Tampere University of Applied Sciences

Exploring and Comparing Alternatives to Liferay Headless CMS

Benny Cascarino

BACHELOR'S THESIS
May 2025

Bachelor's Degree Program in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor’s Degree Program in Software Engineering

CASCARINO, BENNY:
Exploring and Comparing Alternatives to Liferay Headless CMS

Bachelor's thesis 45 pages, appendices 6 pages
May 2025

The objective of this work was to find an alternative to the Liferay Headless CMS tool. The client Twoday Oy has set the goal of identifying a different solution on the market that meets the company’s and the project’s needs.

This work involved identifying the requirements needed for a different solution by conducting interviews to find out the use-cases for the system and creating a comparison between the different solutions.

As the end result, a list of alternatives was provided for the Liferay Headless CMS tool, helping the company make an informed decision on selecting the best solution for the use case.

Key words: CMS, Software development

CONTENTS

1 INTRODUCTION 6
2 BACKGROUND 7
2.1 What is a CMS? 7
2.1.1 History of CMS 7
2.1.2 Rise of open-source platforms 8
2.1.3 How does traditional CMS work? 8
2.1.4 Benefits of a traditional CMS 9
2.1.5 Drawbacks of a traditional CMS 9
2.2 What is headless CMS? 9
2.2.1 How does headless CMS work? 10
2.2.2 Benefits of a headless CMS 10
2.2.3 Drawbacks of a headless CMS 11
3 TWODAY'S USE CASE 12
3.1 The need for a different CMS solution 12
3.2 Requirements 12
3.2.1 Role-Based-Access-Control 12
3.2.2 Multi-tenant architecture 12
3.2.3 Customizable content editing interfaces 13
3.2.4 OpenID Connect authentication 14
3.2.5 Multi-language support 14
4 COMPARISONS AND OVERVIEWS 15
4.1 Introduction 15
4.2 DecapCMS 16
4.3 PayloadCMS 18
4.3.1 Setting up Keycloak with Azure Entra ID for proof of concept 19
4.3.2 Setting up Payload 21
4.4 Gentics Mesh 26
4.5 JHipster 30
4.5.1 JHipster setup 30
4.6 Comparisons 35
5 CONCLUSION 36
6 DISCUSSION 37
REFERENCES 38
APPENDICES 40
Appendix 1. Twoday interview 40

Appendix 2. DecapCMS config.yml 41
Appendix 3. PayloadCMS keycloak.ts 42
Appendix 4. payload.config.ts 43
Appendix 5. NewsArticle.ts in Payload..... 44
Appendix 6. blog.jdl..... 45

GLOSSARY

CMS	Content Management System
API	Application Programming Interface
DOM	Document Object Model
RBAC	Role-Based-Access-Control
OIDC	OpenID Connect
SSO	Single-Sign-On

1 INTRODUCTION

The purpose of the following work is to evaluate alternatives to Liferay headless CMS tool. The company Twoday Oy has set a goal to find a new headless CMS that is more suitable for special use cases where Liferay headless CMS does not fit in. This thesis is intended for technical decision makers, software architects, and developers seeking guidance in choosing a headless CMS solution in an enterprise environment.

This work includes a comprehensive introduction to content management systems and their functionality, comparisons between different headless CMS solutions and the provided features each CMS includes that target the requirements set by Twoday Oy.

The research methodology includes technical evaluations, proof-of-concept implementations, and feature examinations. Each CMS is tested for authentication integration with Keycloak and ease of deployment in real-world scenarios.

Twoday Oy Is a Nordic technology company that specializes in software engineering, Data & AI, Digital experiences, and business applications for both private and public sectors (LinkedIn. N.d.).

2 BACKGROUND

2.1 What is a CMS?

In our digital world, more and more businesses need to create an online presence with a website to reach consumers to sell their products or advertise their services. But not everyone has the technical knowledge to create and manage websites and content on the internet. This is where CMS (Content Management System) can help.

CMS (Content Management System) is a system that helps users create and manage content on a website without the need for technical knowledge. In practice, a CMS offers a user-friendly interface where a user can create a website without writing a single line of code.

2.1.1 History of CMS

In the early days of the internet, before CMS, websites were created as simple static HTML web pages without the need for a backend that would handle requests like the creation of a user. In 1997, dynamic content was introduced with the Document Object Model (DOM). The DOM is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page (MDN, N.d.). Dynamic HTML was revolutionary, letting developers request and receive data to update a web page without reloading the page (Contentstack 2025). As the web evolved from static sites to more interactive sites with dynamic content, the need for more interactive content grew, and this meant more managing and updating the website. This is the point where CMSs come in and create a user-friendly environment for non-technical people to allow adding new content to websites, such as job postings or promoting a new product or service.

2.1.2 Rise of open-source platforms

In the early 2000s, because of high-cost enterprise CMSs, open-source CMS solutions started to rise such as WordPress, Drupal and Joomla. These open-source solutions offered both paid and free application development. This meant that even small businesses could create their own platform free of charge, and CMSs would become mainstream.

At the start WordPress wasn't technically a CMS, WordPress became popular as an open-source solution focusing on blog content delivery (Contentstack, 2025). But later in 2006-2010 WordPress evolved into a full-fledged content management system (wpbeginner, 2025).

2.1.3 How does traditional CMS work?

A traditional or a "monolithic" CMS ties the backend where the content is managed and the frontend that displays the content together. This means that if a business wants to display their content on another platform like a smart watch, rewriting and redesigning the application would be necessary.

PICTURE 1. Monolithic CMS (Source: What is a CMS, Contentstack)

2.1.4 Benefits of a traditional CMS

Traditional CMSs are the perfect solution for small businesses that only need a simple website to display and offer their content. Most CMSs have built-in templates that help design the webpage. Additionally, it is easy to maintain and update since everything is in one place.

2.1.5 Drawbacks of a traditional CMS

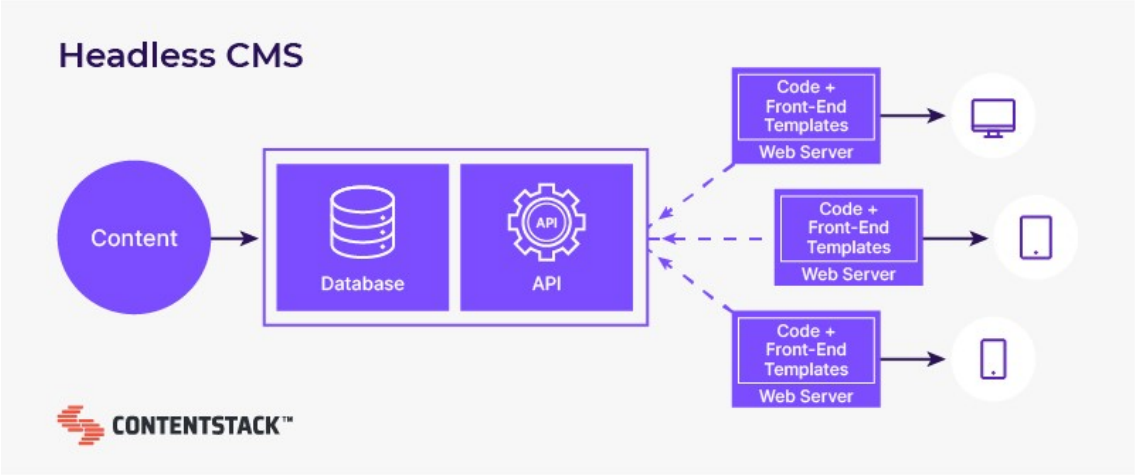
The downside to a traditional CMS is that it is limited to only simple webpages that can be accessed through the internet browser, which is perfect for small businesses that are not looking to invest in omnichannel. Omnichannel in this context means businesses that are providing content on multiple platforms like smart watches, virtual reality, and mobile apps. Some organizations that are stuck with monolithic CMSs have had to come up with custom code, plug-ins, and a lot of workarounds (Contentstack, 2020).

These workaround organizations relying on keeping up with current trends create security concerns, that could create errors anytime they add new features or update existing ones. This means businesses that use traditional CMSs are always one step behind the newest technological trends.

2.2 What is headless CMS?

Headless CMS fixes all the problems presented with the traditional CMS, removing the ties from the backend to the front end making them separate from each other. This gives businesses freedom and flexibility to deploy content across any platform.

2.2.1 How does headless CMS work?



PICTURE 2. Headless CMS (Source: What is a CMS, Contentstack)

As discussed before, the headless CMS means that the front-end and back-end are not tied to each other, and for the front-end to get the information from the backend it needs an application programming interface (API). API is a software intermediary that allows two applications to talk to each other (Mulesoft, N.d.). This API allows developers to create different front ends that display the content from the backend to a custom front-end that could be built to mobile devices or any other device, since the back end stays the same all the developer needs to do is create a new front-end for each device, reducing the development time and cost.

2.2.2 Benefits of a headless CMS

Headless CMS platforms support omnichannel that the traditional CMS does not. And this means more opportunities for businesses to deliver content to different platforms. And since the front-end and backend are not tied together, businesses can build any interface with any technology suitable for them, because all the content is delivered from the API, so as long the front-end can send requests to the API, it will deliver content. Headless CMS is also a futureproof option, it can be easily adapted to any new technology and is always ready for newer updated front ends.

2.2.3 Drawbacks of a headless CMS

Headless CMS has a few disadvantages. A headless CMS solution requires a software developer to manually code a front end that communicates with the CMSs API and display content. Making it not a good option for non-technical consumers.

While headless CMS is flexible, it can also have some performance challenges, since there are many different interfaces created to interact with the CMSs API, that means there a lot of traffic going into the CMS, and that could affect load times and the responsiveness of the interface. So, organizations must consider this and optimize their performance using content delivery networks for example.

3 TWODAY'S USE CASE

3.1 The need for a different CMS solution

Currently, Twoday is using a headless CMS solution provided by Liferay. Liferay headless CMS provides comprehensive features that satisfy many enterprise companies. Even though Liferay's solution is respected and still used in today's projects, the purpose of this work is to find another headless CMS solution that is lightweight and more customizable than Liferay. The new headless CMS solution would be used in special use-cases, where Liferay's headless CMS is too much or too heavy. To find suitable CMS solutions, interview was held with Twoday, where use-cases and requirements were defined. During this interview, it came clear that DecapCMS is one option for the solution. But the drawback of DecapCMS is that it is not Java based. The full interview transcript can be found at Appendix 1.

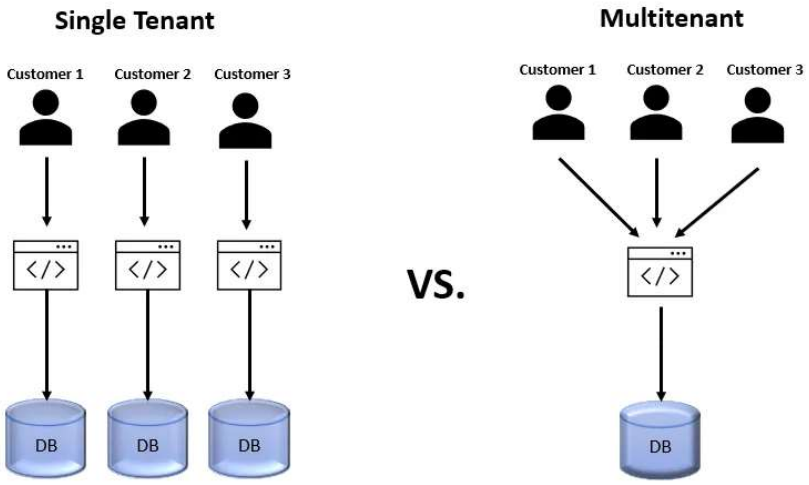
3.2 Requirements

3.2.1 Role-Based-Access-Control

Role-Based-Access-Control (RBAC) is a simple concept involving roles for each user in a software, for example admin, viewer and editor. This requirement in this works context means the maintainer of the system can add roles for each employee that would use the CMS. The environment is more controlled when users have specific roles with certain permissions, and this prevents the situation where the non-technical person uses the CMS and does something that would either break the system or do some damage that could mean calling the maintainer to fix the issue.

3.2.2 Multi-tenant architecture

Multitenant software architecture is when one instance of a software can be shared among many individuals or groups of people. Each tenant's data is isolated from, and invisible to, the other tenants sharing the application instance, ensuring data security and privacy for all tenants (IBM, n.d.).



PICTURE 3. Multi-tenancy architecture (Source: multitenancy for dummies, Medium)

Multitenant software architecture is mostly seen on SaaS (Software as a Service) applications, because when the software has many users, it is more efficient to use multitenant architecture than single tenant.

3.2.3 Customizable content editing interfaces

This feature doesn't have a specific name, each headless CMS has created an own special name for this feature or have not named it at all. Twoday wishes that they could fully customize the interface to meet the customer and use-case needs. And by customizing it makes the usability and ease of use a priority for the customers who do not have a lot of technical knowledge. For example, if the use-case was a news site, the interface could include text fields for a header, content etc., an upload field for images and a submit button. And the person who would like to add the content would only need to fill the required fields and press submit, making it very easy to add content. This prevents the possible scenario of the content creator breaking the system when adding content.

3.2.4 OpenID Connect authentication

OpenID is listed as one of the requirements for the headless CMS, OpenID Connect (OIDC) is an identity authentication protocol that is an extension of open authentication (OAuth) 2.0 (Microsoft n.d.). OpenID enables the possibility to login using Microsoft Entra ID which is a standard in organizations. Like example logging in to a system using the organizations Microsoft account.

3.2.5 Multi-language support

Multi-language support in this context means that for each post, there should be another version of the post but in a different language, this could be interpreted for example in the URL when fetching the post, there could be an ‘en’ for English or ‘fi’ for Finnish post. In practice this means that on the website the end user is at, there should be a language option, so the content can be displayed in different languages.

4 COMPARISONS AND OVERVIEWS

4.1 Introduction

There are four different headless CMS solutions that have been picked for this comparison. Each has their own strengths and weaknesses, and the next phase of this work is to go through each software and compare them. There will be demonstrations of each software, so that it is a proof of concept that these are viable options. DecapCMS was a suggested solution by Twoday. Payload is mentioned in most headless CMS articles and is very popular. ScreamingBox that is a blog website released an article of the seven most popular headless CMS systems and wrote the following: ScreamingBox has chosen Payload as the headless CMS for our new website which we are currently building (ScreamingBox, 2024). Genetics Mesh was a recommendation from a Reddit post, where the author was asking about Java based headless CMSs. Genetics Mesh fit very well this works requirements, fitting almost all the requirements. Here is a link provided to the Reddit post: https://www.reddit.com/r/java/comments/brfcwz/experiences_with_javabased_headless_content/. JHipster was a recommendation by OpenAI ChatGPT, given the prompt “What is an open-source Java based headless CMS that could be customized to fit different use-cases. The requirements are: Keycloak, Java, Spring-boot, Multitenant, open-source, customizable”. Though, JHipster is not particularly a headless CMS it can be built to act like one, and with JHipsters technology stack being Java Spring-boot with build in support for Keycloak, it is a fair option. With requirements like multitenancy, SSO, RBAC, that usually are enterprise features, not many open-source solutions give these features for free. That’s why companies like Strapi give their SSO feature on the paid version of the platform. And Payload does the same thing, locking the enterprise features behind a paid version of Payload.

4.2 DecapCMS

DecapCMS is an open-source content management system that ties with your Git workflow. The basic idea of DecapCMS is to run beside the frontend framework, meaning that there is no need for a server that runs the CMS. With DecapCMS the content is created with the UI provided and it saves the content to a markdown file directly to the project. When new content is uploaded, it will commit the changes to Git and publish them to the website, this also integrates well with Gits version control. DecapCMS was mentioned in the interview as an option that Twoday have been keeping an eye on.

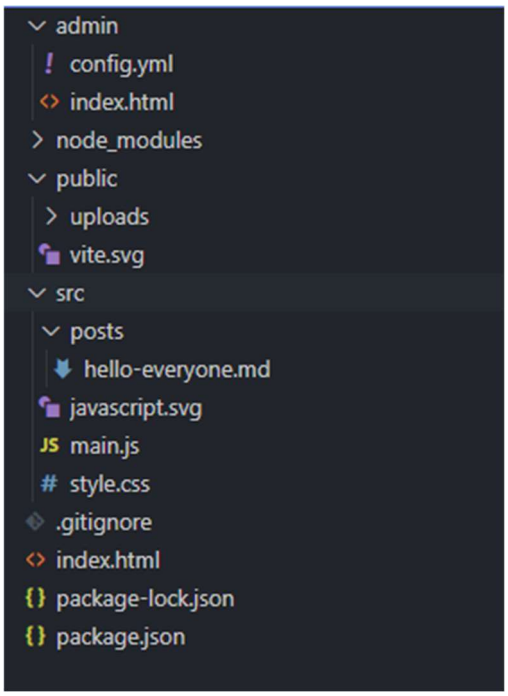
To create content, the user must create a collection. Collections are created in the configuration file. Collections define the structure for the different content types on your static site (DecapCMS N.d.). This following demonstration includes creating a collection for a blog post. This would require a title and body. Title for the blog post title and body for the blog post content.

```
collections:
  - name: "posts"
    label: "Posts"
    folder: "src/posts"
    create: true
    slug: "{{slug}}"
    fields:
      - { label: "Title", name: "title", widget: "string" }
      - { label: "Body", name: "body", widget: "markdown" }
```

FIGURE 1. Collection for posts in config.yml

After creating the collection for posts in figure (1), it should be visible in the admin panel of the website hosted. Creating new content is simple, with the intuitive user interface the button to create new content is in the main page with the example “New Posts”. The whole config.yml file can be found in Appendix 2.

After creating new content, the content is stored in a markdown file under the folder that was defined in the configuration.



PICTURE 4. File structure of a simple DecapCMS project.

In the picture (4), a blog post that was created is stored as “hello-everyone.md”. Each new post would create a new markdown file. The problem is that DecapCMS does not support multi-language for posts. If the user wants to create posts with different languages, that means adding another collection, that is the same but different language. The user must also write the different languages manually to the other posts, there is no automation built in.

DecapCMS out of the box does not support external authentication providers like Keycloak, but there are some third-party plugins that could make that a possibility. Even so, DecapCMS is MIT licensed meaning that the software is fully customizable, so building a custom system to support for example, Keycloak is feasible. This demonstration uses the default login method that is provided with the local backend for testing, that does not require any setting up.

In DecapCMS documentation, it is stated that repositories stored on Azure, the Azure backend allows CMS users to log in directly with their Azure account. Note that all users must have write access to your content repository for this to work (Decap, N.d.). This means that if the CMS is hosted in Azure, it is possible to use Azure Entra ID, which is a cloud-based identity and access management service (Microsoft, 2025). Hosting with Azure would mean that tenants could access the

CMS with organization accounts. But still, this does not allow the use of third-party authentication providers.

The multitenancy requirement is irrelevant with DecapCMS, because DecapCMS is tied with the frontend of the application, there is no server needed. So, each client would have their own instance of DecapCMS tied to their application.

Because DecapCMS is serverless, and serves static content from markdown files, it means that it does not have an API, so there are no URL endpoints for content, so the content can only be displayed in that frontend it is tied with. This could be a problem if the content must be displayed in two or more different environments, for example if there was a native mobile app that needs to also display the content, the application cannot get that content if it is not running beside that DecapCMS instance.

DecapCMS is the most lightweight out of the three different options, and the easiest to set up. A great option for smaller projects, and particular use cases.

4.3 PayloadCMS

PayloadCMS is a very popular MIT licensed Next.js based headless CMS, that offers mostly everything that Twoday Oy has set as the requirements. When downloading the free version of PayloadCMS does not offer all the enterprise level features, the user must either create them or download plugins. PayloadCMS offers the multitenant feature as a plugin and provide documentation on how to install and use it, they also provide an example project that has the multitenant feature ready to go. What they do not offer is the OAuth plugin, that feature is only available on the paid version of PayloadCMS. But because PayloadCMS is widely used and has a big community surrounding them, other users have developed their own OAuth plugins that anyone can download and use, these plugins are also open source.

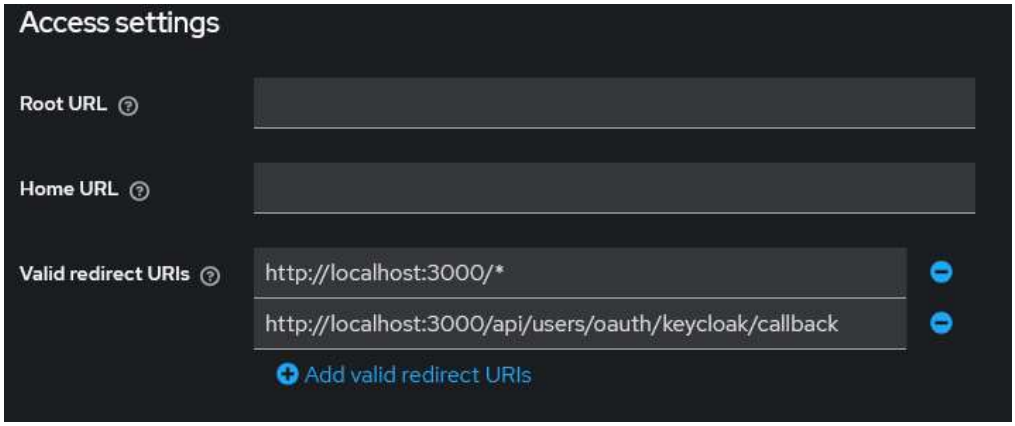
4.3.1 Setting up Keycloak with Azure Entra ID for proof of concept

This demonstration for PayloadCMS includes a very simple setup of Keycloak with Microsoft identity provider for authentication. Keycloak is a powerful open-source identity and access management platform (Mi Do, 2024). The purpose of this demonstration is to show the possibilities and for proof of concept.

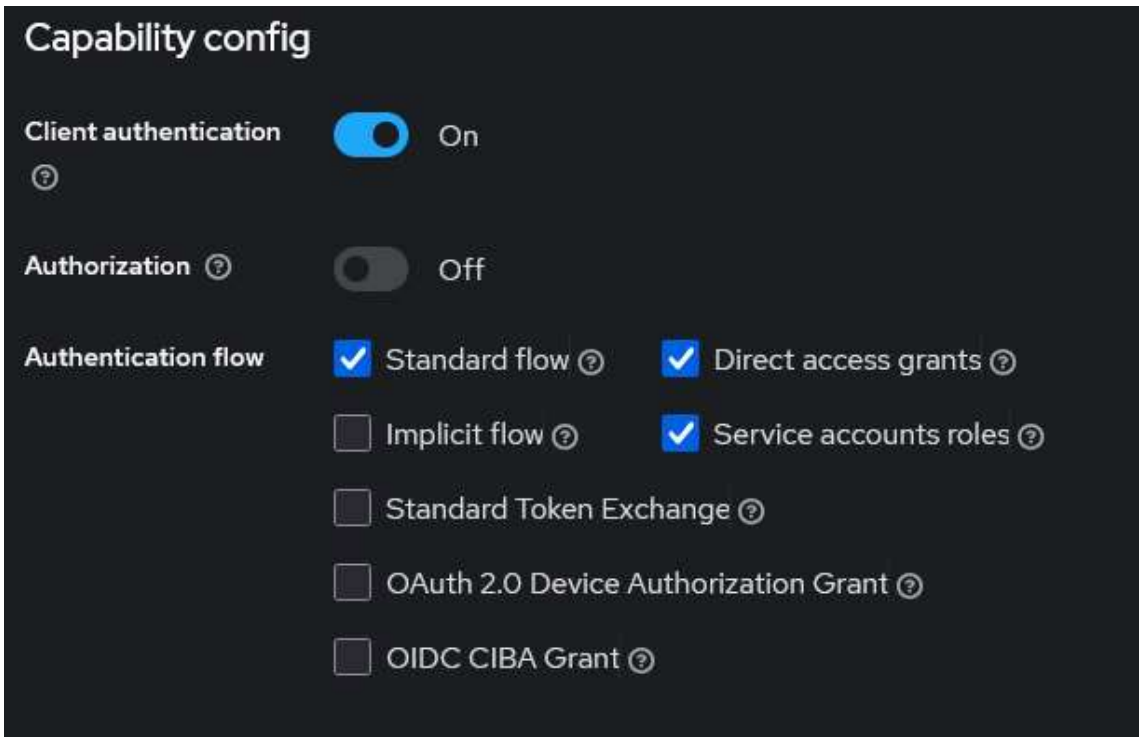
```
Benny@DESKTOP-UL6UE5I MINGW64 /
$ docker run -p 8080:8080 -e KC_BOOTSTRAP_ADMIN_USERNAME=
admin -e KC_BOOTSTRAP_ADMIN_PASSWORD=admin quay.io/keyclo
ak/keycloak:26.2.0 start-dev
```

PICTURE 5. Docker run Keycloak image.

After opening Keycloak's administrator panel, a Realm needs to be created. According to Mi Do (2024), a Realm in Keycloak represents an isolated space, where users, applications, roles, and groups are managed. For this demonstration, a Realm called "payload" will be created. After creating a new Realm, a Client needs to be created. Clients in Keycloak represent applications and services that can use Keycloak for user authentication and authorization (Mi Do, 2024). Clients can use different protocols such as OpenID Connect (OIDC). Configuring a Client for the Payload application for simple demonstration purposes is simple, since the only thing that needs careful attention is the "Valid redirect URIs" that are responsible for the redirection after a successful login.

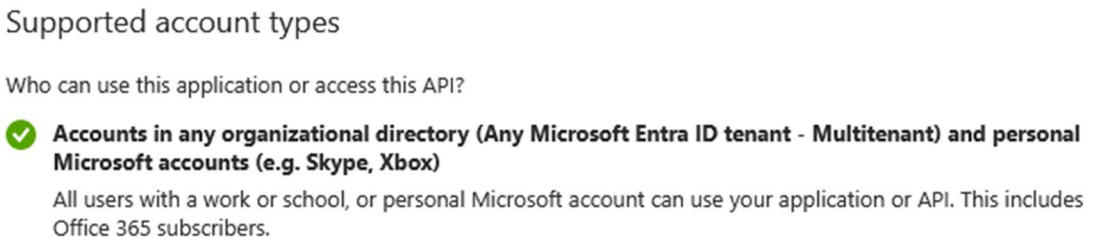


PICTURE 6. Keycloak valid redirect URIs.



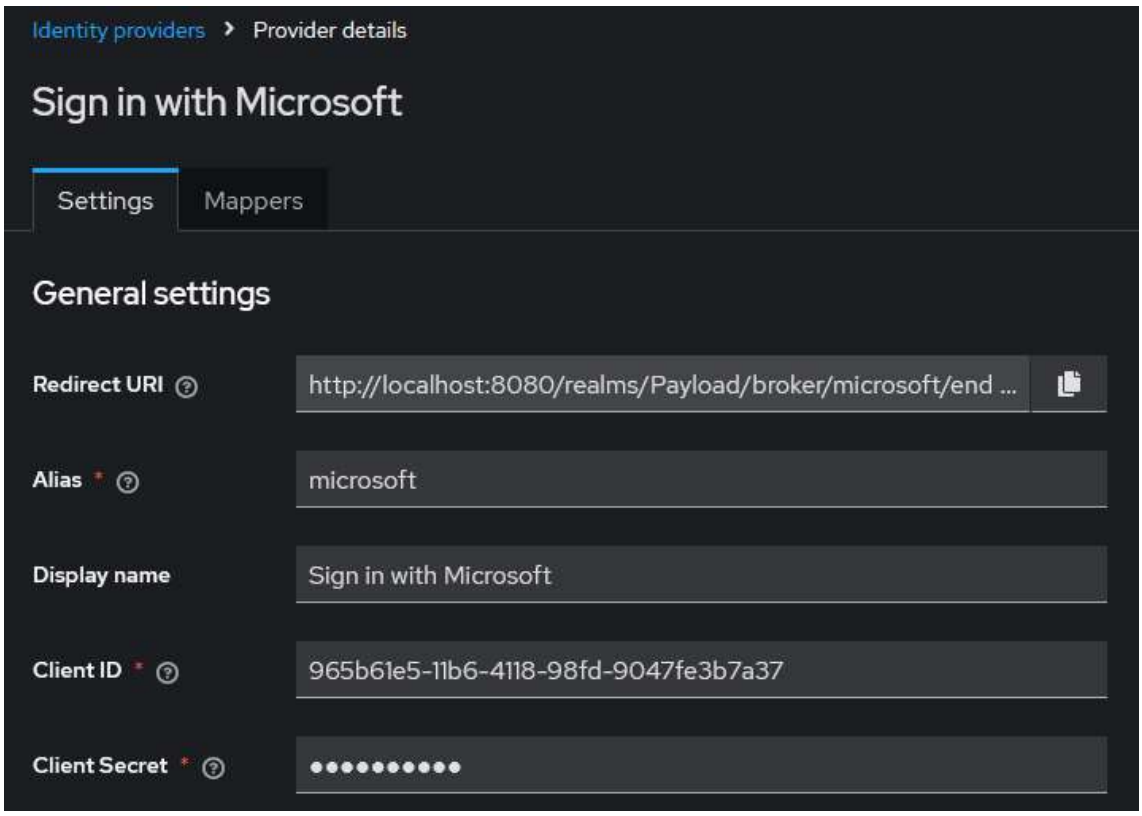
PICTURE 7. Capability config in Clients.

With the Clients configurations complete, the next step is to register an application in Azure in “App registration”. When registering the application, the important step is to add the correct redirect URI, that is in this case “http://localhost:8080/realms/payload/broker/microsoft/endpoint”. For the supported account types choose the accounts in any organization and personal Microsoft accounts as displayed in picture (8).



PICTURE 8. Azure app registration supported account types.

Finally, in Keycloak admin panel navigate to identity providers and choose Microsoft, and as in picture (9), fill in the following fields with the correct details. The “Client ID and “Client Secret” is found in Azure in the app registration done before.



PICTURE 9. Microsoft identity provider.

What is visible in picture (9) enables the possibility to login using Microsoft account, the basic idea is to login using organization accounts.

4.3.2 Setting up Payload

Payload was setup using the “multi-tenant” example, created with the command “npx create-payload-app –example multi-tenant”.

Connecting Payload with Keycloak that was setup earlier, needs an OAuth plugin. The plugin used in this demonstration is an open-source plugin created by the community named payload-oauth2. Next step is to create a file that holds the Keycloak setup through the payload-oauth2 plugin. The full code for this can be found in Appendix 3. After this the “keycloakOAuth” needs to be connected to the “payload.config.ts” file found in the root of the project.

```
plugins: [
   keycloakOAuth,
   multiTenantPlugin<Config>({
     collections: {
     },
```

```
      tenantField: {
```

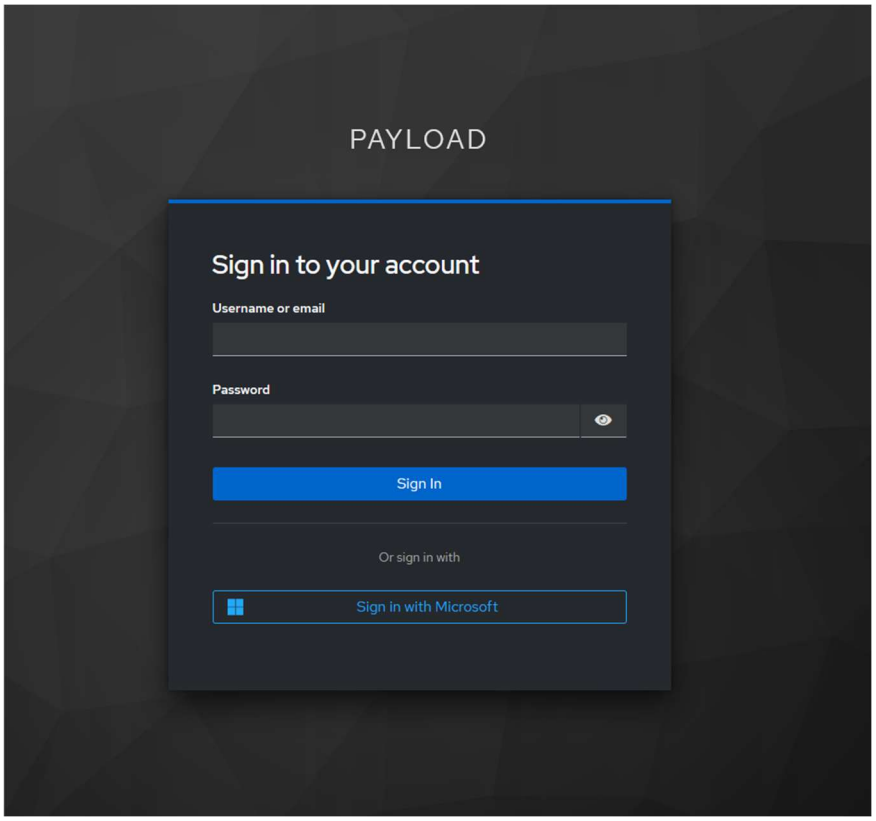
FIGURE 2. Added keycloakOAuth to plugins.

The full “payload.config.ts” file is available in Appendix 4. After this, a “.env” file is needed in the project root directory, that holds all the variables containing the details of the Keycloak Realm, Client, and other.

```
DATABASE_URI=<yourmongouri>
PAYLOAD_SECRET=PAYLOAD_MULTI_TENANT_EXAMPLE_SECRET_KEY
PAYLOAD_PUBLIC_SERVER_URL=http://localhost:3000
KEYCLOAK_ISSUER_URL=http://localhost:8080/realms/Payload
KEYCLOAK_TOKEN_URL=http://localhost:8080/realms/Payload/protocol/openid-connect/token
KEYCLOAK_USERINFO_URL=http://localhost:8080/realms/Payload/protocol/openid-
connect/userinfo
KEYCLOAK_CLIENT_ID=payload-test
KEYCLOAK_CLIENT_SECRET=client-secret
KEYCLOAK_AUTH_URL=http://localhost:8080/realms/Payload/protocol/openid-connect/auth
```

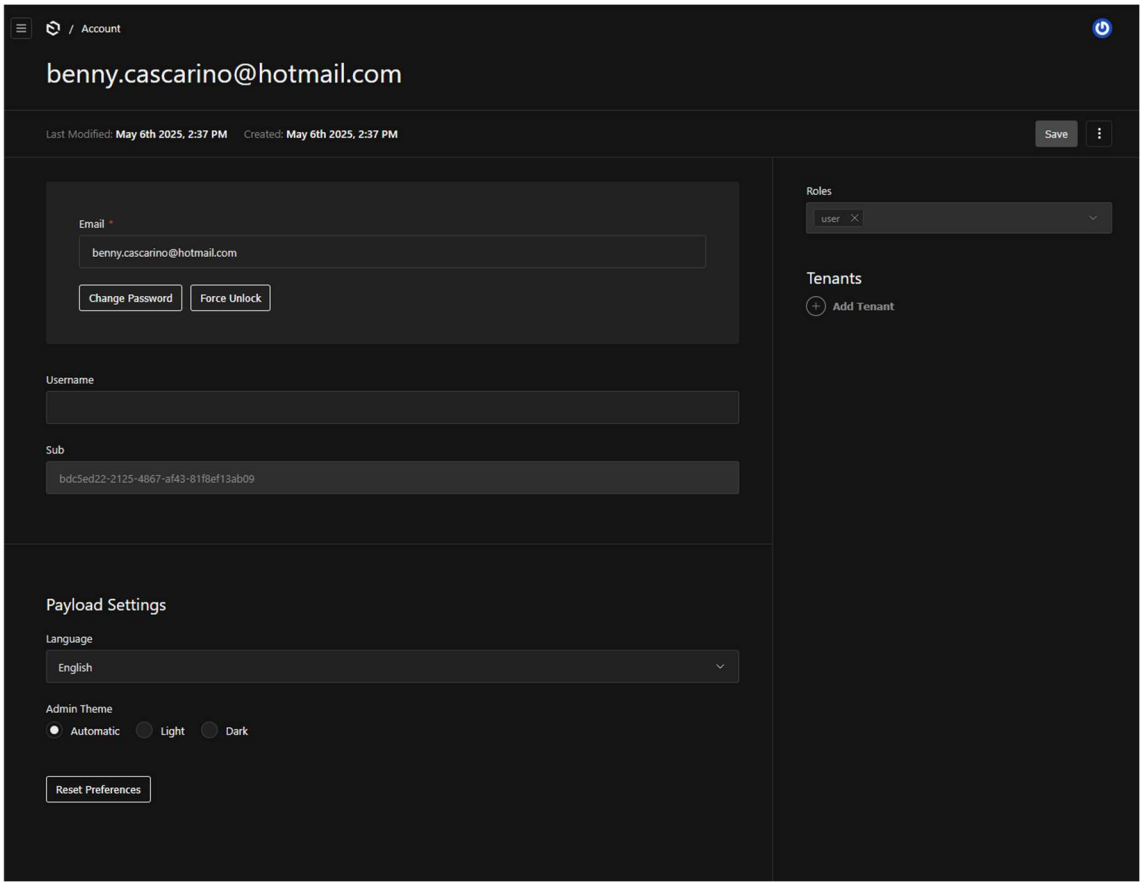
FIGURE 3. “.env” file.

After adding all the necessary code and environment variables, PayloadCMS is now accessible on “http://localhost:3000/” after running the command “npm run dev”. Navigating to the url “http://localhost:3000/api/users/oauth/keycloak” will throw the user to the Keycloak login page.



PICTURE 10. Keycloak sign in.

As seen on picture (10), the “Sign in with Microsoft” is now enabled. This helps the customers use the application seamlessly with their current organization accounts. This feature makes the application easier to use, and more importantly customers do not need to create separate accounts to use the application.



PICTURE 11. Logged in with Microsoft account.

Before adding content, each user must be assigned to a tenant. This can only be done with the super-admin profile. One way to create a super-admin is to manually change the role to “super-admin” in the database.

With the super-admin profile, create new tenants and assign users to each tenant. In Payload, tenants hold multiple users.

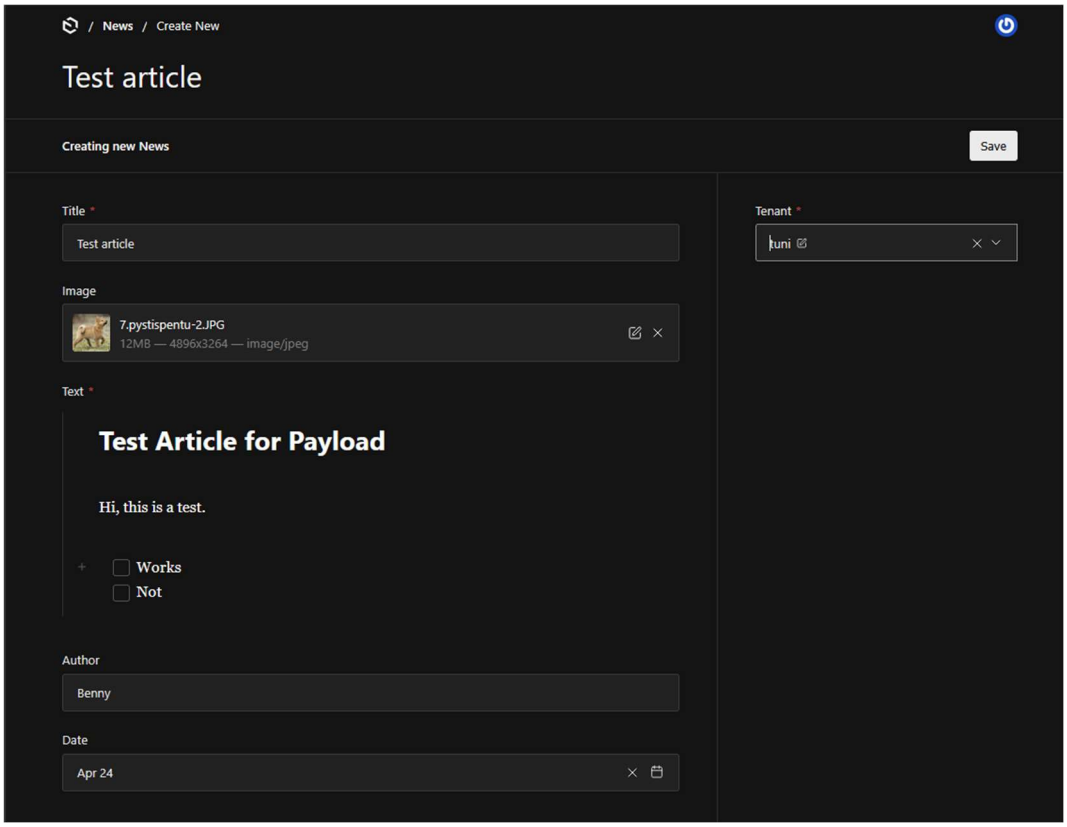
To create content with Payload, it requires “Collections”. A Collection is a group of records, called Documents, that all share a common schema (Payload Docs, N.d.). In this context, creating a collection, for example a news article requires fields such as title, image, text, author, and date. This collection is built using Payloads “CollectionConfig”.

```
import type { CollectionConfig } from 'payload'

const News: CollectionConfig = {
  slug: 'news',
  admin: {
    useAsTitle: 'title',
  },
  access: {
    read: () => true,
  },
  fields: [
    {
      name: 'title',
      type: 'text',
      required: true,
    },
```

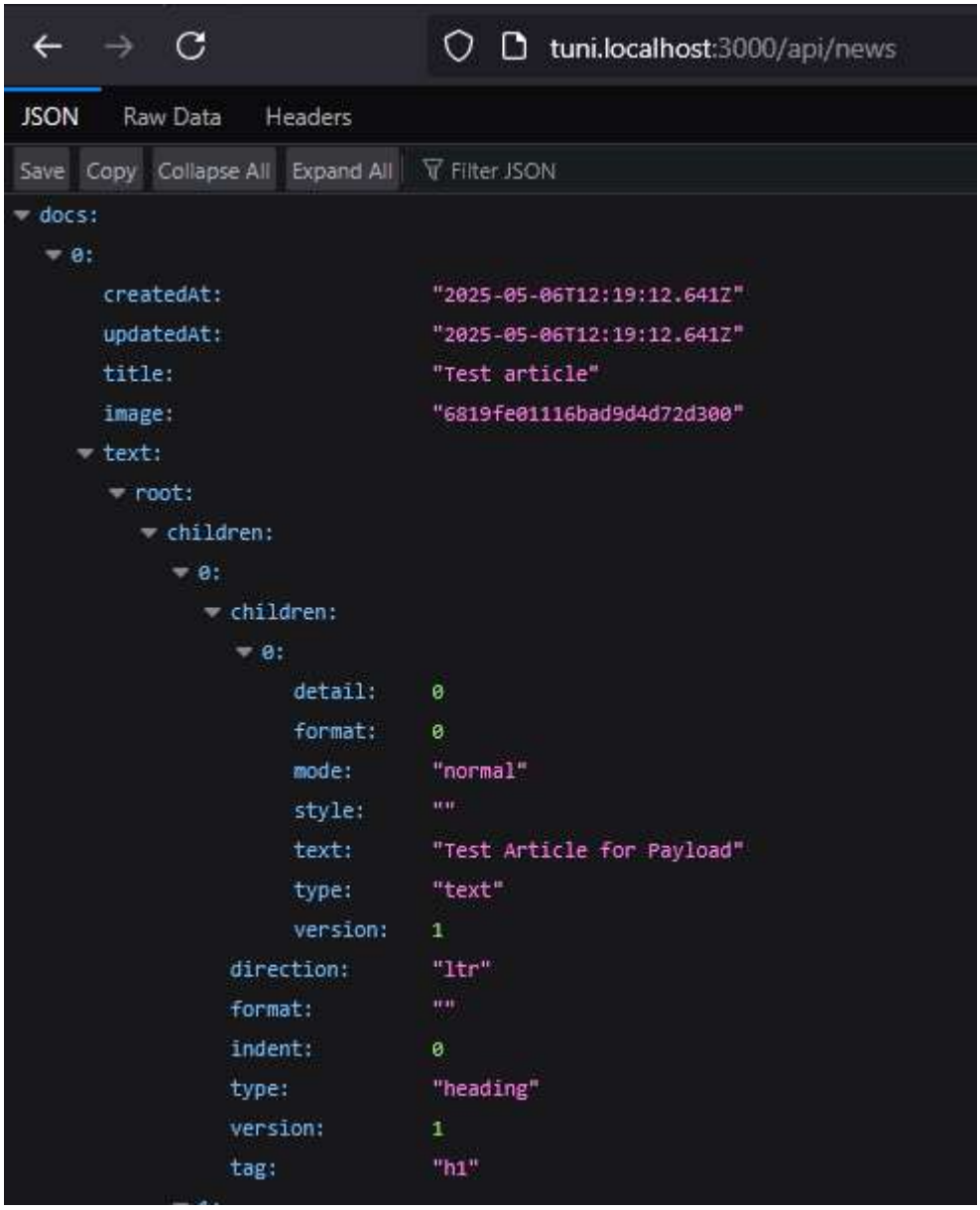
FIGURE 4. News collection.

The full code for the news collection shown in figure (4) is available in Appendix 5. Creating the news collection means that now it is visible in the user interface for every user. So, in practice, with this setup each collection is visible to everyone, which is unusual in multitenant software. This can be changed with some modifications, but in this demonstration each collection is public.



PICTURE 12. Test article.

After saving the “Test article” in picture (12), the content is only visible to the users in that tenant. After publishing the content, the content can be fetched using the API.



PICTURE 13. Tenant tuni news articles.

Each tenant has a subdomain as in picture (13) the URL to fetch “tuni” tenants’ news content is “http://tuni.localhost:3000/api/news”. This will not return any news created by other tenants.

Comparing PayloadCMS with DecapCMS, PayloadCMS needs to be deployed on a server unlike DecapCMS, and unlike DecapCMS the content published is stored in a database and not as a markdown file inside the project. The downside

to PayloadCMS is that it can take some time to configure properly, this demonstration is just proof of concept.

4.4 Gentics Mesh

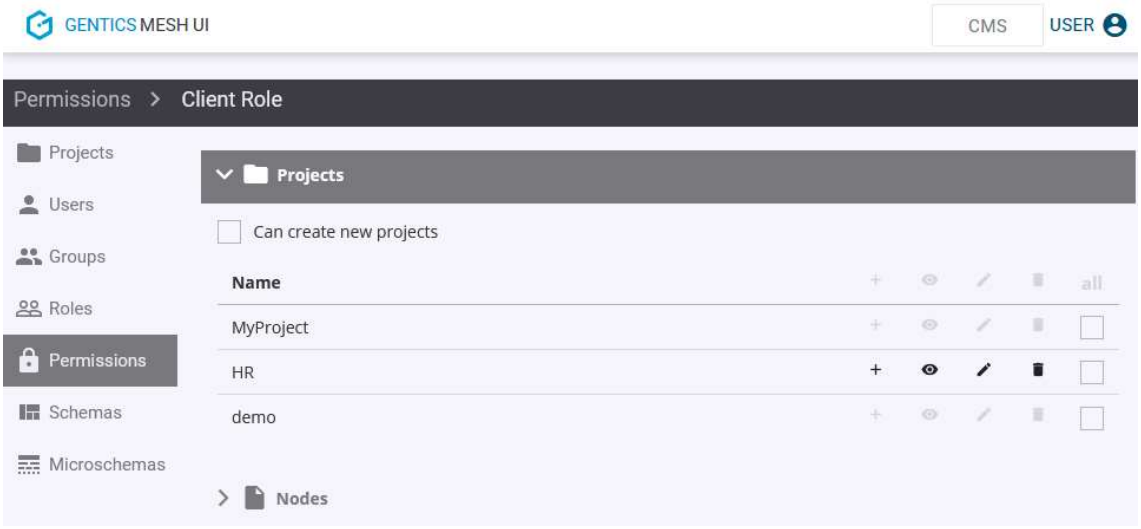
Gentics Mesh is an open-source headless CMS based on Java that offers enterprise features as well as an easy-to-use user interface. Gentics Mesh has a commercial plugin for Keycloak, so in this demonstration there will not be any Keycloak integrations.

With Gentics Mesh multitenancy is not supported but offers comprehensive Role-Based-Access-Control settings.

Role permission	Element path	Permission	Recursive
Can create new groups	/groups	Create	FALSE
Can read all groups	/groups	Read	TRUE
Can update all groups	/groups	Update	TRUE
Can delete all groups	/groups	Delete	TRUE
Can read group	/groups/:uuid	Read	FALSE
Can update group	/groups/:uuid	Update	FALSE
Can delete group	/groups/:uuid	Delete	FALSE
Can read group and its users	/groups/:uuid	Read	TRUE
Can update group and its users	/groups/:uuid	Update	TRUE
Can delete group and its users	/groups/:uuid	Delete	TRUE

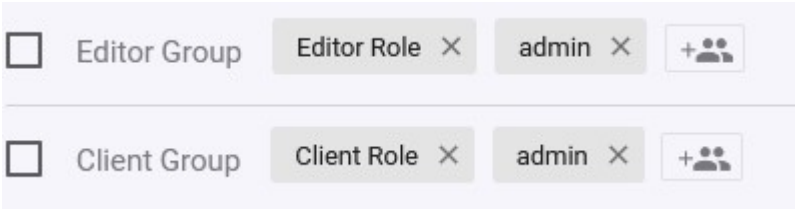
PICTURE 14. Role permissions on a specific group and multiple groups (Source: Permissions, Gentics Mesh).

As seen in picture (14), Gentics Mesh offer a wide range of different role settings that can be setup for each user.

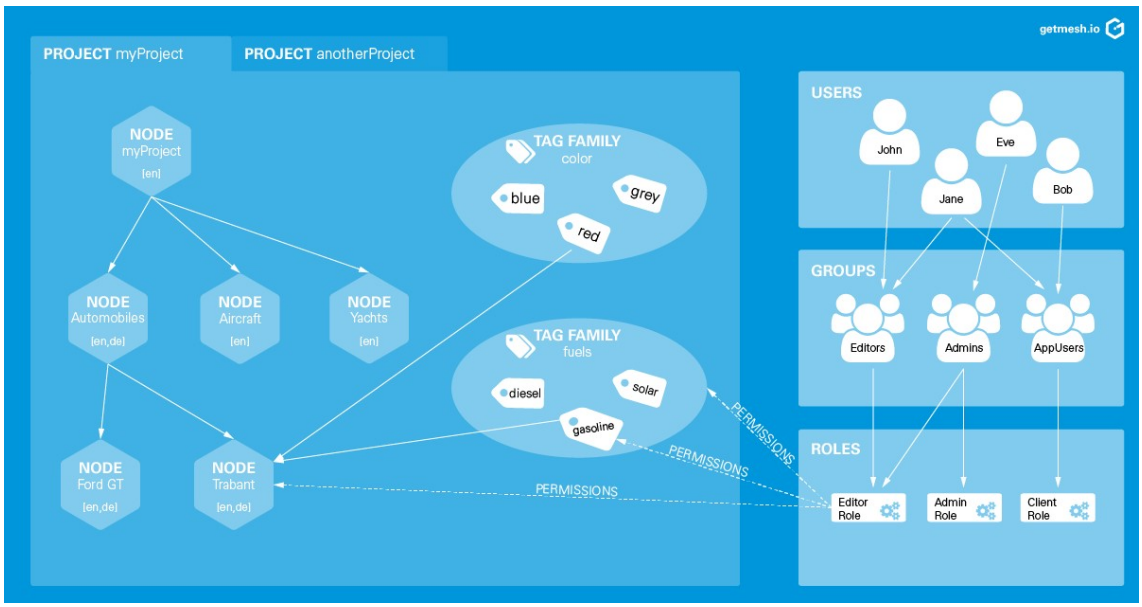


PICTURE 15. Gentics Mesh permissions for Client Role.

In picture (15) there is an example of permissions for the role “Client Role”, this role is just an example, but if this was a client, the role name would be “Client x Role”, x being the clients organization name. And so, each role is set with permissions like for an example what projects can be viewed or edited. Then these roles are added to different “Groups”. In picture (16), there is an “Editor Group” and a “Client Group”. For these groups they are assigned for the “Editor Role” and “Client Role” with specific permissions.



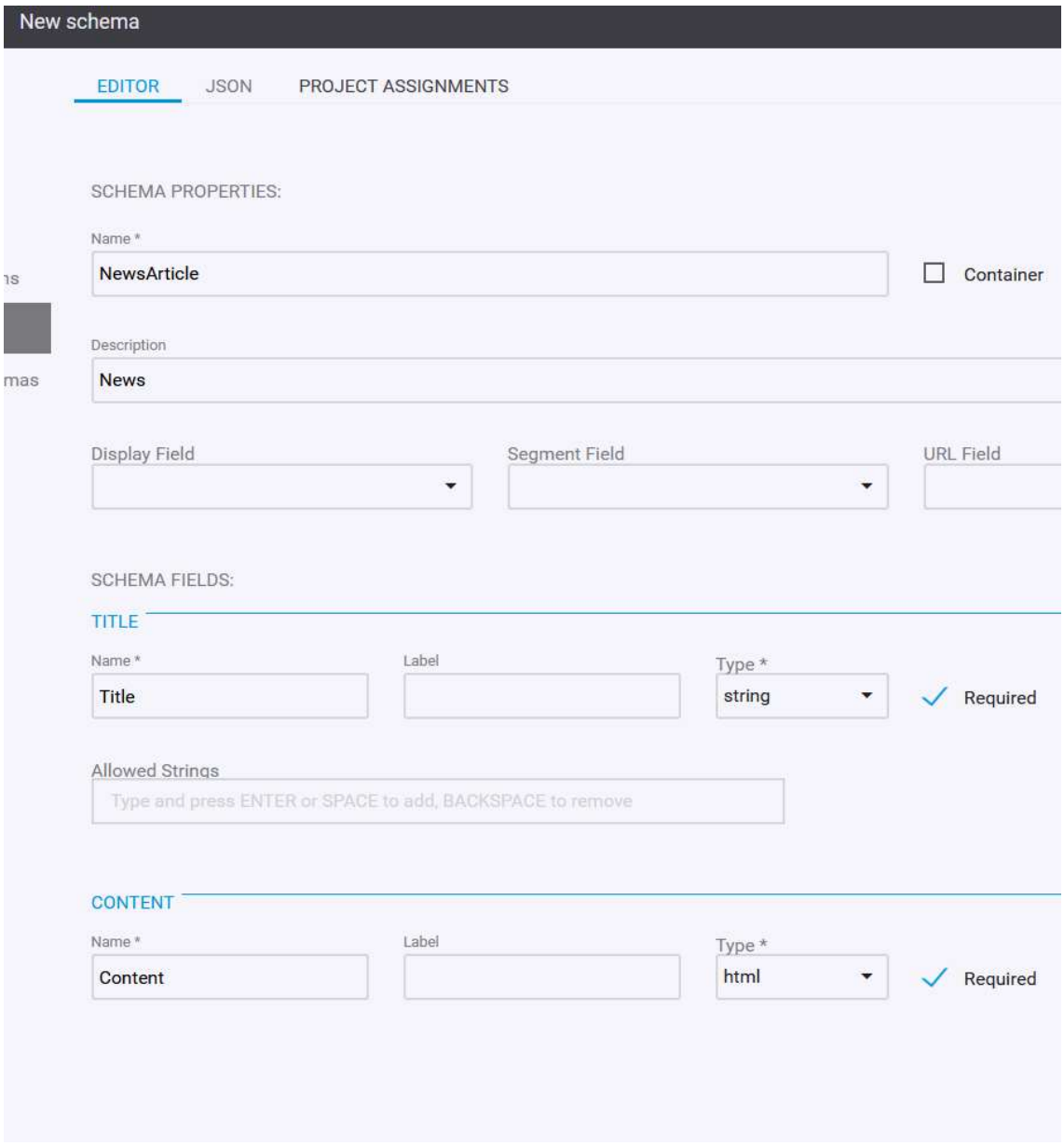
PICTURE 16. Gentics Mesh groups



PICTURE 17. Gentics Mesh “TL; DR” (Source: getting-started, Gentics Mesh)

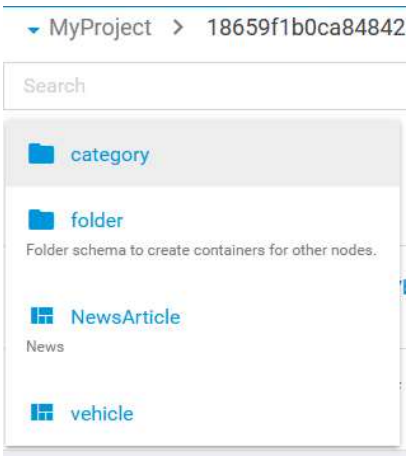
Gentics Mesh supports multi-language, where when calling the API, the user can query individual contents by appending the “?lang” query parameter (Gentics Mesh Docs, N.d.). An example endpoint would be “/api/v2/:projectName/nodes/:uuid?lang:en,fi.

PICTURE 18. Different languages in “Service” node.



PICTURE 19. Creating new schema called “NewsArticle”.

Creating new schemas in the user interface is easy and intuitive. Each field can be set to “required” that can help with maintaining structure in content. Now NewsArticle is visible when creating a new node as seen on picture (20)



PICTURE 20. NewsArticle in the list of schemas.

In picture (20) is a dropdown list that shows the NewsArticle schema, but before posting news articles with the new schema, first there needs to be a folder created for the news articles, so for the base there needs to be the folder with its name for example, “news” and then inside that folder, create news articles with the schema “NewsArticles”.

Gentics Mesh is a promising headless CMS option, with a solid user interface and easy to understand and broad documentation. Even though the Keycloak plugin is a commercial plugin, with Apache-2.0 it is possible to code the plugin or an authentication system manually.

4.5 JHipster

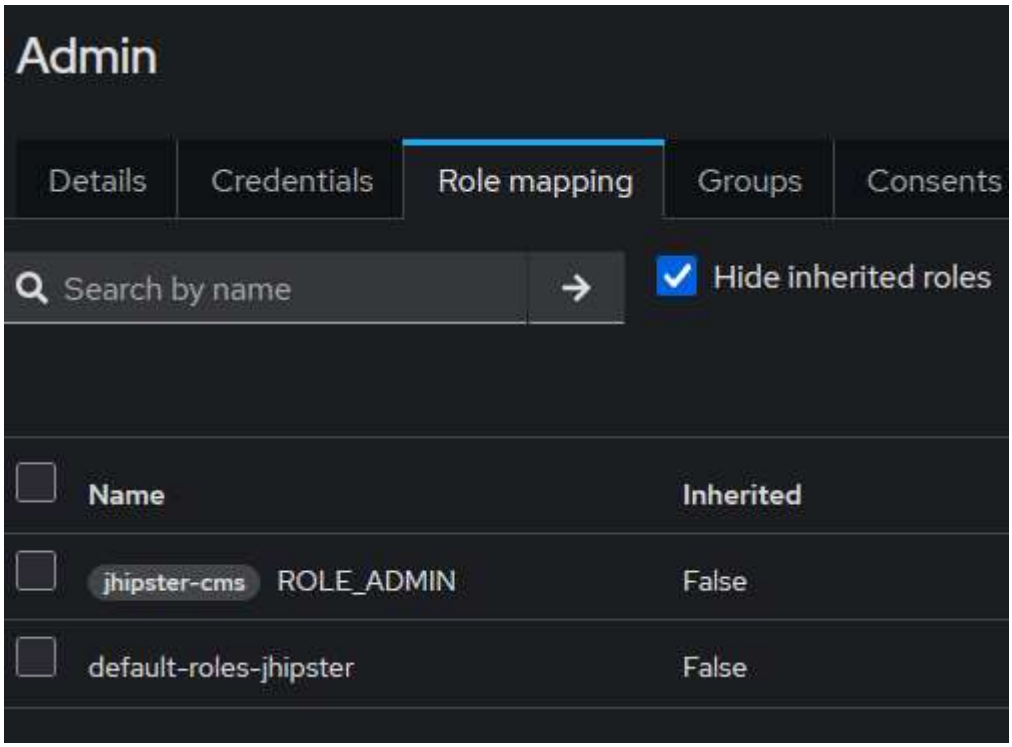
JHipster is an open-source development platform to develop and deploy web applications and microservices. JHipster is not a headless CMS out of the box, but it can be built to be a headless CMS. JHipster is based on Java spring-boot, and has built in integration with Keycloak, and supports MariaDB and many other databases.

4.5.1 JHipster setup

For this demonstration I followed along a video on YouTube, that goes through the basic installation of JHipster (Matt Raible, 2023).

After installing the basic setup of JHipster, adding Keycloak was the next step. Similar setup was used with Keycloak in JHipster as in PayloadCMS demonstration, so the pictures (6), (7), (9) are relevant in this case.

First, in this demonstration a admin user has been created in Keycloak admin panel, creating a user in the “Users” panel, with the name “Admin” and email “admin@cms.fi”. After creating the user, a custom role called “ROLE_ADMIN” must be created, and this role aligns with JHipster’s system.



PICTURE 21. Role mapping for admin.

Configuring Keycloak in JHipster is similar to PayloadCMS, in JHipster the Keycloak ClientID, Client-secret etc. will be added to the application.yml file that holds most of the configuration for the project.

```yaml
security:
  oauth2:
    client:
      provider:
        oidc:
          issuer-uri: http://localhost:8080/realms/jhipster
      registration:
        oidc:
          client-id: jhipster-cms
          client-secret: client-secret
          scope: openid, profile, email
```

FIGURE 5. Keycloak information in application.yml

After adding the Keycloak configurations to the application.yml, build the project using maven with the command “./mvnw”. And the project should be visible in the port that was configured. This port can be checked from the application-dev.yml file.

```
server:
  port: 8081
  # make sure requests the proxy uri instead of the server one
  forward-headers-strategy: native
```

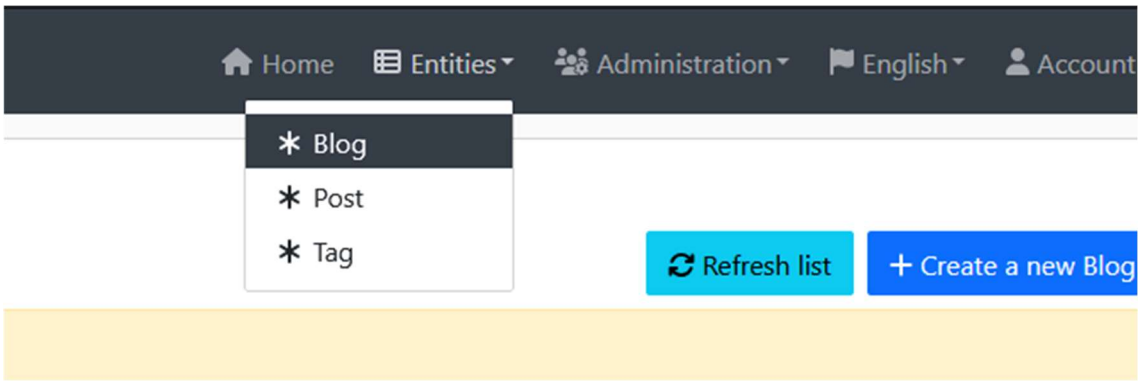
FIGURE 6. Server port for application.

Opening the port in “http://localhost:8081” will display the main page, with a link to login.

Creating entities in JHipster is done using JHipster Domain Language (JDL). The JDL is a JHipster-specific domain language where you can describe all your applications, deployments, entities and their relationships in a single file (or more than one) with a user-friendly syntax (JHipster Docs, 2023)

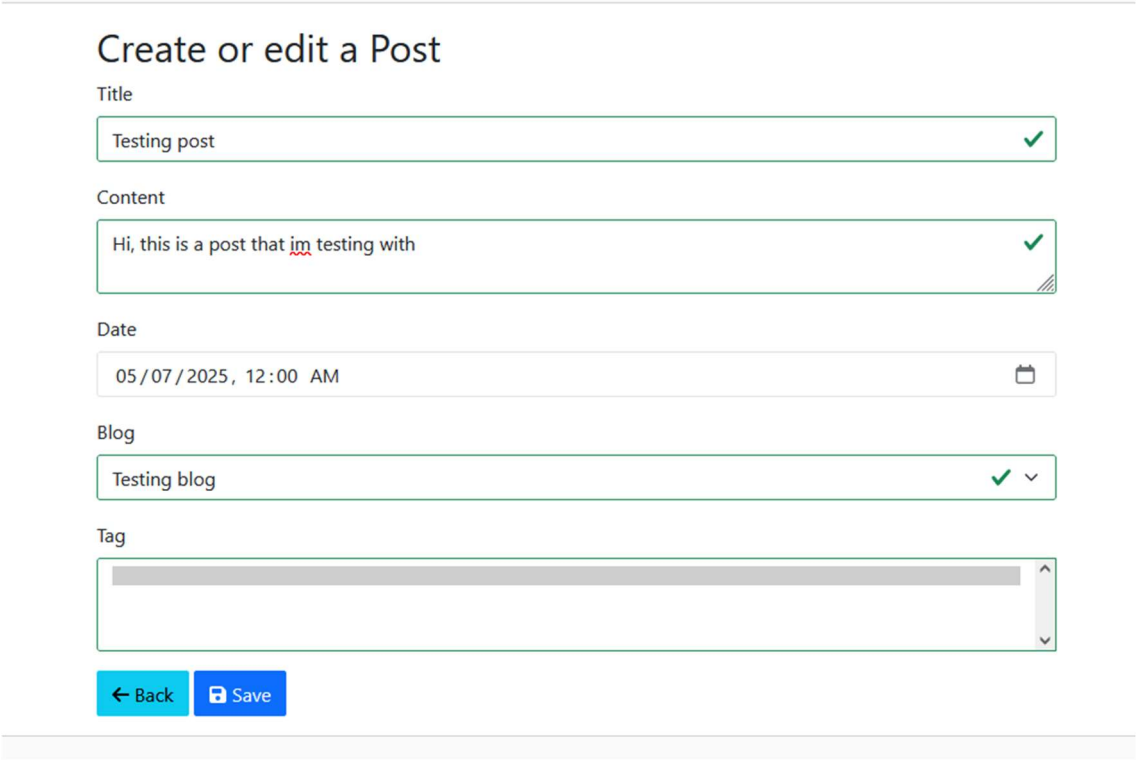
Using the JDL Studio (https://start.jhipster.tech/jdl-studio/) is an intuitive way to create the entities. This demonstration uses a “blog” entity for example that can be found in Appendix 6.

After installing the jdl using the command “jhipster jdl <file>.jdl”, the entity should be visible in the frontend. See picture (22).



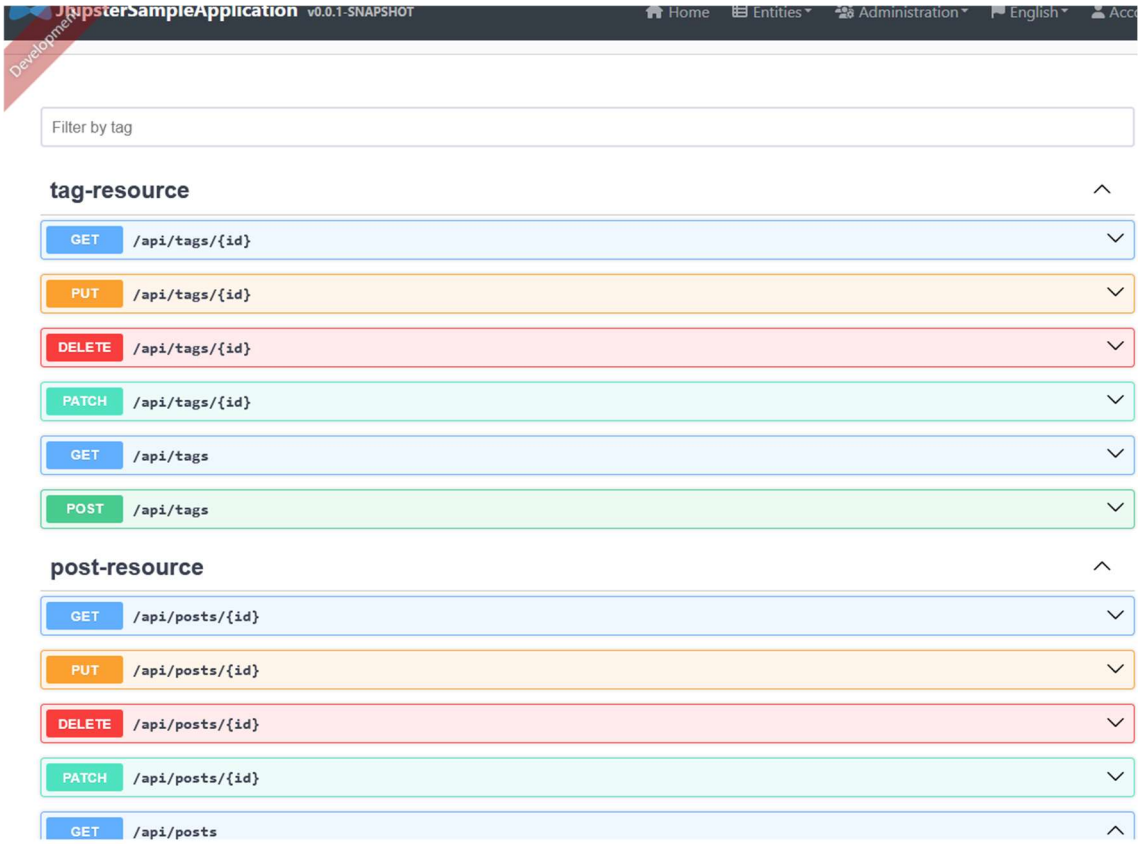
PICTURE 22. JHipster Entities.

In this demonstration, the Blog is the parent of the post, so each post is connected to a certain blog. See picture (23) for example post.

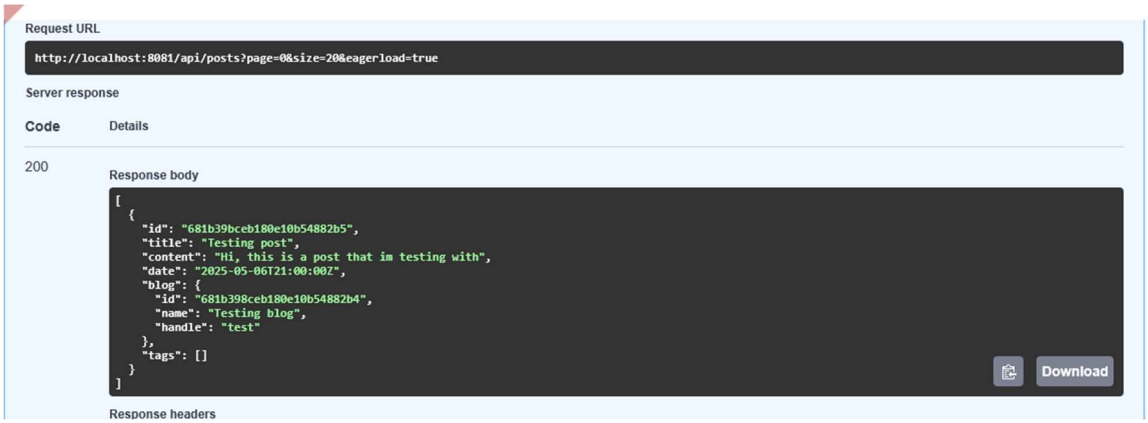


PICTURE 23. Example post with JHipster.

JHipster provides built-in Swagger UI for endpoints. Swagger UI is a user-friendly way to view endpoints in APIs.



PICTURE 24. Swagger UI.



PICTURE 25. Testing endpoint using Swagger UI.

JHipster is a fully customizable system that could be used for any use-case. The benefits of this solution are:

- Java spring-boot based.
- Fully customizable entities.
- Out of the box Keycloak support.
- MariaDB support.

4.6 Comparisons

TABLE 1. Headless CMS comparisons.

Software:	DecapCMS	PayloadCMS	Gentics Mesh	JHipster
RBAC:	False	True	True	True
Multi-tenant:	False	True	False	True
Customizable:	True	True	True	True
Keycloak integration:	-	True	Commercial Keycloak plugin.	True
Multi-language:	False	True	True	False
Supported databases:	-	MongoDB, Postgres, SQLite	MariaDB	MariaDB, Postgres
Language built with:	JavaScript	Typescript	Java	Java
License:	MIT	MIT	Apache-2.0	Apache-2.0

After comparing the different CMSs, objectively Genetics Mesh is the most impressive of the group, having the best documentation, a comprehensive Role-Based-Access-Control system, multi-language support, and the most fitting technology stack for Twoday. Even though the Keycloak plugin is a commercial plugin that might need a licenced version of the software, because Gentics Mesh is Apache-2.0 licensed, it is possible to create a custom OAuth plugin that enables OIDC.

5 CONCLUSION

After a comprehensive evaluation based on Twoday Oy's set requirements, such as support for multitenancy, single-sign-on (SSO) integration, it became clear that a few open-source solutions can meet these requirements without licensing costs.

Among the evaluated options, Gentics Mesh and DecapCMS are the most promising candidates. Gentics Mesh offers a robust, traditional headless CMS architecture that is built with Java and offers enterprise features like SSO and multi-language support. In contrast, DecapCMS introduces a unique Git-based content workflow, making it easy to manage and deploy with its lightweight architecture.

While both solutions have their strengths, the final recommendation depends on the specific project and customer needs.

6 DISCUSSION

The aim of this thesis was to explore and compare alternatives to Liferay's headless CMS. The platforms that stood out the most were:

- DecapCMS for its Git based workflow and simple installation.
- Gentics Mesh for its comprehensive features and well written documentation.

The platform that was a disappointment was Payload. Payload was listed in many of the articles that go through the best headless CMS options, but Payload in this case was a disappointment, the multi-tenant feature is messy with bad documentation, and overall, the documentation was bad and not very user-friendly. Future research could include Strapi in the list. Because Strapi is also open source, but the reason why it wasn't in this report is because there were not any real options for Keycloak integration. But in the future with the growing community there will be a third-party plugin available for Strapi to use OIDC with Keycloak. In addition, this thesis is of importance to Twoday Oy and their goals to expand their development.

REFERENCES

Contentstack. 2025. History of content management systems and rise of headless CMS. Read on 4.4.2025. https://www.contentstack.com/blog/all-about-headless/content-management-systems-history-and-headless-cms

Contentstack. 2020. What is a CMS? Read on 7.4.2025. https://www.contentstack.com/cms-guides/what-is-a-content-management-system

Di Mace. 2022. History of WYSIWYG and CMS. Read on 4.4.2025. https://www.tiny.cloud/blog/history-of-cms-wysiwyg/

Decap. N.d. Documentation overview. Read on 24.4.2025. https://decapcms.org/docs/intro/

Decap. N.d. Collections. Read on 7.5.2025. https://decapcms.org/docs/configure-decap-cms/

Decap. N.d. Azure. Read on 7.5.2025. https://decapcms.org/docs/azure-backend/

Gregg Lindemulder. 2024. What is RBAC? Read on 20.4.2025. https://www.ibm.com/think/topics/rbac

Gentics Mesh. N.d. Documentation. Read on 28.4.2025. https://www.gentics.com/mesh/docs/

IBM. n.d. What is multi-tenant? Read on 20.4.2025. https://www.ibm.com/think/topics/multi-tenant

LinkedIn. N.d. Twoday Read on 4.4.2025. https://www.linkedin.com/company/twoday/about/

Microsoft. n.d. What is OIDC? Read on 22.4.2025. https://www.microsoft.com/en-us/security/business/security-101/what-is-openid-connect-oidc

Microsoft. 2025. What is Microsoft Entra ID? Read on 7.5.2025. https://learn.microsoft.com/en-us/entra/fundamentals/whatis

Mulesoft. N.d. What is an API? Read on 7.4.2025. https://www.mulesoft.com/api/what-is-an-api

Mdn. n.d. Introduction to the DOM. Read on 4.4.2025. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Matt Raible. 2023. Get Started with JHipster 8. Published on 18.12.2023. Watched on 7.5.2025. https://www.youtube.com/watch?v=IfyjKCt6YHE

Oracle. N.d. What is omnichannel. Read on 7.4.2025.
https://www.oracle.com/in/retail/omnichannel/what-is-omnichannel/

OpenAI. (2025). ChatGPT (May version) [Large language model].
https://chat.openai.com/chat

Payload. N.d. What is Payload? Read on 21.4.2025.
https://payloadcms.com/docs/getting-started/what-is-payload

Payload. N.d. Multi-Tenant Plugin. Read on 21.4.2025.
https://payloadcms.com/docs/plugins/multi-tenant

Payload. Collections. Read on 7.5.2025.
https://payloadcms.com/docs/configuration/collections

payload-oauth2. GitHub. Retrieved on 22.4.2025, from
https://github.com/WilsonLe/payload-oauth2

Stephen J. Bigelow. 2024. What is multi-tenancy. Read on 18.4.2025.
https://www.techtarget.com/whatis/definition/multi-tenancy

Screamingbox. 2024. 7 Of the Most Popular Headless CMS Systems in 2024. Read on 8.5.2025. https://www.screamingbox.net/blog/7-of-the-most-popular-headless-cms-systems-in-2024

Tarek El Deeb. 2022. The history of CMS. Read on 4.4.2025.
https://www.storyblok.com/mp/the-history-of-cms

Tom Rankin. 2024. Comparing the best multilingual CMS platforms. Read on 5.5.2025.
https://translatepress.com/best-multilingual-cms/

Viacheslav Lushchinskiy. 2023. The Pros and Cons of Headless Architecture. Read on 7.4.2025.
https://medium.com/elca-it/the-pros-and-cons-of-headless-architecture-1151d7633f3c

WilsonLe. N.d. payload-oauth2. Read on 22.4.2025.
https://github.com/WilsonLe/payload-oauth2

APPENDICES

Appendix 1. Twoday interview

Twoday Interview (16.4.2025)

Okay, let's think about our needs. So Liferay includes the headless CMS API, there you can find a very comprehensive REST API that you can use to retrieve content. But Liferay is too heavy a system in a way for when you have a bunch of HTML content somewhere that you need to retrieve and display. We would like a much lighter solution. That's why we are interested in new options for this. In CMSs, there can be completely static content, for example, text pieces or some visual content, there can be news or some kind of newsletter, whatever we want, so that the customer can manage that content through the CMS. Now we want a lighter solution that doesn't have all that extra stuff in it. It's enough for the customer to log in, for example, local user IDs or Azure Entra ID. They can log in with their own IDs, then roles are needed, and there is a kind of maintenance side where those structures are built. And those structures have to be clean. You need to be able to build your own customized structures for certain types of content that the client's content producer can just use. We can build different structures because we are the administrators, and we manage those structures. In the worst case, the client can break something in the system. That's why roles are important in this case.

Question: Do you already have a solution in mind or are you interested in a specific one?

We've been looking at Strap, but the problem with Strap is that it has a paid and free version, but the paid version has the features we need, for example SSO. And SSO is an important feature and what customers need because they don't want to have their own accounts for each system, but they have Microsoft accounts and use them everywhere. It makes the customer's life a lot easier. And another one that has more potential is DecapCMS, where the content goes to Git, and is stored there as files in the project.

One important requirement is that the software must have sufficient support, i.e. in principle the larger the user base the better. I would like it to be Java-based, it would be fine if it is Node-based. But one of those. Translations must be supported, i.e. the articles must be able to be created in several different languages. Finnish, Swedish, English are the most important right now. OpenID Login, we use Keycloak and it would be good if you could get it with Azure Entra ID connected to it. And it would be a plus if you could get it as a multi-tenant. Here are the requirements and use cases roughly.

Appendix 2. DecapCMS config.yml

```yaml
backend:
  name: proxy
  proxy_url: http://localhost:8081/api/v1
  branch: master

local_backend: true


media_folder: "public/uploads"
public_folder: "/uploads"

collections:
  - name: "posts"
    label: "Posts"
    folder: "src/posts"
    create: true
    slug: "{{slug}}"
    fields:
      - { label: "Title", name: "title", widget: "string" }
      - { label: "Body", name: "body", widget: "markdown" }
```

Appendix 3. PayloadCMS keycloak.ts

```ts
import { OAuth2Plugin, defaultGetToken } from 'payload-oauth2'
import { PayloadRequest } from 'payload'
import axios from 'axios'

export const keycloakOAuth = OAuth2Plugin({
  enabled: true,
  strategyName: 'keycloak',
  useEmailAsIdentity: true,
  serverURL: process.env.NEXT_PUBLIC_URL || 'http://localhost:3000',
  clientId: process.env.KEYCLOAK_CLIENT_ID || '',
  clientSecret: process.env.KEYCLOAK_CLIENT_SECRET || '',
  authorizePath: '/oauth/keycloak',
  callbackPath: '/oauth/keycloak/callback',
  authCollection: 'users',
  tokenEndpoint: process.env.KEYCLOAK_TOKEN_URL || '',
  providerAuthorizationUrl: process.env.KEYCLOAK_AUTH_URL || '',
  scopes: ['openid', 'profile', 'email'],

  getUserInfo: async (accessToken: string, req: PayloadRequest) => {
    const { data: user } = await axios.get(process.env.KEYCLOAK_USERINFO_URL || '', {
      headers: {
        Authorization: `Bearer ${accessToken}`,
      },
    })
    return {
      email: user.email,
      sub: user.sub,
      name: user.name || user.preferred_username,
    }
  },

  getToken: async (code: string, req: PayloadRequest) => {
    const redirectUri = `${process.env.NEXT_PUBLIC_URL ||
'http://localhost:3000'}/api/users/oauth/keycloak/callback`
    const token = await defaultGetToken(
      process.env.KEYCLOAK_TOKEN_URL || '',
      process.env.KEYCLOAK_CLIENT_ID || '',
      process.env.KEYCLOAK_CLIENT_SECRET || '',
      redirectUri,
      code,
    )

    req.payload.logger.info(`Received token: ${JSON.stringify(token)}`)
    return token
  },

  successRedirect: () => '/admin',
  failureRedirect: () => '/admin/login',
})
```

Appendix 4. payload.config.ts

```ts
import { mongooseAdapter } from '@payloadcms/db-mongodb'
import { lexicalEditor } from '@payloadcms/richtext-lexical'
import path from 'path'
import { buildConfig } from 'payload'
import { fileURLToPath } from 'url'
import { Pages } from './collections/Pages'
import { Tenants } from './collections/Tenants'
import Users from './collections/Users'
import { multiTenantPlugin } from '@payloadcms/plugin-multi-tenant'
import { isSuperAdmin } from './access/isSuperAdmin'
import type { Config } from './payload-types'
import { getUserTenantIDs } from './utilities/getUserTenantIDs'
import { keycloakOAuth } from './auth/keycloak'
import Media from './collections/media'
import News from './collections/Pages/twoday/NewsArticle'

const filename = fileURLToPath(import.meta.url)
const dirname = path.dirname(filename)

export default buildConfig({
  admin: {
    user: 'users',
  },
  collections: [Pages, Users, Tenants, Media, News],
  db: mongooseAdapter({
    url: process.env.DATABASE_URI || '',
  }),
  editor: lexicalEditor({}),
  graphQL: {
    schemaOutputFile: path.resolve(dirname, 'generated-schema.graphql'),
  },
  secret: process.env.PAYLOAD_SECRET as string,
  typescript: {
    outputFile: path.resolve(dirname, 'payload-types.ts'),
  },
  plugins: [
    keycloakOAuth,
    multiTenantPlugin<Config>({
      collections: {
      },
      tenantField: {
        access: {
          read: () => true,
          update: ({ req }) => {
            if (isSuperAdmin(req.user)) {
              return true
            }
            return getUserTenantIDs(req.user).length > 0
          },
        },
      },
      tenantsArrayField: {
        includeDefaultField: false,
      },
      userHasAccessToAllTenants: (user) => isSuperAdmin(user),
    }),
  ],
```

Appendix 5. NewsArticle.ts in Payload

```ts
import type { CollectionConfig } from 'payload'

const News: CollectionConfig = {
  slug: 'news',
  admin: {
    useAsTitle: 'title',
  },
  access: {
    read: () => true,
  },
  fields: [
    {
      name: 'title',
      type: 'text',
      required: true,
    },
    {
      name: 'image',
      type: 'upload',
      relationTo: 'media',
    },
    {
      name: 'text',
      type: 'richText',
      required: true,
    },
    {
      name: 'author',
      type: 'text',
    },
    {
      name: 'date',
      type: 'date',
      admin: {
        date: {
          pickerAppearance: 'dayOnly',
        },
      },
    },
    {
      name: 'tenant',
      type: 'relationship',
      relationTo: 'tenants',
      required: true,
      admin: {
        position: 'sidebar',
      },
    },
  ],
}

export default News
```

Appendix 6. blog.jdl

```
entity Blog {
   name String required minlength(3)
   handle String required minlength(2)
}

entity Post {
   title String required
   content TextBlob required
   date Instant required
}

entity Tag {
   name String required minlength(2)
}

relationship ManyToOne {
   Blog{user(login)} to User with builtInEntity
   Post{blog(name)} to Blog
}

relationship ManyToMany {
   Post{tag(name)} to Tag{post}
}

paginate Post, Tag with infinite-scroll
```