Creating an Azure CI/CD pipeline for a React web application

Joonas Saarenpää

2020 Laurea
Creating an Azure CI/CD pipeline for a React web application
Laurea University of Applied Sciences

Degree Programme in Business Information Technology
Business Administration

Joonas Saarenpää

Creating an Azure CI/CD pipeline for a React web application

Year 2020  Number of pages 58

The purpose of this bachelor’s thesis was to examine temporary web hosting services in Azure and to find a suitable platform for future application development. The web hosting service was needed for the React web application demo named “Nummelan kylähistoria”, which needed to be deployed to operational state. During the project, the client organization was changed to Laurea University of Applied Sciences (Laurea UAS). Laurea organization wanted to know how the Azure platform and its microservices can be utilized with student development projects that need a web hosting service. The integration of the demo application to Azure was treated as a case study. The client was represented by the lecturer, Arvind Sharma.

The original commission for this project came from the non-profit association Vihdin-Nummelan kylähistoria Ry, who needed a new web application solution for their pre-existing website. Laurea UAS and its students had previously worked on the demo application in collaboration with the association. The demo application needed to be deployed so that the old website content could be transferred to the demo application. The author was assigned to study the related technologies and deployment options in Azure. Microsoft Azure is a cloud service provider that is used by Laurea UAS.

The knowledge base for this thesis includes Azure, Azure Web App Service, Azure DevOps, and various integrated automation and agile tools. DevOps and Agile concepts were studied to understand the background and features of Azure services. React, WordPress, and Headless CMS were the main technologies used in the demo application, so these technologies were examined as well. Together, these services and ideas comprised the thesis’ subject: creating a continuous integration and continuous deployment (CI/CD) pipeline for the React web application. The framework for the project was the systems development life cycle (SDLC) because Azure is a comprehensive cloud service system that includes multiple dedicated microservices. An experimental approach was used to combine different new technologies together as one cohesive cloud operation.

The development tasks included the integration of the demo to Azure and to its microservices, Azure DevOps and Azure App Service. The application was integrated successfully to Azure DevOps and deployed using the Azure Web App Service. The automated processes in the build and release pipelines improved the application operations and the future development platform was established. Smaller development tasks were also conducted in the thesis work.

The development of the React web application was tentatively suspended. The association and project supervisors from Laurea decided to plan a new system architecture for the association’s software. The new system must be able to support the technical requirements set by the association. The thesis work provided practical information about the platform and its automated software delivery processes.

Keywords: Continuous integration, continuous deployment, Azure DevOps, DevOps, web application
Glossary of terms and abbreviations used

AI = Artificial Intelligence

API = Application Programming Interface. It is a set of methods and protocols that allow applications or its components to interact with other applications. APIs are used by external resources like microservices, operating systems or software.

API key = An identifier for authentication to an API service.

Back-end = In software architecture, a back-end refers to operations and tasks made in server-side, like data sharing. These operations are not accessible to a client.

Backlog = A list of unfinished development tasks or work.

Build = A version of source code that is still in development.

Browser = A software application that connects to a server and displays a specific content (websites, photos, videos) on the user's device. The content is classified with a unique URL on the Web.

CBA = Cost-benefit analysis. CBA presents an estimation of the positive or negative economic outcomes of a project.

CI/CD = Continuous Integration and Continuous Delivery/Deployment.

Client-server model = A client (e.g., website) sends a request to a server which gives a response via a browser back to the client.

Cloud computing = A model for enabling on-demand network access to a shared pool of scalable and configurable computing resources and services, like e.g., server and network infrastructure, applications, and cloud storage, to a client system. Cloud service providers host different types of computing resources via the Internet (or “cloud”) and clients typically rent them on a pay-as-you-go basis.

CMS = Content Management System

Continuous Delivery = An extension of continuous integration. The process to build, test, configure and deploy a build to a production-ready environment. All the steps of the deployment are automated except the production launch, which happens manually.

Continuous Deployment = Continuous deployment includes continuous integration, continuous delivery, and automated deployment to production. Every code change goes through the automated CI/CD pipeline and gets deployed into production without human intervention. When an application is deployed, it is put into operation.

Continuous Integration = The automation of the build and testing phase. A code is compiled, run, and tested before changes are made to version control.

Dashboard = A type of graphical UI where users can manage or monitor operations.

Debugging = Fixing bugs in software. Bugs are errors or faults in a computer program.

Demo = Demonstration of a software product.
DevOps = Development and IT Operations. A set of integration and automation tools and practices that help streamline software development processes. DevOps practices aim to combine the efforts of the development team and operation teams.

Domain name = A set of characters that inform the location of a website in the Internet.

Environment variable = Key-value pairs can be stored as environment variables in a local system. It is used to store sensitive information (e.g. API key) and to avoid exposing the information to public by including them in the code.

Feedback loop = Code change (input) goes through a system and causes a bad or a good output. Feedback loop affects the same system either negatively or positively.

Front-end = In software architecture, a front-end refers to the client-side development and actions. A client can see and interact with the “front” side of a system.


Integrate = To share data between systems.

Internet = A technology that allows servers to communicate with each other in a global network.

IT = Information Technology

Iteration = the repetition of sequence of operations in a process. Iterations continue until a desired outcome has been achieved.

Key-value pair = A data representation of application settings. A value or variable can be accessed with a key.

KUDU = Azure Web App’s background service for monitoring. It contains e.g., deployment logs, a debug console and configuration parameters.

LbD = Learning by Developing

Local development environment = A PC system that hosts the application files.

Microservice = An independent service that serves only one purpose within a larger, decoupled system. Decoupled system includes one or more standalone microservice components.

Migrate = to move data from one place to another.

NPM = Node Package Manager. A utility system that has a command line interface (CLI) tool and an online repository of open-source software packages. Clients who want to use NPM receive packages from package creators via CLI.

Open source = A source code that designed to be publicly available for modification and redistribution.

Pipeline = Pipeline includes all the CI/CD tasks and events to create a software release. It has automation tools and a “pipeline” like workflow.

PWA = Progressive Web App. PWA is an unofficial term for a web application that uses specific tools and technologies of both web and native applications.
React = An open-source JavaScript library. It is used to build interactive user interfaces. It uses client-side rendering (instead of server-side) to display parts of the content on browsers.

REST = REST means Representational State Transfer and it is the set of rules for creating web services.

REST API = An architectural style and approach to exchange resources between a client and a server.

Scope creep = Scope creep happens when a project keeps adding new goals after completing older ones, resulting a possible delay of deliverables, expanded cost or a failed project. Scope creep can happen when the goals of the project are not defined in detail.


Server = A system software that manages services and resources in response to request from one or more clients. A key part of the client-server model. For example, a web server manages web content and network server manages network traffic.

Source code = The human-readable form of a computer program. It is contained in single or multiple files.

Staging = In Azure it means the process of selecting the deployment server (test, development, production) for the build. Each stage contains one deployment target. Staging is the last phase before deployment.

System software = A platform that runs application software.

Tech/technical stack = A set of technologies, tools or software used to create a website, web application or mobile application.

UI = User Interface

URL = Uniform Resource Locator. Also known as the web address for a resource on the Web.

Version control = A software that tracks code changes. It manages all versions of a code, but only presents a single version at a time.

Web = Short for World Wide Web. A way to share and access data over the Internet that is accessible with web browsers.

Web application = An application software that runs on servers and is accessible via a web browser. Simple web applications can be built using CSS, HTML5 and JavaScript.

YAML = Stands for “YAML Ain't Markup Language”. It is a human-readable data serialization standard for writing configuration files. YAML file can modify the build policy and it can be managed with other source files in the root of repository.
# Contents

1 Introduction .............................................................................................................. 9
   1.1 Vihdin-Nummelan kylähistoria Ry ................................................................. 10
   1.2 Laurea University of Applied Sciences .......................................................... 11

2 The project ............................................................................................................. 11
   2.1 Project goals ..................................................................................................... 13
   2.2 Project limitations and risks ............................................................................ 14

3 Research and organization methodology .............................................................. 15
   3.1 React web application .................................................................................... 16
   3.2 WordPress as headless CMS .......................................................................... 16
   3.3 Microsoft Azure ............................................................................................. 17
   3.4 Azure App Service ......................................................................................... 18
   3.5 Agile ................................................................................................................ 18
   3.6 DevOps and Azure DevOps ............................................................................ 19
   3.7 CI/CD pipeline ............................................................................................... 20
   3.8 Systems development lifecycle (SDLC) .......................................................... 21
      3.8.1 Requirements analysis ........................................................................... 22
      3.8.2 System Analysis and feasibility studies .................................................. 22
      3.8.3 Development ......................................................................................... 23
      3.8.4 Implementation ..................................................................................... 23
      3.8.5 Maintenance ......................................................................................... 24

4 SDLC implementation .............................................................................................. 24
   4.1 Requirement analysis ..................................................................................... 24
   4.2 System analysis and feasibility studies ........................................................... 25
   4.3 SDLC with Agile approach ............................................................................. 25
   4.4 Development ................................................................................................... 27
      4.4.1 Local development environment ............................................................ 28
      4.4.2 Web App Service .................................................................................... 29
      4.4.3 Azure DevOps ....................................................................................... 31
   4.5 Implementation of CI/CD pipeline .................................................................. 33
      4.5.1 Creating a build pipeline ........................................................................ 34
      4.5.2 Creating a release pipeline ..................................................................... 36
      4.5.3 Post-deployment phase .......................................................................... 37
   4.6 Additional development tasks ......................................................................... 38
   4.7 Maintenance and security .............................................................................. 39
      4.7.1 Secret variables ...................................................................................... 39
      4.7.2 Kanban and Azure Boards ..................................................................... 40
1 Introduction

There is a great amount of information about completed and actively developed software, but with that comes a great variety of used techniques and build tools that help to achieve similar developments goals. Experienced programmers or engineers have the knowledge and skills to analyze, adapt and implement customized technical solutions from the miscellaneous information pool. Those are the main reasons why the outcomes of a software development task depend heavily on the skills of the programmers and engineers. The global demand of skilled software developers is easy to understand: to build a viable software, companies, organizations, or individual entrepreneurs need either financial resources to create one or technical expertise to make one themselves.

Automation tools are built to reduce the dependency of software developers or engineers. Automation makes the software development work more effective -in both time and cost- when resources can be allocated on tasks that need creative thinking or technical consideration and effort, like system integration, feature design or debugging. Partially or fully implemented automation reduces the time needed to work on often repeated, menial tasks which in return, reduces operational costs. The continuous integration and continuous deployment (CI/CD) pipeline tools help to bring the latest software version to end-users, and developers can be sure that every release build matches the desired code quality when it has successfully gone through the same pipeline processes. CI/CD pipeline includes automation tools for development, integration, testing, and deployment. With these tools, developers can reduce the delivery process complexity, reduce chances of human errors, and increase the effectiveness of the development work. Frequent updates and maintenance are achieved by implementing shorter actions cycles, which are called iterations. Smaller changes in the code are easier to manage and are less disruptive to the software delivery process. Together the iterative workflow and automated CI/CD pipeline provide fast feedback loop and more frequent, reliable code delivery.

CI/CD pipeline is also a tool to adopt agile practices and mindset between business experts, developers, and software engineers. Modern software products and services follow iterative processes and agile practices during the development to bring value to the end-user. Software development projects can range from simple to complex and have evolving requirements, but the end game is always the same: to produce a high-quality, production ready release. That is why I chose to present this thesis to the stakeholders: by creating a continuous integration and continuous deployment for the demo application in Azure, the future development team can focus on the development work and less with supporting
operations. At the same time, these tools enhance the web application delivery process by going “agile” (appendix 2).

The original development commission came from Vihdin-Nummelan Kylähistoria Ry, a non-profit association who had collaborated with Laurea UAS to create a web application to replace an older website. Laurea students Larisa Pyykölä & Edina Petróczki had created a React application demo, called “Nummelan kylähistoria”, for the association (appendix 4) in early 2020. The original objective of this thesis was to continue the development work by creating a working server-side for a React web application and then deploying it using Azure services. Later, an additional objective was included: to examine the Azure and its microservices for future development implementations that will use Laurea UAS’ subscription in Azure. The preconfigured development environment could work as a case study of how to use Azure services with student development projects that need a web hosting service. A CI/CD pipeline in Azure was designed to support all the activities and goals with built-in agile tools.

This thesis includes numerous different technologies, terms and abbreviations that are used regularly in the IT industry. I have included a glossary of terms in the beginning of this thesis and they that are used extensively in this work and are closely connected to the subject area. Some of these terms have one or more meanings. Therefore, I only included a single definition per term. Some of the abbreviations and technology concepts are explained in simplified manner.

1.1 Vihdin-Nummelan kylähistoria Ry

For this thesis, Vihdin-Nummelan Kylähistoria Ry is referred either by its name or as the association. The origins of the association began when a group of volunteers from the Community College of Vihti started to gather local historical information about Vihti in the late 1990’s. The operation started as casual hobby, but as the operation grew more people joined in the effort to collect local history: the most notable member being Veikko Helle, the former member and spokesperson of Finnish Parliament. By this time, the project was named “Koti-Nummela” and the group began collecting information more systematically. In the early stage the group would send brochures to villagers of Vihti, in which they described the project and encouraged people to share their local knowledge with them. Villagers shared historical information and photos about the local area and their surroundings: living residences, streets, and roads. The operation also collected villager interviews and memoirs. The project’s goal was to upload the historical data to the Internet for public use.

In 1999, as the Internet technology became more commonplace, Koti-Nummela published their own website. Koti-Nummela gathered a large collection of photographs and texts for over 16 years by Raimo Parikka and Ritva Miettinen. During that time, Koti-Nummela
organized 4 exhibitions in the Library of Vihti where they showed the collected material. Koti-Nummela got financial support from different sources e.g., from the Toini and Veikko Helle fund, from the Ministry of Education and from Länsi-Uudenmaan Säästöpankki. The website was hosted in the University of Helsinki’s servers and managed by Parikka, who was working there until he retired in 2015. His retirement ended the collaboration with the University of Helsinki, which meant that the Koti-Nummela website had to be moved to the Vihti library’s servers. Later that year, Miettinen retired from her duties of collecting and maintaining the data. This signified the official end to the Koti-Nummela project.

A non-profit association “Vihdin-Nummelan Kylähistoria Ry” was registered in 2019. The association has 40 members, and it is based in Vihti. Heikki Lindfors is the chair of the association and Armi Salenius is the secretary. In 2018, members of the association planned to continue the work of Koti-Nummela project and contacted Laurea UAS representatives. The members of the association were interested of creating a modern website or application by using the historical data as content. The long-term goal for the association is to create a mobile application where a user could use geolocation services to look for specific areas of Vihti municipality and review the history of local houses, streets etc. mentioned in the application. These features were implemented in the demo application by using Google Maps and WordPress REST API.

1.2 Laurea University of Applied Sciences

Laurea University of Applied Sciences (formerly Vantaa University of Applied Sciences and Espoo-Vantaa University of Applied Sciences) began operating in Vantaa in 1991. The current name, Laurea University of Applied Sciences, was adopted in 2001. Laurea UAS has eighteen degree programmes for students to enroll in 6 different campuses. (Laurea 2020b.). Laurea UAS operates in close interaction with employers and non-profit organizations, who are interested to include students with their project and cooperate with them. Laurea UAS (Laurea 2020a) calls this operating method as Learning by Developing (LbD): In LbD projects students and clients work together and try to find a solution to a client’s business problem. Laurea UAS has cooperated with the association for two project phases (Figure 19, 43). Former Laurea students had created a study and a demo application for the association’s application development project.

2 The project

Vihdin-Nummelan Kylähistoria Ry and Laurea UAS began the collaboration project of creating a modern platform for the association’s historical data, which included approximately 4000 pictures and 8000 pages of text of local Vihti history. The original website (appendix 1) is still
in use, but it was created with technologies that are not viable anymore. A new solution is needed for hosting and maintaining the association data. The project was first taken on by Markus Salonvaara, who was issued to study development options for a mobile application. The organization (precursor organization of Vihdin-Nummelan Kylähistoria) needed a new platform for their website content because the current website’s technology was unpractical, and it had scaling issues with mobile devices. Salonvaara concluded (2018, 27) in his thesis that a progressive web application (PWA) developed with React or Polymer would be a feasible option.

In the next project phase, Larisa Pyykölä & Edina Petróczki designed the specifications of the new application based on Salonvaara’s suggestions. The development team, Pyykölä and Petróczki, decided to create a web application for the association. The team developed a web application based on a headless CMS (WordPress) and React. This system architecture (appendix 4) was used for two main reasons. React is a modern JavaScript library that enables the option to later convert the web application into a mobile application. The second reason was that the architecture makes it possible to transfer the old website content to the WordPress REST API and simultaneously continue the development work. The web application solution was designed to be responsive, accessible, intuitive, mobile-friendly and most of all, easier to operate without deep technical skills and minimal costs, as described in their thesis report (2020, 9). Their work did not include a web hosting service, because the final web hosting service was undecided by the client at the time. The development team used Google Cloud during the development, but it was discontinued when the team left the project. The development team had collected feedback from other students about different app functions and features. The team had performed usability, functionality, and UI testing. This thesis project began when Pyykölä shared the development team’s work with the author, which included the application prototype documents, the application’s source code and instruction material for the application and the WordPress REST API.

In the initial virtual meeting (2020) the association members stated that they wanted to move the content first to the new platform and then look for more permanent solution for the web hosting service. The association planned to transfer of the original website’s content independently to the web application’s API platform (WordPress) while the application is being developed further by Laurea students. The association did not have any new requirements for the application, because they were still in line with the requirements that were mentioned in the development team’s thesis report. Lindfors mentioned that Vihti municipality was ready to add the application to their servers in the future. The content transfer was the association’s most immediate requirement. The association did not specify a timetable for the next project phases, but they were interested to continue the development collaboration with Laurea UAS.
Me and thesis project supervisors Arvind Sharma and Jouni Takala conducted a meeting where we discussed the best course of actions in regards of the project. It was decided that this project does not focus on the front-end development of the demo application and instead the focus is on the server-side of the application. New tasks for the project were to find a web hosting service and deploy the demo application in a convenient platform, so it would be possible to see how the data transfer in WordPress REST API interacts with the demo application. The deployment is successful when the application can share the data with WordPress API (appendix 4) and users can interact with the application. After this, the application development could perhaps continue with new students. The stakeholders also discussed about using Laurea UAS’s Azure subscription for creating a functional server side for this project. I was tasked to examine the available options in Azure and setup a development environment for the next development phases.

2.1 Project goals

The development team had created a demo version of the React application for Vihdin-Nummelan Kylähistoria Ry. Their thesis work (2020, 44 –45) mentioned client requirements and missing features list for future development. Pyykölä provided the project material to me and to the association. The material contained: the source code of the web application, tutorial videos about WordPress, prototype files, and instruction material of how to migrate WordPress files into a new web host and how to use API keys. The WordPress material contained a sample data for the development.

This thesis’ first goal is to deploy the demo application in Azure. This action is needed for the data transfer project, where the old website’s content is manually added to WordPress. The second goal is to setup a working development environment for the project by using Azure services. This action is needed for the future development phases. Previous project phases created the application base, but now the project needs to be integrated to Azure hosting services. When the project has been successfully integrated to Azure, it is possible to deploy the application and then continue its development and operations. The thesis project is, in a sense, an interphase in the association’s project lifecycle.

After the initial virtual meeting, the project team decided that the primary client for the thesis is now Laurea UAS, since the project focus was related to the back-end operations in Laurea’s Azure cloud environment and the benefits were more aligned with Laurea UAS’ goals. The back-end development in this case meant combining the source code, WordPress REST API, and a web hosting service into a functional and sensible operation, with the future development in mind. Laurea UAS uses Azure IT architecture for their learning and communications systems. The organization’s ongoing Azure subscription allows IT lecturers to use Azure as a learning platform with different study modules. Laurea supervisors wanted to
know more about using Azure and Azure DevOps services for the web application development and how these services could improve future LbD projects related to the development collaborations between companies and Laurea UAS.

2.2 Project limitations and risks

The web hosting costs must be considered in the project because the association has a limited amount of financial resources. The association had applied for funding from the European Union, but as of November 2020, the funding was still undecided, partly because of the global Covid-19 pandemic impacted the funding process. The pending EU funding could help to solve this issue later, when they must pay for the web hosting. The development work itself does not accumulate costs, because students works as the developers as part of their studies.

The association volunteers do not have technical capabilities to partake in the application development work. The technical complexity must also be kept relatively straightforward, so the association members can operate the demo application with relative ease. The association members are working only on a volunteering basis, so the process of transferring the content from the old website to the new one may require additional help from Laurea or some other source. According to Lindfors (2020), a volunteer journalist would do the manual work and the transfer process would take approximately a year to complete. As a result, the transfer process may generate additional costs.

The demo application creates some limitations as well: because the technology has already been decided, new developers must adjust to the project needs and learn to use the “technical stack”. Learning a new set of tools or software can be a time-consuming commitment which needs to be considered. For example, I had limited experience with Azure and no previous experience with React when the project started. I was free to study and familiarize myself with the project during the summer holiday season.

The demo application went through one development iteration, so multiple requirements were waiting to be developed. Most of these requirements are not included in this work, because it would need front-end development, which was not the focus for the thesis. To improve the quality of the demo application, new developers need to conduct systematic testing. The previous development team did not have time to implement comprehensive testing.

There is a risk that the requirements might change during the development. Smaller changes are expected, since the demo application is still in development. However, if stakeholders want to make more drastic changes, they must return to the pre-development phase to plan a new application version. If that happens, stakeholders must end this development project and start a new one with Laurea students or other developers. If the demo application has new
technologies that does not serve the association’s purposes, there is no need to continue the development. The association may not want to commit to a new system if the system architecture is too complicated to operate.

The global Covid-19 pandemic caused some limitations to this project. All the project meetings were conducted virtually via Zoom and other communication methods were phone calls and emails. These precautions were the result of Finnish Government’s regional recommendation to switch to remote work and to limit people gatherings. Laurea UAS and the local libraries were briefly closed when the project started, but it did not create major limitations to the availability of research material. Covid-19 pandemic is a contagious disease that causes mild and severe symptoms (for example a fever, a cough or breathing difficulties). The disease can lead to serious complications, like pneumonia or death (THL 2020).

3 Research and organization methodology

This thesis is primarily a development project, because it involves a product that needed system development and planning of next project phases. I conducted interviews and discretionary approach to the system development because there were multiple factors to consider: what tools to implement and how extensive this thesis will be. The server-side was designed to be a temporary solution, until the association is ready to move on to a more permanent solution. System development needs experimental approach to implement the current technologies (demo application and the API services) to Azure’s ecosystem, because there is a great number of unpredictable variables, but also freedom to implement different solutions.

The primary research material is gathered from the project meetings, private conversations, and the development team's documentation. I use secondary research material (videos, documentation, articles) to study the development and Azure platform features. The research material and software documentations are collected mostly from online because the IT technology in general is constantly developed and the most up to date information is available online. Written material is used to analyze the related topics and concepts. The scope of this project is on the application deployment and to provide practical information about CI/CD pipeline in Azure DevOps. The research methodologies were selected because the concepts are related to the used technologies and are relevant within the IT industry. The thesis work represents a subjective view of the concepts, but it can provide new and meaningful ideas to Laurea UAS to future development projects.
3.1 React web application

React is a component-based JavaScript library for building user interfaces (UIs). React is sort of a view layer that renders the updated application data through multiple React components. They split the UI into independent, reusable pieces that are either JavaScript classes or functions. React is developed by Facebook, but the technology is open-sourced, and it is widely used as a front-end development tool. (React documentation 2020a.) React is primarily designed for web applications, but it is a multi-platform solution. React components can perform platform-specific operations on any target device, like mobile devices or laptops. (Boduch & Derks 2020, 15–16.)

A web application uses HTML, CSS, and JavaScript to build website applications that are delivered via the Internet. A web application works on mobile and desktop devices and users can access it via a browser. All PWA’s are web applications, but the exact definition of PWA and its features are different between browser developers. Native applications are developed for one mobile system, like Android or iOS. Hybrid application is a combination between the web app and native app: it runs through a browser but can also store some data on the device.

3.2 WordPress as headless CMS

WordPress is an open-source content management system (CMS). In CMS users can add, edit, and delete content of a website. It can be used to create a website or as a development tool. WordPress REST API uses a WordPress website as the primary content repository and the website exchanges the content data with other web services using the REST API technology. REST API is a REST architectural style to process HTTP data requests in an API. When a request has been made, then the API transforms the data to a specifically formatted response. In the demo application, the received response format is JSON. (WordPress 2020.)

Headless CMS is a back-end focused solution to the traditional (CMS). The content layer is for adding and managing content in CMS and the front-end layer, “the head”, is separated in a headless CMS (Camden & Rinaldi 2017, 139). The demo application takes care of fetching the data from the WordPress API and presenting the visual layer to the end-user. According to the development team, this implementation enables future development teams to “reuse the codebase and later create a mobile application” (Pyykölä & Petróczki 2020, 20). The main benefits of using WordPress as headless CMS are the flexibility for future development work and the option of managing the content from one place.
3.3 Microsoft Azure

Microsoft Azure is a public cloud computing service platform that offers computing resources to clients over the Internet. Azure is owned by Microsoft Corporation. Azure is a multipurpose cloud service provider where its services are used for example: to set up a cloud storage, host web applications, analyze data with AI, or develop applications. Cloud technology makes it possible that clients do not have to own any server hardware, but rather use the computing resources of a cloud service provider, like Azure. Soh, Copeland, Puca, and Harris (2020, 5) writes that the Azure infrastructure is geographically identified globally, regionally and by zones to provide high availability to customers. Azure Portal is a web-based platform where users can build, manage, buy, and monitor all the cloud resources by using the graphical user interface. An Azure subscription represents a collection of Azure services for identifying, managing, and providing governance to the service resources and an Azure account can have multiple associated subscriptions (Soh et al. 2020, 8 – 9). Azure Active Directory (AD) is an identity and access management service where global administrators can allocate access to resources based on the user’s role.

Azure’s main service models are Infrastructure as a service (IaaS) and platform as a service (PaaS). The IaaS model offers network, storage, and virtualization infrastructure to businesses. The PaaS model includes infrastructure like in the IaaS model, but hardware and software tools are made available over the internet to customers without them having to worry about the hosting infrastructure. It is used by software and application developers who want to build, test, deploy, manage, or update their work in Azure. The third main type of cloud computing is Software as a service (SaaS), in which the service is made available on-demand over the internet via a third-party provider, usually for a monthly subscription fee. Customer only need to create data and the service provider manages all aspects of the application environment. (Introduction to Azure fundamentals: What is cloud computing 2020.)

Cloud services provide a cost-effective alternative to on-premises infrastructure, according to Webber-Cross (2014, 30). Customers can avoid making large initial investments in hardware and software licenses and on-premises utility costs, like electricity and rent. With cloud services, customers pay only for the cloud computing resources they use. Cloud services are categorized in terms of access, size, and ownership.

There are four deployment models for cloud services: public cloud, private cloud, hybrid cloud, and community cloud. Public cloud services are hosted by a cloud service provider which makes the services available for the public to use. Private cloud services are similar to public cloud services, but the client organization hosts the services internally or externally and has total control over the cloud infrastructure. Community cloud is a private cloud shared
between a limited number of users and hybrid cloud is a mixture of the other three models. (Webber-Cross 2014, 30)

Azure has 22 different categories of cloud services, including services like Analytics, Blockchain and DevOps (Microsoft Azure 2020a). Microsoft Azure is one of the largest cloud service providers on a global scale. Other well-known cloud service providers are Google Cloud Platform and Amazon Web Services (AWS). They support various programming tools, languages, and frameworks for managing software applications and services through their own network of data centers.

3.4 Azure App Service

Azure App Service is a HTTP-based microservice of Azure that can build and host web applications, mobile applications, and REST APIs with different programming languages like .NET, Node.js, Python, Java or Go (The Developers Guide to Azure 2020c, 7). Other languages are also supported with extensions. Azure Web App Service uses an IIS (Internet Information Services) server or an Apache server to make web applications available in the web and it is one of the most often used PaaS in the Azure service catalogue, according to Soh et al. (2020, 233).

Web App Service is used to deploy applications to a domain name. The default domain name in Web App Service ends with the suffix “azurewebsites.net”, but custom domain names are supported in the paid plans. Azure App Service can utilize other Azure microservices in the Azure ecosystem, like Azure DevOps pipelines. Azure App Service enables automated management of a web application. With the combination of DevOps pipelines and the App Service, users can utilize continuous deployment from various Git repositories (e.g., GitHub, Azure Repos) by pulling in the latest updates, according to Azure documentation (2020f).

3.5 Agile

The origin of DevOps came from the need to streamline the software development processes. According to Kennaley (2010, 18—22), until the turn of the millennium software projects followed a linear “single-pass” process models to deliver software products. These models’ simple and sequential steps were time-consuming, and they did not respond well to changing needs. In 2001, the Agile alliance formed by industry professionals met in Utah conference and created the Agile Manifesto. It contained a set of agile principles (appendix 2) that paved a new way to approach software development. It emphasized “individuals and interactions over processes and tools” (Girvan & Paul 2017, 17). Over the years Agile frameworks and methods have evolved from the original manifesto and nowadays agile practices are adopted in different fields of activity (Macaulay 2019). Agile way of thinking was the logical precursor for DevOps, according to Sharma (2017, 39).
### 3.6 DevOps and Azure DevOps

DevOps is an aggregation of two types of activities: development and IT operations. IT operations are generally viewed as a separate department from development and it entails the people and management processes that are related with IT infrastructure and IT services like helpdesk support, server and device management and application delivery. The structure and tasks of IT operations are based on the needs of the organizations, so the structure and the task can vary.

![DevOps Loop Diagram](image)

**Figure 1:** The continuous DevOps loop. This kind of loop is used to visualize the DevOps process in software development projects.

DevOps can be defined as an organizational structure, framework, mindset, trend, or a set of practices or tools. The IT industry does not have an exact definition, because DevOps can be used in multiple ways. According to Carstensen, Golden and Morgenthal (2012, 197), DevOps incorporates agile quality assurance, system operations and development into a new collaboration delivery unit. This kind of cross-functional collaboration activity is often depicted with a DevOps loop (Figure 1). Guckenheimer (2018) describes the DevOps framework in software development as “essential practices that include agile planning, continuous integration, continuous delivery, and monitoring of applications”. Guckenheimer states that the overall objective is to automate operations more and enhance the development time cycle by working on smaller action and information batches. DevOps can be used to improve code quality in software development projects or to improve the performance of operations in system development. DevOps aims to deliver more value to end users and project teams by enabling continuous delivery of products and services.

Azure DevOps is a set of software tools created by Microsoft to provide DevOps practices and tools as services. Azure DevOps is a microservice of Azure and it can be integrated with other Azure services on to external services like GitHub and Jenkins. Azure DevOps provides tools
for source control, testing, support, and automation. Azure DevOps tools can be configured and used individually or in combination with other tools.

3.7 **CI/CD pipeline**

CI/CD stands for continuous integration and continuous delivery/deployment pipeline. The term CI/CD pipeline is a DevOps practice where the software delivery process is automated (Figure 2). The code can be released into different development and production environments: development, test, or production for example. Continuous integration (CI) means that the codebase is build, tested, and merged with the build pipeline automatically, when a new code commit is made to a Git repository. A Git repository holds all the versions of a code, so it is distributed a version control system (Tsitoara 2020, 11). The integration process can also be manual. CI is followed by continuous delivery. It means that the codebase of software, a build, is ready to be built, tested, configured, and deployed to a specific target at any time. That target can be either a production environment, development environment, or a testing environment. Continuous delivery is manually deployed to a target. Continuous deployment includes all the preceding steps, except the deployment is automatically triggered to the target environment.

![Figure 2: Automated practices of software delivery.](image)

The pipeline helps developers and organizations to focus on the development work by automating certain tasks in the application development. A pipeline is comprised one or more stages. When a pipeline meets certain trigger conditions (e.g., new build was created) each stage within that pipeline is executed manually or automatically. A stage contains one or more jobs, which are series of smaller steps defined in the pipeline. Steps can be scripts or tasks. A job needs an agent to perform scripts or tasks, because the agent is assigned to execute tasks and scripts. (Soh et al. 2020, 512.) These concepts are also explained in Table 1 (21). The automation tools in the pipelines helps to minimize the amount of work between different project phases. CI/CD tools affect the supporting infrastructure of a software system, like testing, code release process, deployment, and monitoring. The pipeline process includes automated stages that are triggered by jobs and tasks in the Azure DevOps ecosystem (table 1, 21). There are various configuration options for build runs and releases:
for example, in Azure DevOps, the email notification system can inform the team members via email when a build succeeds.

<table>
<thead>
<tr>
<th>Trigger</th>
<th>A trigger tells a Pipeline to run.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>A pipeline is made up of one or more stages. A pipeline can deploy to one or more environments.</td>
</tr>
<tr>
<td>Stage</td>
<td>A stage is a way of organizing jobs in a pipeline and each stage can have one or more jobs.</td>
</tr>
<tr>
<td>Job</td>
<td>Each job runs on one agent. A job can also be agentless.</td>
</tr>
<tr>
<td>Agent</td>
<td>Each agent runs a job that contains one or more steps.</td>
</tr>
<tr>
<td>Step</td>
<td>A step can be a task or script and is the smallest building block of a pipeline.</td>
</tr>
<tr>
<td>Task</td>
<td>A task is a pre-packaged script that performs an action, such as invoking a REST API or publishing a build artifact.</td>
</tr>
<tr>
<td>Artifact</td>
<td>An artifact is a collection of files or packages published by a run.</td>
</tr>
</tbody>
</table>

Table 1: Key concepts of Azure pipelines (2020b).

3.8 Systems development lifecycle (SDLC)

The framework for this project is the Systems Development Life Cycle (SDLC) as described by Turban, Volonino & Wood (2015, 433). SDLC is a framework for system development and maintaining information systems. SDLC is also described as software development process model and several different SDLC processes exists that follow this SDLC framework. Software development lifecycle uses the same abbreviation with systems development lifecycle and both forms are used interchangeably. In this thesis case SDLC refers to system development lifecycle since Azure and its microservices (like Azure App Service and Azure DevOps) work on PaaS model level and offer a cloud computing system of dedicated services for the React web application. SDLC is a structured process that includes different models of SDLC framework, like waterfall model, iterative model, V-model and Agile model (Kazim 2017, 15). All methodologies in SDLC follow a set of sequential and iterative processes where the goal is to improve the overall software quality and development process. In SDLC there are five main development phases: requirements analysis, system analysis and feasibility studies, development, implementation and maintenance.
3.8.1 Requirements analysis

SDLC starts with the planning phase which includes the identification of functional and data requirements. A group of stakeholders meet and analyze the current system’s deficiencies or possibilities of improvement. Stakeholders discuss new system opportunities and risks in detail. The goal is to define the overall scope of the project and what resources are already available. More time spend on the analysis phase often increases the chances of success for the project, according to Turban et al. (2015, 433).

3.8.2 System Analysis and feasibility studies

This phase determines if the proposed new system solves the business problem. The proposed system is designed based on the client’s requirements. The blueprint of the new system comprises details of how the proposed system will solve the functional requirements alongside other fundamental factors (e.g. security, additional features). During the analysis, the project team conducts a feasibility study. The feasibility study can range from simple to complex, as well as free to costly (Litten 2018). The study plays a crucial part in the project, because it gives the stakeholders valuable information about the potential of the proposed system, and how likely the project will succeed. Feasibility study consists an assessment of technical, economic, organizational, legal and behavioral feasibility (Table 2, 23).

<table>
<thead>
<tr>
<th>Technical feasibility</th>
<th>Economic feasibility</th>
<th>Organizational feasibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determines the technical requirements of hardware and software and how well they work on the proposed system. The study also takes into account the possibilities of using the organization’s existing technology to solve the business problem.</td>
<td>Evaluates the possible financial benefits and costs and the projected timetable regarding the project. A multifaceted cost-benefit analysis (CBA) can tell how the benefits outweigh the costs. A CBA features tangible and intangible metrics, like customer satisfaction. The project team decides what monetary value they give to certain intangibles or how they are incorporated in the study.</td>
<td>Determines how the proposed system affects the company policies and politics.</td>
</tr>
</tbody>
</table>
Legal feasibility involves an investigation of legal, regulatory or environmental requirements. The project has to meet legal obligations and regulatory standards. Legal feasibility study also considers the ethical factors.

Behavioral feasibility evaluates how the project affects human behaviour. The proposed system might require a new set of skills which, in turn, might create resistance within workforce. Behavioral feasibility study answers the questions "can they use the new system" and "will they use the system" (Turban et al. 2015, 433).

Table 2: A table of the SDLC feasibility studies

The feasibilty analysis results can be reviewed formally or informally with the relevant stakeholders. The project team will decide if the project precedes as planned or not. If the system analysis results are insufficient or the project conditions have changed unfavourably, the team can re-examine the proposed system. Other option is to discard the project temporarily and wait for better circumstances to arise in the future.

3.8.3 Development

This phase starts the production of new system. The proposed system is developed and tested based on the design specifications and company standards. In SDLC, this can mean either the development of the IT infrastructure or the development of database or code. Developers should follow the best practices and use relevant and modern programming tools for the systems development.

3.8.4 Implementation

Implementation, or deployment, is the installation phase of the new system. It also starts the process of converting from the old system to the new system. The conversion can happen in multiple ways. In parallel conversion the old and the new version can operate simultaneously for a while. With direct conversion, the old system is discontinued and the new system is deployed at a certain point in time. A pilot conversion introduces the new system in a specified location where it can be tested. A phased conversion introduces a certain module of the new system in different phases to the public. When that module is working properly, a new module is introduced. This process continues until the new system is operational. (Turban et al. 2015, 434.)
3.8.5 Maintenance

When the new system’s operations are stabilized, the project can move to the maintenance phase. The final version is released to production. The system software can be developed further by updates, performance upgrades, bug fixes and new requirements. The new system operations are being monitored and improved, based on the feedback from system users.

4 SDLC implementation

Figure 3: SDLC theory and how it was followed in practice.

4.1 Requirement analysis

Requirement analysis was conducted with the thesis supervisors Sharma and Takala, as shown in Figure 3. Sharma and I investigated the project files and analyzed the best course of actions. Because Sharma was also the client for this project, he could provide access to Laurea's Azure resources during the operation. In practice these resources were available after the summer holiday season was over, so during that time I continued to analyze the development files and instruction material. The association's requirements were acknowledged in the client discussion, but at this point they served more as a reminder of the remaining development work. The development team had created new requirements based on the feedback they got from the pilot version.
4.2 System analysis and feasibility studies

I had previously studied the fundamentals of Azure but did not have any practical experience prior to this work. In terms of technical feasibility, Azure seemed to be the best option to be a temporary hosting platform and a development platform (Figure 3.24). The precise steps needed for the demo application to be integrated and later to be deployed in Azure were not clear from the start, but the initial conclusion suggested that the project is feasible. Azure provided several key benefits:

- The Azure platform and its microservices supporting CI/CD pipeline
- Opportunity to use pre-existing technology
- Hosting by Web App Service
- Familiar environment, at least for supervisors
- Integrated DevOps tools, like Azure Boards
- Student and collaborator changes in the team are easier to maintain
- Version control

The technical feasibility was possible, because Laurea lecturers had launched websites and web applications before using the Azure platform. The Azure DevOps architecture is built to use several Azure microservices, so I did not see a reason to add new, external services to the already complicated system architecture. Adding more systems with their own service dependencies to the project was not optimal. Economic risks were also minimal, because Laurea had an ongoing subscription with Azure and the project’s schedule was open. Because the application is still in development and the platform is not marketed to end-users, the amount of data moving across networks is limited. Any costs that may occur are either minimal or nonexistent. Behavioral feasibility study generated an unknown variable: will Laurea students and supervisors use these tools during the development? Legal and organizational feasibility studies were not conducted, because I assumed the project does not create any challenging issues.

4.3 SDLC with Agile approach

Agile methods are usually implemented alongside SDLC to improve the efficiency and output of the development team. Agile techniques help to split the project process into smaller, more manageable steps in situations where there are lot of uncertainty involved. Complex or large IT projects are more likely to succeed when the project team successfully completes one small part of the project at a time. Juvonen writes that software development projects that utilize agile methods are almost four times as likely to succeed compared to projects that follow the linear waterfall model (2018, 74). On the other hand, Juvonen states that in practice, the process model’s significance is not as important with small-scale projects.
Girvan and Paul (2017, 260) advise that for the agile methodology to be successful in a project, it needs to be introduced to a project team first. Team members need to be familiar with the practices and tools before they can adopt agile practices. The advice from Girvan and Paul was one of the reasons why I chose to present the idea of agile because Agile practices are closely related to the Azure DevOps tools.

When the project started, it was not clear how the thesis work would advance the association’s project. As I learned more about the related subjects, I decided to implement an Azure CI/CD pipeline to as a case study for Laurea UAS’ business objectives since it matched well with the association’s immediate requirements. My role was more of an operator or “a solution developer”, as Girvan and Paul (2017, 118) have described when defining the stakeholder roles in agile development: “The solution developer interprets the business requirements and translates them into a deployable solution that matches the non-functional and functional requirements.” In agile systems development, a solution developer can be equally part of the development work and the “behind the scenes” operations, like building software components and perform unit testing. Or the work can entail only the aspects of IT operations that goes beyond the software development. A solution developer’s work is defined on a case-by-case basis and the role is flexible, but the main objective is to ensure that the desired product or service is delivered.

The future development process would proceed like this: the iteration loop (Figure 4, 27) for development would continue until the final production release is created or when the project ends. Each iteration would have a specific bug fix, old or new requirement, or an improvement that students would work on and implement them to the demo application one by one. The DevOps platform and CI/CD pipeline makes it possible to have multiple deployments for different branches and constantly develop the application further. New development teams can independently decide how they would approach each issue. They can use whatever framework and practices they want, but students and Laurea UAS should implement agile practices when possible (appendix 3) in order to complete the unfinished development project and to produce a modern software. This thesis work created a possible foundation for new teams to take over the application development.
Figure 4: The designed lifecycle of the demo application development project. Developers could be more efficient if they adopt the agile mindset and aim for continuous cycle of improvement.

The idea of creating a build and release pipeline came from personal observation of the situation. The most immediate task was the deployment, but what happens after it? I tried to look beyond this project phase and create a solution that would serve both the short- and long-term goals of the demo application project. This system could help to mitigate the impact when students or project administrators leave the project. New collaborators could be included by using the already provisioned Microsoft accounts. New collaborators are in this case the students and Laurea UAS personnel. The project teams are better prepared for organization changes because the configuration work can be done more easily in Azure. Projects teams can enforce the adopted agile practices to newcomers and introduce agile concepts in a more pragmatic way. These observations in project management were partly originated from the examination of the administrative identity management conducted by Carstensen et al. (2012, 131–134).

4.4 Development

The application development was initially started by the previous development team. The demo application technology is explained in more detail by the development team (Pyykölä & Petróczki 2020), but I have included the demo application's system architecture (appendix 3) in this thesis. This work contains system development and minor development tasks that were needed to have a functioning web application. The integration process included creating a Web App service, starting an Azure DevOps project, and making small changes to the source code, so it can access WordPress.
Figure 5: The project timeline shows the selected Azure tools and technologies. Options marked with grey font are well-known alternative options. This figure is a modified version of the original illustration made by Jackson (2019).

Figure 5 shows the project’s timeline and the actions that were conducted in Azure environment. The source code was first uploaded from GitHub to Azure Repos. CI/CD pipeline started whenever I made a change to the codebase or if I manually started a build. The build pipeline and release pipeline resulted a “build”, which was then released to Azure Web App Service. It deployed the application to a website domain that was shown in the App Service settings (Figure 6).

Figure 6: App Service "NummelaVihtiAPPv1" dashboard. App Service configurations were made in Azure Portal.

4.4.1 Local development environment

The work started with acquiring the source code from the previous project collaborators. The documentation for the future development provided instructions for continuing the front-end and back-end work and the source code was inspected by me on a local development
environment. I used VS Code as code editor to make changes to the source code and installed NodeJS, which is needed for React development. Npm has two main components: command line interface (CLI) and the registry (npm 2020).

In figure 7, the first command line is the project directory, where the files are located. The second command, “npm install” installs all the software packages (dependencies) that are needed for the React application to work. For example, “react”, “react-dom”, “emailjs-com” and “googlemaps-react”. These packages are defined and listed in a “package.json” file. After this setup, I typed “npm start” command which runs the app in the development mode. The application was now accessible to view via a browser.

**VS Code terminal**

```
   cd G:\localCodingProjects\Nummela_project-master
   npm install
```

Figure 7: After installing NodeJS to my PC, I installed necessary packages by using the VS Code terminal. Text lines marked with blue are commands.

4.4.2 Web App Service

One Web App Service was used to deploy the WordPress database and restore it from the migration file and another Web App Service to host the React demo application. Hosting a web app or a website means that a web server stores the system and delivers them to users through a browser. These systems are distinguished by a unique domain name. Using two different app services was not the initial idea, but in which the project settled upon. Two different app services in this case are a more secure solution, because the two layers — back-end and front-end — are isolated from each other. The WordPress database is managed in one domain and the presentation layer in another domain. For this project phase, this kind of implementation was appropriate since the project was planned to be temporarily deployed in Azure.

During this phase I worked with the client, because he had the administrative access to Azure and its services. We created a resource group in Azure Portal and then created a Web App Service. This service enabled that the application has a hosting service for the WordPress API. After the Web App Service creation, the application was ready to be deployed. The default web app page was working and was visible when visiting the corresponding “azurewebsites.net” domain. We followed the same steps described in the Microsoft Azure documentation (2020b). The Web App Service configuration were outlined to match Laurea UAS’ desired values.
Then we checked the WordPress site, where we had admin credentials. The application had to be migrated from a local “wp-file” to the new WordPress website by using a plugin called “All-in-One WP Migration” tool. This plugin was also used to create a copy of the WordPress website. We installed the plugin and then imported the local “wp-file” to the platform. We imported the file to the website and used older credentials to login to WordPress and update the settings: I added the Azure Web App Service URL to the WordPress website settings. This URL fetched the WordPress posts data to the application’s front-end side.

```javascript
import axios from 'axios'
const baseUrl = 'https://nummelavihtivillage.azurewebsites.net/nummela/wp-json/wp/v2

// fetches data for "projektista" page
const getHomePage = async () => {
  const response = await axios.get(`${baseUrl}/homepage`)
  return response.data
}

export default { getHomePage }
```

Figure 8: Code from HomePage.js.

Figure 8 displays the updates I made to the React application codebase: in the baseUrl, I added “/wp-json/wp/w2” to the end of baseUrl path. I repeated this process with all the files that had the same baseUrl code. After this the WordPress API connection was established. With the second Web App Service I wanted to change the runtime stack from PHP to ASP.NET, because I had examined new setup options for integrating Web App Service to the pipeline. Runtime stack is a framework, platform or library that runs the code on server-side.

I used the command “npm run build” to build the application to production in the build folder. These files were first uploaded to GitHub and later to the Azure Repos. These files represented a production build. It was created for enhancing the system performance and optimizing the build files. The original development build was still available in a zip-file. In software development, there are usually three different branches developers work on while programming an application: development branch is for testing new versions or features, test branch is for debugging, and production branch is for releasing a stable version of the project to the end-users (Tsitoara 2020). According to Dan Radigan (2020), this branching system allows development teams work in parallel and support sustainable development: developers have a stable release version to revert to if issues are encountered.

I created a private GitHub repository for the project, so that I could deploy the application in Netlify. Netlify was connected to my remote GitHub repositories where it pulled the source code and deployed it by using their content delivery network (CDN) (Netlify 2020). Netlify deployment was used for preview and testing purposes. I deleted the Netlify deployment
after I had seen the application working properly on a browser. GitHub had an option to create “GitHub Free for organizations”, but it was not suitable for this project. Students leave development projects as they graduate and administrating access in a GitHub organization’s repositories (GitHub 2020) would have created unnecessary steps and layers to the development. Administrating access in Azure Repos was a better option, since lectures can manage access there instead of students managing access in GitHub.

4.4.3 Azure DevOps

The client and me created an Azure DevOps project called “NummelavihtiApp” to test the service features and deploy the demo application. To use Azure DevOps services, we needed to have a Microsoft account which were already provided by the university. The administrative access was once again needed to create the Azure DevOps project, so the client’s presence was needed. The client was the project administrator in Azure DevOps and could give permissions to the group members. The Azure DevOps group included me and the client. The client gave me the necessary permissions, like the permission to create service connections in Azure DevOps.

I imported a GitHub repo manually to Azure Repos by selecting “import repository” option from a drop-down menu in DevOps project page. The GitHub repository was private, but I authorized the service connection in Repos and entered my GitHub profile’s credentials (Figure 9, 32). After this, the import process began, and I had successfully uploaded the source code to Azure Repos. Azure DevOps now had the project files and I moved on to the next phase: creating a build and release pipeline.
I imagined the demo application could benefit from a steady CI/CD pipeline. Because one of my tasks was to study the CI/CD pipeline, I had to analyze how the future development and operations would work in Azure DevOps (Figure 10) in Laurea. My thesis progress started with the plan phase and went through the cycle to the monitor phase. The process loop started again when I made smaller development tasks to the demo application, like EmailJS integration.

Figure 9: A prompt to import a Git repository. This image is an example from Azure DevOps Documentation (2020a).

Figure 10: The designed DevOps loop for the project. The project would start with a new plan and go through the cycle continuously while the demo application development is ongoing.
According to documentation, Azure DevOps Services supports two types of version control: centralized version control TFVC (Team Foundation Version Control) and distributed version control Git (2017). Both version control systems manage all versions of a source code and keeps track of the changes the software development team makes. With version control system, users can compare new code to previous versions and create specific branches for different development tasks. Most of the modern software projects use Git as the version control provider and it is also the default option in Azure DevOps. The Microsoft documentation (2020) recommends using Git unless users have a specific need for centralized version control features.

Azure Repos is a source code repository that supports TFVC and Git. Azure DevOps also supports external repositories like GitHub, but for future code management purposes I decided to use Azure Repos instead. Laurea University had a pre-existing Azure account and Vihdin-Nummelan Kylähistoria Ry had planned to continue the development work with Laurea University in the future, therefore it was sensible to keep using the same administrative controls with Azure Repos. With GitHub, it would had been necessary to create a new organization account.

4.5 Implementation of CI/CD pipeline

To create a CI/CD pipeline, I needed to have an Azure DevOps project and a Web App Service ready for the pipeline. A private DevOps project, “NummelavihtiApp”, was created with the client (Figure 11, 34). The Git repo was the Azure Repos. Azure pipelines were created and configured with the Classic editor (Figure 12, 34). The build pipeline created an artifact that was consumed in the release task. The task was defined to use Web App Service to deploy the release build into production. It was possible to make configurations for different development branches, but for this thesis I only worked on the master branch of the application. Some configuration details and related technologies are not mentioned in this thesis for the sake of brevity.
Figure 11: The DevOps project UI. The colored icons under the plus icon on top left are Overview, Boards, Repos, Pipelines, Test Plans and Artifacts.

Figure 12: The process of how to define a pipeline with the Classic user interface. The image is from Azure DevOps documentation (2019a).

4.5.1 Creating a build pipeline

The creation of build pipeline started with selecting the pipelines option from the vertical menu in the DevOps project panel (Figure 11), and then selecting the “create new pipeline” button. I used the classic editor to set up the pipeline. I selected from the options the source of the pipeline (Azure Repos), the team project (NummelavihtiApp) and the branch (master). After this step I selected an empty job for the build jobs. The first task of the agent installs npm, the second task runs the custom command “run build” in the files, and the third task publishes the build artifact that will be used in the release pipeline (Figure 13,35). I defined the name of the artifact, the file path in which it will be published and the location for the artifact to be in Azure Pipelines. The YAML file (Figure 14, 35) was also created in the root of the repository. I also enabled continuous integration from the options, which in this
case meant that when a new code commit was made to the master branch, it triggers the build pipeline (Figure 14, 36).

![Azure DevOps](image)

**Figure 13:** The build pipeline, named as “NummelavihtiApp-CI”.

```
1 # Node.js with React
2 # Build a Node.js project that uses React.
3 # Add steps that analyze code, save build artifacts, deploy, and more:
4 # https://docs.microsoft.com/azure/devops/pipelines/languages/javascript
5
6 trigger:
7 - agent:
8
9 pool:
10  vnetmap: 'ubuntu-latest'
11
12 steps:
13 - task: NodeTool@@
14 inputs:
15  versionSpec: '10.x'
16  displayName: 'Install Node.js'
17  script: |
18  npm install
19  npm run build
20  displayName: 'npm install and build'
21
22 - task: CopyFiles@2
23 inputs:
24  contents: 'build/**' # Pull the build directory (React)
25  targetFolder: '${Build.ArtifactStagingDirectory}'
26
27 - task: PublishBuildArtifacts@0
28 inputs:
29  pathtopublish: '${Build.ArtifactStagingDirectory} dist or build files'
30  artifactName: 'www' # output artifact named www
31
32```

**Figure 14:** The YAML file that includes the tasks of the build pipeline.

After creating these settings, I selected “run pipeline” from the menu. This starts the agent job, and it runs through all the tasks I manually added and the related background processes.
When the build process was finished after a few minutes, I could view the summary of the run and check if there were any issues (Figure 15) in the run process. It took me several efforts to find correct setting for the build pipeline, so the above-mentioned process represents the path I took for a working solution.

Figure 15: A build pipeline run status. The artifact is published, and it is consumed by a release pipeline. Any commits I created in the master branch triggered the pipeline.

4.5.2 Creating a release pipeline

After the build pipeline was created successfully, I moved on to creating the release pipeline. The artifact I created in the build pipeline was needed, because the release pipeline will consume it: the release process deploys the component by using the Web App Service (Figure 16, 37). With continuous integration, the published artifact is linked automatically to the release pipeline. The release pipeline could be configured multiple ways, for example I could have created a scheduled release trigger, pull request trigger or a specific stage trigger. The trigger event for this project was launched whenever a new version of the build artifact was available.

A release in this context means the package or container that has all the actions or task defined in the release pipeline: stages, tasks, the values of task parameters and variables, release policies like triggers, approvers, and release queuing options. Multiple releases are supported, and they are stored in the release pipeline. Administrators can create certain
policies with each release, for example they can define the retention policies like how long to keep runs, tests and releases in the pipeline. (Azure DevOps documentation 2020d.) A release is not automatically deployed to a target, because it must first pass a set of circumstances (triggers, approvals etc.) that are defined in the pipeline. Developers can add pre-deployment approvals for the releases and link variables groups to a specific release.

It is possible to automate the entire release process between different stages and manage multiple releases in the release pipeline. Authorized team members can control the level of automation they want to implement to the release pipeline. I chose continuous deployment for this project. This meant that whenever a commit was made to the source code, build and release pipelines were automatically executed. I linked a variable group to the release that contained a set of API key values used in the demo application. The initial deployment was made manually, but after I had seen the system working, I switched to continuous deployment.

4.5.3 Post-deployment phase

When the CI/CD pipeline was done, I could access the demo application via a browser. The demo application’s URL was initially set up by the client when we created the Web App
Service. During this phase I could monitor the performance and usage of the application in the Azure Portal. When the application was deployed for the first time, external API services were not functioning properly.

4.6 Additional development tasks

The development team had implemented a mapping feature, where users could locate houses on a map. That feature uses a Google Map as an API. Another feature in the demo application uses EmailJS library to send emails from the “Yhteydenotto” contact page to an email address. A key-value pair is a data representation of application settings, where a key has an assigned value that contains configuration information. That value can be accessed with the key. The initial idea was to integrate these services to the Azure DevOps and test them in the demo application. This development ended when the demo application project was put on hold. I had planned to insert the variable group values with the association data, but this plan was not implemented.

Adding a new Google Maps API key was a straightforward task because the development team had left instructions related to the Google Maps API. I created a new Google Maps API key in the Google Cloud Platform with using my pre-existing Google account. I restricted the API key usage to only use the demo application’s website in the referrer section. The map feature in the demo application (“Kylähistoria” tab in the horizontal menu) at its current state shows a warning “Google Maps not working properly”. The browser’s console gave a specific error details and it informed me that I need to attach an active billing account to get full access to the Google Maps API. This action was not needed, because I only used my account to test the demo application’s map feature. I got different console error messages when I used different browsers: Google Chrome error indicated “enable billing” error and Firefox indicated “no API keys” error. I decided to continue developing other features because the map feature development was not a priority in the thesis.

EmailJS implementation began with checking the documentation and instructions in the EmailJS website. I created a new account for EmailJS and then created a new service with my personal Gmail account. From the EmailJS website I got the user ID and the service ID. The template was also created with its own ID, and together with three ID’s I could use the integrated EmailJS features in the demo application. I added the EmailJS ID key-value pairs to the variable group in the Azure DevOps submenu Library (Figure 17, 40).
4.7 Maintenance and security

4.7.1 Secret variables

A secret variable is a standard variable that is encrypted. These variables are encrypted and stored with a 2048-bit RSA key. Secret variables can be used at global level, meaning they are available on the agent for tasks and scripts to use. (DevOps documentation 2020c.) Secret variables are saved on the server, so it provides additional layer of protection to the deployment, according to Azure DevOps documentation (2020e). Once a value has been set to be a secret and saved to the system, it cannot be viewed by other release users.

I had problems integrating the environment variables in Azure DevOps. These variables contained key-value pairs, that were used to connect with the Google Maps API and EmailJS services. After some research I found a solution that described how to use environment variables in Azure pipelines as secret variables (Bertram 2020). I created a variable group for the pipeline which stored key values as one group. These values included API service keys that were used to connect the demo application’s components to the external API services like the Google Maps API and EmailJS. The variable group was created in Pipelines submenu called Library. A variable group can also be assigned to a specific stage in the release pipeline. The values in the group could be hidden from the view by the administrator. Other option to store secrets in Azure would have been using the Key Vault which is a paid plan service.

I linked the group in the release pipeline, but I had small problems with it. When I manually added pipeline variables in the release, they were used by the demo application. When I used the created variable group in the release pipeline, it was not used. More testing of variable groups is needed in the future. The variable group was shared across all pipelines after its use has been authorized (Figure 17, 40), but I may have missed some additional steps in the configuration.
40

Figure 17: The variable group settings. This service enabled a secure way to store API keys in the pipelines editor. By clicking a padlock icon, I could change the variable type.

4.7.2 Kanban and Azure Boards

One of the most common problems in software development is inadequate communication amongst developers. Because this project uses the association volunteers and students as developers, valuable information about the development might get lost in the process when the development responsibilities are handed from old to new collaborators. For new collaborators, it is essential to know what work objectives has been planned, what objectives are work in progress (WIP) and what objectives are finished. Kanban board is often used with other Agile methods since it provides a relatively simple way to visualize software development workflow and WIP limits. Kanban is a Japanese word literally meaning “board” and it is based on six core practices: visualize, limit WIP, manage flow, make process policies explicit, implement feedback loops and improve collaboratively (Girvan & Paul 2017, 71).

Azure Boards is a microservice of Azure DevOps which creates a Kanban Board for the project. According to Azure documentation (2018), Boards are automatically created when users create a project or add a team to Azure DevOps. The basic idea of Kanban board usually includes a set of specific columns: in this case “To Do”, “Doing” and “Done”. As work progresses, you add a work item on the board or update an existing one by moving it from left to right. Juvonen (2018, 24) states that Kanban boards are particularly used in the maintenance phase of software. Team members can track the overall progress and make
changes to the board accordingly. The Board tasks can be cross-referenced in the build, and it is possible to automatically update the progress (for example, WIP status) on the Board.

Figure 18: A collection of requirements visualized using the Azure Board. I added example tags “Prototyped” and “new idea” to certain work items to describe the level of implementation.

I planned to use Azure Board for visualizing the feature requests from the association and the previous development team (Figure 18). The purpose for this was to make the current progress as clear as possible to any new collaborators and stakeholders. For testing purposes, I added a collection of the development requirements to the board and assigned them in their respective columns. It was possible to assign certain work items between team members and create iterations. I could have added work items to a backlog or include them in the created sprint “Sprint 1”. The term sprint is used with Agile-related SCRUM methodology.

Azure Boards seemed functional and a good option to track a team member’s individual work and shared work items. It is not guaranteed if new developers want to use it. There are other
Kanban boards or similar tools available online to track work progress. Other option for developers is to use a physical Kanban board.

5 Future development

Multi-stage pipeline was not implemented for this thesis, because the project needed only the development stage and I did not have permissions to create new stages in DevOps (appendix 5). I tested adding more stages to the system, but I did not have access to use Laurea UAS’ subscription in the release pipeline configuration phase. The project could benefit from using at least two different deployment environments: one for a stable version of the application and one for the development team. One deployment could be assigned for the association, so that they could see how the data transfer in the WordPress API affects the application. The development team could use another one to test new features and deploy improved releases. Azure App Service can deploy applications to different domains with deployment slots, and these are included in the Standard, Premium, or Isolated service plan tiers (Azure 2020d).

CI/CD pipeline and DevOps services could be improved with so called container technology. This concept was not studied enough to provide a realistic approach for this thesis project, but according to Preibisch (2018, 153 –154) it could help with DevOps services. Container technology makes it possible for developers to package all the parts they need in for deployment: the software application, its dependencies, libraries, and configuration files (Rubens 2017). Each container serves one purpose, but the technology makes it possible to build an application with multiple containers.

When the additional development was in progress, I discussed with the project team and Lindfors about the next project tasks. In the discussion the stakeholders decided to look for a new software architecture for the application. New technical solutions were introduced to the association, and they agreed to change the development focus for more streamlined solution that could be managed in one platform. WordPress was one of the suggestions and it is possible it might be used to create the next web application version. This decision meant that the development for the demo application was put on a tentative hold. The milestones of the demo application development can be seen in Figure 19 (43).
6 The demo application development timeline

Figure 19: The demo application development milestones

7 Conclusion

The thesis project managed to achieve the goals that were initially set in the project meetings. The demo application was integrated to Azure and it was deployed to a new domain. The Azure DevOps was used to automate the deployment and the service provided additional tools for future development work. The source code in Azure Repos can be later accessed by other collaborators, and they can download it to their preferred development environment. The Azure platform and Azure DevOps provided sufficient tools for version control, deployment, testing, monitoring, and automation. The operations in Azure DevOps might be useful with future development project, but the benefits and risks need to be examined carefully within the Laurea UAS organization (Table 3, 44).

<table>
<thead>
<tr>
<th>BENEFITS</th>
<th>Predictable and repeatable operations</th>
<th>Better response to changes</th>
<th>Team and client collaboration</th>
<th>Focus on development</th>
</tr>
</thead>
<tbody>
<tr>
<td>LbD + Azure services + Agile</td>
<td>Continuous improvement</td>
<td>Automated, agile deployment</td>
<td>Flexibility with tasks</td>
<td></td>
</tr>
</tbody>
</table>
Client budget. Possible costs in Azure

Knowledge risk → Need for expertise

Increased complexity

Resource planning in LbD projects

Scope creep

Security and operations in Azure are depended on administrators

Resistance when adopting new systems and ideas

Details and plans perhaps lost with agile

Table 3: The implementation benefits and risks analyzed from the author’s perspective.

From the association’s point of view, the tangible results were small but needed. By hosting and deploying the application, the association members could start the data transfer project in WordPress and create a more prominent web application. The association members will need guidance to use the WordPress API in the future, but the instruction material will help to understand the platform. Additional guidance from Laurea UAS might also be needed. The Azure platform was not designed to be used by the association members, but they can still provide valuable information to developers by giving them feedback about the performance of the application or by giving more details to the project requirements.

7.1 Project evaluation

The project did not achieve the set goals immediately: I had to research multiple factors beforehand and try different settings several times before I could see the application site working on a browser. I relied heavily on the documentation, video tutorials and articles about the subject matter, because this project was practically made in isolation. The official documentation made by Microsoft was helpful, but I needed additional sources of information to support my understanding the subject and to examine the specific technologies that were used to create the demo application. I needed time to internalize different concepts and to practice using various new technologies. Combining Headless CMS technology to Azure Web App Service was challenging, because I did not find precise guides or documentation of how to do it properly with Azure. The system implementation should be reviewed — and improved upon if needed — to avoid future issues and to increase the performance and security of the application. Later I learned better ways to solve certain issues that would have helped in the implementation phase.

I had occasional virtual meetings with the client, but the communication and cooperation with the client were sparse. This project would have benefitted from more frequent face to
face meetings and communication. I had occasional problems with the system security because I had limited access to Azure services and these issues had to be cleared with the client. Collaborating with the client more and with a person with expertise in Azure could have helped to ease my learning process. Expert knowledge of Azure was needed to fully understand the platform tools and opportunities. When I was experimenting with Azure DevOps services, I had limited knowledge of the system which led to unnecessary trials. The requirements of using a Web App Service were also partly misunderstood, because the implementation was started early in comparison to my knowledge and skill level at the time. Azure Web App toolset includes a monitoring and debugging tool named KUDU that could have helped me to locate some of the deployment issues I had earlier. KUDU console was briefly visited, but it was underutilized in this project.

The association’s development project will be continued when new Laurea student developers are found. A more detailed requirement analysis together with a client could have perhaps prevented making an inadequate application version. That could have deterred the misuse of workforce resources. The new solution must be able to be later converted to a mobile application if the project will still follow the initial requirements made by the association. A review of the technical requirements might reveal more details about what functions and features are desired from the application and if they can be implemented to the future technical solutions. Lindfors later mentioned in a phone interview that they were interested to use the application in cooperation with another association called Vihdin Matkailu ry. This tourism association operates on a local level in Vihti and they were interested to use the association’s content in their projects.

I am satisfied about the work that was produced. The obstacles I faced were expected because I knew it will take time to achieve even surface-level knowledge about the technologies used, and the concepts that were related to Azure and its microservices. The practical approach slowly helped me understand how these tools and ideas would work together as one cohesive system. The project overall provided a meaningful learning experience. Modern concepts like DevOps, Agile, Azure, and headless CMS are important to know for professional growth. The obtained knowledge will carry over towards of becoming an IT professional.

7.2 Acknowledgement

I want to acknowledge my appreciation to the association members for starting this project and for collaborating with students. I want to thank Laurea UAS supervisors for the support and assistance. I hope that the association continues the collaboration with Laurea students, so they get a change as well to improve their skills as developers and eventually create a product that will preserve the unique, historical data of Vihti area and its surroundings.
References

Printed


Electronic


Unpublished


Lindfors, H. & Salenius, A. Chair of the Association and Association Secretary. Vihdin Nummelan Kylähistoria Ry. 18 May 2020. Virtual meeting.
Figures

Figure 1: The continuous DevOps loop. This kind of loop is used to visualize the DevOps process in software development projects. ............................................................. 19
Figure 2: Automated practices of software delivery. ......................................................... 20
Figure 3: SDLC theory and how it was followed in practice. ............................................. 24
Figure 4: The designed lifecycle of the demo application development project. Developers could be more efficient if they adopt the agile mindset and aim for continuous cycle of improvement.................................................................................................. 27
Figure 5: The project timeline shows the selected Azure tools and technologies. Options marked with grey font are well-known alternative options. This figure is a modified version of the original illustration made by Jackson (2019). ...................................................... 28
Figure 6: App Service ”NummelaVhtiAppv1” dashboard. App Service configurations were made in Azure Portal. .............................................................................................. 28
Figure 7: After installing NodeJS to my PC, I installed necessary packages by using the VS Code terminal. Text lines marked with blue are commands. ........................................ 29
Figure 8: Code from HomePage.js. ....................................................................... 30
Figure 9: A prompt to import a Git repository. This image is an example from Azure DevOps Documentation (2020a). .......................................................... 32
Figure 10: The designed DevOps loop for the project. The project would start with a new plan and go through the cycle continuously while the demo application development is ongoing. 32
Figure 11: The DevOps project UI. The colored icons under the plus icon on top left are Overview, Boards, Repos, Pipelines, Test Plans and Artifacts. ................................. 34
Figure 12: The process of how to define a pipeline with the Classic user interface. The image is from Azure DevOps documentation (2019a). .................................................. 34
Figure 13: The build pipeline, named as “NummelavihtiApp-CI”. .................................. 35
Figure 14: The YAML file that includes the tasks of the build pipeline. ......................... 35
Figure 15: A build pipeline run status. The artifact is published, and it is consumed by a release pipeline. Any commits I created in the master branch triggered the pipeline. ...... 36
Figure 16: Release pipeline ”NummelavihtiApp-CD” task settings. .................................... 37
Figure 17: The variable group settings. This service enabled a secure way to store API keys in the pipelines editor. By clicking a padlock icon, I could change the variable type. . 40
Figure 18: A collection of requirements visualized using the Azure Board. I added example tags “Prototyped” and “new idea” to certain work items to describe the level of implementation. ............................................................................................. 41
Figure 19: The demo application development milestones ........................................... 43
Tables

Table 1: Key concepts of Azure pipelines (2020b). .................................................... 21
Table 2: A table of the SDLC feasibility studies ........................................................ 23
Table 3: The implementation benefits and risks analyzed from the author's perspective...... 44
Appendices

Appendix 1: The old website of Nummelan kylähistoria .................................................. 54
Appendix 2: The original 12 principles of Agile Manifesto .............................................. 55
Appendix 3: The system architecture of the demo application as described by the original
development team (Pyykölä & Petróczki 2020, 30) .................................................... 56
Appendix 4: The desktop and mobile views of the demo application ............................. 57
Appendix 5: Error message in the release pipeline. The idea was to add a production stage
and a test stage to the pipeline ..................................................................................... 58
Appendix 1: The old website of Nummelan kylähistoria

NUMMELAN KYLÄHISTORIA

Paikallishistoriaa, kansalaismuistitietoa ja lähes 20 vuotta kestänyttä ja edelleen jatkuvaa tietojen keruuta Nummelan kylän elämästä.


Taloja ja elämää Nummelassa


Appendix 2: The original 12 principles of Agile Manifesto.

<table>
<thead>
<tr>
<th>12 principles mentioned in the Agile Manifesto:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.</td>
</tr>
<tr>
<td>2. Welcome changing requirements even late in development. Agile processes harness change for the customer’s competitive advantage.</td>
</tr>
<tr>
<td>3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.</td>
</tr>
<tr>
<td>4. Business people and developers must work together daily throughout the project.</td>
</tr>
<tr>
<td>5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.</td>
</tr>
<tr>
<td>6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.</td>
</tr>
<tr>
<td>7. Working software is the primary measure of progress.</td>
</tr>
<tr>
<td>8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.</td>
</tr>
<tr>
<td>9. Continuous attention to technical excellence and good design enhances agility.</td>
</tr>
<tr>
<td>10. Simplicity—the art of maximizing the amount of work not done—is essential.</td>
</tr>
<tr>
<td>11. The best architectures, requirements, and designs emerge from self-organizing teams.</td>
</tr>
<tr>
<td>12. At regular intervals, the team reflects on how to become more effective then tunes and adjusts its behavior accordingly.</td>
</tr>
</tbody>
</table>

https://agilemanifesto.org/principles/
Appendix 3: The system architecture of the demo application as described by the original development team (Pyykölä & Petróczki 2020, 30).
Appendix 4: The desktop and mobile views of the demo application.
Appendix 5: Error message in the release pipeline. The idea was to add a production stage and a test stage to the pipeline.