Evaluating Headless CMSs role in web development utilizing the Analytic Hierarchy Process

Lappeenrannan–Lahden teknillinen yliopisto LUT

Bachelor’s thesis, Software Engineering

2022

Author: Pyry Santahuhta

Supervisor: Assistant professor Antti Knutas

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT Teknis-luonnontieteellinen

Tietotekniikka

Pyry Santahuhta

Headless CMS:n roolin arviointi analyyttistä hierarkiaprosessia hyödyntämällä

Tietotekniikan kandidaatintyö

2022

36 sivua, 2 kuvaa, 13 taulukkoa ja 3 liitettä

Tarkastaja: Avustava professori Antti Knutas

Avainsanat: headless CMS, monolithic CMS, sisällönhallintajärjestelmä, verkkokehitys arkkitehtuurit, analyyttinen hierarkiaprosessi

Tämä tutkimus arvioi headless sisällönhallintajärjestelmien sopivuutta eri verkkokehitystilanteissa. Perinteiset monoliittiset sisällönhallintajärjestelmät ovat erittäin suosittuja verkkokehityksessä. Usean kanavan sisällöntoimituksen ja entistä dynaamisempien verkkoapplikaatioiden tarpeen kasvu on ongelma perinteisille sisällönhallintajärjestelmille. Näihin ongelmiin mahdollinen ratkaisu voisi olla headless sisällönhallintajärjestelmä Tämä tutkimus tutkii tätä käyttäen analyyttistä hierarkiaprosessia.

Tutkimuksessa ilmeni, että headless CMS voi toimia välimaastona perinteisten sisällönhallintajärjestelmän ja full-stack kehityksen välillä. Headless sisällönhallintajärjestelmät mahdollistavat monipuolisemman sisällön toimituksen, kuitenkin säilyttäen sisällönhallintajärjestelmien backendin yksinkertaisuuden. Lisäksi tutkimuksessa selvisi, että headless sisällönhallintajärjestelmät toimivat parhaiten tilanteissa, joissa budjetti, aika ja joustavuus ovat prioriteetteja. Analyyttinen hierarkiaprosessi soveltui verkkokehitys arkkitehtuurien valintaan hyvin. Analyyttisen hierarkiaprosessin heikkous on kriteerien rajoitettu määrä, jonka takia tärkeitä yksityiskohtia tai ominaisuuksia voi jäädä huomiotta.

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

School of Engineering Science

Software Engineering

Pyry Santahuhta

Evaluating Headless CMSs role in web development utilizing the analytic hierarchy process

Bachelor's thesis

2022

36 pages, 2 figures, 13 tables, and 3 appendices

Examiners: Assistant professor Antti Knutas

Keywords: headless CMS, monolithic CMS, content management system, web development architecture, analytic Hierarchy Process

This thesis determines the suitability of headless content management systems (CMS) in different web development scenarios. Traditional monolithic CMSs are very popular in web development. The emerging trends of omnichannel delivery and more dynamic sites pose a problem for traditional CMSs, which headless could solve. The thesis examines this utilizing a decision-making framework called the analytic hierarchy process.

It was identified that headless CMS can serve as a middle-ground between traditional CMSs and full-stack development. Headless CMSs allow for more diverse content delivery than traditional CMSs, while still retaining their simplicity for backend administration. Additionally, it was discovered that headless CMS works the best in scenarios where budget, time, and versatility are priorities. The analytic hierarchy process was found to be suitable for choosing web development architectures. AHP has a weakness in the limited number of criteria, which can leave important details or features unaccounted for.

ACKNOWLEDGEMENTS

I would like to express my gratitude for my supervisor Antti Knutas for all his help throughout this thesis. This work would also have been impossible without the team that assisted in selecting the comparison weights. Finally, I would like to thank my family and friends for their constant support over the years.

Abbreviations

CMS	Content Management System
HTML	Hypertext Markup Language
REST	Representational State Transfer
API	Application Programmable Interface
UI	User Interface
IoT	Internet of Things
AHP	Analytic Hierarchy Process
JSON	JavaScript Object Notation

TABLE OF CONTENT

1 Introduction

World Wide Web usage has grown exponentially from the year 2000 to the year 2020. In addition to the number of websites, the complexity of the web has also grown dramatically. This means that in the current software engineering climate, efficient web development is vital. With the growth of the web, the amount of content multiplies as well, so content producers must find ways to distribute their content effectively.

Content management systems (CMS) are designed for this purpose, to automate parts of the development process, make development more accessible for non-technical users, and make content delivery straightforward. WordPress is the most popular CMS, up to 29 percent of all websites on the internet are developed using it [1]. Traditionally, WordPress uses a “coupled” architecture. This means that the frontend and backend are coupled together into one entity. A coupled CMS streamlines the creation of simple websites such as blogs, and personal or business websites so that no coding knowledge is required. This allows for efficient and low-cost development of static websites.

Although they work well with desktop-based websites, coupled CMSs do come with disadvantages. The more complex a site becomes, the harder it becomes to develop using a coupled CMS. Because the front and backend are coupled together, additional frontend devices are not possible. Additionally, the developer must accept the web framework and layout element editing tools imposed by the CMS. Search engine optimization is impossible or very limited because the frontend is created from ready-made components. Dynamic data poses another problem, since coupled CMS websites are static, which means that the Hypertext Markup Language (HTML) code is fixed, and the same information is presented to every user.

Headless CMS is similar to coupled CMS architectures, but it lacks the frontend portion of the system. The system contains a Representational State Transfer (REST) Application Programmable Interface (API), which allows for the attachment of any sort of user interface (UI) or “head” to the CMS. This architecture design solves many of the problems coupled architectures face, while also automating the whole backend development process. Dynamic data webpages, for example, suddenly become possible, since the frontend doesn't have to adhere to a layout determined by the CMS provider. A headless CMS approach also offers the possibility of using multiple UIs with the same system, designed for different types of devices. These could include a traditional web page, an interface for an info screen at a mall and a voice interface to be used by an Internet of Things (IoT) device. Since more and more devices are getting online nowadays, this is becoming increasingly important. Search engine optimization can also be taken into consideration with a headless CMS.

API-driven CMSs enable developers to change technologies and stay nimble. Modularizing a website allows for the replacement of or upgrading of specific components, and headless CMSs are ready to support the technologies of the future. As a result, the systems remain future-proof, which is crucial as the web development industry is constantly evolving. REST APIs are built into WordPress, and there are several plugins that can be used to decouple a system and convert it into a headless CMS. There are currently numerous open-source and commercial headless CMSs available, which means that headless is ready and available for use.

1.1 Goals and methods

A web application's architecture type is very important when it comes to development. The purpose of this thesis is to determine what types of web applications are suitable for headless CMS development. The aim is that when presented with a new application, this thesis can aid in the decision process regarding the suitability of a headless CMS.

To limit the scope of the study, the evaluation is only aimed at architecture. Therefore, the choice of backend databases or server planning will not be discussed in detail. Neither will choices regarding frontend technical options be discussed. Regarding frontend development, headless CMSs are viewed as more versatile, costly, and more time-consuming to develop than coupled CMSs. The evaluation will be between different architectures, and specific choices within those architectures such as choosing the provider of headless CMS will not be taken into consideration. Decoupled CMSs will also not be considered in the evaluation, as they are very similar to headless.

The evaluation will be conducted using the Analytic Hierarchy Process (AHP), a commonly used method to help with decision-making [2]. AHP has been used in various areas, such as selection, analysis, priority, and resource allocation. AHP will be used to evaluate the suitability of Headless CMS against traditional content management systems and full-stack web application frameworks for different types of websites and applications. AHPy and Python will be used as a tool for conducting AHP.

The main research question for this thesis is:

What types of web applications should be developed using headless CMS?

If headless CMS is incompatible with any type of web application, the reasons for that will be discussed.

1.2 Work structure

The thesis is spread into five chapters. The first part is the introduction to the thesis, which gives small opening remarks on headless CMSs and why they could be used. Also included in the first chapter is the thesis's research question and the methods for answering them. The second chapter discusses Headless CMS and Web Development-related research. Here the reader can find additional information on the topic. The findings of the related research are then used to identify possible future research.

The third chapter explains how the study was conducted. The chapter covers AHP in detail. It also justifies its use and describes how it was utilized in the study. The fourth chapter collects the findings of the evaluation and discusses how the results answer the research question set in the introduction. The fifth chapter discusses the key findings and results from the evaluation, and how they could be applied in practice. Finally, the thesis concludes by summarizing the key findings of the study and by having a look into where the topic might go in the future.

2 Related research

This chapter introduces content management systems and covers related research on headless content management systems. After going through content management systems at large, monolithic and Headless are examined separately. In addition, the two will also be compared. As a relatively new technology, headless CMS has had few scientific papers written about it. However, the general principles of CMSs apply to Headless CMS as well.

2.1 Content Management Systems

In their patent for Content Management System, Baxter and Vogt describe it as "A system that organizes the content of the information separately from the appearance of the presented information" [3]. Additionally, Deane Barker defines a content management system as a "Software package that provides some level of automation for the tasks required to effectively manage content" [4]. Simply put, content management systems are tools for displaying and storing content.

As Boiko describes in his book, content is not the same as data [5]. While both content and data are considered information, the difference is in who the information is intended for. Boiko goes into detail on the differences, but the main takeaway is that content is made by humans, for humans, and data is made for computers to process. Barker agrees with Boiko and defines content as information produced through an editorial process and ultimately intended for human consumption via publication [4]. Barker also mentions the distinction between management and delivery that is revealed by the definition.

A CMS provides the tools to control content in several ways, including how it is displayed, who can view, edit, input, or delete it, and how it is indexed [4]. A CMS provides developers with stylizing templates that allow them to design structures for websites without requiring technical expertise [6]. The traditional architecture of a CMS contains an input area for content, which is managed in the backend usually with a UI interface.

The ease of use with CMSs doesn't come free though, as there are compromises associated with their use. Firstly, is the initial effort of starting up or adapting to a CMS. There is always a learning curve with this process, as CMS is just a set of tools to be utilized for the creation and management of a site. As CMSs are generally used for several different types of sites, they are not pre-configured to perform any one task particularly well [4]. Additionally is the loss of developer freedom, nearly any CMS will likely impact decisions or require compromises around the operating system and version upgrade cycle of the server [7]. Another pitfall associated with using CMSs is when they are implemented carelessly and treated more like an installation than an implementation [6].

In an article done by Black, the suitability of different CMSs was evaluated for their suitability to be used for an academic library website [8]. The library started to consider a change because its website had grown increasingly inconsistent and deprecated, a situation that CMSs are designed to handle. As noted in the article, CMSs are a good option for an application such as a library website, but the process of adapting to one is not only technical. For the content providers to use the CMS, they must learn a new process for providing content specific to each CMS. The article compared three CMSs: Drupal, MODx, and SilverStripe, and the evaluation deemed SilverStripe as the most suitable. [8]

The first CMSs were not server-based, rather they were client-side templating tools, these included CityDesk MarsEdit and Radio UserLand [4]. From there on they grew into server-based monolithic CMSs, that dominated the web in the early 2000s. Blogging gained popularity, and WordPress was the leading CMS designed for them. CMSs thrived, later evolving into web 2.0 applications where users could post content. The growing number of mobile devices led to the development of decoupled and later headless CMS, as monolithic was designed to deliver web content for PCs. [9]

2.1.1 Monolithic CMS

CMS architectures with monolithic or coupled content management systems are most common. The coupling in a monolithic CMS stands for the fact that management and delivery of content are inseparably linked together in the same environment [4]. With monolithic architectures, there usually is only one interface for editing the application, and it happens through a GUI. Since the developers need only know one technology, such as WordPress or Drupal, this type of architecture makes the development process simple, but rather limited. A visualization of the monolithic CMS architecture can be seen in Figure 1. The diagram was made using the draw.io (Version 14.3.1) web application.

Advantages of monolithic architectures include the ease of development as there is only one technology, and that technology requires little to no coding knowledge. Publishing of content has also been made very simple, and once the developers and publishers are familiar with the technology, the process becomes smooth. Personal or small community sites are also suited for monolithic architectures as the initial setup is fast and easy [10].

The problems in monolithic architectures arise when you want to change something. As time goes on, technologies and systems progress, and the need for updating one's systems increases. In the case of a monolithic CMS, migrating parts of it elsewhere is impossible as everything is inside one system. As the complexity of a site increases, the coding requirements become extensive and costly [9]. Additionally, the predeterminity of the templates and structures for the site makes advanced customization problematic.

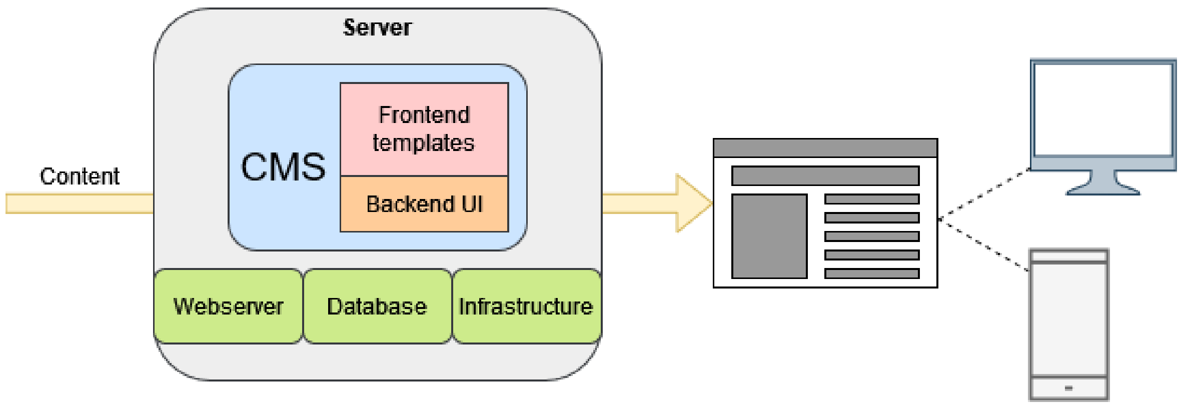

Figure 1

As seen in figure 1, content flows through the CMS, into a presentable view created by the frontend templates. New content is created or added through the backend content management UI. After creation content is stored in the database, from which the frontend templates retrieve it for presentation when needed. The whole CMS and database reside on the same server.

2.1.2 Headless and Decoupled CMS

A CMS usually has a frontend and backend. In CMS, the "head" refers to the frontend, so a headless CMS does not deliver content. In the ultimate guide to CMS, Contentstack describes Headless CMSs as API-driven CMS [11]. This is because the provider focuses a lot of attention on the quality of the APIs, which are the only way to connect new “heads” to the CMS. On his website, Barker describes Headless CMS as a “content-focused data source”, which is essentially what a headless CMS is [12]. A visualization of the headless CMS architecture can be seen in Figure 2. The diagram was made using the draw.io (Version 14.3.1) web application.

Decoupled is not the same as headless, as decoupled CMS is still concerned with content delivery, the delivery and management environments are just separated. A headless CMS is reactive, as it manages the content and then waits to be called. A decoupled CMS on the other hand is proactive, as it readies and pushes content into a delivery environment for presentation. [12]

The biggest advantage of a headless CMS compared to a monolithic one is the future-proofing of an application. Since the headless CMS is neutral by nature, it can be adapted to use new technologies or microservices as they develop [13]. The flexibility of development is another upside, as the developers can pick and choose the tools and technologies to be used in the application.

As the frontend code or “head” for all supported devices must be written for a headless CMS, it requires significantly more development than a monolithic CMS. This increases the initial setup time since the frontend usually has to be made from scratch [10]. Another one of the drawbacks of headless CMS is the lack of a live preview of the content since the CMS only provides the backend and the API [14]. This makes marketing for the application harder, as no clear representation of the app can be presented to stakeholders or clients [13]. The tech stack of a headless web application can also grow complex and fragmented, as different technologies may be used for different heads.

Web content is no longer confined to browsers and desktops [15]. With the rise of IoT, web applications have taken on many new forms. Headless CMS provides an ideal solution for the new forms, as the same API is capable of delivering consistent content on any device, consumed by multiple frontend applications [14].

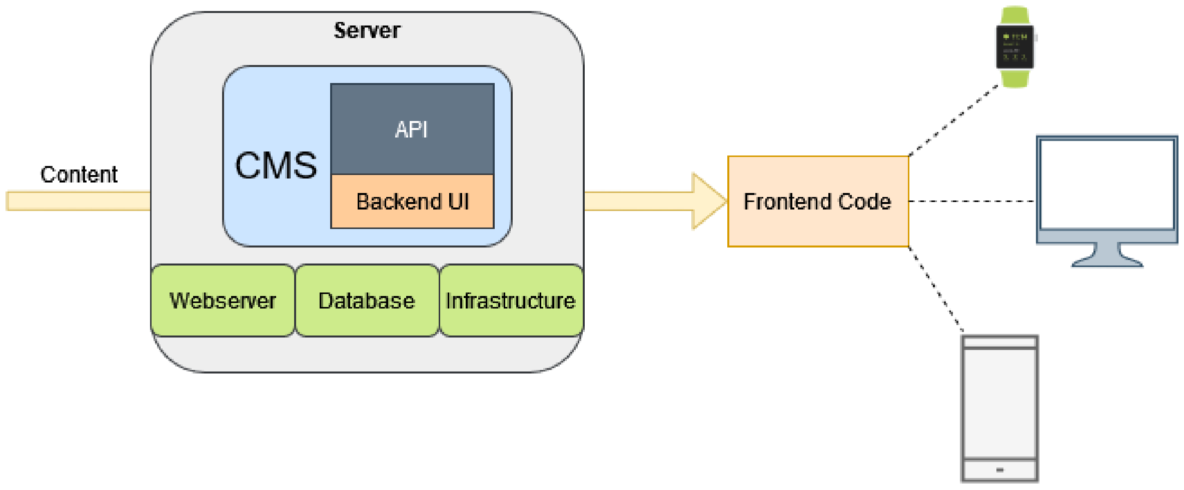

Figure 2

As seen in figure 2, content flows a little differently compared to figure 1. The content is still added through a backend UI and stored in a database but is now provided through an API. Frontend code must be developed outside the CMS and server to call the API, from which content can be accessed by devices.

2.2 Comparisons of content management systems

In The ultimate guide to CMS Vol 02, Contentstack weighs the pros and cons of traditional or monolithic CMS architectures against headless CMS [10]. Monolithic CMS advantages include a low barrier to entry, the multitude of templates available, and monolithic being an all-in-one solution. Headless, on the other hand, offers flexibility in creating content, programming language freedom, and security due to the separation of the database and the publishing platform. The difference in the strengths mostly consists of development-related factors, as monolithic is easier to get into for non-technical people, while headless provides greater customization and developer freedom. Petr Palas's list of pros [16] is very similar, going more in-depth into headless's ability to create microservices from the API. In Lalit Singla's comparison [13], the omnichannel capabilities of headless come up as its core strength.

On the downside, the developer freedom that comes with headless CMS comes with a catch, you need a competent developer to create the necessary frontend platforms [10]. Similarly, the biggest drawback of monolithic CMSs is connected to their advantages, as the templates that allow for ease of development also restrict design choices, technology options, and delivery channels [13]. Another downside of headless is that it is not possible to preview what a site will look like, similar to monolithic's What You See Is What You Get editor approach [10]. In addition, marketing a headless site or web application is more difficult than a monolithic one because there are no previews before frontend interfaces are implemented.[16]. The technology stacks of both headless and monolithic can also grow complex and fragmented with improper use [10].

Since most of the advantages and disadvantages of these CMS architectures can be viewed as trade-offs, it becomes apparent that monolithic and headless are both useful in different situations. On the other hand, if the wrong CMS solution is chosen for a project, the consequences may be severe.

3 Method description

This thesis considers the choice of web application architecture and compares alternatives using the AHP decision-making tool. Study alternatives include monolithic CMS, headless CMS, and full-stack development. As part of the thesis's decision-making and choice procedures, the technical choices inside the different alternatives were not considered, e.g., which programming language to use for the frontend of the headless CMS is not part of the thesis's scope. In addition, the scope of the study will be limited to examining only the aforementioned alternatives in AHP. The pairwise comparisons were decided on by a group of experts working in and or studying software engineering.

Based on the study's broad approach (looking at only three alternative architectures in four different imaginary cases), the results will reflect on which types of situations are different architectures suitable for. The study will not provide a straight answer to every real-life scenario, but rather guide what architecture would be most suitable.

3.1 Analytic Hierarchy Process as an evaluation tool

In the analysis Analytic hierarchy process (AHP) was used. Mathematician Thomas L. Saaty developed Analytic Hierarchy Process at the Wharton School of Business of the University of Pennsylvania [17]. He has written numerous books and papers and developed software on AHP and the Analytic networking process (ANP). Saaty describes it as a problem-solving framework, which organizes the basic rationality by breaking down a problem into its smaller constituent parts and then calls for only simple pairwise comparisons [18]. The comparisons are based on the Saaty scale, which ranges from 1-to 9, with 1 being equal importance and 9 being extreme importance over the other [19]. Saaty distinguishes three core principles required in solving problems using logical analysis: constructing hierarchies, establishing priorities, and logical consistency [20], these can all be found in AHP.

AHP can be used for a variety of different problems. It is a common tool in decision-making [2]. Humans have many limits in their capacity to process information. As Saaty puts it “They are limits on the number of sensations, impressions, or distinctions that can be held in mind briefly and grasped at once, or used as a basis for making judgments” [21]. One of

AHP's major strengths is the use of pairwise comparisons since humans can easily identify which alternative is better. AHP also removes bias and emotion from decision-making. The method achieves this by hierarchically organizing decision problems to avoid people making reactive and unplanned decisions [22]. Due to these traits, AHP will be suitable for this study, as there could be some bias and prejudice concerning CMS and headless CMS. It is sometimes possible for AHP to lead to incorrect results, but this is usually due to the user and not the method [23]. To achieve the best results, it is important to follow the requirements regarding criteria and the hierarchical structure, when implementing AHP.

AHP will be conducted for four different imaginary cases for different types of needs in web development. The cases were created from real-world examples of web development projects. Furthermore, the cases were intended to be very different from one another. The aim was to relate the cases to as many real-life scenarios as possible. In each of the four cases, monolithic CMS, headless CMS, and a full-stack solution will be compared to each other. The following cases provide an overview of the types of use cases for which Headless architecture is appropriate. The cases will be “A start-up of a few people creating a fitness application”, “A large enterprise company moving to their own intranet application”, “A young graduate creating a portfolio site” and “A software engineering company creating a web application for government taxing.”. Their requirements are vastly different, and that will become evident in the results.

In this study, pairwise comparisons for the criteria and the architectures will be conducted by a team of experts in the field. The team will make up of people who work in or study software engineering. Using the data from the team, AHP will be performed using AHPy.

3.1.1 Criteria

The criteria and their values are an important choice for the success of AHP. In his 2003 paper [21], Saaty and Ozdemir define that the number of criteria used in AHP should be limited to seven plus or minus two. This is due to increasing inconsistencies with higher counts of criteria. Identifying and correcting the inconsistencies becomes increasingly challenging after seven criteria are met. Consequently, this study will use five criteria.

The criteria chosen for the study are performance, budget, versatility, ease of development, and time to market. These criteria will touch on the most important aspects of web development, while also keeping the scope manageable. Performance can be described as how efficiently and fast the architecture can be estimated to accomplish different tasks. The budget considers all financial costs related to the development of an application. These include development costs, maintenance costs, and fees related to service providers. Versatility accounts for the ability to react to change, whether it’s horizontal or vertical scalability, adopting new technologies, or adding new delivery channels. Ease of development refers to how easy it is for a developer to learn the architecture and keep the codebase manageable. Finally, time to market can be characterized by the time between the start of the project and the time when users can use the application. These criteria play different roles in different situations. For example, in big codebases used by hundreds of developers, ease of development is crucial, while for a single developer developing their project, the budget might be the most important factor.

3.1.2 Utilizing AHPy

AHPy is an open-source python library for conducting AHP [24]. The analysis in this study will be performed with AHPy. PyAhp is another alternative Python library that implements AHP [25], however, the AHPy library was selected for this study due to its comprehensive documentation and recent updates. Furthermore, AHPy's terminology is more general and descriptive, making it clearer to present, compared to PyAhp’s numeric approach.

In the documentation for AHPy, a few examples are presenting the stages of conducting AHP using it [24]. Going through an example is relevant since the examples closely follow how the library will be utilized in this study. Initially in AHP all distinct candidate pairwise comparisons are made in every evaluation criterion. Upon comparing candidates, the criteria are also compared one to another to determine their importance. In AHPy, the elements are stored in Python dictionaries in which the order of the comparison elements matters, as the first element, is compared to the second [24]. For example, ('Budget', 'Development time'): 5 means that budget is strongly more important than development time, and ('Budget', 'Time to market'): 1/3 means that time to market is moderately more important than budget on the Saaty scale [19]. As the next step in AHPy compare objects are created from the criteria. Finally, the criteria are added as children to the criteria object and after that, the results can be printed to display them in the desired manner.

3.2 Method stages

Firstly, each architecture was compared to the other two in every criterion. Sources for comparisons[11], [13], [16] and the group's knowledge of software engineering were used to make the decisions about the criteria. The selected criteria weights will be used for all case comparisons in AHP. The selected criteria weights can be seen below.

Tables 1-5: Criteria comparisons

Performance			
	Monolithic CMS	Headless CMS	Full-stack
Monolithic CMS	1	1/6	1/9
Headless CMS	6	1	1/2
Full-stack	9	2	1

Budget			
	Monolithic CMS	Headless CMS	Full-stack
Monolithic CMS	1	2	6
Headless CMS	1/2	1	5
Full-stack	1/6	1/5	1

Versatility			
	Monolithic CMS	Headless CMS	Full-stack
Monolithic CMS	1	1/7	1/9
Headless CMS	7	1	1/2
Full-stack	9	2	1

Ease of development			
	Monolithic CMS	Headless CMS	Full-stack
Monolithic CMS	1	2	7
Headless CMS	1/2	1	7
Full-stack	1/7	1/7	1

Time to market			
	Monolithic CMS	Headless CMS	Full-stack
Monolithic CMS	1	2	5
Headless CMS	½	1	5
Full-stack	1/5	1/5	1

Secondly, an AHPy program was written for the study. The program was written in a way that allows it to be easily reusable when checking for inconsistencies. The AHP program can be referenced in appendix 1. Following this, all the pairwise case comparisons were done individually by members of the team, which were then collected into one definitive table.

These results were then checked for inconsistencies using the program. Then the final comparison data was ready to be input into the AHPy program. The program asks users for each pairwise comparison and calculates AHP according to the input. The results were then checked and recorded for discussion.

4 Results

The results will be covered in order of the cases. First in each case is a brief overview of the results and reasoning on how the team of experts judged the criteria. After the explanations, the results gotten from the AHP program are displayed. AHP gives results as weights that add up to 1. A bigger number indicates a more suitable architecture for the given case. Below the architectures, are the criterion weights for the given case. Here you can see what criteria were important for each situation. The pairwise comparison tables which were inputted into the program can be referenced in appendix 2. The JavaScript Object Notation (JSON) results from the program can be referenced in appendix 3.

4.1 Case one, a start-up of a few people creating a fitness application

In case one, Headless CMS was the most favoured alternative as can be seen from table 6. Start-ups are generally risky and uncertain businesses, often depending on their first project to take off. This creates pressure on budget and time to market which was the second and third most wanted qualities as can be seen in table 7. The most sought-after quality, versatility, is inherent to the nature of fitness applications. In addition to desktop computers, fitness-type applications are often delivered via multiple different channels, such as mobile phones and smartwatches. A multichannel app must be able to transition to new frontend delivery methods and channels. Although Full-stack development offers the most flexibility, Headless comes out on top in this case due to the need for a low budget and rapid time to market. Performance was the least important factor. This can probably be attributed to the organization being a start-up with a low budget.

Table 6: Case one Architecture weight results

	Architecture	Weight
1	Headless CMS	0.348
2	Monolithic CMS	0.330
3	Full-stack	0.322

Table 7: Case one criterion weight results

	Criterion	Weight
1	Versatility	0.408
2	Budget	0.200
3	Time to market	0.193
4	Ease of development	0.139
5	Performance	0.060

4.2 Case two, a large enterprise company moving to their own intranet application

In case two, Full-stack was the most favoured alternative, with headless CMS coming in second as can be seen from table 8. This can be attributed to the utmost need for performance in the web application, with little regard to budget or time to market as can be seen from table 9. The reason for these criterion weights comes from the fact that in this imaginary case a large company that requires efficient intranet communication is moving to its own application. This means that they already have a solution, so time to market is not important at all. In addition, as a big company, the costs of an intranet application are trivial. Monolithic CMS was the least suitable for this case since it is not well suited for performance optimization.

Table 8: Case two Architecture weight results

	Architecture	Result
1	Full-stack	0.455
2	Headless CMS	0.342
3	Monolithic CMS	0.203

Table 9: Case two criterion weight results

	Criterion	Weight
1	Performance	0.489
2	Versatility	0.230
3	Ease of development	0.129
4	Time to market	0.080
5	Budget	0.072

4.3 Case three, a young graduate creating a portfolio site

In case three, a young graduate is creating a portfolio site for themselves. Monolithic CMS was the preferred alternative in this case, with headless CMS coming in second as can be seen from table 10. Since they are relatively inexpensive and easy to use, CMSs are good options for creating simple portfolio websites. Since Monolithic requires no programming experience to start a website, it is especially suitable for those who are not familiar with coding. In this case, the budget was the most important factor by a wide margin as can be seen from table 11. Graduating students aren't likely to be very wealthy, so they hope to set up their sites at the lowest possible cost. For an application such as this, CMS providers can usually manage a website for free, which would be ideal for an application such as this. Time to market is the least preferred quality in the application because the graduate does not have any deadlines or reasons to release the website quickly.

Table 10: Case three Architecture weight results

	Architecture	Result
1	Monolithic CMS	0.400
2	Headless CMS	0.343
3	Full-stack	0.257

Table 11: Case three criterion weight results

	Criterion	Weight
1	Budget	0.523
2	Performance	0.172
3	Versatility	0.167
4	Ease of development	0.094
5	Time to market	0.043

4.4 Case four, a software engineering company creating a web application for government taxing

Case four covers a situation where a software engineering company is chosen to create a web application for government taxing services. Table 12 shows that full-stack was the most favoured architecture in this scenario followed by headless CMS. As software engineering companies have the required knowledge, resources, and pipeline to create full-stack web applications, this is perfectly suited. A full-stack architecture's strengths are performance and versatility, both of which were valued highly in case four as can be seen from table 13. Since the government is funding the application, the budget was of little concern.

Table 12: Case four Architecture weight results

	Architecture	Result
1	Full-stack	0.438
2	Headless CMS	0.343
3	Monolithic CMS	0.219

Table 13: Case four criterion weight results

	Criterion	Weight
1	Performance	0.418
2	Versatility	0.269
3	Ease of development	0.131
4	Time to market	0.107
5	Budget	0.076

5 Discussion

In the Analytic Hierarchy Process, the differences and similarities of the three types of web development architectures were highlighted. Larger organizations with larger budgets tended to steer towards full-stack development. When versatility and budget were both of concern headless CMS was favoured. And when budget and ease of development were the most important factors, monolithic CMS took the lead. These differences were highlighted because all four cases were very different. The cases had huge contrasts in budget, resources, and knowledge available for the application. This was done deliberately to cover as many situations as possible. Real-life scenarios are bound to be less precisely definable, but they can all be compared to the four cases in some regard.

5.1 Headless CMS use cases and future

This thesis's primary research question was "What types of web applications should be built using headless CMS?". In case one "A start-up of a few people creating a fitness application" headless CMS was the most favoured option. Because of this, we can conclude that headless CMS is a good choice when a web application project is tight on their budget, and time and versatility of development is important.

Based on their properties and strengths, headless CMSs can be seen as a kind of middle ground between monolithic CMS and full-stack development. This becomes apparent from the other cases, where headless CMS wasn't the top choice. Aside from case one, headless CMS was the second most ideal option for every case. Based on the information presented above, headless CMS is a viable choice for almost all web application development scenarios, combining CMS ease of use with the flexibility of full-stack development. In this sense, headless CMSs are an excellent way of moving monolithic content management systems into the omnichannel age.

As most monolithic CMS users are using it because it requires no programming experience, headless will not replace monolithic CMS. That does not mean however that headless will not be used. Headless is more suitable for web applications with some developers involved and when the flexibility of development and delivery is a priority. Headless CMS is also future-proof since it does not rely on one technology.

The scientific research on the use cases of headless CMS is very limited, and it is mostly found in articles and blogs. In Contentstack's article [10], the majority of headless CMSs' strengths were related to versatility. Contentstack compares headless CMS only to monolithic CMS, which have similar budgets and development speeds, so those advantages of headless are not discussed. Similarly, in Singla's article [13], developer and content flexibility were listed as advantages of headless CMS. In his blog, Barker also lists non-web content publishing and multi-channel publishing as important values of headless [26]. This thesis's findings were very similar, with flexibility, budget, and speed of development showing up as headless's strengths. As a result of AHP's limited criteria, properties of CMSs such as publishing efficiency, marketing, and content aggregation were not considered in this thesis.

5.2 AHP's suitability for selecting web development architectures

The AHP approach was satisfactory for this thesis and thus selecting web development architectures. Although AHP's decisions seemed reasonable, no further conclusions can be drawn about their correctness. The method was successful because the cases and comparison criteria were very distinct. The cases and criteria worked so well because they were created with AHP in mind. If they were defined by a real company deciding on their web development architecture, the criteria might be less suitable. Nonetheless, AHP provided clear feedback about how strongly an alternative is favoured, and what criteria are valued the most. AHP can at the least be used for developing general frameworks and creating ideas for real decisions.

Choosing a real web development architecture using AHP would prove more difficult than the process in this thesis since in real life the criteria are more precise than just "Versatility" for example. Web applications have strict requirements for things like search engine optimization or cross-platform deployment. These cannot be taken into consideration when using AHP because of the magic number of seven-plus minus two [21]. With limited criteria, the decision-maker is directed to general criteria, leaving important details ambiguous. Some details or features, such as a live chat or dynamic data may render some architectures unusable. Since AHP cannot recognize such situations, it is the individual's responsibility to notice them.

6 Conclusion

This thesis answered the question “What types of web applications should be built using headless CMS?” using the analytic hierarchy process. The thesis compared headless CMS architecture to traditional monolithic CMS and full-stack development in four different distinct cases. The best use cases for Headless CMS are when budget, time, and flexibility are priorities. to allow omnichannel delivery while maintaining the simplicity of CMSs for backend administration. It also will not replace monolithic CMS due to the development required. It was found that AHP works reasonably well when selecting web application architectures. However, due to the limited number of criteria, important details may be lost when selecting the criteria. In addition, AHP cannot recognize situations where an application is not possible with a certain architecture.

The results of this thesis can be used as a guide when selecting an architecture for web development. Although real-life scenarios won't exactly match the cases, they can still serve as a reference. It is also shown in this thesis that headless CMS is better than other architectures in some scenarios. This can give assurance to individuals considering headless CMS for their projects.

Most of the pros of headless CMS in Contentstack’s article were related to versatility [10]. Furthermore, in Singla’s article [13], developer and content flexibility were listed as advantages. The aforementioned sources compare headless CMS to monolithic CMS, which both are valued for their low budget and efficient development, so these advantages of headless are not mentioned. The results of this thesis were very similar with versatility being one of the strengths of headless CMS along with budget and time to market.

Despite being a relatively new technology, headless CMSs are already a viable option for web development. Future improvements will undoubtedly result in making headless even more appealing. As omnichannel delivery becomes increasingly important, headless architecture will surely gain popularity. For future studies with AHP, it is recommended that the number of criteria is kept low to minimize inconsistencies. Thus, the subject to analyse should be able to be condensed to a few criteria.

References

[1] J. Cabot, ‘WordPress: A Content Management System to Democratize Publishing’, *IEEE Softw.*, vol. 35, no. 3, pp. 89–92, May 2018, doi: 10.1109/MS.2018.2141016.

[2] O. S. Vaidya and S. Kumar, ‘Analytic hierarchy process: An overview of applications’, *Eur. J. Oper. Res.*, vol. 169, no. 1, pp. 1–29, Feb. 2006, doi: 10.1016/j.ejor.2004.04.028.

[3] USPTO.report, ‘Content management system’, *USPTO.report*. https://uspto.report/patent/grant/6,651,066 (accessed Feb. 03, 2022).

[4] D. Barker, *Web Content Management: Systems, Features, and Best Practices*. O’Reilly Media, Inc., 2016.

[5] B. Boiko, *Content Management Bible*. John Wiley & Sons, 2005.

[6] C. Short, ‘Web content management: CMS for competitive advantage’, *J. Direct Data Digit. Mark. Pract.*, vol. 11, no. 3, pp. 198–206, Jan. 2010, doi: 10.1057/dddmp.2009.32.

[7] E. Bradford Lee, *Content management systems*. Emerald Group Publishing, 2006.

[8] E. L. Black, ‘Selecting a Web Content Management System for an Academic Library Website’, *Inf. Technol. Libr.*, vol. 30, no. 4, pp. 185–189, Dec. 2011, doi: 10.6017/ital.v30i4.1869.

[9] B. Heslop, ‘History of Content Management Systems and Rise of Headless CMS’, *contentstack.com*. https://www.contentstack.com/blog/all-about-headless/content-management-systems-history-and-headless-cms/ (accessed Jan. 25, 2022).

[10] ‘ultimate-guide-cms-vol-2.pdf’. Accessed: Feb. 03, 2022. [Online]. Available: https://info.contentstack.com/rs/489-WNI-383/images/ultimate-guide-cms-vol-2.pdf

[11] ‘ultimate-guide-cms-vol-1.pdf’. Accessed: Feb. 03, 2022. [Online]. Available: https://info.contentstack.com/rs/489-WNI-383/images/ultimate-guide-cms-vol-1.pdf

[12] ‘The State of the Headless CMS Market’, 00:00:00Z. https://deanebarker.net/tech/blog/state-of-the-headless-cms-market/ (accessed Feb. 07, 2022).

[13] L. Singla, ‘The Ultimate Guide to Headless CMS’, *Insights - Web and Mobile Development Services and Solutions*, Nov. 04, 2021. https://www.netsolutions.com/insights/what-is-headless-cms/ (accessed Feb. 03, 2022).

[14] J. Attardi, *Using Gatsby and Netlify CMS: Build Blazing Fast JAMstack Apps Using Gatsby and Netlify CMS*. Berkeley, CA: Apress, 2020. doi: 10.1007/978-1-4842-6297-9.

[15] D. Monroe, ‘Adding a Head to a Headless CMS’, *EContent*, vol. 40, no. 6, pp. 22–23, Dec. 2017.

[16] P. Palas, ‘The Ultimate Guide to Headless CMS’, 2019. https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWLvVL (accessed Feb. 15, 2022).

[17] ‘ISAHP2020 | About the Analytic Hierarchy Process (AHP)’. https://www.isahp.org/about/ (accessed Feb. 14, 2022).

[18] T. L. Saaty, ‘The Analytic Hierarchy Process: Decision Making in Complex Environments’, in *Quantitative Assessment in Arms Control: Mathematical Modeling and Simulation in the Analysis of Arms Control Problems*, R. Avenhaus and R. K.

Huber, Eds. Boston, MA: Springer US, 1984, pp. 285–308. doi: 10.1007/978-1-4613-2805-6_12.

[19] D. Ergu, G. Kou, Y. Peng, Y. Shi, and Y. Shi, ‘The analytic hierarchy process: Task scheduling and resource allocation in cloud computing environment’, *J. Supercomput. - TJS*, vol. 64, pp. 1–14, Jun. 2013, doi: 10.1007/s11227-011-0625-1.

[20] T. L. Saaty, *Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World*. RWS Publications, 1990.

[21] T. L. Saaty and M. S. Ozdemir, ‘Why the magic number seven plus or minus two’, *Math. Comput. Model.*, vol. 38, no. 3, pp. 233–244, Aug. 2003, doi: 10.1016/S0895-7177(03)90083-5.

[22] T. L. Saaty, ‘What is the Analytic Hierarchy Process?’, in *Mathematical Models for Decision Support*, Berlin, Heidelberg, 1988, pp. 109–121. doi: 10.1007/978-3-642-83555-1_5.

[23] R. Whitaker, ‘Criticisms of the Analytic Hierarchy Process: Why they often make no sense’, *Math. Comput. Model.*, vol. 46, no. 7, pp. 948–961, Oct. 2007, doi: 10.1016/j.mcm.2007.03.016.

[24] P. Griffith, *ahpy: A Python implementation of the Analytic Hierarchy Process*. Accessed: Feb. 14, 2022. [OS Independent]. Available: https://github.com/PhilipGriffith/AHPy

[25] A. Mishra, *PyAhp: Analytic Hierarchy Process Solver*. pyAHP, 2022. Accessed: Feb. 14, 2022. [Online]. Available: https://github.com/pyAHP/pyAHP

[26] ‘Use Cases for a Headless CMS’, 00:00:00Z. https://deanebarker.net/tech/blog/use-cases-for-headless-cms/ (accessed Mar. 28, 2022).

Appendix 1: AHPy program GitHub page

https://github.com/Pyry-Santahuhta/Web-Dev-Architectures-AHPy

Appendix 2: AHP pairwise comparison tables

Case one	Performance	Budget	Versatility	Ease of development	Time to market
Performance	1	1/5	1/3	1/3	1/5
Budget	5	1	1/4	2	1
Versatility	3	4	1	3	2
Ease of development	3	1/2	1/3	1	1
Time to market	5	1	1/2	1	1

Case one, a start-up of a few people creating a fitness application

Case 2	Performance	Budget	Versatility	Ease of development	Time to market
Performance	1	6	4	3	4
Budget	1/6	1	1/3	1/2	1
Versatility	1/4	3	1	3	3
Ease of development	1/3	2	1/3	1	2
Time to market	1/4	1	1/3	1/2	1

Case two, a large enterprise company moving to their own intranet application

Case 3	Performance	Budget	Versatility	Ease of development	Time to market
Performance	1	1/5	1	3	4
Budget	5	1	4	4	7
Versatility	1	1/4	1	2	5
Ease of development	1/3	1/4	1/2	1	3
Time to market	1/4	1/7	1/5	1/3	1

Case three, a young graduate creating a portfolio site

Case 4	Performance	Budget	Versatility	Ease of development	Time to market
Performance	1	6	2	3	3
Budget	1/6	1	1/4	1/2	1
Versatility	1/2	4	1	2	3
Ease of development	1/3	2	1/2	1	1
Time to market	1/3	1	1/3	1	1

Case four, a software engineering company creating a web application for government taxing

Appendix 3: AHPy program result JSONs

Case one:

```
{
   "name": "Criteria",
   "global_weight": 1.0,
   "local_weight": 1.0,
   "target_weights": {
      "Headless CMS": 0.347700192,
      "Monolithic CMS": 0.330038997,
      "Full stack": 0.322260811
   },
   "elements": {
      "global_weights": {
         "Versatility": 0.408206648,
         "Budget": 0.199823025,
         "Time_to_market": 0.192881426,
         "Ease_of_development": 0.139301836,
         "Performance": 0.059787065
      },
      "local_weights": {
         "Versatility": 0.408206648,
         "Budget": 0.199823025,
         "Time_to_market": 0.192881426,
         "Ease_of_development": 0.139301836,
         "Performance": 0.059787065
      },
      "consistency_ratio": 0.08248288
   }
}
```

Case two:

```
{
  "name": "Criteria",
  "global_weight": 1.0,
  "local_weight": 1.0,
  "target_weights": {
    "Full stack": 0.45476251,
    "Headless CMS": 0.341920201,
    "Monolithic CMS": 0.203317289
  },
  "elements": {
    "global_weights": {
      "Performance": 0.489378041,
      "Versatility": 0.229633502,
      "Ease_of_development": 0.129220899,
      "Time_to_market": 0.079802291,
      "Budget": 0.071965267
    },
    "local_weights": {
      "Performance": 0.489378041,
      "Versatility": 0.229633502,
      "Ease_of_development": 0.129220899,
      "Time_to_market": 0.079802291,
      "Budget": 0.071965267
    },
    "consistency_ratio": 0.045877919
  }
}
```

}

Case three:

```
{
   "name": "Criteria",
   "global_weight": 1.0,
   "local_weight": 1.0,
   "target_weights": {
      "Monolithic CMS": 0.400052156,
      "Headless CMS": 0.343410203,
      "Full stack": 0.256537643
   },
   "elements": {
      "global_weights": {
         "Budget": 0.522508116,
         "Performance": 0.172199818,
         "Versatility": 0.167329422,
         "Ease_of_development": 0.094787033,
         "Time_to_market": 0.043175612
      },
      "local_weights": {
         "Budget": 0.522508116,
         "Performance": 0.172199818,
         "Versatility": 0.167329422,
         "Ease_of_development": 0.094787033,
         "Time_to_market": 0.043175612
      },
      "consistency_ratio": 0.052441846
   }
}
```

Case four:

```
{
    "name": "Criteria",
    "global_weight": 1.0,
    "local_weight": 1.0,
    "target_weights": {
        "Full stack": 0.43768734,
        "Headless CMS": 0.342991887,
        "Monolithic CMS": 0.219320773
    },
    "elements": {
        "global_weights": {
            "Performance": 0.418239727,
            "Versatility": 0.268506247,
            "Ease_of_development": 0.130594964,
            "Time_to_market": 0.106835509,
            "Budget": 0.075823553
        },
        "local_weights": {
            "Performance": 0.418239727,
            "Versatility": 0.268506247,
            "Ease_of_development": 0.130594964,
            "Time_to_market": 0.106835509,
            "Budget": 0.075823553
        },
        "consistency_ratio": 0.016847918
    }
}
```