality from the controllers to paths and methods.

Let's call the file with the above functions methodController.js (which is not really a good name, but oh well..).

When looking at the project as a tree, it looks as follows.

Chart

Description automatically generated with low confidence

## Working with Form and Data

When learning the basics of HTML, we took a peek at creating [forms](https://fitech101.aalto.fi/web-software-development/2-hypertext-markup-language/3-forms/). Here, we'll learn to use forms as a part of a web application. In general, when working with forms and sending data to the server, we use the HTTP POST -method, which is set using the method-attribute of the <form>-element.

Let's use the following document as the first example. The method is POST and as there is no action-attribute, the form data will be sent to the current address -- i.e. the address where the form is viewed at. The form itself is simple -- it has only a submit-button, which is used for submitting the form.

Logo

Description automatically generated with medium confidence

When the form is submitted, the form data will be sent to the server using a specific format. Information about the format is added automatically by browsers to the request -- this information resides in request headers. Request headers can be accessed through the headers-property of the request. The headers property contains a [Headers](https://developer.mozilla.org/en-US/docs/Web/API/Headers) object, which has a method get for retrieving header values. The method returns a string that contains the value of a name given to the method as a parameter.

Let's next look into how data sent using a form is handled. In this example, we create a form that has an input field that can be used to enter a name.

Graphical user interface

Description automatically generated with medium confidence

When the form has input fields, the data entered to the form is sent within the body of the request (in the case of the POST request that we use). For each input element, the body of the request will contain a key-value -pair that contains the name of the input field (the value of the name-attribute) and the value entered into the input field.

The body of a request can be transformed into a [FormData](https://developer.mozilla.org/en-US/docs/Web/API/FormData) object using the asynchronous formData method that is provided by the [Request](https://developer.mozilla.org/en-US/docs/Web/API/Request) object. The [FormData](https://developer.mozilla.org/en-US/docs/Web/API/FormData) object has a number of useful methods, including keys that returns an iterator with all the keys that correspond to the names in the submitted form, get that returns a value for a given key, and has that returns a boolean that indicates whether a given key exists.

## Cookies and Sessions

Cookies are a mechanism for storing small amounts of data on the client, which then sends that data to the server on every request. Cookies are often used for tracking users across requests.

In practice, cookies work so that when a client makes a request to the server, the server, in the response, asks the client to set a cookie for future use. Then, on subsequent requests to the server, the client sends the cookie within the request. This allows the server to keep a track of the user across requests.

Cookies are implemented using HTTP protocol headers. When a client makes a request to the server, the server adds a Set-Cookie header to the response. A Set-Cookie header could, for example, look like the following one -- in the following example, we create a cookie with the name visits that has the value 0.

Oak framework abstracts away the need to work with HTTP headers and provides direct access to cookies. Cookies can be accessed through the context object, which has a variable called cookies.

The method get of cookies retrieves the value of a cookie sent by the client, while the method set is used to add a Set-Cookie header to the response. The method get takes one parameter, which is the name of the cookie for which the value is being retrieved, while the method set takes two parameters, the name of the cookie and the value of the cookie.

Both methods are asynchronous.

In the following example, we use and set a cookie for keeping track of the number of requests made to the server.

Graphical user interface, text, application

Description automatically generated

Next, we look into sessions, which also use cookies for tracking users across requests. In addition to sending a cookie to the client, the server stores session data that can be retrieved based on the cookie. In practice, when using sessions, the cookie contains a random string that is passed back and forth between the server and the client. The cookie, on the server, is then resolved to an object stored on the server, which contains data related to the particular cookie and client

Using [Oak Sessions](https://deno.land/x/oak_sessions@v4.0.5/mod.ts) library in oak requires importing the library, creating an application, and attaching the session functionality to the application through a middleware that can be retrieved from a Session object.

The following example shows how the library is taken into use in oak.

Text

Description automatically generated

When the session functionality is added to oak, a variable session is added to the state object of context. Similar to when working with cookies, the session variable has get and set methods, which are used to retrieve and set values related to the session. When using those functions, we do not set or modify the cookie data, however, but set or modify the session data stored on the server.

Both methods are asynchronous.

## Authentication and Authorization

An authentication process consists of the user sending credentials to the application, often a username or an email and a password. These details are then verified against the data on the server. If the sent credentials match those on the server, the user has successfully authenticated.

A very crude application for authentication could be as follows. The following application expects that a password is sent in the request body. If the user sends a password 'asparagus', then the user is shown a message 'User authenticated'. Otherwise, the user is sent the HTTP status code 401, indicating bad credentials or unauthorized access.

Session-based authentication is based -- as the name implies -- on the use of sessions. When the user sends the credentials, e.g. a password, to the server, the server verifies the credentials. Once the user has authenticated for the first time, i.e. the credentials match those on the server, information about the user and authentication is added to the session. From this point on, when the user makes a request to the server, the server authenticates the user based on the session.

Text

Description automatically generated

In a simple form, this could work similarly to the following application. The application checks whether the user's session has a variable 'authenticated'. If the variable does not exist, credentials are read from the request body and checked against those on the server. If the password sent to the server is correct, the server sets the 'authenticated' variable to the session and responds with the message 'User authenticated by password'. If the password is incorrect, the server responds with the status code 401.

If the user's session has the variable 'authenticated', then the user is shown a message 'User authenticated by session'. In this case, no password is needed; instead, the session identifier that is sent in the request (as a cookie) acts as the credentials.

The term authorization refers to the process of verifying that the user has the rights to perform the actions that the user is trying to perform. In addition, it also refers to the process of defining access rights to specific resources in an application.

Let's start with an example of broken authorization, after which we look into authorization functionality

## Data Validation

At this point, we've learned to work with forms and databases. When adding data to the database, we have not, however, been concerned with the correctness or the validity of the data. For example, we have not cared whether submitted form data has been empty or not.

The term *data validation* refers to the process of verifying that the data is in an expected format. This includes (1) verifying the data type, e.g. numeric fields should only accept numeric data and email addresses should be formatted like email addresses; (2) checking the input data against a known list of accepted values, e.g. when entering country information, colors, or other enumerable data; (3) checking that numeric data falls within a range, e.g. when entering a year of birth, depending on the application, it is meaningful to verify that the year of birth has four digits and cannot be in the future; (4) checking that the data follows a specific format, e.g. the Finnish social security number has 11 characters, which encode date of birth and gender; similarly, dates are formatted differently in different countries; and (5) checking for uniqueness, e.g. verifying that a specific email should be entered only once to a list of emails

When developing web applications, there are multiple locations for data validation. For our purposes, the most relevant locations are (1) client-side, (2) server-side, and (3) database.

* Client-side data validation refers to verifying that the users type in data in a specific format. This is done with specific input fields as well as with client-side JavaScript. In practice, client-side validation is done for usability -- anyone who knows a bit about how the web works can bypass client-side data validation and send invalid data to the server.
* Server-side data validation refers to verifying that the data that is sent to the server follows specific rules. In practice, this functionality is added to the functions that handle incoming requests. Server-side data validation may also include verifying that data sent from the server to the user is valid.
* Data validation in the database refers to verifying that the data stored in the database follows the correct format. With relational databases, some of this can be automated -- for example, using a column that accepts only numeric data can be used to restrict inputs. Relational databases also feature constraints (e.g. NOT NULL, UNIQUE, CHECK, PRIMARY KEY, FOREIGN KEY), which can be further used to restrict what can be added to the database, and also, to verify that the data follows specific rules (e.g. a an account can be only added to an existing user).

Our focus here is on server-side data validation.

[Validasaur](https://deno.land/x/validasaur@v0.15.0) provides validation functionality that we can use in our applications. When validating data with Validasaur, we import the validation rules and a function called validate, define an object with attributes that are linked to lists of validation rules, then define the object that should be validated, and finally validate the object using the validate function and the defined validation rules.

The function validate returns a list with two values. The first value contains a boolean variable telling whether the validation passed or not (i.e. whether no errors were found), and the second value contains an object with validation errors.

In the following example, we validate an object with an age. The age is required (i.e. it cant be empty) and it should be numeric (i.e. it cannot contain text). If the validation passes, i.e. no errors were found, we log a message indicating that no validation errors were found. Otherwise, we log the validation errors.

Graphical user interface, text, application, chat or text message

Description automatically generated

When we run the application, we notice that the validation did not pass. Instead, we see an object that outlines which errors were obsered. The object specifies the attribute for which the validation errors failed, and for that attribute, which rules failed. The library also provides clear-text explanations outlining the reasons for the failure.

## Web Security Basics

In 2013, an article published in [Forbes](https://www.forbes.com/) suggested that every day, on average, about [30,000 new websites are hacked](https://www.forbes.com/sites/jameslyne/2013/09/06/30000-web-sites-hacked-a-day-how-do-you-host-yours/). As web applications (and other applications) continue to be under scrutiny from malicious users, it is not surprising that [there is a cybersecurity workforce shortage](https://atlasvpn.com/blog/global-cybersecurity-industry-faces-a-workforce-gap-of-3-12-million-in-2020). It is important that software developers know at least the basics of building secure software, and know how to maintain secure software.

Indeed, most common issues with application security are related to developers' lack of awareness or rigor as well as poor development practices. With lack of awareness or rigor, developers may leave bugs into their applications which allow attacks towards the application or the users of the application. With poor development practices, developers may store e.g. passwords or other crucial information in locations that can be accessed or breached, not actively keep the used libraries and other software up to date to mitigate arising security concerns, or may even -- again, as an example -- host applications on a server that has other software with security flaws, which can lead to attacks towards the applications through the bugs exposed by other software.

The [Open Web Application Security Project® (OWASP)](https://owasp.org/) foundation works to improve software security and to provide resources for learning about securing web applications. They, for example, maintain a guide on [Web Application Security Testing](https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/) and keep track of the (current) [most critical security risks in web applications](https://owasp.org/www-project-top-ten/).

Here, we look at a few of the common issues, including injection flaws, cross-site scripting, and broken authentication. For further readings on the topic, visit the cyber security courses offered by Aalto and the [Cyber Security Base](https://cybersecuritybase.mooc.fi/) course series by F-Secure and the University of Helsinki.

Note that it is illegal to look for security flaws in web services unless the owner of the web service has given explicit permission. Here, we visit the topic for learning purposes, and the few assignments that you will work on will be on an application running on your own computer.

Injection flaws

Cross-site scripting

Broken Authentication

# Implementation Process of web application

We create an application for handling condominium chores. A condominium -- from Latin words con (together) and dominium (ownership) -- is a collection of individually owned homes (be they in a block of flats, a row house, or other sort of homes) where common areas such as playgrounds, carbage can shelters, and so on are collectively owned.

The application allows the creation of chores, reserving chores, and completing chores. Each chore is related either to a home of the individual creating the chore (such as mowing the lawn of a specific home), or to the common areas (such as mowing the lawn of the playground). Each chore will have a description, a due date, and a fee; fees in the application are handled using a virtual currency called chorecoins.

Anyone living in the condominium is able to create chores, and anyone living in the condominium can claim chores for themselves. The application will feature a ranking that shows who has created most chores. The application naturally works as a web application, but it also provides an API that allows checking for open chores. Chores related to individual's homes and the common areas do not need to be technically separated; the location of the chore can be entered in the chore description.

The application will use a 3-tier architecture (client, server, database) and a layered architecture with four layers (views, controllers, services, database). We use Oak as the base framework for building the application.

Note! When you start building the application, keep the application running all the time using the --watch flag. Effectively, whenever you make changes to the sources, the changes are reloaded to the application.



## Usecase Analysis

Table 1 – The main features for admin

|  |  |
| --- | --- |
| **Feature** | **Feature description** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table 2 – The main features for users

|  |  |
| --- | --- |
| **Feature** | **Feature description** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Designing Database

## Almost a Walking Skeleton

## Adding and Listing Chores

## Claiming Chores

## Completing Chores

## Registration and Authentication

## User Chorecoins

## Statistics

## Validation

## Styles

## Deployment

# CONCLUSION

The currently version of house management offers only room renting services and the software ran on desktop and web environment. The research of this system can be further developed for the bigger purpose of usage such as homecare services and more specifically the software can be built on the Android or IOS environment in future. The future development should be successfully completed by studying all requirements and executed in logical ways to integrate with the existing ones. The system will be upgraded by incorporating better performing record management software, modern technologies in another programming language and open-source frameworks. Thus, designing a system in future will be significantly optimized in the consumption of time and work.

**REFERENCES**

Deno <https://deno.land/manual@v1.29.1/introduction>

VSCode <https://code.visualstudio.com/docs>

Elephant Postgresql <https://www.elephantsql.com/docs/index.html>

Google Chrome <https://www.google.com/chrome/?brand=YTUH&gclid=Cj0KCQiA5NSdBhDfARIsALzs2ECbu4lU0JhO-6uXuSjHlP_6tqyTfyK6VCXxDyr76pq0QfY0E7ZDZwEaAnntEALw_wcB&gclsrc=aw.ds>

Docker <https://docs.docker.com/desktop/>

Docker Compose <https://docs.docker.com/compose/>

Monolithic Architecture <https://microservices.io/patterns/monolithic.html>

HTML <https://developer.mozilla.org/en-US/docs/Web/HTML>

CSS <https://developer.mozilla.org/en-US/docs/Web/CSS>

JAVASCRIPT <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

---------------------------------------------------------

Lano, K. 2005, Advanced Systems Design with Java, UML and MDA, Elsevier Science & Technology, Burlington.

Dennis A. Wixom B.H., Roth R.M 2012, System Analysis and Design, John Wiley & Sons. Available: [https://www.oreilly.com/library/view/system-analysis%20and/9781118057629/17\_chap08.html. Accessed 18 April 2022](https://www.oreilly.com/library/view/system-analysis%20and/9781118057629/17_chap08.html.%20Accessed%2018%20April%202022).

Hanmer, RS. 2013, Pattern-Oriented Software Architecture for Dummies, John Wiley & Sons, Incorporated, Somerset.