Bachelor's thesis

Information and Communications Technology

2024

Emilia Heinonen

Developing a Help Centre with a Retrieval-Augmented Conversational AI System

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2024 | 54 pages

Emilia Heinonen

Developing a Help Centre with a Retrieval-Augmented Conversational AI System

This thesis aimed to improve the user support experience in the commissioner’s archival software by developing an online platform for essential software instructions, allowing users to troubleshoot common issues, and freeing up support and development teams' time. This thesis's final products were a help centre site and a retrieval-augmented conversational AI system. The main technologies used were Laravel and Tailwind CSS. One of the main goals was to explore AI implementation possibilities in a data-privacy-sensitive application. The AI system was built on top of the base site, leveraging LlamaIndex as the retrieval-augmented generation framework, OpenAI's GPT-3.5 Turbo model, and Ada text embedding model for natural language processing tasks.

Compared to the more conventional help centre, the AI system's main disadvantage was a lack of adequate open-source large language models that supported Finnish and inconsistent answers. It offered more possibilities for tailored support and easier upkeep. The testing results imply that the AI system would not work well on its own, but when combined with the traditional Help Centre, it shows potential for quick, personalised assistance. More development and research are required for a production-ready system.

Keywords:

Customer Support System, Artificial Intelligence, Large Language Models, Retrieval-Augmented Generation

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2024 | 54 sivua

Emilia Heinonen

Ohjekeskuksen kehittäminen RAG-pohjaisella keskustelullisella tekoälyjärjestelmällä

Tämä opinnäytetyö pyrki parantamaan käyttäjätukea toimeksiantajan arkistointiohjelmistossa kehittämällä sivuston, joka tarjoaa olennaiset ohjelmisto-ohjeet. Sivusto auttaa käyttäjiä ratkaisemaan yleisiä ongelmia itsenäisesti vapauttaen samalla tukitiimin ja kehittäjien aikaa muille tehtäville.

Lopputuloksena syntyi ohjekeskus-sivusto ja retrieval-augmented generation-tekniikkaa hyödyntävä keskustelutekoälyjärjestelmä. Pääasiallisesti käytetyt teknologiat olivat Laravel ja Tailwind CSS. Yksi tärkeimmistä tavoitteista oli selvittää tekoälyn käyttöönottomahdollisuuksia sovelluksessa, jolle on tiukat tietosuojavaatimukset. Tekoälyjärjestelmä rakentui perussivuston päälle hyödyntäen LlamaIndexiä retrieval-augmented generation -ohjelmistokehyksenä ja OpenAI:n GPT-3.5 Turbo-mallia sekä Ada-tekstiupotusmallia luonnollisen kielen käsittelytehtäviin.

Verrattaessa ohjekeskukseen, tekoälyjärjestelmän heikkoudeksi nousi suomen kieltä tukevien avoimen lähdekoodin kielimallien puute sekä vastausten vaihtelevaisuus. Se tarjosi enemmän mahdollisuuksia yksilöllisen avun ja helpon ylläpidon suhteen. Testaus osoitti, että tekoälyjärjestelmä ei toimisi hyvin itsenäisenä järjestelmänä, mutta yhdistettynä perinteiseen ohjekeskukseen se osoittaa potentiaalia nopeaan ja personoituun asiakastukeen. Tuotantovalmis järjestelmä vaatii lisää kehitystä ja tutkimusta.

Asiasanat:

asiakastukikeskus, tekoäly, suuret kielimallit, retrieval-augmented generation

Contents

Figures

List of abbreviations

Abbreviation	Explanation of abbreviation
AI	Artificial Intelligence
API	Application Programming Interface
CSS	Cascading Style Sheets
CSRF	Cross-Site Request Forgery
FAQ	Frequently Asked Questions
HTML	Hypertext Markup Language
LLM	Large Language Model
NPM	Node Package Manager
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
RAG	Retrieval-augmented Generation
SQL	Structured Query Language
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

1 Introduction

This thesis is a commissioned project for Arcista, a cloud software designed to streamline information management and storage for businesses, owned by Älyarkisto Oy. This thesis refers to Arcista's version 2.2.0703. The issues and findings presented in this thesis may not apply to future versions of Arcista. The commissioner of this project is Taika Group Oy, the parent company of Älyarkisto Oy.

The primary motivation for the project stems from Arcista's lack of a dedicated help centre. Some of Arcista's user companies have expressed interest in integrating AI features into Arcista, prompting this thesis as a first step towards exploring the feasibility of doing so.

With Arcista version 2.2.0703, the Arcista Support Team handles all inquiries, and users receive guidelines via email in PDFs. It is a complicated system, especially since the Arcista development team is constantly developing and adding new features to the software. The PDFs users receive can become outdated quickly.

A dedicated help centre would significantly improve the user experience by providing a site where users can easily access essential software usage information. It would also improve communication and provide a centralised source for frequently asked questions and guides that could be regularly updated with the software. This would enable users to troubleshoot common issues without having to contact support. This reduction in inquiries would free up the support team's time to address more complex problems and enhance overall efficiency. Additionally, it would decrease the volume of requests forwarded to the development team concerning the more complex module functions prone to usage errors. The development team could focus better on improving the software rather than addressing minor difficulties that could be resolved using guides and tutorials provided in a help centre.

1.1 Final Product

The final products of this thesis include a knowledge base-type help centre with a focus on providing a conceptual site for guide articles already existing as PDFs. As a proof-of-concept case, an alternate version of the base help centre was developed, integrating a retrieval-augmented conversational AI system into the help centre to explore the potential for future implementations. One of the purposes was to showcase the potential advantages and drawbacks of implementing this new technology for user support in applications with strict data privacy needs, like Arcista.

Both the help centre and the AI system were developed solely in Finnish, as Arcista is primarily targeted at Finnish companies, and the current version of Arcista 2.2.0703 does not support any other languages.

1.2 Objectives and Limitations

1.2.1 Objectives

This project has five main objectives.

1. Identifying the current situation and the solution
2. Developing a base help centre
    - developed to provide users with a centralised location to find all important information about Arcista software, as well as a baseline to compare the AI system to.
3. Developing a retrieval-augmented conversational AI system
    - Serves as a proof-of-concept case for research into using new technologies to improve the user support experience and internal processes.
4. Comparing the AI system with the base help centre

- This is done to provide information on the possible drawbacks and advantages of the AI system and how it compares to the more traditional base help centre.

5. Documenting the results for future reference.

1.2.2 Limitations

Because of the way Arcista is structured as software, there are some limitations set for this project. Specifically, these limitations have an impact on the component selections for the AI system.

1. All data must stay inside Finland.
    - Each component used needs to be local or self-hostable.
    - Avoid any components that rely on cloud services.
2. Prefer open-source solutions with an MIT license.
    - Ideally, there are no additional costs associated with licensing or service fees.
    - This is not a hard limit, but rather a preference.
    - Due to a lack of suitable large language models, this has to be overlooked to develop a functioning proof-of-concept system.

2 Technologies

2.1 Artificial Intelligence

Artificial intelligence (AI) refers to technologies that enable computers, machines, and systems to simulate human intelligence and problem-solving capabilities. [1], [2], [3]

2.1.1 Brief History

The idea of "a machine that thinks" has been around for centuries. Alan Turing, known as the "father of computer science" and a "pioneer of artificial intelligence," published his paper Computing Machinery and Intelligence in 1950, marking the first appearances of AI. In his paper, Turing proposed a criterion that later became known as the Turing test to determine whether an artificial computer can think. [2], [4]

In 1956, the term artificial intelligence was coined at the first-ever AI conference at Dartmouth College, and the Logic Theorist, the first-ever running AI software program, was created.

Advances in neural networks gained popularity during 1960–1980 as a way to simulate human brain functions in computers. This era also saw advancements in natural language processing and expert systems. [5], [6], [7], [8]

Figure 1 summarises some of the important points of AI's history in a timeline.

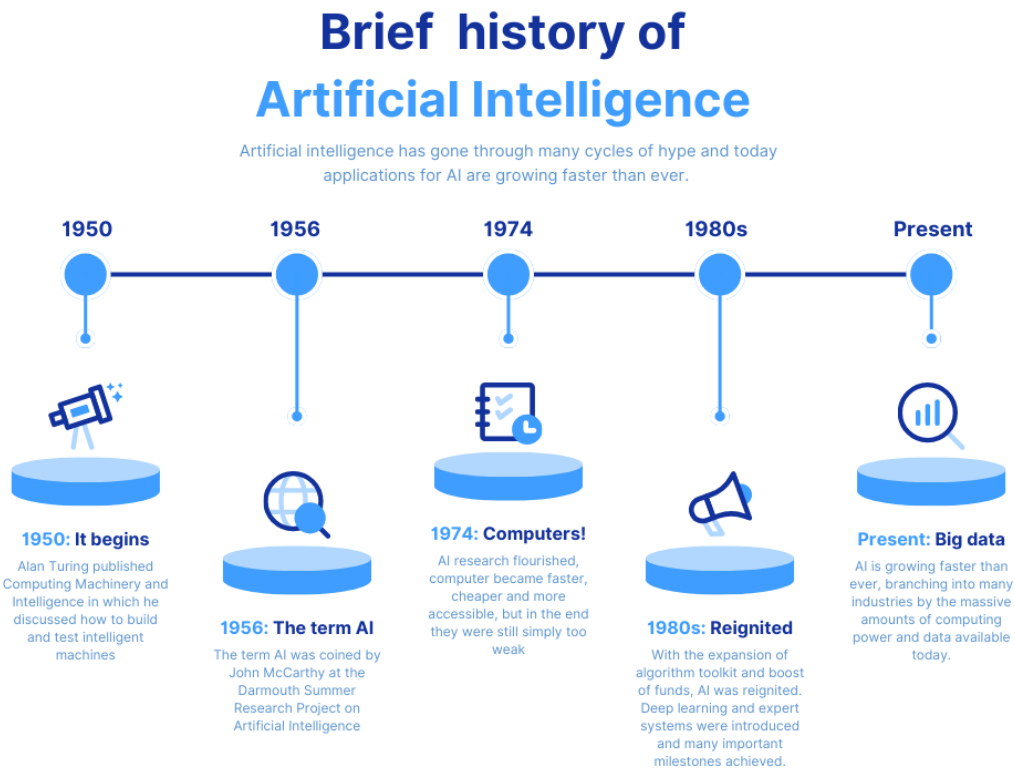

Figure 1. A brief history of AI.

Types of AI

Narrow AI, also known as *weak AI* or *artificial narrow intelligence*. Many modern AI applications use this type of AI, trained and focused on specific tasks. These include AI for enhancing graphics in video games [9], virtual assistants like Apple's Siri and Amazon's Alexa, and autonomous vehicles. [3]

Strong AI comprises of artificial general intelligence (AGI) and artificial superintelligence (ASI). AGI, or general AI, is a theoretical form of AI where a machine would possess human-level intelligence. It would be self-aware, able to solve problems, learn, and plan for the future. ASI would exceed the intelligence and capabilities of the human brain. Although strong AI is currently only a theoretical concept without practical applications, AI researchers are actively investigating its advancement. [2]

2.1.2 Machine learning and deep learning

Machine learning (ML) is a sub-discipline of AI that focuses on developing algorithms that allow the incorporation of intelligence into machines by learning from data and making predictions or decisions without explicit programming. [10], [11]

Deep Learning

Deep learning is a class of ML algorithms that use neural networks to solve complex problems. Neural networks are a type of ML model inspired by the function and structure of human brains. Similarly to human brains, neural networks are composed of interconnected layers of artificial neurons called nodes.

After initial training with labelled or structured datasets, deep learning models can process unstructured data in its unprocessed form. They are used to detect patterns or data groupings from unlabelled datasets, without human intervention. This is also known as unsupervised learning. Unsupervised learning models work independently to find the intrinsic structure of unlabelled data. They still require human interaction to validate the output variables. For example, an unsupervised deep learning model can detect that online buyers frequently purchase many things at the same time. The data analyst must still verify the logic of the recommendation based on the purchases. [12], [13]
These algorithms are especially beneficial for fields like computer vision and natural language processing, where the volume of labelled training data required to train models can be enormously large. [13], [14]

Classical, or "non-deep," machine learning algorithms rely more on structured datasets to learn using supervised learning. Supervised learning uses labelled training sets to teach models to get the desired results. Training datasets contain both inputs and correct outputs, allowing the model to learn over time. The

algorithm employs the loss function to gauge its accuracy, adjusting until it effectively reduces the error. [13], [14], [15], [16]

Figure 2 illustrates the relationship between the AI subsets.

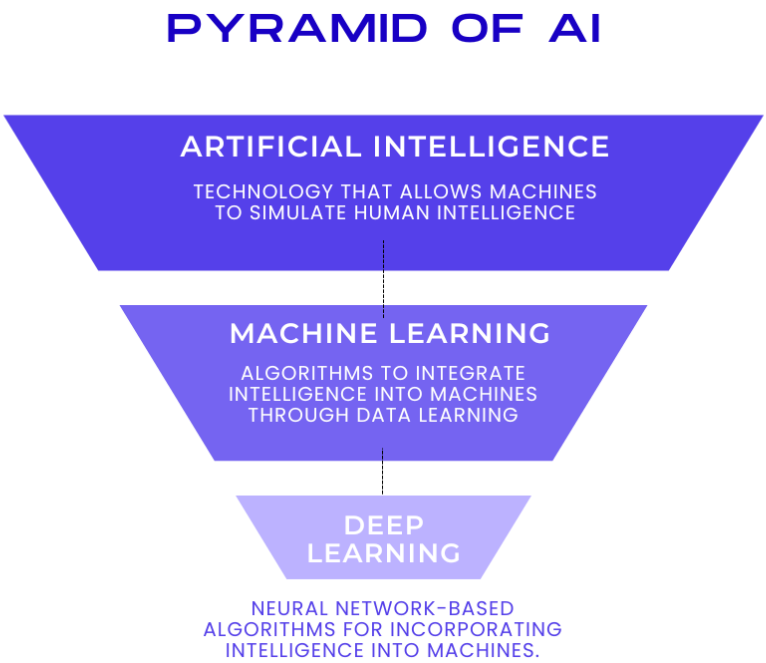

Figure 2. Artificial intelligence subsets in relation to one another.

Large Language Models

Large Language Models (LLMs) are machine learning systems that process and analyse large amounts of natural language data and respond to user prompts. [17]

LLMs are generative AI, meaning they are designed to generate, in this case, human-like language. LLMs can also perform other natural language processing (NLP) tasks, such as translation and summarization. They are frequently utilised in applications such as chatbots, sentiment analysis, and question-answering. [18], [19]

LLMs primarily belong to a category of deep learning systems called transformer networks. A transformer model is a neural network type that learns context by recognising patterns in sequential input, like the words in a sentence. A transformer usually consists of several transformer blocks, commonly referred to as layers. It has self-attention layers, feed-forward layers, and normalisation layers that all work together to parse input and forecast streams of output during inference. The layers can be stacked to create more complex transformers and sophisticated language models. Vaswani et al. at Google Brain and Google Research first introduced transformers in their 2017 paper, "Attention Is All You Need." [4], [20], [21]

Like any technology, LLMs are not without flaws. The most notable flaw that appears in almost all LLM models is hallucinations. When asked a question, to which the LLM can’t formulate a direct answer, the LLM may "hallucinate" or give inaccurate answers. These misinterpretations occur for a variety of reasons, including overfitting, training data bias, inaccuracy, and high model complexity. A noteworthy example of hallucinations occurred when Google's Bard chatbot falsely stated that the James Webb Space Telescope captured the world's first photographs of a planet outside our solar system. [22] Hallucinations can lead to the propagation of misinformation and should be considered when employing LLMs for fields requiring accurate responses based on data. [18], [22], [23]

When employing LLMs for consumer-facing products or research, a few potential risks must be considered. LLMs can be biased in their responses, generating stereotypical or discriminatory information. This occurs because the models are trained on enormous datasets, which may contain biased data or the data amounts to a biased pattern. Despite the safeguards in place to avoid this, LLMs can still occasionally generate harmful stereotypical or discriminatory information either on their own or because a user has been able to manipulate the LLM into producing such content. This is commonly referred to as prompt hacking, and it can be used to trick the LLM into creating improper or malicious content. It is critical to be aware of these potential risks while employing LLMs, particularly in consumer-facing applications. [18], [23]

2.2 Retrieval-augmented Generation

Retrieval-augmented Generation (RAG) is an architectural approach to improve the accuracy and dependability of generative large language models by leveraging resources that were not in the original trained dataset. [24]

LLMs, particularly open-source ones, are commonly trained on large datasets with publicly available information. They are not trained on company-specific and confidential data, which is often distinctive to the problem the company is attempting to address. This can lead to issues when trying to use LLMs for tasks that require an understanding of company-specific nuances and language. The LLM may not understand and handle the requests effectively, providing inaccurate responses, or hallucinations or it may not provide answers at all. Often, the solution to this is to fine-tune the language model based on the company's specific data. But fine-tuning creates its own set of challenges, such as requiring large amounts of labelled data and expertise in machine learning. Unsupervised fine-tuning and training can lead to unexpected and unwanted results such as biased models. Fine-tuning also does not address the issues of hallucinations and untraceable reasoning very effectively. [24], [25]

Retrieval-augmented Generation is a simpler, more popular way to reduce hallucinations and use company-specific data, providing it as part of the prompt that you query the LLM model with. Instead of relying primarily on knowledge obtained from training data, an RAG pipeline extracts useful information from external sources and combines static LLMs with real-time data retrieval. This allows the model to access a wider range of specialised information, which can be easily updated later. This approach results in more accurate and relevant responses to user queries. [24], [26], [27]

A typical RAG pipeline can be divided into five key stages [24]:

Loading: the process of bringing data into your pipeline from where it currently resides, such as text files, PDFs, another website, a database, or an API.

Indexing: the process of constructing a data structure that allows the data to be queried. For LLMs, this almost invariably entails developing vector embeddings, numerical representations of the meaning of your data, and a variety of other metadata tactics to make it simple to identify contextually relevant material.

Storing: Once the data is indexed, it is important to store the index and metadata to prevent the unnecessary re-indexing.

Querying: At its most basic, querying is simply a direct call to an LLM: it can be a question and an answer, a request for summarization, or a much more complicated instruction. More complicated questioning may employ repeated/chained prompts and LLM calls, or a reasoning loop spanning across numerous components.

Evaluation: An important phase in any pipeline is determining how effective it is in comparison to other tactics or when making changes. Evaluation provides objective metrics for how accurate, dependable, and timely the responses to inquiries are.

2.3 Application Core

This section provides basic information on the frameworks and technologies used to develop the application core.

2.3.1 Laravel

Laravel is an open-source PHP-based web application framework that includes a variety of pre-built solutions for more complicated concepts like authentication and authorization. Laravel advertises itself as a “progressive” framework that grows with the developer’s skillset. Although Laravel is inherently more of a backend framework, it allows developers to make full-stack applications by either leveraging PHP and Laravel’s built-in Blade templating engine or JavaScript frameworks such as Vue and React.

Laravel follows the MVC architecture, meaning the basic structure of applications is divided into three elements: models, views, and controllers.

Models are responsible for managing the data of the application and fetching it from the database. For this Laravel offers helpers like query builder and Eloquent ORM (Object-Relational Mapping). It is Laravel's built-in ORM implementation that provides a simple ActiveRecord implementation for working with the application database. Each database table has a corresponding "Model" that is used to interact with that table. Laravel also offers migrations that are like version control inside the application for the database, allowing development teams to modify and share the application's database schema.

Views are responsible for presenting the data to the user in a readable format. For creating views Laravel comes with Blade, a simple yet powerful templating engine. Blade is an extremely lightweight templating engine with its Blade syntax that provides a convenient, short syntax for displaying data, iterating over data, and more. Blade makes web development in Laravel easier by allowing the creation of dynamic and reusable HTML structures and combines the versatility of PHP with the readability of HTML, by supporting a variety of features such as

embedding PHP code, conditional expressions, and loops directly in the HTML code. Additionally, it supports and encourages the creation of reusable view components. It is easy to divide the web pages into smaller Blade templates that can be combined to create whole layouts. [28]

Controllers are responsible for handling user input and interactions. They are the middleman between models and views, processing user requests and updating the model as needed.

The MVC architecture chart in figure 3 illustrates the basic operation style of software developed with MVC architecture. [29]

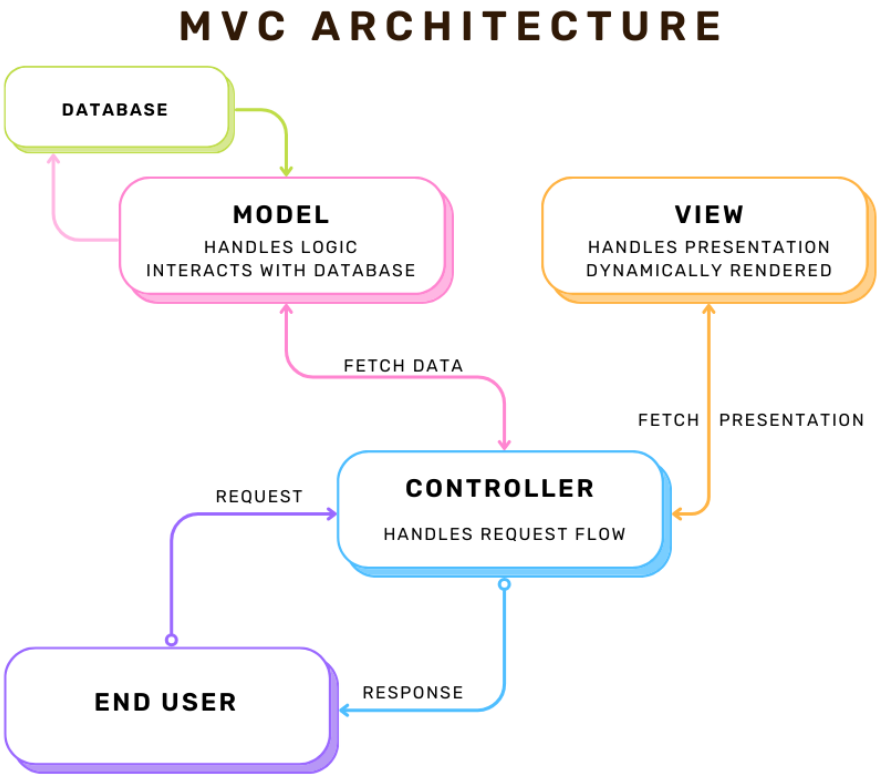

Figure 3. Model-View-Controller architecture chart.

Even though it is not explicitly stated, routing is an essential part of the MVC approach. Routing is a mechanism where an HTTP request is linked with a certain logic, typically a controller action. When a request comes into Laravel, it is dispatched to a route that then decides which controller action to call. The

controller action then interacts with the necessary models and data and loads a view to send as the HTTP response. So, in a sense, routing is the first step in handling a request and deciding which controller and subsequently, which view should be used.

Most of Laravel's built-in features interact using Artisan Console, a command-line interface that includes numerous useful functions for basic operations like model creation, migrations, testing, and more. The Artisan console also includes support for task scheduling and custom commands for any application-specific commands.

Laravel also provides several ways to secure your application, including a simple way of organising authorisation logic and controlling access to resources. [28]

2.3.2 Tailwind CSS & UI

Tailwind CSS is a free, open-source, utility-first CSS framework packed with multiple classes that can be composed to build any design, without ever leaving the HTML code. Tailwind CSS takes pride in its ability to be very responsive, performant, and low-level “API for your design system”. [30]

Tailwind UI, on the other hand, is a commercial collection of responsive UI components created with Tailwind CSS. It consists of a library of UI components, such as forms, cards, navigation, and more, that were developed and coded by the creators of Tailwind CSS. These components are fully responsive and customisable, and they can be copied right into your project. They also support JavaScript frameworks such as React and Vue.

It's important to note that, while Tailwind CSS is open source and free to use, Tailwind UI is a paid proprietary product. The license supplied by Taika Group allows for its use in this project. [30]

2.4 Databases

A database is a collection of organised data that is stored in a computer system. It is designed to store, manage and manipulate large amounts of data in a structured framework.

2.4.1 Relational Database

One of the core components of the Help Centre is a relational database.

A relational database stores data in tabular format, with rows and columns reflecting different data attributes and the relationships between them. Relational databases use Structured Query Language (SQL) programming language for storing and processing information. SQL statements can be used to store, update, remove, search, and retrieve information from the database. They can also be used to maintain and optimise database performance. [31]

This project relies on the MariaDB Server and HeidiSQL.

MariaDB Server is one of the most widely used open-source relational databases. It was created by the original developers of MySQL and is promised to remain open source. Most cloud providers include it, and Linux distributions make it the default. It is founded on the principles of performance, stability, and openness, and the MariaDB Foundation assures that contributions are approved based on technical merit. [32]

The Windows version of MariaDB installation includes HeidiSQL, a free Windows client that allows users to view and edit data and structures from computers running one of the following database systems: MariaDB, MySQL, Microsoft SQL, PostgreSQL, and SQLite. HeidiSQL, invented by Ansgar in 2002, is one of the world's most popular MariaDB and MySQL utilities. [32], [33]

2.4.2 Vector Database

A Vector Database or a vector store is a collection of data stored as mathematical representations. Vector databases are utilised mostly with machine learning models, making it easier for machine learning models to remember previous inputs, allowing it to be used to power search, recommendations, and text generation use cases. Data can be identified using similarity measures rather than precise matches, allowing a computer model to interpret data contextually.

Vector databases function by presenting each object or item as a vector, whether it's an image, a video, a phrase, a document, or any other data type. These vectors are likely to be extensive and complex, describing each object's location across dozens, if not hundreds, of dimensions. [34]

The vector database used in this project is LlamaIndex’s in-memory SimpleVectorStore, which is initialised as a part of the default storage context. LlamaIndex uses this by default, as it is great for quick experimentation. For actual production-grade applications, a more robust, external vector store might be a better solution. LlamaIndex has over 20 different integrations for vector stores. [24]

3 Development

3.1 Help Centre

3.1.1 Project Setup

The first step to begin development, was to install the main framework Laravel. As Laravel is made with a popular general-purpose scripting language PHP [35], it is essential to have a correct version of PHP installed on the development environment. To manage and update all the package dependencies, Laravel requires Composer to be installed. Composer helps with declaring, managing, and installing dependencies on PHP projects. A new Laravel project can be created using Composer's "create-project" command, or by globally installing the Laravel installer, as was done in this project.

In the Laravel documentation, it is also recommended that Node.js and Node Package Manager (npm) be installed. Node.js is an open-source, cross-platform JavaScript runtime environment. [36]

Npm is similar to Composer as it is a package manager but for Node.js packages. Node.js and npm are especially important to this project as some of the core frontend technologies, like Tailwind CSS and Tailwind UI depend on it. Tailwind CSS is a Utility-First CSS framework for creating modern websites with varying designs. Tailwind UI is a collection of ready-made components created by Tailwind CSS's authors and made with Tailwind CSS classes.

Tailwind CSS is installed through npm alongside a PostCSS Autoprefixer. PostCSS is a tool for transforming styles with JavaScript plugins and Autoprefixer is a plugin for it that parses the CSS and adds vendor prefixes to the CSS rules. Vendor prefixes are special prefixes added to CSS rules to make them render correctly in each type of browser rendering engine. For example, Google Chrome uses the Chromium engine, Firefox has the Gecko engine and Apple's Safari has

the WebKit engine. Each of these engines handles and renders CSS with a little bit different names so they need their prefixes to work correctly.

Tailwind UI does not need to be installed to function as it is a component library. However, some of the components used required the installation of Tailwind CSS's official plugins, such as Typography and Forms. These plugins were installed through npm and added to the “tailwind.config.js”- file as a plugin so that the base Tailwind CSS installation can recognize it.

For a localhost development server, this project requires two development servers running simultaneously. To get the site up and running, a PHP artisan development server is required. To get the site to render properly, especially with Tailwind CSS, it is important to have a npm server running as well, as it compiles the CSS and runs a series of scripts related to the development server setup. By default, Laravel uses Vite to compile the application's CSS and JavaScript into production-ready components and that too relies on Node.js and the npm server.

3.1.2 Frontend

To build the required views, new Blade files were created to the "resources/views/"-folder with the file ending “. blade.php". These files will include the HTML, with Tailwind CSS classes in the markup and ready-made components from the Tailwind UI library.

This project uses Blade, Laravel's built-in templating engine, to create the views for the user interface. Blade files or Blade views, albeit containing HTML, are inherently PHP and compiled into plain PHP. This enables really useful shortcuts for standard PHP control structures such as if statements, loops, and displaying data supplied to the view by simply wrapping it between two curly braces making it a Blade echo statement. Any PHP function results can be echoed using the Blade echo statement and they are automatically sent through PHP’s functions to prevent cross-site scripting (XSS) attacks. Using Blade for views also allows developers to define sections, template inheritance and components. Defining a layout or section is useful when working with larger applications as it allows

developers to create child views that inherit a section or layout mitigating repetition.

This site's design language is loosely based on the design of Arcista and its website, as well as the Tailwind UI component's design style. Before starting the actual creation of the pages, some layout inspiration research was made from Dribbble.com. [37] It is a community platform where creatives and designers can share their designs, connect, grow, and find work. The main layout inspiration leaned into the most was this simple Help Centre homepage design shown in figure 4 created by Valentin Salmon.

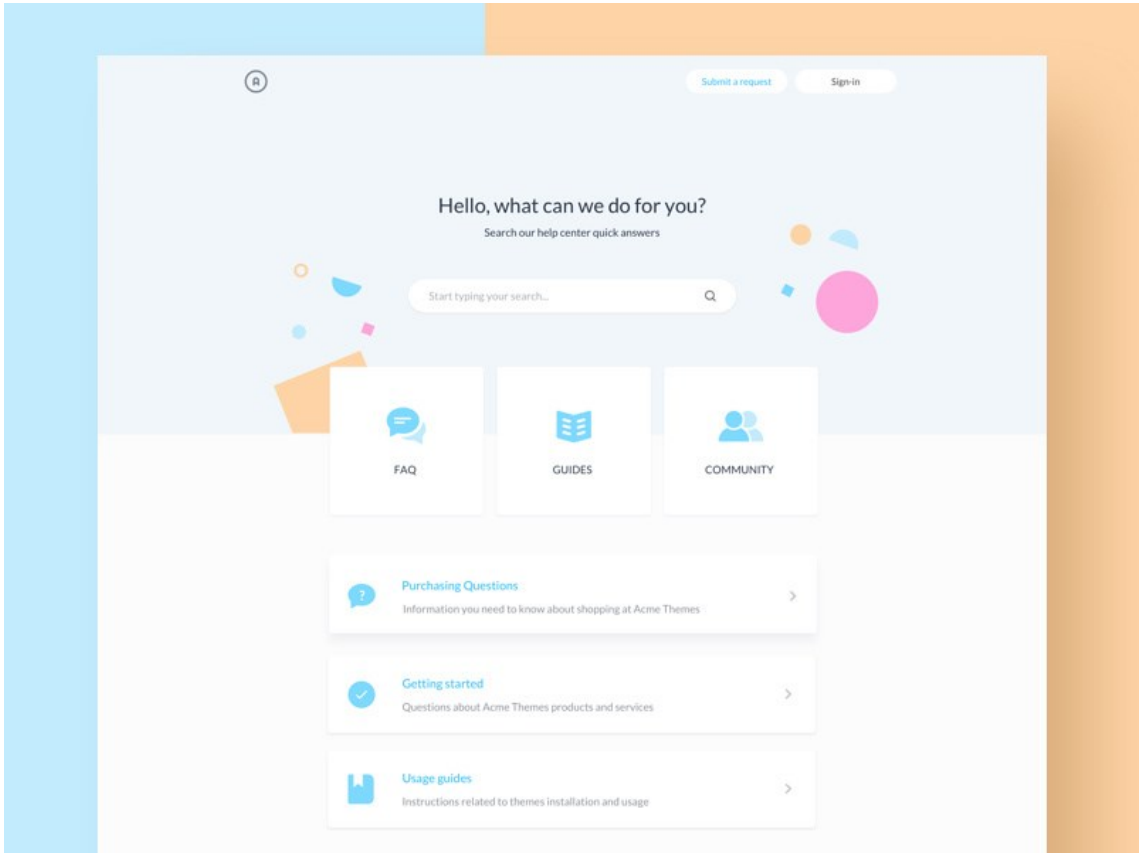

Figure 4. Help centre - Homepage by Valentin Salmon on Dribbble.com.

The homepage of the site is a simple welcome page with a little message, a quick search bar with the possibility to delimit to only certain categories and quick links to things like starter guide articles, FAQs, and support contacts. The figure 5

illustrates the final layout and design of the Help Centre's homepage in the light theme variant.

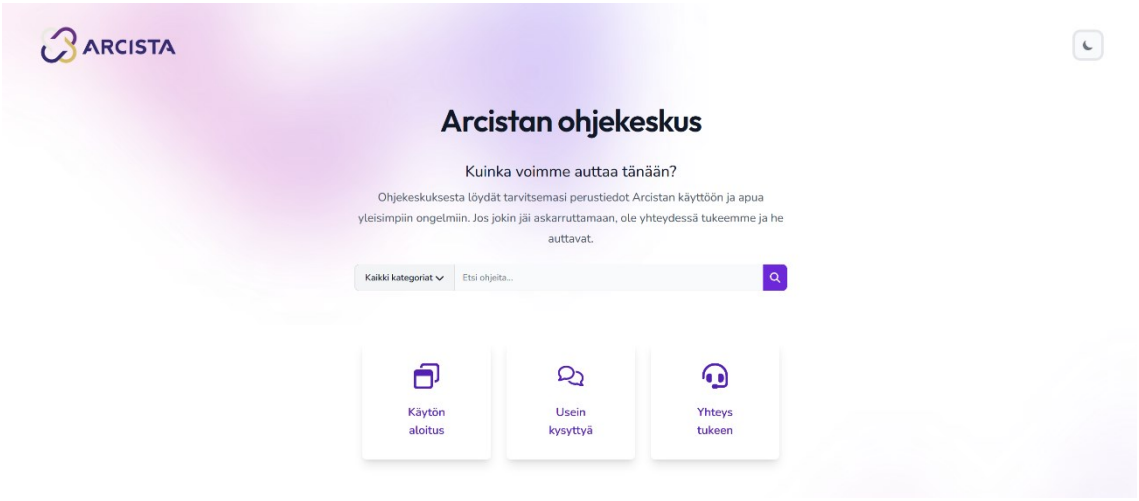

Figure 5. Top of the homepage, light variant.

Each page was designed to work with a light and dark theme variant, changed by a button toggle in the upper right corner. The figure 6 demonstrates the upper half of the homepage in the dark theme variant.

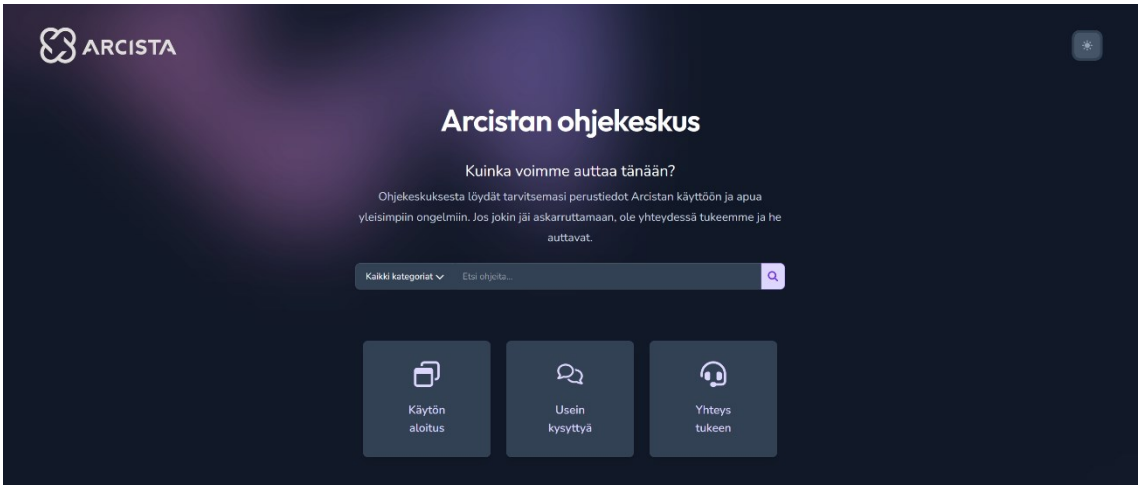

Figure 6. Top of the homepage, dark variant.

Continuing ahead in the bottom part of the homepage displayed in figure 7, the category-specific buttons are presented with a footer including important

information links such as about, privacy policy, licencing, and contact, as well as the required copyright information.

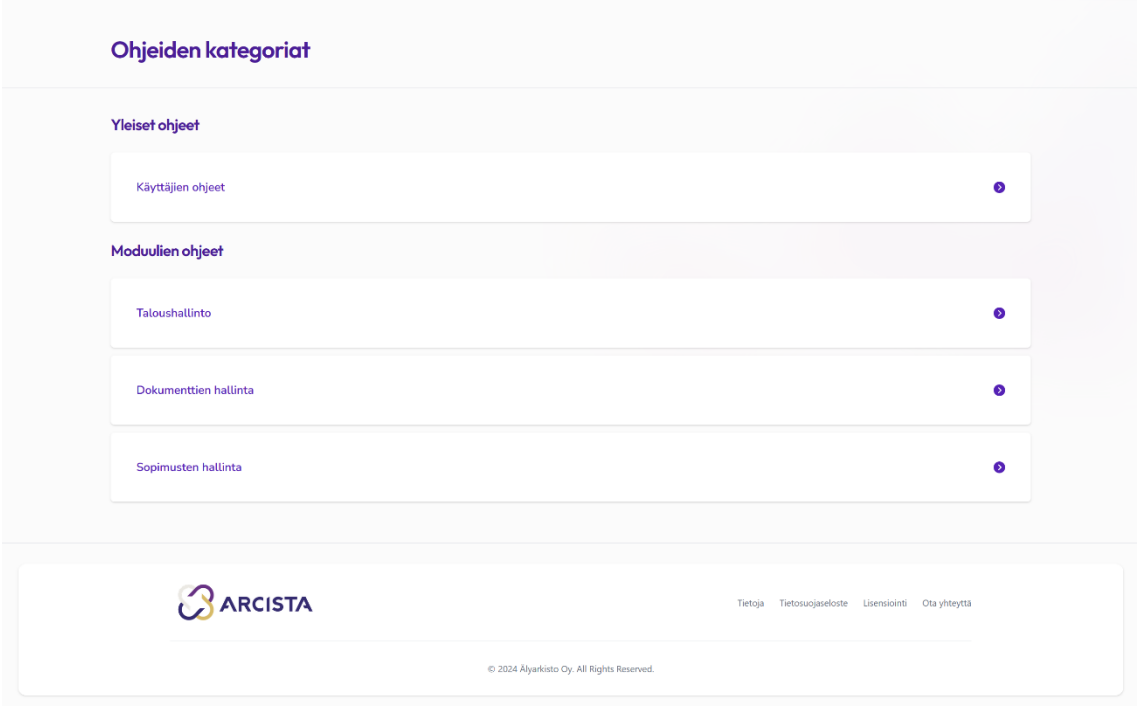

Figure 7. Bottom of the homepage, light variant.

The article collection pages make use of the Blade views the ability to define layouts that can be extended through multiple views. The collection page base consists of two sections: the sidebar on the left and the search bar with the theme toggle area on the top. This was constructed as its own Blade view that has a separate @yield('content')-tag in the spot where the additional child views will be placed when rendered. The figure 8 demonstrates the article collection page layout without any child-view rendered content.

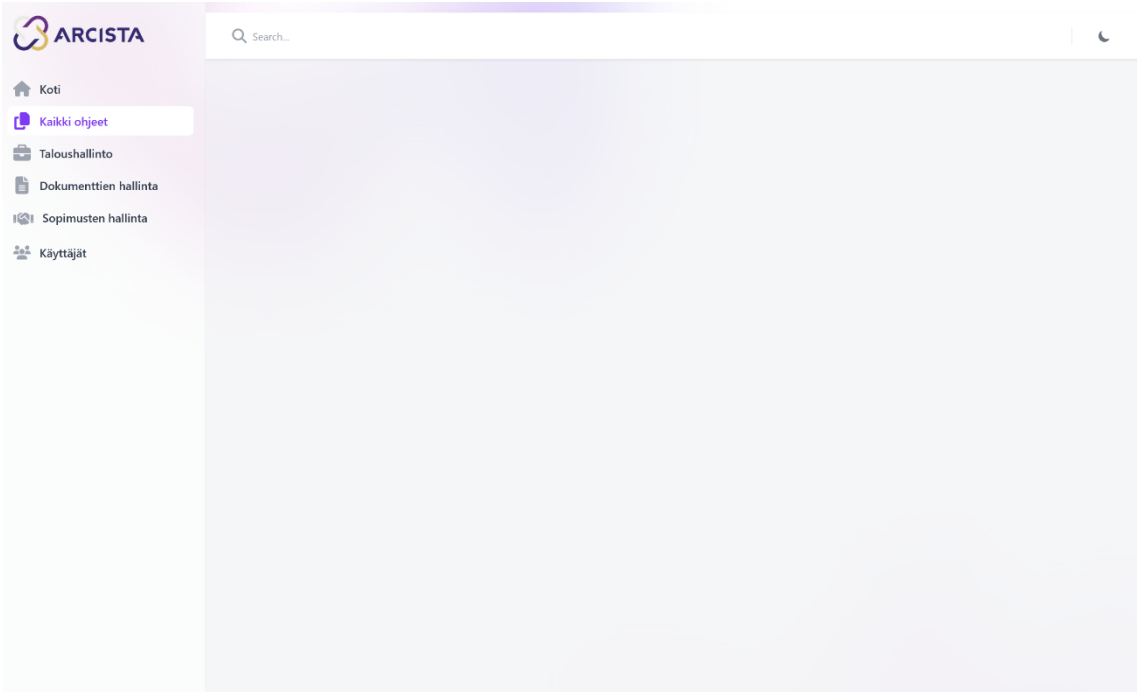

Figure 8. Article collection page base layout.

This way no matter which category the user selects, the sidebar and the top sections always remain the same. The figure 9 demonstrates the article collection page and category selection with child view content with guide articles from the document management category.

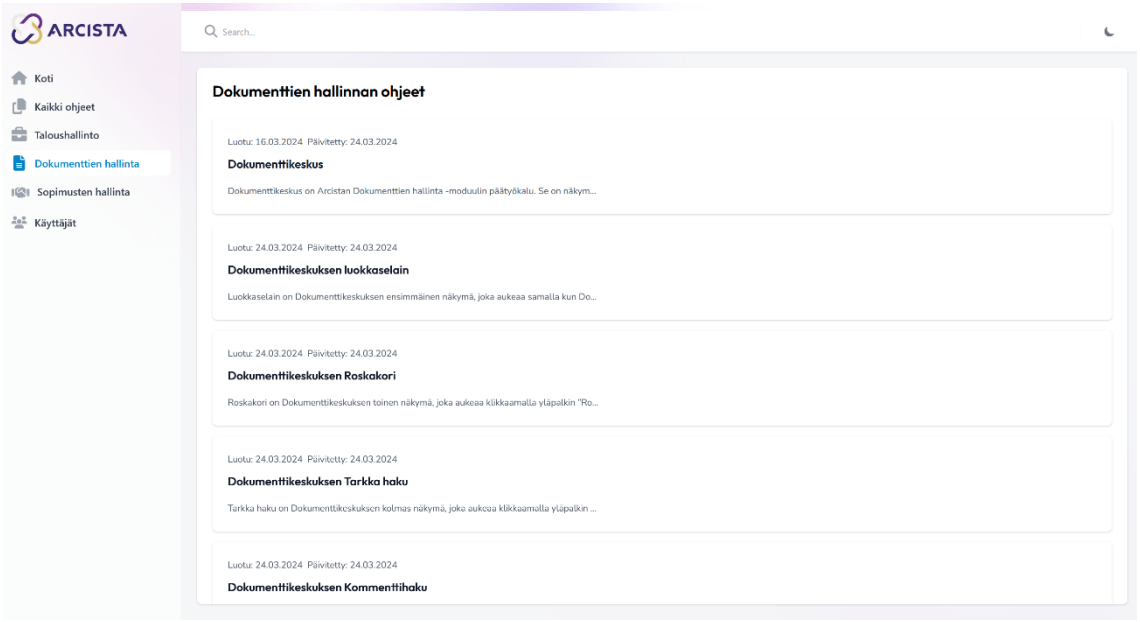

Figure 9. Article collection page, category selection.

Each of the sidebar's category buttons has its own hover and active state colours corresponding to the actual Arcista module colours. Because of their universality, the home, and all guides category buttons are the same purple. The home button directs the user back to the landing page. The figure 10 demonstrates the selection of colours in action.

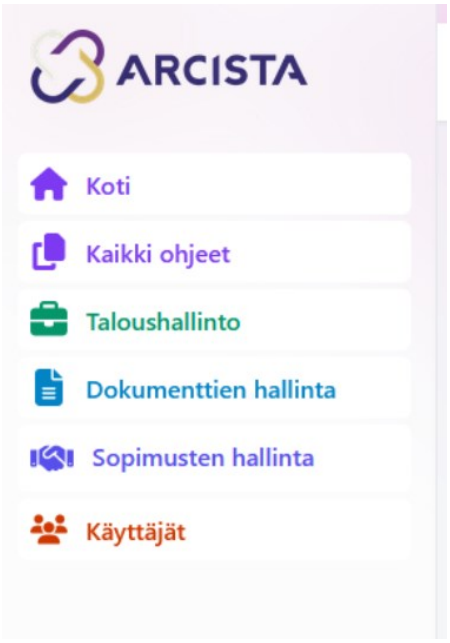

Figure 10. Category selection colours.

Arcista's current version has three modules: financial management, document management, and contract management. These modules are stand-alone and can be purchased separately. The guide articles are organised into categories based on their subject module. Guide articles that discuss user-related subjects, such as login, are also divided into separate categories for ease of use.

When a user hovers over a listed guide article, it is separated from others by slightly changing its colour. Figure 11 demonstrates this state before a user has clicked on any of the guide articles.

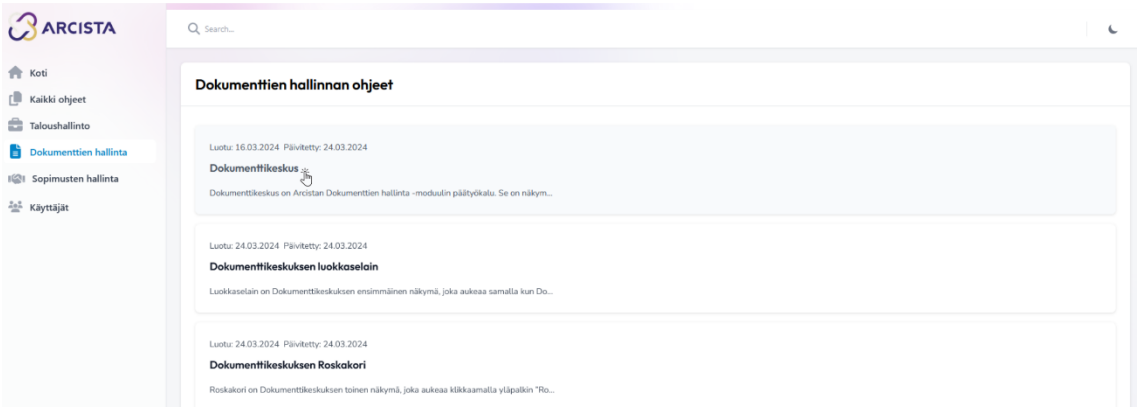

Figure 11. Hover over the listed article.

When a user clicks on any of the listed articles on the collection page, the article-specific page is opened. The article-specific page displays the article's contents, including a back button, article category, and article title. Everything on this page is fetched from the database when the page loads. Figure 12 demonstrates the article-specific page with a guide article open from the document management category.

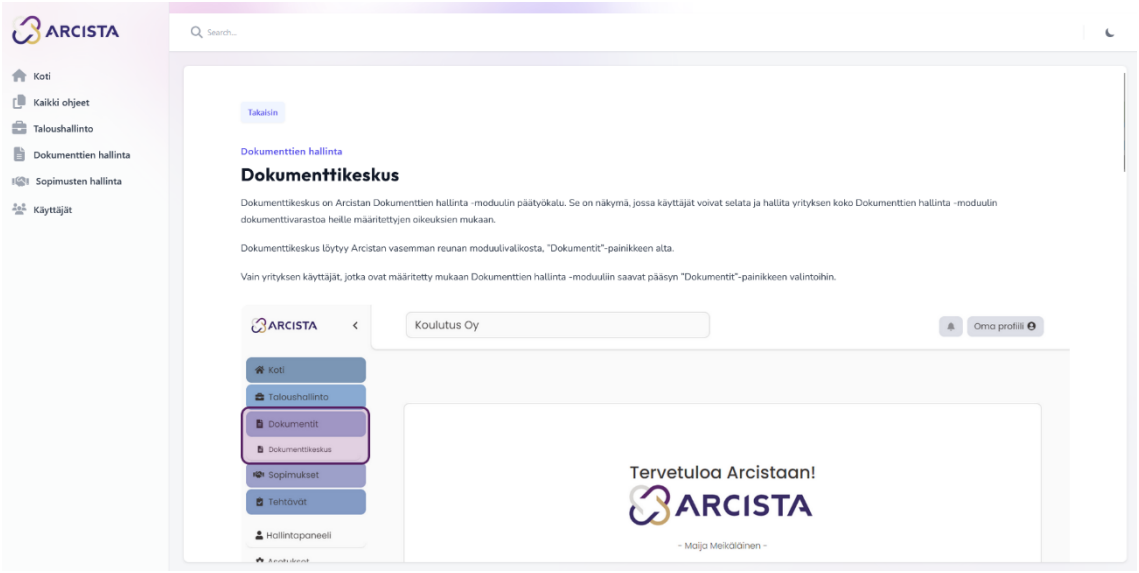

Figure 12. Article specific page.

3.1.3 Backend

Routing

For the server to be able to point to the right view, for example when entering the site, it needs to understand how to interpret incoming URLs and map them to the appropriate functionalities within the application. In Laravel, this is achieved through a routing system. By defining routes, we establish a set of instructions that translate URLs into specific actions responsible for handling user requests and generating the corresponding responses. This structured approach to routing streamlines application development promotes code maintainability and facilitates the separation of concerns between the presentation layer (views) and the functional logic (controllers).

Routes are defined in the “routes/web.php”-file. This file defines routes for the web interface that are then assigned to the web middleware group, which includes features such as session state and Cross-Site Request Forgery

protection. In its most basic variant, a route takes a URI, the user request and a closure, which is typically a function. The routes defined in the "routes/web.php" file can be accessed by typing the defined route's URL into a browser. Route functions can be defined directly in the "routes/web.php" file or a separate controller. [28], [38]

For the simplest routes, such as the landing page, the route functions were defined directly in the "routes/web.php" file, as they only return a Blade view with no need for additional data. For routes with more complex structures, such as the guide category route, separate controllers were established to handle the request functions.

Guide Storage

The guide articles need to be accessible, thus stored somewhere so they may be efficiently queried when needed. An SQL relational database powered by MariaDB Server was the solution in this project because it can be easily adjusted to fit Arcista's underlying architecture. Relational databases excel in storing information with inherent relationships, making them excellent for situations in which articles are connected based on topic, category, or other relevant criteria. This structured method helps to organise the article information in a way that makes it easier to search and retrieve. MariaDB Server is an open-source relational database system to create databases. HeidiSQL provides a user-friendly interface to manage the created database schema and to insert, update, and query the stored information.

The database has two main tables: guides and categories. The guides table stores the information about the guide article snippets in 6 columns with the g_id column being the unique auto-incremented primary key. The title column saves the title of the article snippet, making it easier to use in various locations on the page. The content column stores the actual guide content formatted in Markdown format, which can then be converted to HTML after being obtained from the database. Markdown is a lightweight markup language for text formatting. The

main idea behind Markdown is that it’s easy to read and write. The markdown can easily be converted into valid HTML. This enables quick styling with Tailwind CSS's Typography plugin.

The “category_id” column stores the category’s number to which the article snippet belongs. This column is linked to the categories table id column with a foreign key relationship. The columns created_at and updated_at offer useful additional data about the articles. They are completely optional and hold more value in future versions with administrative toolsets.

The categories table contains two columns: id and name. The id column reflects each category's unique number, which also serves as the table's primary key. The name column contains the written name of each category. This is mostly used on the article collection and article-specific pages to always indicate which category's article the user is reading.

The figure 13 represents the database schema to visualise the database structure more clearly.

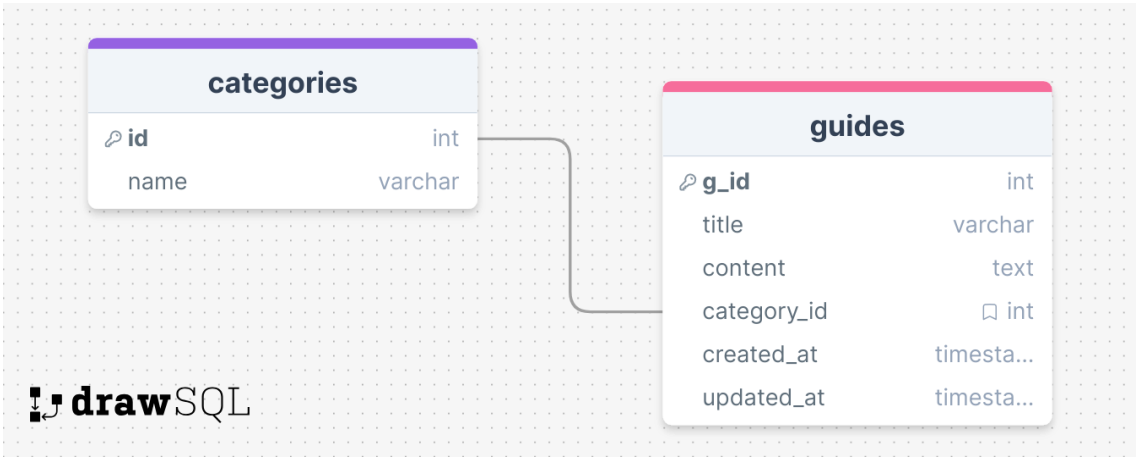

Figure 13. Database schema

Retrieval

When a user enters one of the pages with dynamically rendered content, such as the articles, the data needs to be retrieved from the database. This especially

takes advantage of Laravel's ability to name routes. Named routes enable the convenient recognition and generation of URLs or redirects for specific routes by their assigned name. It is essentially like giving the route a nickname, defined in the “routes/web.php”-file. The route can later be referred to with said nickname for example, inside a Blade view for a button link. In this particular case, the route name is used to match and select which category guides are obtained from the database with an if-else structure.

The routing done in the web.php file is instructed to hand off specific URL functions such as the categories, to the Guide controller to handle. The Guide Controller has a function, named index, that takes the received request and checks if the requested route matches any of the specified category route names with the “routeIs()”-method. If it matches one of the specific named routes, such as document management, Laravel’s Eloquent ORM is used to query the database for the specific articles in that category based on the category’s id.

Once the data has been correctly obtained from the database, the user is redirected to the correct view, and the retrieved articles are submitted alongside the view as an associative object array, that can then be used to access the data inside the Blade view.

If the route name does not match any of the specified category route names, thus all stored articles and category information are retrieved and provided as an associative object array with the appropriate view.

Named routes also enable easy linking in Blade files with the Blade syntax. This is used on many occasions in the project, like the Guide collection page sidebar to toggle the correct button to active, indicating which page the user is now on.

Markdown processing

The article-specific page consists mostly of dynamic content fetched from the database. This provided a unique challenge of how to process and style the pure Markdown retrieved from the database. Similarly to the other pages, the base of

the article-specific page consists of a Blade view with HTML and Tailwinds CSS utility classes, extending the sidebar layout. This page is noticeably simpler in terms of code compared to the other pages, as it only contains a back button, category name, article title and the content itself.

The figure 14 represents the article-specific page view with a guide article from the Contract Management -category.

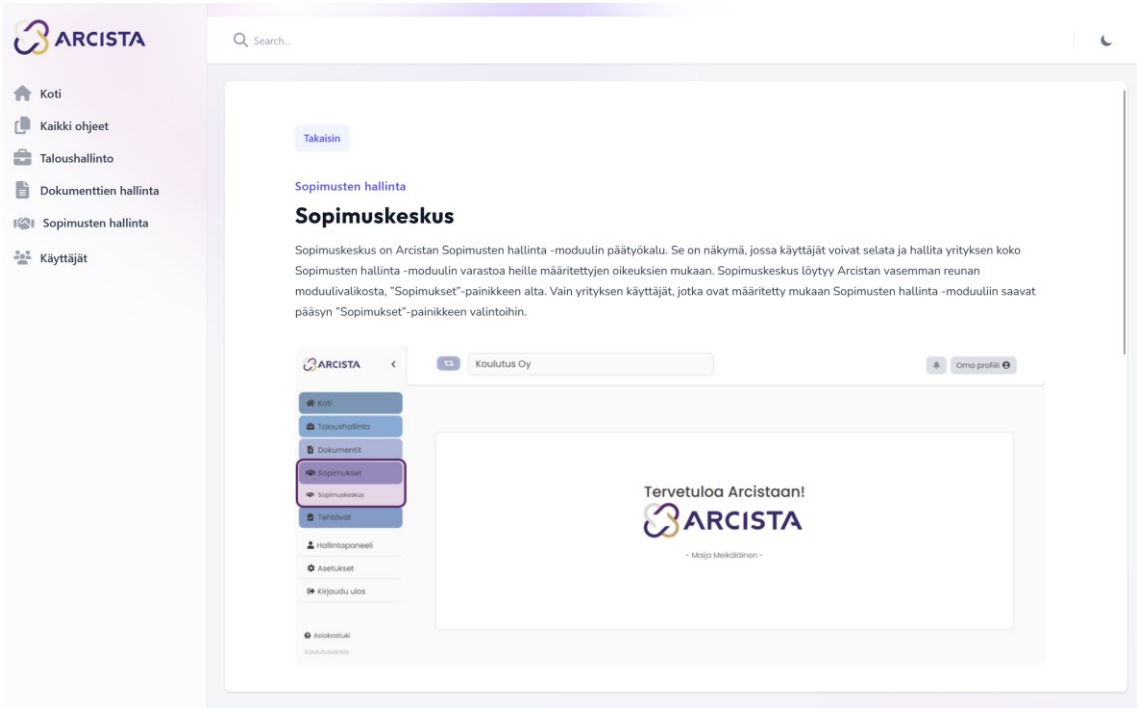

Figure 14. Article specific page

To convert the markdown formatted text to HTML, the “Str::markdown()”-function was used from Laravel's Str-utility class. It is a class that provides static methods for working with strings.

For styling, the transformed HTML, the Tailwind CSS Typography plugin includes a set of utilities for designing rich text material like blog posts and markdown files. When applying the prose class to an element, the Typography plugin styles all of its child elements, including headings, paragraphs, lists, blockquotes, tables, and more. This allows for easy application of consistent typographic styles to a large

block of text. These automatic stylings can also be tailored to each application using element modifiers.

3.2 Retrieval-Augmented Conversational AI System

3.2.1 Frontend

The AI system can function with two different style possibilities. One is a full-page version with an animated robot mascot. The robot mascot was designed specifically for Arcista and appears mainly on the maintenance screen of the software. The animations such as idle, waving, and thinking animations were added since studies have shown that adding anthropomorphic cues to a virtual chat agent can be beneficial for achieving a positive user experience in the cases of mishap or malfunction. [39], [40], [41]

The figure 15 illustrates the full-page chat screen.

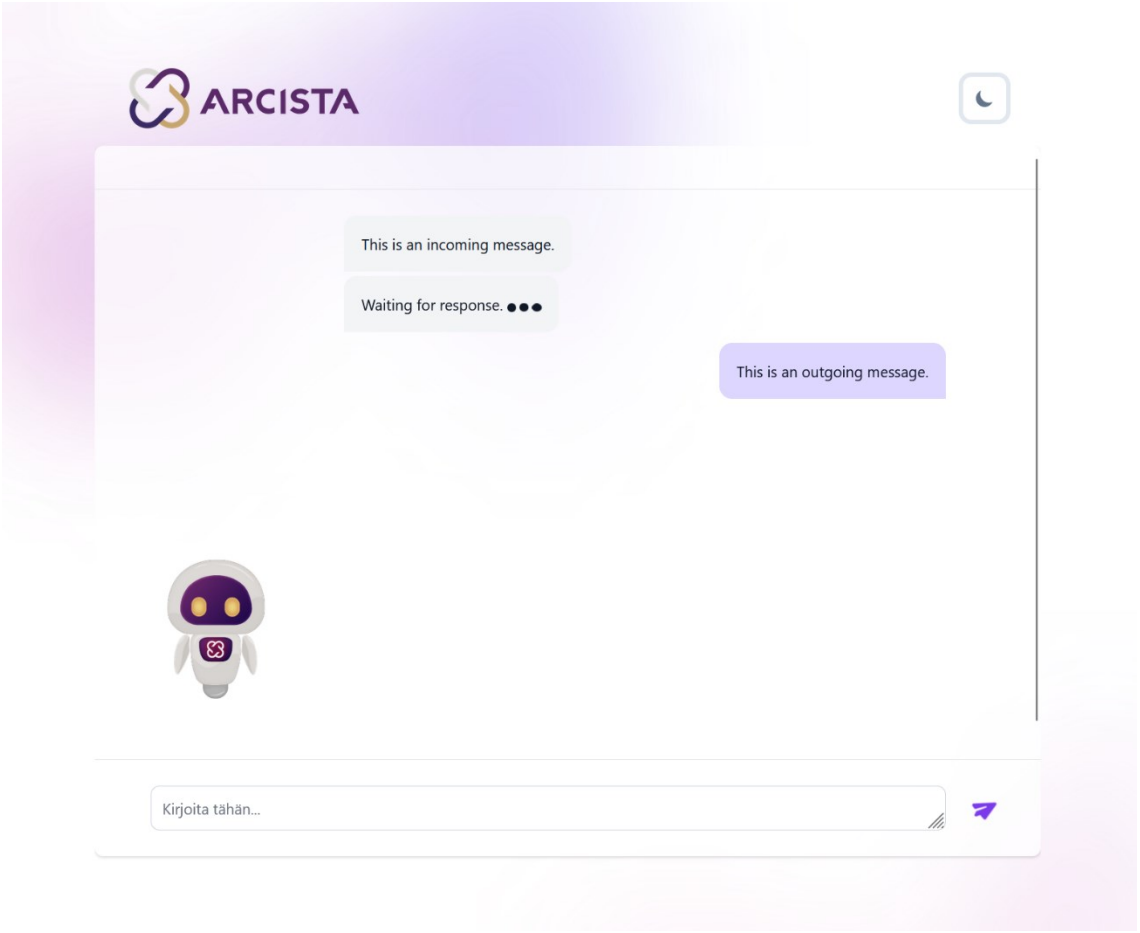

Figure 15. Chatting screen.

The figure 16 illustrates the chat widget possibility. This type of widget is already widespread on many websites and is rather simple to access via a static button. The animated robot mascot could also be added to this version of the chat. To use space more efficiently, it was not designed into this iteration of the chat widget.

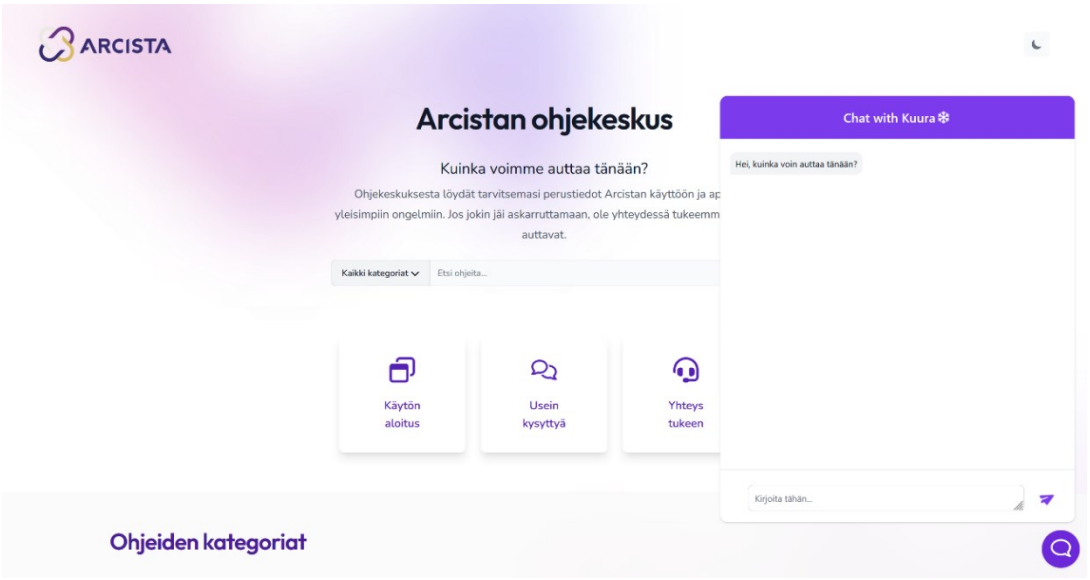

Figure 16. Chat widget on the homepage.

3.2.2 Backend

The inner workings of the AI system consist of multiple parts. The messages are handled on the frontend server by JavaScript. When the user presses the enter or the send button after typing their question, a handler function processes it. The handler function takes the user's query and if it’s not empty, the message is assigned a class, in this case, “c-outgoing”. This means that it’s user user-originated message and should be styled accordingly.

The handler function passes the user’s query and the class it was assigned, forward to the message creator function. The message creator function is responsible for creating the list element and the div element, inside of which the message is going to be put. The list element in HTML is used to group a set of related items in lists. In this case, the listed items are div elements styled with CSS to look like chat bubbles. The div element is by default a block element, which means it takes up all available width and includes line breaks before and after. It can be styled and is commonly used to group web page sections, such as text and images. [42]

The message creator function assigns the necessary Tailwind CSS utility classes to the div, to style it properly to present the user a chat bubble. The final content, which contains a new list item with the style div and the user's message inside, is returned to the handler function that appends it to the chat frame. Appending is a way to dynamically add content to a web page after it loads. The Document Object Model (DOM) represents the structure of a web page as a tree of elements. One way to manipulate this structure is through the "appendChild"- method. This method, available in most modern browsers' JavaScript engines, allows for appending a child element, like a paragraph, a div or a button, to an existing parent element within the DOM tree. This enables functionalities like adding new content to a page based on user interactions or updating content based on server responses received through Asynchronous JavaScript and XML (AJAX) requests. AJAX lets web pages communicate with servers in the background, without reloading the entire page. This is achieved by JavaScript code in the browser sending requests and receiving responses, often in JSON format, from the server. This allows for a more responsive user experience as content can be updated dynamically without interrupting the user's flow.

After the user's message has been appended and is visible, the handler function uses AJAX to pass the user query from the frontend server to the backend server, while creating a temporary waiting-message bubble to the chat. The AJAX makes a POST request to a route that can accept it and point it to the correct controller. A POST request is a specific type of communication method used within the HTTP protocol, the foundation of data exchange on the web. In an ideal situation, the controller receives the POST request and the contents of the user request. The controller pre-processes the user request into a form that it can then be given to the RAG pipeline. This pre-processing consists of sanitising by converting the query to UTF-8 encoding, eliminating any leading or trailing whitespace, and applying basic XSS protection via the "htmlspecialchars" -function. Once this is done, the input is then escaped for shell argument usage with the "escapeshellarg" -function. This prevents the shell from misinterpreting the string's contents, especially if it contains special characters or spaces. The shell

is used to run the Python script that is responsible for communicating with the LLM and producing a relevant answer based on the user's query and the additional information provided by the RAG pipeline.

The Python script is run through the "shell_exec" method, passing the sanitised and escaped user message as a shell argument. The "shell_exec" method is used since the script's output will most likely be multiple lines long, and it is necessary to obtain all of the lines. The script's output is captured and stored in the variable response. If the Python script produces no output i.e., the response is null, the method sets it to the message "Error executing Python script". This easily indicates to developers that something is wrong with the Python script. The method returns the response which can be the Python script's output or an error message.

The Python script contains LlamaIndex to facilitate the ingestion and indexing of documents, enabling the LLM to access and process information for tasks like question answering and conversation generation. LlamaIndex is the RAG framework used to implement natural language processing into the chat system. LlamaIndex can be thought of as a bridge between LLMs and the company data that exists in different formats such as PDFs.

The way LlamaIndex and RAG generally work can be compared to having a pile of unorganised documents and information in various languages. LlamaIndex acts like a librarian, taking this information and organizing it into a structured format called an "index." This index is a processed collection of data from the original documents and makes it easier for the LLM to find relevant pieces of information quickly. Different information comes in different forms, like text documents, website responses, or even data from databases. LlamaIndex uses data connectors to handle this variety of data types. These connectors act like translators, converting the information from its original format into a format the LLM can understand. LLMs, as powerful as they are, still need some guidance. LlamaIndex helps by providing context through system prompts. These prompts are like instructions that tell the LLM how to approach and interpret the information in the index. By preparing the information and guiding the

LLM, LlamaIndex can be used to build chatbots that can conversationally answer questions like it is utilised in this project, or to power search engines that can find relevant information from data collection much faster than a human could.

This project utilises the Python version of LlamaIndex, with OpenAI's GPT 3.5 Turbo as the default LLM and ada-002 text-embedding model to create the vector embeddings. Vector embeddings are numerical representations of the relationships of words, sentences, and other kinds of data. Vector embeddings transfer an object's main traits or attributes into a concise and organised array of numbers, allowing computers to quickly access the information. Similar data points are clustered closer together when transformed into points in a multidimensional space. The usage of vectors in machine learning enables the search for related objects. A vector-searching algorithm needs only to locate two vectors that are close together in a vector database. [43], [44]

The script starts by gathering the necessary tools such as the API key and the necessary packages that need to be imported. Next, the system sets the system prompt. Then, the script checks if it already has an index ready to use. If there is no index already existing, the script gathers the documents provided in a local folder and builds the index using a tool called "SimpleDirectoryReader". Data connectors take the information and convert it into a format the system can understand. This conversion process is also referred to as ingestion. The data connectors ingest data from various sources, format and convert it into a Document representation that consists of text and basic metadata. LlamaIndex describes a Document as a container for any data source, such as a PDF, an API response, or data collected from a database. By default, a Document holds text as well as information and relationships to other documents.

Once the original data has been ingested into Documents, the script begins to generate the Index. Indexes store data as Node objects and extend a Retriever interface for further setup and automation possibilities. During indexing, the loaded Documents are broken down into chunks and parsed into Node objects, which are lightweight abstractions of text strings that keep track of metadata and

relationships. The Node Objects are turned into vector embeddings, ready to be queried by the LLM.

After the index is generated, it is often recommended to save it for later use because having a ready-to-use index speeds up the processing time by eliminating the need to re-index every time a query is made. Re-indexing is most likely only required when updating or modifying the original documents. Index storage is usually done with a vector store, and LlamaIndex provides a variety of vector store solutions, as well as the built-in, default Simple Vector Store, which is ideal for fast experimentation purposes. It is an in-memory vector store that allows the index to persist on the disc so that in cases where an Index already exists on the local disc, it is loaded instead of creating a new one. This greatly improves loading times and reduces unnecessary duplication of resources.

As the index is ready, either by loading or generation, the script moves on to initialising the query engine. A query engine acts as an intermediary between the queries and the underlying data. It takes the query as input and analyses the wording trying to understand the intent behind the question. Leveraging the pre-built index created by LlamaIndex, the query engine searches for relevant information and based on the search results, The retriever components retrieve the most relevant portions of the indexed data. This might involve fetching entire documents, specific sections, or even smaller snippets depending on the nature of the question. As the retrieved information might not be directly usable as an answer the query engine might need to further process it, potentially using the LLM itself, to generate a clear and concise response that addresses the question asked. As this project's AI system is intended to work as a chat, LlamaIndex's chat engine was used instead of the query engine. The chat engine is a stateful analogy of the query engine. It can answer queries with previous context in mind as it preserves the conversation history.

After the process of forming the answer is done, the query engine returns the answer. The original PHP controller that initialised the Python script with the shell execute command now reads the lines containing the answer from the shell and passes it on, back to the frontend server. The JavaScript creates a chat bubble

for it and appends it to the chat. It is quite marvellous how all of this happens in under 5 seconds at best. With more optimising, there is a high chance that even faster results could be obtained.

The figure 17 presents the backend process flow as a simplified chart for better visualisation.

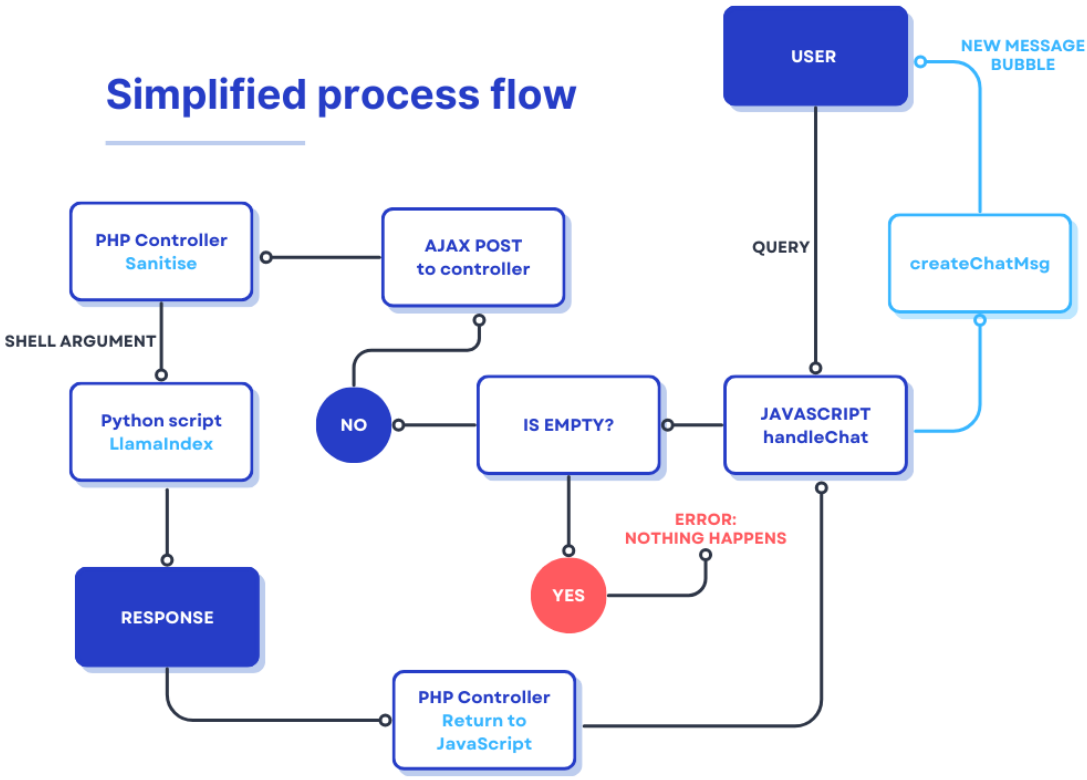

Figure 17. Simplified process flow

Evaluation

There are many methods for evaluating the responses. LlamaIndex provides essential elements for measuring the quality of the generated results. They also provide essential components for measuring retrieval quality.

Response Evaluation: Does the response correspond to the retrieved context? Does it also match the query? Does it match the reference answer or the guidelines?

For response evaluation, LlamaIndex offers LLM-based evaluation modules to measure the quality of results. This uses a "gold" LLM (e.g. GPT-4) to decide whether the predicted answer is correct in a variety of ways. Many of the current evaluation modules do not require ground-truth labelling. Evaluation can be performed using a combination of the question, context, and response, together with LLM calls. In addition to assessing searches, LlamaIndex can utilise the data provided to produce evaluation questions. This means that it is possible to build questions automatically and then use an evaluation pipeline to see if the LLM can accurately answer them using the data. [24]

Retrieval Evaluation: Are the retrieved sources related to the query?

The concept of retrieval evaluation is not uncommon; given a dataset of questions and ground-truth rankings, one can evaluate retrievers using ranking metrics such as mean reciprocal rank, hit rate, precision, and others. [24]

However, none of these evaluation options were used for this project. Since this project was primarily an early prototype and proof-of-concept example, it would not receive sufficient activity to generate enough relevant data. It was also thought to be more efficient to manually validate the responses rather than implementing the evaluation tools.

This project began with a somewhat different approach to testing and focus. The testing was expected to be done with legitimate Arcista users, with the primary focus being on user responses and opinions on this AI system. The original intention for testing was to collect data on the performance of both the traditional and AI systems through a brief usage test and an experience survey.

The participants would have been randomly separated into two groups, and testing would have been done on an individual basis. Each tester would have been given a question that they needed to answer using either the traditional or

AI system. The participant's time spent finding the answer would have been recorded for comparison. The survey section would have addressed more of the actual user experience and the testers' thoughts on the tested technology. These findings were intended to be used to lightly assess and influence the choice of whether or not to continue building the AI system.

However, these plans changed due to a lack of answers to the email campaign conducted to gather participants for this testing. It was determined that the development team would conduct small-scale testing, with a focus on basic test usage and manual analysis of response relevancy. This strategy provided valuable information on what needed more development in the future for prospective production-level versions.

4 Discussion

This chapter focuses on both the potential and limitations of two self-service support systems designed for software with strict security policies: a static help centre and a conversational AI system. These technologies were designed to address the difficulty of delivering fast and user-friendly help while maintaining security standards that restrict external access to program documentation.

4.1 Evaluating the Help Centre

The static help centre has various benefits. Its simplicity of deployment and minimal ongoing costs make it an appealing option. Once developed, the material is easily accessible and reliable for users who can find the appropriate articles. However, the efficiency of the help centre is heavily reliant on the quality and accessibility of its content. Creating and maintaining informative and up-to-date information requires a concerted effort, especially when the software is often updated. Likewise, if the article's organisation is unclear, readers may struggle to find the information they need, resulting in frustration and wasted time.

Due to its manual nature, expanding and upkeeping the help centre’s content in multiple languages might be costly. Translating technical material requires expert knowledge to ensure accuracy and clarity, which adds significant cost to the overall solution.

4.2 Evaluating the conversational AI system

The conversational AI system shows potential for providing a more personalised and efficient user experience. These systems, employing machine learning algorithms, can process and respond to user queries in real-time, enabling a more interactive type of self-service support. The system’s capacity to learn and develop based on user interactions provides a major benefit over static resources. Another compelling feature of conversational AI is its ability to use

current software documentation as is, removing the requirement for dedicated help articles. As many large language models are trained on multilingual datasets, the AI system has the potential to simplify the content generation process, removing language barriers with relatively low additional cost. However, developing a properly optimised multilingual system may demand further consideration.

The conversational AI system developed in this thesis has limitations that must be addressed before it can be considered a viable solution. The system's performance varies, with response times fluctuating and replies not always matching the user's inquiry. In ideal conditions, responses can be generated in less than 6 seconds; but, if the system has just been opened, it may take up to 30 seconds. At this point, the request often gets terminated since it took too long. The request termination threshold can be adjusted, but this issue must also be addressed. It is unacceptable to have such long generation times, even in ideal conditions.

In certain circumstances, the system just pulls text from the original documents without truly understanding the user's intent. The AI can also misinterpret the user's question, resulting in useless responses, that technically match some keywords from the user query. These constraints necessitate additional research and refinement to ensure dependable and accurate user help.

The most notable drawback of this AI system is the lack of suitable MIT-licensed, open-source LLMs. Arcista has strict privacy policies that essentially prohibit the use of third-party applications in such cases where the data would leave Finland’s borders. Without an open-source, locally hostable LLM model, the conversational AI system cannot be built into Arcista, and it cannot be used to manage any type of customer data. This is a significant issue, that needs to be solved to create a production-ready system.

4.3 Limitations and Future Work

Both the static help centre and the conversational AI system present potential solutions to the problem of delivering user support for software that has strong security requirements. The static help centre is a convenient and cost-effective solution, but it demands user effort to access information and ongoing content management. The conversational AI system has the potential to provide a more engaging and customised experience, but its existing limitations necessitate further research to ensure accuracy in the user support it provides.

The conversational AI system developed in this thesis is a prototype with limited functionality. Additional research is required to resolve performance issues and improve the system's ability to understand and respond correctly to user requests. Throughout the limited usage testing conducted within this project, it became evident that the AI system requires functional expansions such as multi-modality, which is the ability to handle and display relevant images.

Furthermore, the analysis of the help centre and the AI system concentrated on their theoretical strengths and limitations. Future research could and should include user studies that compare the user experience with both approaches and assess their effectiveness in providing self-service assistance. A hybrid system that incorporates both the static help centre and the conversational AI, may be worth researching further when the AI systems implementation requirements are met.

5 Conclusion

This thesis investigated the feasibility of developing self-service support solutions for Arcista, an archival software with strict security policies and no existing self-service support solutions. Two potential solutions were developed and explored: a static help centre and a conversational AI system.

The developed static help centre demonstrates ease of implementation and low ongoing costs. However, maintaining informative and up-to-date content requires dedicated effort and might not be scalable for complex software.

The conversational AI prototype was developed to research using AI in software like Arcista. This thesis successfully demonstrated that while it is possible, it has severely restrictive factors. Arcista’s strict security policies set limitations for the component selection of the AI system. Each component must be open-source, MIT-licensed and local or self-hostable. The lack of suitable open-source Finnish large language models prevents full implementation of the AI system until a suitable solution has been found or developed.

This thesis provides valuable insights for Arcista's future development. The static help centre will serve as the foundation for the development of an Arcista-integrated help centre to give quick user support. The conversational AI prototype demonstrates the potential for a more advanced support system, paving the way for future exploration in this area. The AI system offers more possibilities for tailored support and easier upkeep without the need to create specialised content. The testing results indicate that the AI system developed in this thesis would not function effectively as a standalone support approach, but when paired with the traditional Help Centre, it has the potential for quick search and personalised assistance.

Future work on the conversational AI system should focus on identifying or developing a suitable open-source Finnish LLM model and expanding the AI system capabilities to multimodal to handle relevant imagery. Additionally,

exploring hybrid solutions combining the help centre with AI functionalities or integrating the AI directly within Arcista are promising areas for further research.

This thesis successfully established the groundwork for developing self-service support solutions for Arcista. While the conversational AI system requires further development, the static help centre offers a practical foundation for the developers. This thesis lays a solid foundation for Arcista's future user support efforts and opens the door to additional research into AI-powered customer service solutions.

References

[1] IBM, ‘What is Artificial Intelligence (AI)?’ [Online]. Available: https://www.ibm.com/topics/artificial-intelligence.

[2] Investopedia, ‘What Is Artificial Intelligence (AI)?’ [Online]. Available: https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp.

[3] SAS3, ‘Artificial Intelligence (AI): What it is and why it matters’. [Online]. Available: https://www.sas.com/en_us/insights/analytics/what-is-artificial-intelligence.html.

[4] NVIDIA, ‘What are Large Language Models? | NVIDIA’. Accessed: May 12, 2024. [Online]. Available: https://www.nvidia.com/en-us/glossary/large-language-models/

[5] B. Copeland, ‘Alan Turing | Biography, Facts, Computer, Machine, Education, & Death’. [Online]. Available: https://www.britannica.com/biography/Alan-Turing.

[6] ‘History of Artificial Intelligence - AI of the past, present and the future! - DataFlair’. Accessed: May 21, 2024. [Online]. Available: https://data-flair.training/blogs/history-of-artificial-intelligence/

[7] ‘History of AI: How generative AI grew from early research | Qualcomm’. Accessed: May 21, 2024. [Online]. Available: https://www.qualcomm.com/news/onq/2023/08/history-of-ai-how-generative-ai-grew-from-early-research

[8] ‘The History of Artificial Intelligence - Science in the News’. Accessed: May 21, 2024. [Online]. Available: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

[9] C. University, ‘AI in Video Games | Columbia University’. [Online]. Available: https://ai.engineering.columbia.edu/ai-applications/ai-video-games/.

[10] J. Hurwitz and D. Kirsch, *Machine Learning IBM Limited Edition*. 2018.

[11] ‘What Is Machine Learning (ML)? | IBM’. Accessed: Jun. 03, 2024. [Online]. Available: https://www.ibm.com/topics/machine-learning

[12] ‘Supervised vs. unsupervised learning: What’s the difference? | IBM’. Accessed: May 30, 2024. [Online]. Available: https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning

[13] IBM, ‘What Is Unsupervised Learning?’ Accessed: May 19, 2024. [Online]. Available: https://www.ibm.com/topics/unsupervised-learning

[14] IBM, ‘Five machine learning types to know - IBM Blog’. Accessed: May 19, 2024. [Online]. Available: https://www.ibm.com/blog/machine-learning-types/

[15] IBM, ‘What is Deep Learning?’ Accessed: May 12, 2024. [Online]. Available: https://www.ibm.com/topics/deep-learning

[16] IBM, ‘What Is Supervised Learning?’ Accessed: May 19, 2024. [Online]. Available: https://www.ibm.com/topics/supervised-learning

[17] databricks, ‘A Compact Guide to Large Language Models, databricks’.

[18] Z. Ji *et al.*, ‘Survey of Hallucination in Natural Language Generation’, *ACM Comput Surv*, vol. 55, no. 12, Feb. 2022, doi: 10.1145/3571730.

[19] NVIDIA, ‘Generative AI – What is it and How Does it Work?’ Accessed: May 12, 2024. [Online]. Available: https://www.nvidia.com/en-us/glossary/generative-ai/

[20] A. W. S. Amazon, ‘What are Transformers? - Transformers in Artificial Intelligence Explained - AWS’, *December*, [Online]. Available: https://aws.amazon.com/what-is/transformers-in-artificial-intelligence/.

[21] A. Vaswani *et al.*, ‘Attention Is All You Need’, 2017.

[22] IBM, ‘What Are AI Hallucinations?’ Accessed: May 28, 2024. [Online]. Available: https://www.ibm.com/topics/ai-hallucinations

[23] Learn Prompting, ‘Pitfalls of LLMs’. [Online]. Available: https://learnprompting.org/docs/basics/pitfalls.

[24] LlamaIndex, ‘LlamaIndex documentation’. [Online]. Available: https://docs.llamaindex.ai/en/stable/.

[25] Turing, ‘Fine-Tuning LLMs : Overview, Methods, and Best Practices’. [Online]. Available: https://www.turing.com/resources/finetuning-large-language-models.

[26] databricks, ‘What is Retrieval Augmented Generation (RAG)? | Databricks’. Accessed: May 12, 2024. [Online]. Available: https://www.databricks.com/glossary/retrieval-augmented-generation-rag

[27] NVIDIA, ‘What Is Retrieval-Augmented Generation aka RAG | NVIDIA Blogs’. Accessed: May 12, 2024. [Online]. Available: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/

[28] L. H. Inc, ‘Laravel Documentation’, *Laravel Holdings Inc*, [Online]. Available: https://laravel.com/docs/11.x.

[29] GeeksforGeeks, ‘MVC Framework Introduction’. Accessed: May 30, 2024. [Online]. Available: https://www.geeksforgeeks.org/mvc-framework-introduction/

[30] T. L. Inc, ‘Tailwind CSS - Documentation’, *Tailwind Labs Inc*, [Online]. Available: https://tailwindcss.com/docs/installation.

[31] AWS, ‘What is SQL? - Structured Query Language (SQL) Explained - AWS’. Accessed: May 12, 2024. [Online]. Available: https://aws.amazon.com/what-is/sql/

[32] M. Foundation, ‘MariaDB’, *MariaDB Foundation*, [Online]. Available: https://mariadb.org/.

[33] A. Becker, ‘HeidiSQL’. [Online]. Available: https://www.heidisql.com/.

[34] Cloudflare, ‘What is a vector database? | Cloudflare’. Accessed: May 12, 2024. [Online]. Available: https://www.cloudflare.com/learning/ai/what-is-vector-database/

[35] ‘PHP: Hypertext Preprocessor’. Accessed: May 21, 2024. [Online]. Available: https://www.php.net/

[36] ‘Node.js — Run JavaScript Everywhere’. Accessed: May 21, 2024. [Online]. Available: https://nodejs.org/en

[37] ‘Dribbble - Discover the World’s Top Designers & Creative Professionals’. Accessed: May 30, 2024. [Online]. Available: https://dribbble.com/

[38] Tutorialspoint, ‘What are Named Routes in Laravel?’ Accessed: May 19, 2024. [Online]. Available: https://www.tutorialspoint.com/what-are-named-routes-in-laravel

[39] X. Lv, Y. Liu, J. Luo, Y. Liu, and C. Li, ‘Does a cute artificial intelligence assistant soften the blow? The impact of cuteness on customer tolerance of assistant service failure’, *Ann Tour Res*, vol. 87, p. 103114, Mar. 2021, doi: 10.1016/J.ANNALS.2020.103114.

[40] Q. Hu and Z. Pan, ‘Is cute AI more forgivable? The impact of informal language styles and relationship norms of conversational agents on service recovery’, *Electron Commer Res Appl*, vol. 65, p. 101398, May 2024, doi: 10.1016/J.ELERAP.2024.101398.

[41] M. Song, H. Zhang, X. Xing, and Y. Duan, ‘Appreciation vs. apology: Research on the influence mechanism of chatbot service recovery based on politeness theory’, *Journal of Retailing and Consumer Services*, vol. 73, p. 103323, Jul. 2023, doi: 10.1016/J.JRETCONSER.2023.103323.

[42] ‘HTML Div Tutorial’. Accessed: May 30, 2024. [Online]. Available: https://www.w3schools.com/html/html_div.asp

[43] ‘What are Vector Embeddings? | Definition from TechTarget’. Accessed: May 30, 2024. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/vector-embeddings

[44] ‘What are embeddings in machine learning? | Cloudflare’. Accessed: May 30, 2024. [Online]. Available: https://www.cloudflare.com/learning/ai/what-are-embeddings/