Tu Nguyen

Java Spring Framework in developing the Knowledge Article Management application

A brief guide to use Spring Framework

Helsinki Metropolia University of Applied Sciences

Bachelor's Degree

Information Technology

Bachelor's thesis

1 March 2018

Helsinki Metropolia University of Applied Sciences

Authors	Tu Nguyen
Title	Java Spring Framework in developing the Knowledge Article Management application
Number of pages	35 pages + 3 appendices
Date	1 March 2018
Type of Project	Thesis
Degree Program	Information Technology
Instructors	Peter Hjort, Senior Lecturer, Main supervisor Sonja Holappa, Senior Lecturer, Language supervisor

This thesis aims to build an application called “Knowledge Article management system” for internal use. Many companies, especially large corporates, have encountered difficulties in creating and managing information within the organization due to lack of intuitive interface as well as insufficient IT skills of employees. This system modifies conventional tools and provides additional tools for better document editing and improves synchronization via enhanced authorization.

This study combines multiples theoretical modules to create the application. The architect is formulated in 3 major areas: Database server, API server and Web server. Each area consists of different technologies, for example, OAuth2 for authentication, SQL for database and RESTful service. After composing the architecture of application, further details on implementation and deployment processes are presented.

With numerous experiments of various technologies and methodologies, the Knowledge Article management system was created with advanced features compared to traditional text editors such as WordPress or other CMSs. The Spring framework along with its related technologies have proven their efficiency in practice by various applications as well as flexibility in conditional modifications.

In the scope of this paper, it was not feasible to cover the whole project but only a general perspective was presented. It was a firm evidence to the vast potential of techniques described in the literature. The limit does not lie with technology but with the creativity of human. Large companies as well as IT students may find this paper useful. The thesis provides relevant knowledge for further development of this system.

Keywords	Java Spring, Knowledge Article, Wiki Engine, Cassandra

Contents

Acronyms

AJAX: Asynchronous JavaScript and XML

AMQP: Advanced message queuing protocol

AMS: Article management system

API: Application program interface

CMS: Content management system

HATEOAS: Hypermedia as the engine of application state

HTML: Hypertext markup language

IDE: Integrated development environment

IO: Input/output

JPA: Java persistence API

JSON: JavaScript object notation

KAMS: Knowledge article management system

LDAP: Lightweight directory access protocol

MQTT: Message queue telemetry transport

MVC: Model-View-Controller

NoSQL: Not only SQL

OS: Operating system

POJO: Plain old Java object

REST (or **ReST**)**:** Representational state transfer

SQL: Structured query language

XD: Spring XD is name of one of the Spring projects. It's not an acronym for something.

XML: Extensible markup language

1 Introduction

This thesis deals with Java Spring architecture and developing applications using Java Spring, OAuth Protocol and Cassandra database. This final year project also looks at the architecture of Java Spring blending with other components and technologies.

1.1 Objective and outcome

The objective of the thesis is to build a Knowledge Article management application for companies that have large databases and are finding it difficult to manage them efficiently. Many existing management applications are missing some significant features and thus this thesis proposes customized features to meet company requirements, for instance, controllable accessibility, articles versioning and providing different application programming interfaces (APIs) to integrate with other micro-services.

The application created in this thesis is meant to be a web-based application aimed at helping the client to manage the knowledge base storage of their company. The knowledge base storage consists mainly of digital articles written by the employees of the client companies to save information that is used as a reference for answering related issues and questions.

The outcome of this thesis is the application that helps the client to efficiently manipulate the articles.

1.2 Thesis Outline

The thesis covers Model-View-Controller (MVC) theory and Java Spring framework concepts as the base of the application architecture. Technologies such as OAuth and Cassandra are described to show how they are integrated in the application.

The thesis provides an overview to employing the above-mentioned theories and technology to construct an article management system. Firstly, the thesis discusses the the-

ories and technologies as the background of the application. Secondly, the thesis presents the architecture of the application. Thirdly, it gives the implementation details of the application. Finally, it shows how to deploy the application to a server.

1.3 Key Concepts

The thesis uses the following important key concepts.

- Model-View-Controller (MVC): This is a software architecture pattern that is used to divide an application into three component parts to increase the efficient of software developing by allow code reuse and parallel development [1].
- Protocol: This is a defined set of rules that determine how unrelated systems or components can communicate [2].
- Application Programming Interface (API): This is a set of defined communication methods that describe how various software components communicate together [3].
- Open Authenticate (OAuth): This is an open protocol that enables a third-party application to obtain limited access from another web services [4].
- Scalability: This is a capability of a system to handle a growing amount of work, in a capable manner or its ability to be enlarged to accommodate that growth [5].
- Polyglot object mapper: This is an ability of the database driver can map a programming object into various database systems.
- Endpoint: This is a connection point of a web service where the URL of an active server page is exposed and can be accessed by a client application [6].

2 Literature Review

This section discusses the theory used in application design and implementation. It introduces Knowledge Management system's background with several characteristics of existing solutions such as Docker, Spring Framework, RESTful JSON API, OAuth2, Apace Maven, SQL Database, Kundera and JPA and Apache Tomcat.

2.1 Knowledge Article Management System

Knowledge Article Management System (KAMS) is one part of Content Management System (CMS) which is an application that users can have capabilities to manage all or a section of content, data or information of an article, website and so forth. Furthermore, users can manage those contents without knowledge of HTML [7].

In this project, KAMS is a web application on which business clients can collaboratively modify content and structure. Further, KAMS is a content management system, which is similar to Wikipedia but differs in a way that it mainly focuses on knowledge articles. KAMS helps and provides different features for clients to manage the digital content of articles. Evidently, it manages versions because clients can freely come back to any previous ones while they are writing. Equally important, it increases interactions among authors and readers in commending, rating and other features. In addition, it stores, secures and assigns rights to content. In this case, there is a filter for internal and public articles or clients created by the author so that internal clients can only see internal articles and the other way around.

Moreover, KAMS has a flexible article filter, which stores many different article's contents. When clients want to search for something, they can exploit as many subjects as they wish.

2.2 Characteristics of Existing Solutions

Nowadays, WordPress, Joomla and Drupal are the three most popular CMS in the world. Below are some main features and characteristics of them:

- **WordPress:** This is probably the most used and one of the most popular CMS available in the world. It is specifically designed for blogs and blogging [8]. A blog entry can contain any sort of media, so this can be used as an AMS, where each article is posted as a blog post. Latest version 4.9 was released on 15 November 2017 [9].
- **Joomla:** It's also a PHP based CMS first launched in 2005. Joomla claims that this CMS is being used by more than 2 million websites across the globe [10].
- **Drupal:** It's an open source PHP based CMS [11], used in various industries like healthcare, higher education, media and publishing, government agencies [12] etc. The latest version 8.4.0 was released on 4 October 2017 [13].

2.3 Strengths and Weaknesses of Existing Solutions

The following are some of the main advantages and disadvantages of existing solution

Table 1. Strengths and Weaknesses of several Content Management Systems

Existing Solutions	Strengths	Weaknesses
WordPress	• Easy to use • Large community of users • Strong majority of free plugins • Thousands of free and paid graphics templates available [14]	• Modification requires knowledge of PHP • Graphics modification requires knowledge of CSS and HTML • Given functionality can be added by different plugins created by different authors • Lack of PHP security • Tables and graphics formatting is complicated • SQL queries can be complex since it requires additional syntax [14]
Joomla	• Easy to install • Strong majority of free plugins • Has comprehensive navigation system and advanced administration • Good looking URLs [15]	• Limited adjustment options • Some plugins are paid • May occur some frustrating compatibility issues among the plugins [15]
Drupal	• Includes a lot of functionality • Variety of content types • Advanced user management • Big capabilities of design element editing • Flexible page content management • Strong majority of plugins • Has great support in management and modification of the script [16]	• Require advanced knowledge to install and modify • Take some time for users to get used to new solutions • Poor scalability and efficiency [16]

As shown in table 1, the existing solutions have both pros and cons from the user point of view. However, this project focuses on the weaknesses in order to create a better solution. This is because knowledge management is the core aspect in managing different versions and creating professional content without user needing to know advanced knowledge about technology in any field. In other words, KAMS can be used easily even for non-tech savvy persons. Furthermore, the content could be filtered based on user authorization. In addition, interactions between authors and readers is emphatically employed.

2.4 Docker

Docker is a Linux container management tool with a “social” element. It allows users to publish container images and consume those published by others. A Docker image is a recipe for running a containerized process. It will be utilized in this project as the container to deploy the application. Docker offers a suitable environment for building and shipping the application. Deployment using Docker is easy and the created “Docker image” can run anywhere [17].

2.5 Spring Framework

More specifically, this project uses Spring Boot to create the RESTful API and the webpages. Spring Framework is provided by Pivotal Software, Inc. Spring Framework is a set of software packages to enable the quick development of new applications. It is modular by design so any of the following packages can be used as per the need of developers. The “Spring Framework” includes projects [18] provided in Appendix 4.

2.6 RESTful JSON API

An API is needed for enabling the other components and developers to use the data of users of Knowledge Article Management System in the case company. The API will be RESTful. REST is acronym for **Re**presentational **S**tate **T**ransfer. Being RESTful means, it will follow certain architectural guidelines and constraints. The constraints are uniform interface, stateless, cacheable, client server, layered system and code on demand. [19]

2.6.1 Uniform interface

This constraint means that the developers, who know about the REST principles, will partially already know the correct endpoints to call. This particular constraint is guided by the following four principles:

- Resource based: the resources are the models of the data. In object-oriented programming, every type of data is modeled as classes and kept in separate tables in the database. This principle guides that the individual resources can be identified in requests using the requested URI. The models themselves are separated from the data returned in the response. The response is usually HTML, XML or JSON. Here JSON will be returned as much as possible. Only in some rare cases, where it's not possible to return JSON, like in case of user authentication (see OAuth2 below), HTML or XML will be returned.
- Manipulation of Resources through representations: when a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.
- Self-descriptive messages: each message includes enough information to describe how to process the message. For example, which parser to invoke may be specified by an Internet media type (previously known as a MIME type). Responses also explicitly indicate their cache ability.
- Hypermedia as the Engine of Application State (HATEOAS): clients deliver state via body contents, query string parameters, request headers and the requested URI (the resource name). Services deliver state to clients via body content, response codes, and response headers. This is technically referred to as hypermedia (or hyperlinks within hypertext).

2.6.2 Stateless

In a RESTful API, statelessness is the key. Essentially, what this means is that the necessary state to handle the request is contained within the request itself. The URI uniquely identifies the resource. [19]

2.6.3 Cacheable

Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients reusing stale or inappropriate data in response to further requests. Well managed caching partially or completely eliminates some client–server interactions, and that improves scalability and performance.[19]

2.6.4 Client Server

Only the interface is defined and after that clients and server are developed separately. These can be modified, scaled separately as long as the agreed upon interface is not altered.

2.6.5 Layered System

API server has multiple intermediates, and a client does not know whether it's getting the response from an intermediate or from the end server. This is done using caching. Using load balancing, this improves the performance and number of the requests, which the API can serve.

2.6.6 Code on Demand

Servers are able to temporarily extend or customize the functionality of a client by transferring logic to it that it can execute. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

This one is the only optional constraint of REST architecture. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

2.7 OAuth2

OAuth2 is an authorization protocol. The core features of this protocol have been defined mainly in RFC 6749 [20], RFC 6750 [21] and RFC 6819 [22]. It has become the industry norm nowadays. The REST API, which is planned to create, needs to be secured with OAuth2. OAuth2 standard allows the pre-registered 3rd party apps to get the permission from the user to access some secured data. It provides a way for user to control what

data do the apps can access and whether a particular application can modify/delete the data.

2.8 Apache Maven [23]

Maven is maintained by Apache Organization Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

Maven manages dependencies for Java projects. It fetches the dependencies and creates a local maven repository. Maven also provides scripts to test, build, and run the project. It maintains the whole lifecycle of the project.

2.9 Database – SQL or NoSQL

For keeping the data, the system needs to have a database. The choices for database are a lot and those are categorized mainly in two categories – SQL and NoSQL. SQL means the databases that use Structured Query Language and NoSQL simply means **N**ot **O**nly SQL. These are the database, which may or may not use Structured Query Language. MySQL, MSSQL are prominent examples of SQL database servers; while MongoDB, Cassandra are prominent examples of NoSQL databases. Table 2 shows some of the differences between SQL and NoSQL database servers.

Table 2. MySQL and NoSQL Comparison

	SQL	NoSQL
1	SQL databases have a fixed and pre-defined schema [24]	These databases have dynamic structure [24] of the models, when talking in terms of Object Oriented programming.
2	These databases use Structured Query Language (SQL) to manipulate data.	These databases use unstructured query language [24] to query the data. The language differs from database to database.
4	SQL databases emphasize on ACID properties [25]. ACID means Atomicity, Consistency, Isolation and Durability. [26]	Instead of ACID properties, NoSQL databases follow the Brewer's CAP theorem [27] (Consistency, Availability and Partition tolerance) [28]
5	These databases are vertically scalable. [28] That means in order to improve the performance the hardware must be improved.	NoSQL databases are horizontally scalable. [28] Horizontally scalable means the performance is improved by sharing the load among multiple machines.
6	These are called relational databases (RDBMS) because the relations among the tables can be defined.	These are called non-relational and distributed databases.
7	These databases are considered not well suited for hierarchical data.	These databases work well with hierarchical data.
8	Because of poor horizontal scalability, these database systems are not well suited for large amount of data, as in big data systems.	These systems are developed to handle large amount of data with high performance, so these databases are better suited for big data systems.
9	Reasons to use SQL database • ACID properties are needed. • The structure of the persistent data is known beforehand and does not change unexpectedly.	Reasons to use NoSQL database • Storing a large amount of data that often has little to no common structure. • Taking better advantage of cloud computing and cloud storage:

		Cloud often requires database to be distributed so that the load on shared hardware resource is not much. NoSQL databases, such as Cassandra, support distributed deployment out of the box.

In order to make this application handle a large amount of data at phenomenal speed, a NoSQL database appears as a better choice. A NoSQL database server, which supports distributed deployment out of the box, makes it easy to deploy the application at multiple locations across the globe.

A NoSQL database is chosen for this application name Cassandra. Cassandra is maintained by Apache and it is a database management software, which provides high scalability and high availability without compromising performance. Difficulty of integrating Cassandra is similar to that of a SQL database because of Kundera. Kundera is an object mapper which maps the entities stored in database with the Java objects. Kundera is explained more in section 1.10. It is known for good replication across multiple data centers at different locations. It's a proven, fault tolerant, performant, decentralized, scalable, durable, and elastic database.

2.10 Kundera and JPA

JPA is the Java Persistence API. It provides an interface for storing the java models, called entities, in the database. Kundera is a “polyglot object mapper” [29] with JPA interface. Polyglot object mapper means that it can work with multiple database servers, be it RDBMS or not, in a single application. The idea behind Kundera is to make working with NoSQL databases as simple and fun as working with SQL databases is. Kundera is being developed by iLabs at Impetus Technologies and is available open source. Kundera maps tables in a NoSQL database to Java models and provides a consistent and easy to use API for saving, modifying and deleting the data in database using Java API calls.

Kundera supports Cassandra, MongoDB, HBase, Redis, Oracle NoSQL, neo4j, CouchDB, rethink DB, kudu, all relational databases and Apache Spark.

2.11 Apache Tomcat [30]

Apache Tomcat is an application container, which provides multiple libraries to the application to be able to be accessible on the internet. Apache Tomcat runs on JVM. Spring Boot has Apache Tomcat server embedded and deployed the web server and API server. Development, testing, and deployment all these stages are planned to be on embedded Tomcat server.

This container is very different from the Docker containerization. Docker container will run JVM, Tomcat will run on a JVM, our application will run on the Tomcat server.

3 Architecture

This section presents the functional as well as non-function requirement of the application itself and discusses the architecture of the application in terms of components, Scalability and MVC Design Pattern.

3.1 Requirements

In software development process, a set of requirements serve as a guideline to help developers to focus on core components and features of the application and prevent them from the temptation of making a monolithic program. This section mentions requirements of the application in terms of two main category namely functional requirement and nonfunctional requirement.

Functional requirements describe the specifications of the application or, in other words, what the application can do. Firstly, the application needs to manage the article implementing four basic functions of persistent storage such as create, read, update and delete (CRUD). Secondly, the application is able to authenticate users into different groups such as admin group, manager group, internal group and external group. The admin group has full privileges to manage the articles. The manger group has similar privileges to the admin group except that a manager can only soft delete articles. The internal group only allows its users to read all public and private articles while the external group restricts its users to public articles. Thirdly, the application provides a version control to manage changes or articles. Finally, the application provides an Application Programming Interfaces (APIs) that allow other applications to retrieve the document resources.

Non-functional requirements refer to the criterion that can be used to evaluate the operation of the application or, in other words, the properties of the application. Firstly, the application is a standalone micro-service that can be integrated with another micro-service. Secondly, the application is able to handle at least one hundred thousand requests per second. Thirdly, the application can scale into multiple instances. Finally, the application uses common data center Cassandra with other existing micro-services with its own key space.

3.2 Components

The components of this application are standard, and the diagram below shows all the components and the relations between them.

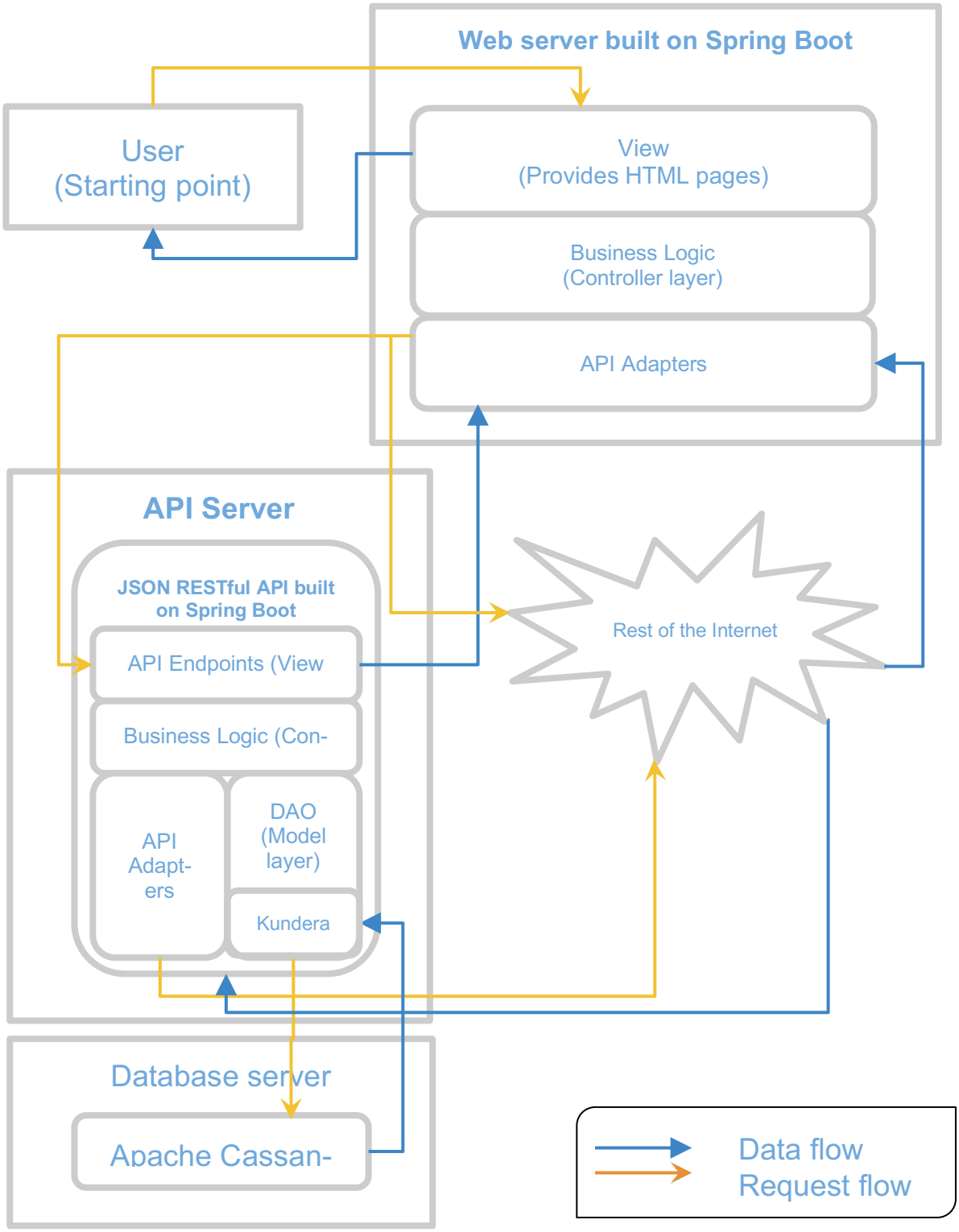


Figure 1. Proposed components of the architecture

In the paragraphs below, all the components shown in this figure will be defined.

3.2.1 Database server

As mentioned in the Introduction chapter, by using Cassandra as the database server, it requires providing a schema of tables and databases. Once the required databases and tables have been initialized with the needed data, it is the time to deploy it on a machine. A connection string, a username and a password will be retrieved, and API server will need these credentials to be able to access the databases.

Only the API server will access the database server. All the requests to access, to modify or to delete the data must come through the API. This ensures that any business logic written in the API server executes every time someone accesses the database, leaving less chance of runtime problems and possible illogical data states.

3.2.2 JSON API server

JSON API server communicates with database server and provides JSON endpoints to enable interaction of third party application with the database. This will be built with the use of Spring Boot framework. Have a look at the Appendix 1 to know how to build and run a Spring Boot application. It will be secured with Oauth2 authorization mechanism. For enabling OAuth2, Spring Security framework will be used.

Data read and written from the Cassandra database will be handled using Spring Data and possibly Cassandra Repository object [31]. This Cassandra Repository interface can be extended to suit important needs. Specifically, the needs are to be able to find articles by title, by category, by time of publishing, by the name of an author and by the ID of an author. To fulfil these needs the project extends Cassandra Repository object as shown in class diagram (Figure 3). This interface clearly shows some of the features, the application is going to be supported.

POJOs [32] (Plain Old Java Objects) can be shared between web server and API server, so these classes must be more or less same in both the components to enable a seamless communication and conversion of the models between these two components. All

the communication between the API clients and API server happens using JSON. The web server is a client for our API server.

Here four POJOs are used but in Cassandra only two entities are being saved, and those are Article and Attachment. The author class contains only three fields namely ID, name and link to his profile. The link to the author's profile can even be an external link, which is accessible publicly on the internet. Author objects can be instantiated with the data stored in table for Articles. Objects of Endpoint Response class are used to wrap the data returned from the API endpoints. For each request made to API, there will always be an instance of Endpoint Response class returned as JSON. Apart from tables for persisting articles and attachments, Cassandra will also have supporting tables created automatically by Kundera and Spring Security for object mapping and OAuth2 implementation. These supporting tables are required for Kundera and Spring security and are created and managed automatically by them.

3.2.3 Web server

Web server consumes the API provided by JSON API server. It follows the MVC pattern, described below. The web server will be built with Spring Boot framework. See Appendix 1 to know more technical details about building and running a Spring Boot application.

The web server will handle all the HTTP requests from the users and will provide mostly HTML response to them. Usually this response will be displayed on the browsers of the users. To get the data and fulfil the request, it will make appropriate calls to API server, and the API server will respond with JSON data. The job of web server will be to convert that data into HTML views and send it to the user over the internet.

The main purpose of keeping the web server and API server separate is to enable development of other sorts of applications using the same API server. The below section on scalability explains this further.

3.3 Scalability

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth [5]. Like any real-life facility or architecture, a software is designed by keeping the future of the system in mind.

Software is designed to handle a certain number of requests, processing and data, but it is also important to keep provisions of various sort to ensure that the system can be scaled in future if more performance is required to handle a bigger number of requests, processing and data. This is sort of future proofing in software design and implementation to keep the software in use for longer periods of time.

All the components in the proposed system are segregated, so these can be deployed on separate machines. These components are modular, which enables developing it by separate teams at the same time thus improving the rate of the development.

Scalability is one of the beautiful aspects of this architecture. Software level scalability can be achieved by the following strategies:

- **Well-known 3-tiered architecture:** The architecture is well known among the development community hence switching developers and finding new developers to speed up the development will not be a problem with this project. This will also reduce the time for new resources to attain their efficiency in the minimum time possible because almost all the component and software environment will look and feel familiar if they have worked on any other properly 3-tiered software development before. This brings a level of uniformity and order in the software development cycle. This tiered architecture ensures that the separation of concerns (SoC) [33] is followed, so that the spotting and fixing bugs can be quicker. Using Spring framework also makes dependency injection (DI) very simple and provides a Spring container for creating and running the application. Dependency injection decouples interface and their implementations [34] and it promotes reusability, testability and maintainability [35]. Dependency injection also reduces boilerplate code in the application because initializing or setting up dependencies is handled by a providing component [36].
- **Microservice architecture:** The primary benefit of using a microservice based architecture is that one set of endpoints can be upgraded and updated without affecting other endpoints and whole API for that matter. This helps a lot in upgrading a system which is being used by a large number of users at most of the times. The system proposed here lacks it, but it can be implemented in future. It is left for now but when that level of scalability is really needed it can be done. Another benefit of the microservice architecture is better monitoring and granular control over the whole API endpoints.

- **Separate JSON API server:** This helps adding more of applications which serve and modify the same dataset. The web server here which will serve a web application to the users and as JSON server is ready served, Mobile apps and desktop apps can also be created which provide data and interface to interact with it natively in the devices of users. One more thing is that it enables other developers to use the API and create their own applications based on the API. Enabling and empowering other developers to use your API requires extensive documentation and with the server, all that will be available thus making it easier to turn it into a small ecosystem of apps involving third party apps and app developers. Here "app" is used in its wider sense, it includes all sorts of software applications, such as Web application, Android app, iOS app, Windows Phone app, Windows 10 app, Windows desktop application, Linux application, macOS application, Java application and others.

3.4 MVC Pattern

API server as well as the web server follow MVC pattern [37] and the code is divided in three distinct layers. These layers are separate and are bound together by Spring container.

3.4.1 View layer (View)

For JSON API server, this is where exposed endpoints are defined, because view of a web server is equivalent to the API exposed by an API server. This layer defines how the app developers will interact with the API and subsequently with the data of an authenticated user served by the API. The endpoints must be well thought out so that they are sufficient and need not to be changed in anyway, because changing an endpoint or modifying the data provided by an endpoint may break existing applications which rely on it. Often changing endpoints must be marked unstable. Stability is one primary quality of a good API. Reliability comes from the stability of the API.

As per figure 3, this view layer contains endpoints to expose three resources, these endpoints will be secured with OAuth2 and after proper authentication users will be able to share their data with third party applications. Exposed resources are-- Articles, Attachments and Authors. These resources can be accessed by third party applications using proper HTML requests, and the JSON API server will return proper HTML status codes

in the response. Each resource sharing class in the view layer contains an endpoint to search the database for a particular resource. This search is accessible with GET request and it needs to have an instance of that particular resource with all the fields set for searching. This searching has a limitation that it searches only for equality. Other endpoint accessible with GET request is to find a resource by its ID, given a ID, it returns JSON form of the resource with 200-OK code status or a blank response with 404 NOT FOUND HTTP status code.

Let us consider the response, given by the endpoints. The response from the endpoints will always be as the Endpoint Response class shown in figure 3. This class takes a generic to return the object of appropriate type. It contains information about the time when the request was fulfilled by the server, a Boolean to indicate whether the request was processed successfully. If the Boolean indicates that there was some error, the message will contain more information so that the developers can take the preventive measures to get around that error. It also contains a list of objects to actually contain the data the third-party applications are interested in. This kind of response along with appropriate HTTP status codes helps a lot with the API integration. As shown in the view layer in figure 3, all the endpoints return Endpoint Response which is taken by JSON marshaller and is converted to JSON immediately to be sent via the network. As already mentioned, conversion from Java objects to JSON is handled by Jackson from FasterXML team.

For web server, this view layer creates and provides HTML views in response to the user requests. HTML is, as usual, accompanied with supporting JavaScript and CSS files. This view gets rendered by the user agent (browser) of the visitors. To make the view, frameworks like jQuery, Bootstrap can be used. These frameworks make the web pages look better and adhere to best practices in the least possible development time. jQuery is a JavaScript library which allows the HTML traversal, manipulation, AJAX requests in much simpler way than plain JavaScript. jQuery supports all the major browsers, so developers don’t have to think about the compatibility issues while writing client-side JavaScript code.

3.4.2 Business Logic Layer (Controller)

This is where the logic goes. This layer provides any specific data, or its processing required to create the view (or API Endpoint). The classes in view layer call this layer.

This way the logic from view is encapsulated. In Java terminology, the objects in this layer may be called managers.

This layer provides methods for the view layer to get the data or to process the data as per the request obtained from the user. This is where the processing engine of the whole application lies. This is where the business logic goes, so this layer is also called 'Business Logic Layer' in some technologies like .NET.

Another interesting point is where the data is not fetched from a database server but from an API server. Any adapter [38] which interact with external APIs exposed by third party web services are also included under this layer.

In JSON API server, this layer gets called by the classes exposing API endpoints and provide them the data which those classes need to convert to JSON and return. Usually this conversion to and from JSON is done automatically by appropriate interceptors. Conversion to JSON is called marshalling and creation of Java objects from JSON is called unmarshalling. Jackson [39] will be used in API server as well as in web server For JSON marshalling and unmarshalling. Jackson is the best JSON parser for Java [40], it's lightweight and does the conversion at the fastest speed. Jackson JAX-RS provider component will be used in our API server [41].

In the web server, this layer gets called by the classes which provide HTML views to the users. This layer provides all the data those classes need to create the HTML views. Usually the templates for the views are saved as static HTML or JSP files and dynamic data is populated in the placeholders. This data which needs to be displayed in that template is provided by the business logic layer. Most of the time third party templating engines like Velocity [45] are also used.

3.4.3 Data Access Objects layer (Model)

This layer interacts with database and provides coherent methods and any specific methods for our business logic layer. This layer is responsible for the data persistence and retrieval. In this application, Cassandra is chosen to be used with Kundera is the Java driver, as only this layer is concerned about persistence, no Java object from these two dependencies must go from data access object layer to business logic layer. Even all

the database specific exceptions must be handled and only the application specific exceptions must be thrown with proper error messages. In this project, this layer database is kept agnostic as much as possible, but if it is not possible, the Business Logic Layer is kept absolutely database agnostic.

4 Implementation

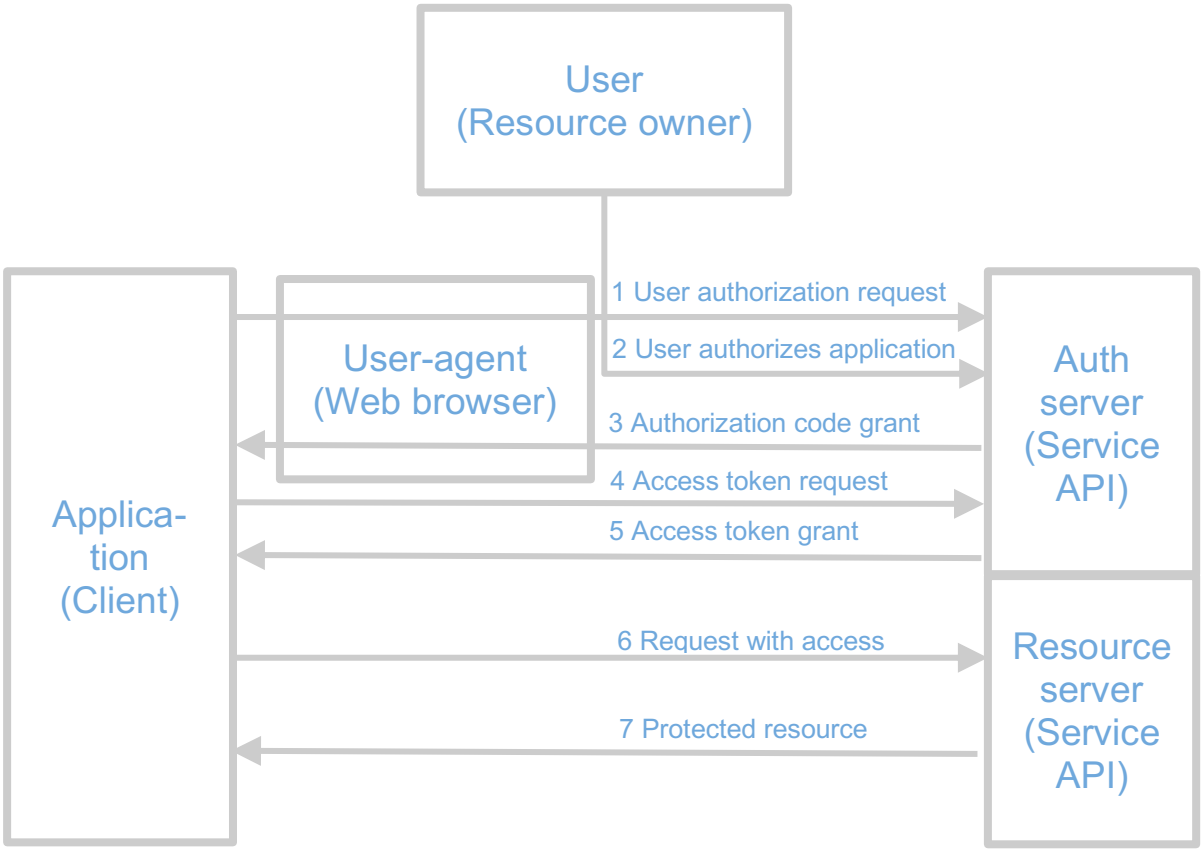
This section presents the implementation of the application in different parts. Firstly, it shows how to authenticate on server side using OAuth 2.0. Secondly, it describes how to construct the application. Finally, it discusses how to customize a Wiki engine.

4.1 Authentication using OAuth 2.0

OAuth2 will be implemented using Spring Security framework. Spring Security is introduced earlier and more about Spring frameworks can be seen in Appendix 4. OAuth 2 contains four types of grants and these follow different flows [22] through which the user can authorize applications to access their data from the API server.

1. Authorization code
2. Implicit
3. Resource owner password credentials
4. Client credentials

Figure 2. Authorization code grant flow



Here in the server, there is only first grant type which is authorization code. This type of grant allows applications, which are hosted on a server and can keep information hidden from the user agents, to ask permission to access and modify users' data from the API server

Describing other types of grants, the grants and flows described below are not implemented because these are not needed in this application.

Implicit grants are used for the applications which run on mobile devices or web browsers, in these environments the confidentiality of client secret cannot be maintained. This is also redirection-based flow, because the client secret cannot be kept safely, so the identity of application is not verified because the content of request can be seen by the users, the identity is solely based on the pre-registered redirection URL. It does not support refresh tokens.

Resource owner password credentials are used for trusted applications such as the applications developed by the same group of people who built the API. Here the application is not registered, and the token is provided for the authenticated user. It is not as secure as the two mentioned above.

Client credentials grant is not used to get access to user data, but this grant is provided to registered applications to access and modify their own data. With their own credentials, the applications can modify their own data, such as name, redirect URL etc.

4.2 JSON API server

A good API server is which has good security and yet provides information in the best possible and coherent way. It must also adhere to the principles of HTTP status codes and in case of any error, the response must contain enough information so that the app developers building apps upon the API can resolve the problems without hassle. The data provided by the API should be consistent. API developers and maintainers must use semantic versioning so that all the app developers who depend on the API can modify their apps accordingly. The API developers and maintainers must also ensure that all the versions of API server are deployed and available at the same time. This is to ensure that any apps using old versions of API do not break. To make it happen, when deploying the next version of API server, do not remove the old version. The old versions must be

removed only when you are sure that none of the apps are making any request to that particular version and even after that a timely notice/circular must be provided to notify the developer community. This instils trust among them hence increasing the popularity of the provided API service.

While discussing the details of its implementation. The first thing to develop is the resource, in this case, an “article” object is in the centre of it and all other resources (models) and whole application develops around it. Cassandra is being used as a database server where writing is cheap and reading from more than one partitions is costlier [31]. Cheap means less resource intensive and less time taking. This means that there can be duplicate data in the database as long as the number of partitions read while executing a query can be reduced, thus the tables are to be created as per the queries which is executed in order to retrieve the data. Here are a few examples which should be supported by application.

- Get articles by date published
- Get articles by categories
- Get articles by author
- Get articles by popularity
- Get articles published in a particular day, week, month or year

Here is a table that can contain multiple partitions, so the partitions are kept fewer, but the data must be divided equally among all the partitions created. [31] Another aspect which is different from SQL databases is that it can easily store lists and arrays in the table, but lacks foreign keys to refer to other tables, so one way it favors composition more than aggregation [42].

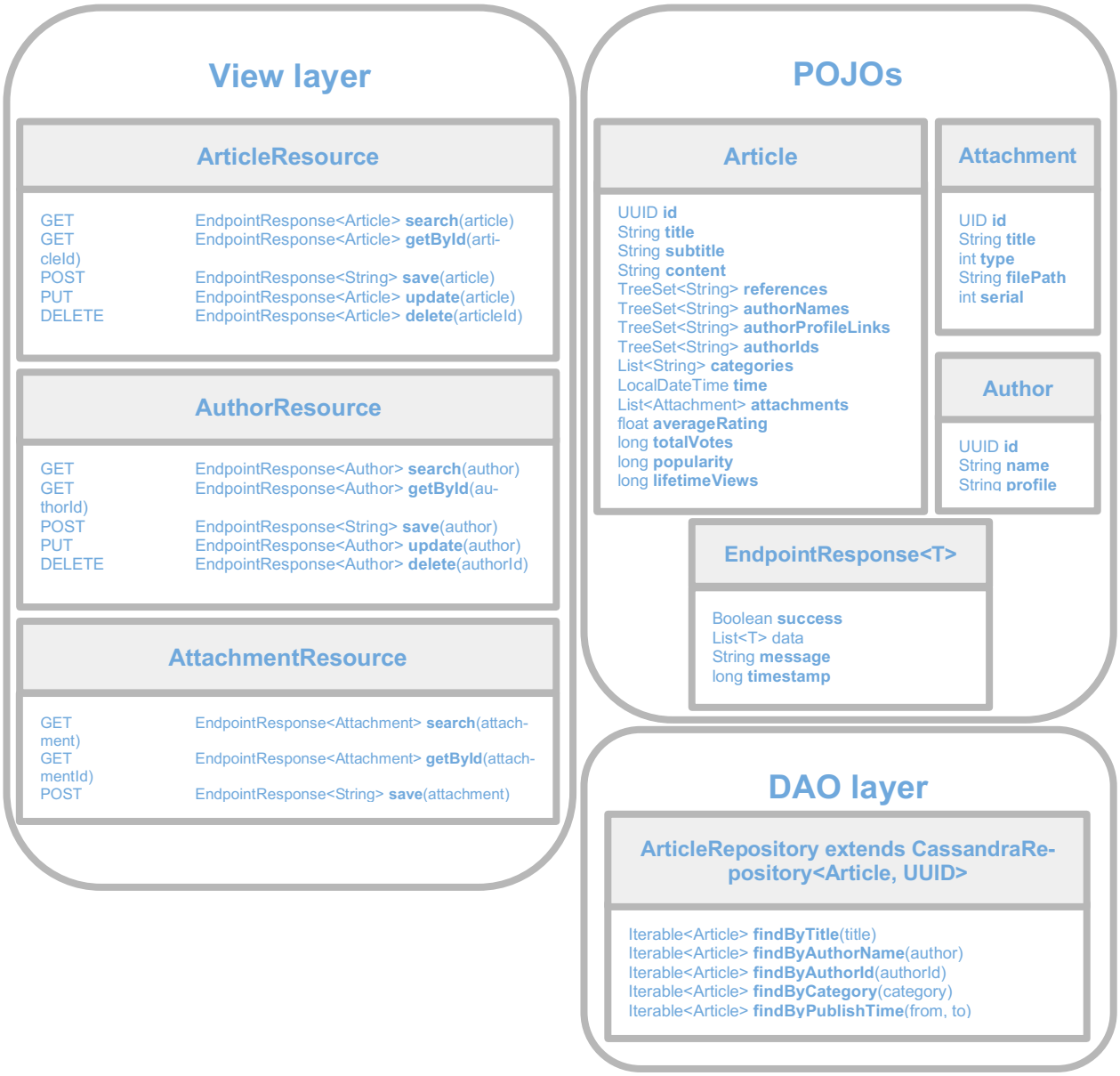
Article object is designed to support the following features.

- There can be more than one authors for an article. This enables putting the articles which are written in collaboration by many authors.
- There can be links to the profiles of the authors. These profiles of authors may be hosted on external portals as well.
- There is a way to keep internal IDs for the authors. This is to enable the selection of articles by the author. By keeping some internal IDs for all the authors, articles written by a particular author can be quickly searched because here the database server just need to compare two numbers instead of two strings.

- There can be more than one reference to the external material. This is a necessity to keep the references properly.
- An article can be categorized or tagged into multiple categories. This helps readers to browse and read articles about related topics.
- An article can have multiple attachments. All the images shown in line with the content should either be a publicly hosted external image or an attachment. Attachments of other types will be available to download as files below the article.
- An article can have content which is basically a markdown formatted string. Markdown makes rendering and parsing the content easier.
- A way to measure the current popularity of the article. This may be really helpful in creating recommendations for the readers. For this reason, a field named “popularity” of type long is kept in this class. This simply records the number of views in past week/month or year as decided beforehand. This value along with average rating should be used for recommending this article.
- Users will be allowed to rate the article they just read, no individual votes will be recorded but total number of votes and average rating is saved. There are two fields “totalVotes” of type long and “averageRating” of type float.
- Finally, an article will also keep the time when it was published. This information is used to find, sort and group the articles.

Figure 3 shows a class level diagram for JSON API server.

Figure 3. Class level diagram (showing only prominent classes and fields) for JSON API server



Article object has attachments, Attachment class is created as well. This class simply has 3 main fields namely title of the attachment, URL of the file to attach, which usually is the file path of the attachment on the server, and type, 0 means image and 1 means non-image. Attachment class also has serial, which can be omitted, it simply tells the renderer the order in which the attachments are to be displayed.

4.3 Wiki Engine

In order to manage the versions of articles, it would be much more convenient and efficient to integrate the Wiki engine than implement all of this feature. Wiki engine is an

open source project that provides feature-rich and build around standard JEE components. For better understanding of Wiki software, it is better to examine what the main components of the software are. In general, the Wiki technology space comprises major components as depicted in figure 4.

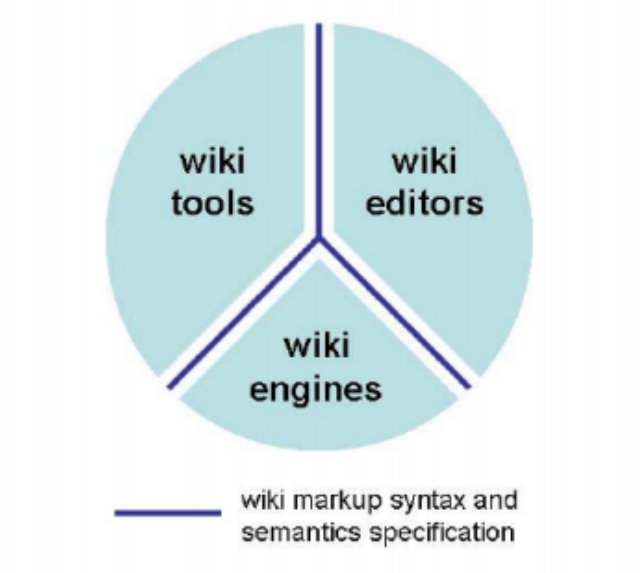


Figure 4. Three components of Wiki software decoupled by Wiki markup syntax and semantics specification. Reprinted from Martin et. al. (2008) [46].

As can be seen in figure 4, the Wiki software is decoupled into three independent parts namely Wiki engines, wiki tools and Wiki editors by Wiki markup syntax and semantics specification. The Wiki engine is the core component of the software and is used to store and retrieve wiki pages. The Wiki editor provides the editing platform that collaborators utilize to produce Wiki contents. The Wiki processing tools enable users to migrate Wiki data into different formats such as word document, pdf document and open office documents. [46]

Discussing integration in more detail, the first thing is to have the engine included in the project. This step can be easily done by fetching the wiki project from maven repositories. The engine can be enable by do some tweak in the configuration file so that the engine can start along with the application. The key things here are to declare two filters class, which are WikiServletFilter and WikiJSPFilter, so that all the requests in and out can be

go through the Wiki engine. With these simple steps, Wiki Engine is enabled in our project. However, as default setting of Wiki Engine, the articles are managed with File System Provider. In other words, all articles and related data is stored as Directory tree in the storage. As a result, a custom provider class is created that redirects the engine to store articles into our Cassandra database. This is the final step and the most essential part to integrate the engine into the application created in this study.

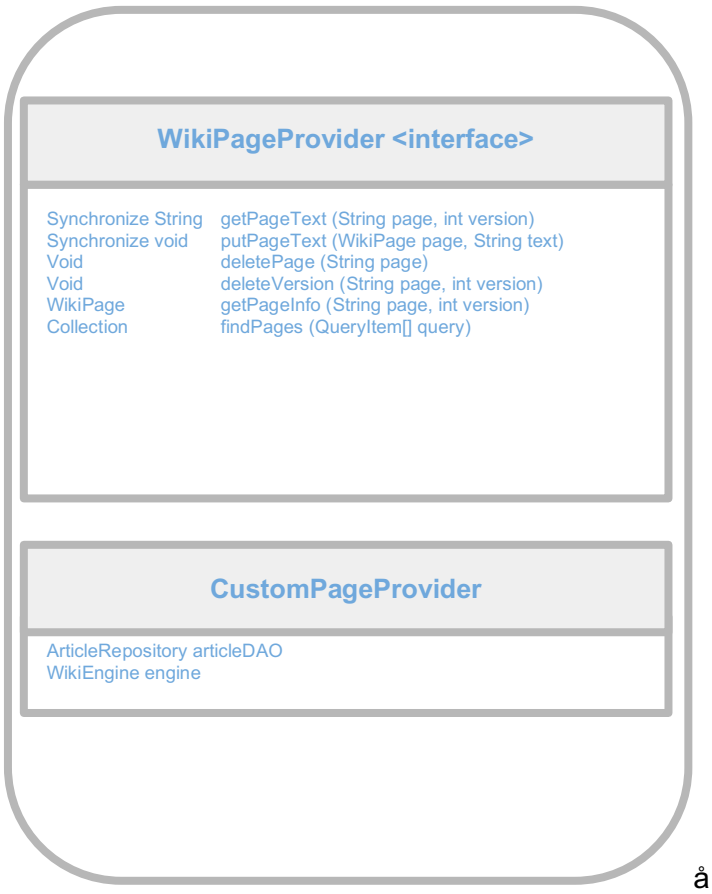


Figure 5. CustomPageProvider class diagram for redirecting Wiki Engine store data into Cassandra

A custom provider should implement WikiPageProvider interface of the Wiki Engine and has the following features:

- Class has an instance of Wiki Engine and Article Data Access Object (DAO). This enable class can have DAO object to save the article info into Cassandra database and retrieve the wiki context if necessary
- getPageText: Method to get the content of article base on the article's name and version number. If version number is negative or not existing, it can be returned the content of latest version

- putPageText: Method to save the articles with given article's name and text content of articles as parameters.
- deletePage: Method to remove the articles by name. All the version of this article will be removed as well.
- deleteVersion: Method to remove a specific version of an article.
- getPageInfo: Method to get all related info of the article include the content base on article's name and version number as the parameters. If the version number is negative of not existing, it can be returned the info of latest version.

Constructing a whole engine to manage the articles would take a lot of resources whilst it can be simpler and less time-consuming with only a few customizations. Wiki engine has provided my project with useful power in manipulating the database.

5 Deployment

This section discusses the deployment of the application in terms of how to build it as well as hardware scalability and containerization.

5.1 Building the application

The application is built using Maven as a build tool. Maven provides complete dependency management and application lifecycle management. To build the application, the way is to simply run the following command, `mvn spring -boot:run`.

To package the app for deployment mvn package is run. This creates an “all-including” JAR file. This JAR file contains embedded Tomcat server and this JAR can be deployed on any compatible JVM without the need of any more tools.

5.2 Hardware Scalability

The application has three separate parts as follows:

- Database server
- API server
- Web server

All these components can be deployed on separate hardware machines. These machines can have hardware more suitable to the kind of server deployed on it.
The database server can have better hard drives such as SSDs to enable faster data storage and retrieval while the API server can have better processing power to be able to process all the data and requests. A web server can have better network capabilities to be able to handle large number of requests. Using Cassandra as our database server helps in scaling it. Cassandra is inherently a distributed database server hence it can be deployed on multiple systems across the world reducing network time and ensuring better backup and fault tolerance.

While talking specifically about web and API servers, the architecture of these components is called “monolith”. That means that all the contents of these components must be kept together at a single hardware machine. This inhibits the hardware scalability a little. For API server specifically, microservice architecture can be done, meaning that

one type of endpoints can be kept on one machine while other part of the API containing different set of endpoints can be deployed independently on different machines, thus increasing the overall power and hardware deployed to serve the overall system.

Having mentioned microservices based architecture, now let us discuss the hardware scalability solutions for our monolith applications first. One thing which will certainly affect the overall quality and responsiveness of the system is network bandwidth connecting these components. The better the connection between these components the faster the system will be to serve a higher number of users. Another thing, which will help the system, is geographically distributed servers. To properly distribute the servers, single location needs to be started and then watched the geography of the users of the system and plan accordingly to distribute the system. While distributing the server, the different instances of the databases need synced up, which is handled by Cassandra itself.

5.3 Containerization

By containerization, this application is saved as a docker image and this docker image can be deployed on a single container. To create a Docker image, a specific setting in the pom file is used so that Maven can create the image needed.

A sample pom file is shown in Appendix 2. This enables the system to create the image fil needed and changes in the configuration is done in order to make it work on a docker container. Once dockerfile and pom.xml files are created, following command is run in order to run the application on Docker container.

```
./mvnw install dockerfile:build
```

6 Conclusion

This paper has presented briefly the implementation of various technologies to formulate Knowledge Article Management System, a web-based application which can handle a large number of users simultaneously.

A few months of studying and working in this project has broadened the author's knowledge about Java Spring framework and some related technology in software developing as well as delivery. Firstly, difficulties were resolved in configuration and setting up Spring components by using an experimental approach that gradually eliminates wrong choices. Secondly, since JSPWiki engine was used to manage the content version of articles, the concepts of Wiki engine and its integration with the Spring Framework had to be studied. Thirdly, gaining understanding of the open standard for the authentication (OAUTH) technology was needed. Finally, this project provided a chance to employ popular Docker and Consul to DevOps.

There are still plenty of opportunities for further research and building based on this study. At the moment, the security of the system is barely mentioned. One can implement better secured layers of protection allowing more confidential articles to be created on the system. Another research direction could be importation and reading of various formats other than office conventional files. Or, drag and drop features can be added to aid creators to be more creative with their contents. Nevertheless, the possibilities are endless. Creativity is not hindered by limitation of technologies but by barriers of imagination.

References

1. What is Model View Controller (MVC)? - Definition from Techopedia [Internet]. Techopedia.com. 2018 [cited 10 March 2018]. Available from: https://www.techopedia.com/definition/3842/model-view-controller-mvc
2. Protocol [Internet]. En.wikipedia.org. 2018 [cited 10 March 2018]. Available from: https://en.wikipedia.org/wiki/Protocol
3. Cite a Website - Cite This For Me [Internet]. Heim.ifi.uio.no. 2018 [cited 10 March 2018]. Available from: http://heim.ifi.uio.no/~frank/inf5040/CBSE/Component-Based_Software_Engineering_-_ch1.pdf
4. Grimes R. What is OAuth? How the open authorization framework works [Internet]. CSO Online. 2018 [cited 10 March 2018]. Available from: https://www.csoonline.com/article/3216404/authentication/what-is-oauth-how-the-open-authorization-framework-works.html
5. Hedge M. Why Is Scalability Important for My Business? - Contegix [Internet]. Contegix. 2018 [cited 10 March 2018]. Available from: https://www.contegix.com/why-is-scalability-important-for-my-business/
6. The Definition of Web Service EndPoint | Techwalla.com [Internet]. Techwalla. 2018 [cited 10 March 2018]. Available from: https://www.techwalla.com/articles/the-definition-of-web-service-endpoint
7. What is Content Management System (CMS) [Internet]. Comentum.com. 2018 [cited 10 March 2018]. Available from: http://www.comentum.com/what-is-cms-content-management-system.html
8. Blog Tool, Publishing Platform, and CMS — WordPress [Internet]. Wordpress.org. 2018 [cited 10 March 2018]. Available from: https://wordpress.org/
9. About » Roadmap — WordPress [Internet]. Wordpress.org. 2018 [cited 10 March 2018]. Available from: https://wordpress.org/about/roadmap/
10. Joomla! The CMS Trusted By Millions for their Websites [Internet]. Joomla!. 2018 [cited 10 March 2018]. Available from: https://www.joomla.org/
11. Ways to Get Involved [Internet]. Drupal.org. 2018 [cited 10 March 2018]. Available from: https://www.drupal.org/contribute
12. Drupal - Open Source CMS [Internet]. Drupal.org. 2018 [cited 10 March 2018]. Available from: https://www.drupal.org/
13. Drupal core [Internet]. Drupal.org. 2018 [cited 10 March 2018]. Available from: https://www.drupal.org/project/drupal
14. WordPress - CMS review, advantages and disadvantages [Internet]. Whichcmstochoose.com. 2018 [cited 10 March 2018]. Available from: http://whichcmstochoose.com/wordpress.html
15. Joomla! - CMS review, advantages and disadvantages [Internet]. Whichcmstochoose.com. 2018 [cited 10 March 2018]. Available from: http://whichcmstochoose.com/joomla.html

16. Drupal - CMS review, advantages and disadvantages [Internet]. Whichcmsto-choose.com. 2018 [cited 10 March 2018]. Available from: http://whichcmsto-choose.com/drupal.html

17. What is Docker? [Internet]. Docker. 2018 [cited 10 March 2018]. Available from: https://www.docker.com/what-docker

18. Spring Projects [Internet]. Spring.io. 2018 [cited 10 March 2018]. Available from: https://spring.io/projects

19. Todd Fredrich P. What is REST? [Internet]. Restapitutorial.com. 2018 [cited 10 March 2018]. Available from: http://www.restapitutorial.com/lessons/whatisrest.html

20. RFC 6749 - The OAuth 2.0 Authorization Framework [Internet]. Tools.ietf.org. 2018 [cited 10 March 2018]. Available from: https://tools.ietf.org/html/rfc6749

21. RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage [Internet]. Tools.ietf.org. 2018 [cited 10 March 2018]. Available from: http://tools.ietf.org/html/rfc6750

22. RFC 6819 - OAuth 2.0 Threat Model and Security Considerations [Internet]. Tools.ietf.org. 2018 [cited 10 March 2018]. Available from: http://tools.ietf.org/html/rfc6819

23. Porter B, Zyl J, Lamy O. Maven – Welcome to Apache Maven [Internet]. Maven.apache.org. 2018 [cited 10 March 2018]. Available from: https://maven.apache.org/

24. SQL vs NoSQL - javatpoint [Internet]. www.javatpoint.com. 2018 [cited 10 March 2018]. Available from: https://www.javatpoint.com/sql-vs-nosql

25. Haerder, T.; Reuter, A. (1983). "Principles of transaction-oriented database recovery". *ACM Computing Surveys*. 15 (4): 287. doi:10.1145/289.291.

26. ACID properties [Internet]. Msdn.microsoft.com. 2018 [cited 10 March 2018]. Available from: https://msdn.microsoft.com/en-in/library/aa480356.aspx?f=255&MSPPError=-2147217396

27. Armando Fox and Eric Brewer, "Harvest, Yield and Scalable Tolerant Systems", *Proc. 7th Workshop Hot Topics in Operating Systems (HotOS 99)*, IEEE CS, 1999, pg. 174–178.

28. DB S. SQL vs NoSQL Database Differences Explained with few Example DB [Internet]. Thegeekstuff.com. 2018 [cited 10 March 2018]. Available from: http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/?utm_source=tuicool

29. Impetus/Kundera [Internet]. GitHub. 2018 [cited 10 March 2018]. Available from: https://github.com/impetus-opensource/Kundera/wiki/Polyglot-Persistence

30. Project A. Apache Tomcat® - Welcome! [Internet]. Tomcat.apache.org. 2018 [cited 10 March 2018]. Available from: http://tomcat.apache.org/

31. Basic Rules of Cassandra Data Modeling [Internet]. DataStax: always-on data platform | NoSQL | Apache Cassandra. 2018 [cited 10 March 2018]. Available from: https://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling

32. What is POJO? Webopedia Definition [Internet]. Webopedia.com. 2018 [cited 10 March 2018]. Available from: https://www.webopedia.com/TERM/P/POJO.html

33. Painter, Robert Richard. "Software Plans: Multi-Dimensional Fine-Grained Separation of Concerns". Penn State. CiteSeerX 10.1.1.110.9227.

34. "the urban canuk, eh: On Dependency Injection and Violating Encapsulation Concerns". *www.bryancook.net*. Retrieved 2015-07-18.
35. "The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 330". *jcp.org*. Retrieved 2015-07-18.
36. "The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 330". *jcp.org*. Retrieved 2015-07-18.
37. Design Patterns MVC Pattern [Internet]. www.tutorialspoint.com. 2018 [cited 10 March 2018]. Available from: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
38. Java Adapter Classes - javatpoint [Internet]. www.javatpoint.com. 2018 [cited 10 March 2018]. Available from: https://www.javatpoint.com/java-adapter-classes
39. FasterXML/jackson [Internet]. GitHub. 2018 [cited 10 March 2018]. Available from: https://github.com/FasterXML/jackson
40. FasterXML/jackson [Internet]. GitHub. 2018 [cited 10 March 2018]. Available from: https://github.com/FasterXML/jackson#jackson-project-home-github
41. FasterXML/jackson [Internet]. GitHub. 2018 [cited 10 March 2018]. Available from: https://github.com/FasterXML/jackson#providers-for-jax-rs
42. Difference between Association, Composition and Aggregation in Java, UML and Object Oriented Programming [Internet]. Javarevisited.blogspot.in. 2018 [cited 10 March 2018]. Available from: http://javarevisited.blogspot.in/2014/02/ifference-between-association-vs-composition-vs-aggregation.html

Appendices

Appendix 1. Building and running a Spring Boot application

A command to run a Spring Boot application on development machine using Apache Maven build-tool.

```
mvn spring-boot:run
```

To create the application package for deploying on the staging or production server, run following on the development machine.

```
mvn package
```

It will create a JAR file containing all the dependencies. This jar will also contain embedded tomcat server, so it can simply run as a standalone JAR. To run the application, transfer this created JAR to the target machine and run following command.

```
java -jar application.jar
```

After running this command, the Spring Boot application will be available

Appendix 2. Containerization of Spring Boot application

Dockerfile

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ADD target/gs-spring-boot-docker-0.1.0.jar app.jar
ENV JAVA_OPTS=""
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.secu-
rity.egd=file:/dev/./urandom -jar /app.jar" ]
```

POM file

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://ma-
ven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework</groupId>
    <artifactId>gs-spring-boot-docker</artifactId>
    <version>0.1.0</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.4.RELEASE</version>
    </parent>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
```

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>com.spotify</groupId>
                <artifactId>dockerfile-maven-plugin</artifactId>
                <version>1.3.4</version>
                <configuration>
                    <repository>${docker.image.prefix}/${project.ar-
                tifactId}</repository>
                </configuration>
            </plugin>
      </plugins>
    </build>
</project>
```

Appendix 3. Spring Project

The spring projects are following.

1. **Spring IO Platform:** This project provides a cohesive, versioned platform for building modern applications. It is a modular, enterprise grade distribution that delivers a curated set of dependencies.

2. **Spring Boot:** Takes an opinionated view of building Spring applications and gets the application up and running as quickly as possible.

3. **Spring Framework:** Spring framework builds the backbone of the applications by providing support mainly for dependency injection, inversion of control and data access etc.

4. **Spring Cloud Data Flow:** This is an orchestration service for creating micro-services utilizing modern runtimes.

5. **Spring Cloud:** This project provides a set of tools for common patterns in distributed systems. It is very useful for building and deploying microservices.

6. **Spring Data:** Spring Data provides a consistent approach to data access – relational, nonrelational, mapreduce, and beyond.

7. **Spring Integration:** This project supports the well-known Enterprise Integration Patterns via lightweight messaging and declarative adapters.

8. **Spring Batch:** Spring Batch project comes handy when very high-volume batch operations are run, or application components is developed.

9. **Spring Security:** Spring Security protects the applications with established and proven authentication and authorization protocols. It is used for securing all the Spring based applications.

10. **Spring HATEOAS:** Simplifies creating REST representations that follow the HATEOAS principle. Spring HATEOAS provides some APIs to ease creating REST representations that follow the HATEOAS principle when working with Spring and especially Spring MVC. The core problem it tries to address is link creation and representation assembly.

11. **Spring REST Docs:** It provides the support for creating extensive documentation of endpoints of RESTful services. It helps documenting both the handwritten and auto generated documentation.

12. **Spring Social:** This allows the new applications to connect with the third-party APIs such as Facebook, Twitter, LinkedIn, and more.

13. **Spring AMQP:** Spring AMQP brings core Spring concepts to the development of AMQP based messaging solutions. This project consists of two parts-- “spring-amqp” is the base abstraction, and “spring-rabbit” is the RabbitMQ implementation.

14. **Spring Mobile:** With its inbuilt device detection, it speeds up the mobile apps development and further with its progressive rendering, it helps them to look smooth.

15. **Spring for Android:** It brings Spring components dependency injection and other features in developing Android applications.

16. **Spring Web Flow:** The web applications, which require controlled navigation, can be easily developed utilizing Spring features by using Spring Web Flow project.

17. **Spring Web Services:** This project facilitates the development of contract-first SOAP web services.

18. **Spring LDAP:** It brings the Spring features in the development of applications using LDAP. It notably brings Spring's familiar template-based approach.

19. **Spring Session:** Spring Session provides an API and implementations for managing a user’s session information.

20. **Spring Shell:** Spring Shell provides a powerful foundation for building command line apps using a Spring based programming model.

21. **Spring XD:** XD simplifies the development of big data applications by addressing ingestion, analytics, batch jobs and data export. The project's goal is to simplify the development of big data applications.

22. **Spring Flo:** It helps creating charts and other visual items using simple web technologies, like HTML5 and JavaScript.

23. **Spring Kafka:** Provides Familiar Spring Abstractions for Apache Kafka[1]. The Spring for Apache Kafka (spring-kafka) project applies core Spring concepts to the development of Kafka based messaging solutions.

24. **Spring Statemachine:** A framework for application developers to use state machine concepts with Spring applications.

[1] Apache Kafka: https://kafka.apache.org