**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ**

1. Изучить фреймворк Play 2.12;
2. Изучить Scala: асинхронность, неявные параметры, коллекции, объект-компоновщик;
3. Изучить ORM Squeryl;
4. Изучить СУВБД Liquibase;
5. Изучить СУБД Postgresql;
6. Разработать онлайн базу знаний с хранением записей в БД, имеющая инициализацию системы, возможность добавления новых элементов и т.д. Должно быть реализовано с использованием описанных выше технологий.

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ**](#_1oahch7mwaf8) **……………………………………………………………………….4**

[**ОСНОВНАЯ ЧАСТЬ**](#_bd943zemny55) **…………………………………………………………...10**

[**ЗАКЛЮЧЕНИЕ**](#_iu3wqaefvfbl) **………………………………………………………………...15**

[**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**](#_4uoi80ypfgav) **……………………………..16**

[**ПРИЛОЖЕНИЕ А**](#_pxxd0flduojl) **……………………………………………………………...17**

# ВВЕДЕНИЕ

**Framework Play**

**Play -** каркас разработки с открытым кодом, написанный на Scala и Java, использует паттерн проектирования Model-View-Controller (MVC). Нацелен на повышение производительности, используя договорённости перед конфигурацией, горячую перегрузку кода и отображения ошибок в браузере. Разработку Play вдохновили такие каркасы как Ruby on Rails и Django.

Известные сайты, которые используют Play:

* BBC,
* Coursera,
* Gawker,
* Gilt,
* Гардиан,
* и др.

**Язык Scala**

Scala - это современный мультипарадигмальный язык программирования, разработанный для выражения общих концепций программирования в простой, удобной и типобезопасной манере. Элегантно объединяя особенности объектно-ориентированных и функциональных языков.

Scala объектно ориентированный

Scala - это чистый объектно-ориентированный язык в том смысле, что [каждое значение - это объект](https://docs.scala-lang.org/ru/tour/unified-types.html). Типы и поведение объектов описаны в [классах](https://docs.scala-lang.org/ru/tour/classes.html) и [трейтах](https://docs.scala-lang.org/ru/tour/traits.html)(характеристиках объектов). Классы расширяются за счет механизма наследования и гибкого [смешивания классов](https://docs.scala-lang.org/ru/tour/mixin-class-composition.html), который используется для замены множественного наследования.

Scala функциональный

Scala также является функциональным языком в том смысле, что [каждая функция - это значение](https://docs.scala-lang.org/ru/tour/unified-types.html). Scala предоставляет [легкий синтаксис](https://docs.scala-lang.org/ru/tour/basics.html) для определения анонимных функций, поддерживает [функции высшего порядка](https://docs.scala-lang.org/ru/tour/higher-order-functions.html), поддерживает [вложенные функции](https://docs.scala-lang.org/ru/tour/nested-functions.html), а также [каррирование](https://docs.scala-lang.org/ru/tour/multiple-parameter-lists.html). Scala имеют встроенную поддержку алгебраических типов данных, которые используются в большинстве функциональных языках программирования (эта поддержка базируется на механизме [сопоставления с примером](https://docs.scala-lang.org/ru/tour/pattern-matching.html), где в качестве примера выступают [классы образцы](https://docs.scala-lang.org/ru/tour/case-classes.html) ). [Объекты](https://docs.scala-lang.org/ru/tour/singleton-objects.html) предоставляют удобный способ группировки функций, не входящих в класс.

Вдобавок к этому, концепция сопоставления с примером логично переносится на [обработку XML-данных](https://github.com/scala/scala-xml/wiki/XML-Processing) используя в качестве примера [регулярные выражения](https://docs.scala-lang.org/ru/tour/regular-expression-patterns.html), при поддержке функционала [объектов экстракторов](https://docs.scala-lang.org/ru/tour/extractor-objects.html). Для еще большего удобства обработки данных представлена схема формирования запросов с использованием [for-выражения](https://docs.scala-lang.org/ru/tour/for-comprehensions.html). Такие возможности делают Scala идеальным решением для разработки приложений по типу веб-сервисов.

Scala статически типизированный

Scala оснащен выразительной системой типов, которая обеспечивает безопасное и гармоничное использование абстракций. В частности, система типов поддерживает:

1. обобщенные классы
2. вариантность типов
3. [верхние](https://docs.scala-lang.org/ru/tour/upper-type-bounds.html) и нижние границы типов
4. [внутренние классы](https://docs.scala-lang.org/ru/tour/inner-classes.html) и члены абстрактного типа, как часть объектов
5. составные типы
6. самоописываемые типы
7. [неявные параметры](https://docs.scala-lang.org/ru/tour/implicit-parameters.html) и неявные преобразования
8. полиморфные методы

Выведение типов означает, что разработчику не обязательно добавлять в код избыточную информацию о типах. Такой функционал обеспечивает основу для безопасного переиспользования абстракций и типобезопасного развития программного обеспечения.

Scala расширяемый

Зачастую разработка приложений для очень специфичных областей требует специфичных для этих областей языковых возможностей, либо отдельных специализированных языков программирования. Вместо этого Scala предлагает уникальные механизмы, для легкой модификации и расширения самого языка.

Во многих случаях такое можно сделать без использования средств мета-программирования, таких как макросы. Например: неявные классы позволяют добавлять новые методы к уже существующим, интерполяция строк позволяет добавлять обработку строк (расширяется разработчиком с помощью интерполяторов).

Scala совместимый

Scala полностью совместим с популярной средой Java Runtime Environment (JRE). Взаимодействие с основным объектно-ориентированным языком программирования Java происходит максимально гладко. Новые функции Java, такие как SAM, [лямбды](https://docs.scala-lang.org/ru/tour/higher-order-functions.html), [аннотации](https://docs.scala-lang.org/ru/tour/annotations.html) и [дженерики](https://docs.scala-lang.org/ru/tour/generic-classes.html), имеют прямые аналоги в Scala.

Те функции Scala, которые не имеют аналогов в Java, такие как [параметры по умолчанию](https://docs.scala-lang.org/ru/tour/default-parameter-values.html) и [именованные параметры](https://docs.scala-lang.org/ru/tour/named-arguments.html), компилируются как можно ближе к Java. Scala имеет такую же компиляционную модель (отдельная компиляция, динамическая загрузка классов), как у Java, что позволяет получить доступ к тысячам уже существующих высококачественных библиотек.

**Squeryl**

Squeryl **-** Scala ORM и DSL для взаимодействия с базами данных с минимальной детализацией и максимальной безопасностью типов.

**Liquibase**

Liquibase - открытая (open source) система для управления миграциями БД. Liquibase помогает организовать процесс внесения изменений в схему БД, каждая миграция будет содержать описание изменений, необходимых для перехода от старой ревизии к новой.

При использовании Liquibase изменения структуры базы данных будут храниться в отдельных файлах (changelogs), поддерживаются форматы XML, YAML, JSON или SQL, что очень удобно. Изменения можно хранить в одном или множестве файлов с последующем включением в основной файл. Второй вариант предпочтительнее, т.к. позволяет гибко организовать применение и хранение чейнджлогов.

В файлах-чейнджлогах изменения представляются в виде чейнджсетов (changesets). Чейнджсет может содержать одно или несколько изменений базы данных. Каждый changeset может быть уникально идентифицирован с помощью атрибутов id и author. Liquibase создает таблицу databasechangelog в базе данных для отслеживания примененных чейнджсетов. При каждом запуске Liquibase будет проверять хэш суммы чейнджлогов со значениями в таблице и изменения будут применяться, если еще не применялись или если используется параметр runAlways.

**Postgresql**

PostgreSQL — это популярная свободная объектно-реляционная система управления базами данных. PostgreSQL базируется на языке SQL и поддерживает многочисленные возможности.

СУБД отличается высокой надёжностью и хорошей производительностью. PostgreSQL поддерживает транзакции (ACID), репликация реализована встроенными механизмами. При этом система расширяемая — можно создавать свои типы данных и индексов, а также расширять поведение при помощи языков программирования.

# ОСНОВНАЯ ЧАСТЬ

Для начала были написаны файлы конфигурации и сборки, в которых подключаются используемые технологии, библиотеки и т.д.

Содержимое файла build.sbt.

name := "knowledge\_base2"

version := "1.0"

lazy val `knowledge\_base2` = (project in file(".")).enablePlugins(PlayScala)

resolvers += "scalaz-bintray" at "https://dl.bintray.com/scalaz/releases"

resolvers += "Akka Snapshot Repository" at "https://repo.akka.io/snapshots/"

scalaVersion := "2.12.2"

libraryDependencies ++= Seq( jdbc ,

ehcache ,

ws ,

specs2 % Test ,

guice,

"org.squeryl" %% "squeryl" % "0.9.5-7",

"org.postgresql" % "postgresql" % "9.3-1102-jdbc4",

evolutions

)

unmanagedResourceDirectories in Test <+= baseDirectory ( \_ /"target/web/public/test" )

Содержимое файла application.conf

play.http.secret.key = "changeme"

play.i18n {

langs = [ "en" ]

}

parsers.text.maxLength = 512k

play.filters {

enabled += filters.ExampleFilter

}

importJson {

path = "app/recursive.json"

}

db.default.driver="org.postgresql.Driver"

db.default.url="jdbc:postgresql://localhost:5432/knowledge\_base"

db.default.username="postgres"

db.default.password="postgres"

После этого создана база данных по следующей схеме:

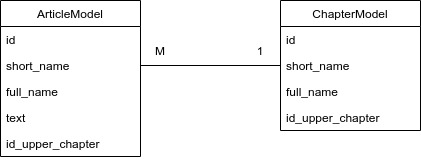


Рисунок 1 - схема базы данных

После этого были созданы модели для соответствующих таблиц и они были связаны с использованием ORM Squeryl.

Описание схемы с использованием библиотек Squeryl:

object AppSchema extends Schema {

implicit val chapters = table[Chapter]("chapter")

implicit val articles = table[Article]("article")

}

Затем были разработаны контроллеры, включающие в себя методы для создания статьи/главы, их обновления и удаления и другие.

Метод ArticleController.delete:

def delete(id: String) = Action { implicit request =>

articleService.deleteArticle(id)

Redirect(routes.MainController.index())

}

Метод ArticleService.deleteArticle:

def deleteArticle(*id*: String): Unit = {

transaction {

AppSchema.*articles*.lookup(*id*) match {

case *Some*(value) => AppSchema.*articles*.delete(*id*)

case None =>

}

}

}

Остальные методы реализованы похожим образом, их код приведен в Приложении А.

Когда вся серверная часть была готова, был разработан клиентский интерфейс. View-файлы будут также приведены в Приложении А.

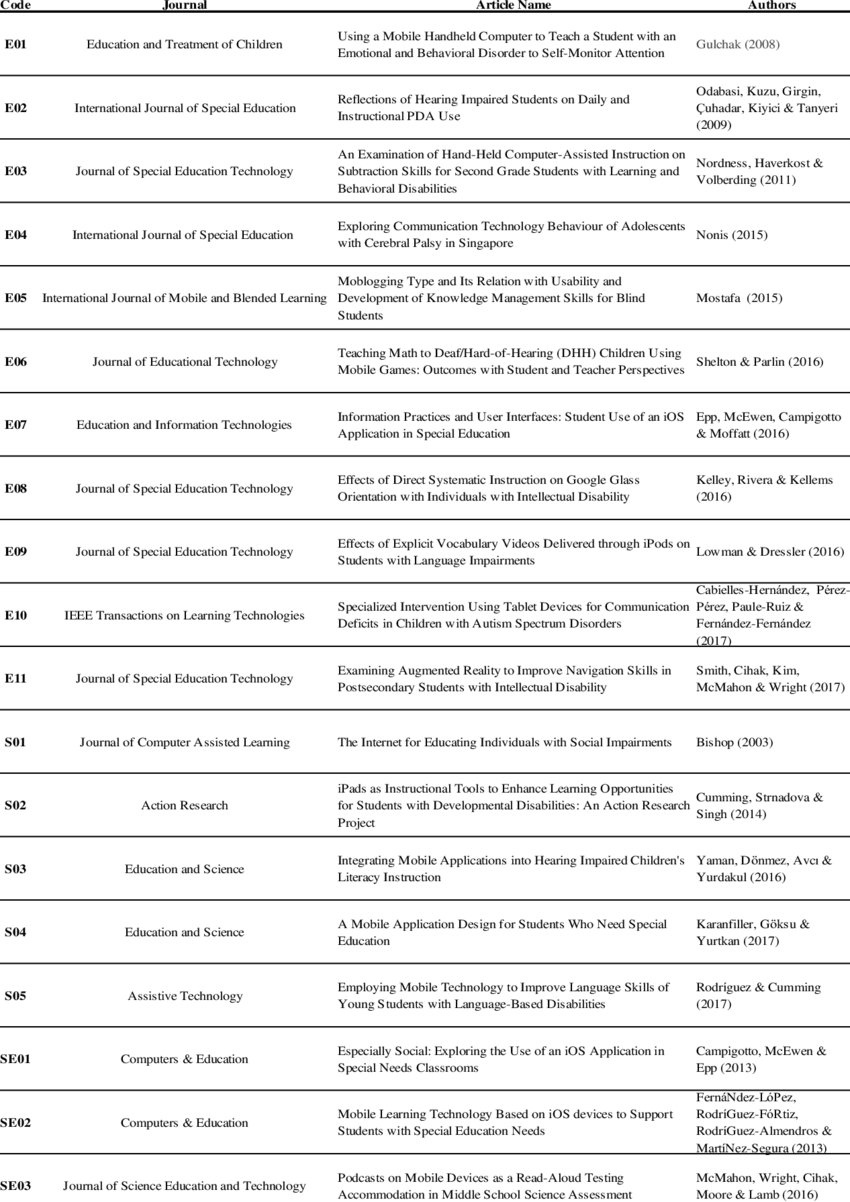


Рисунок 2 - список статей

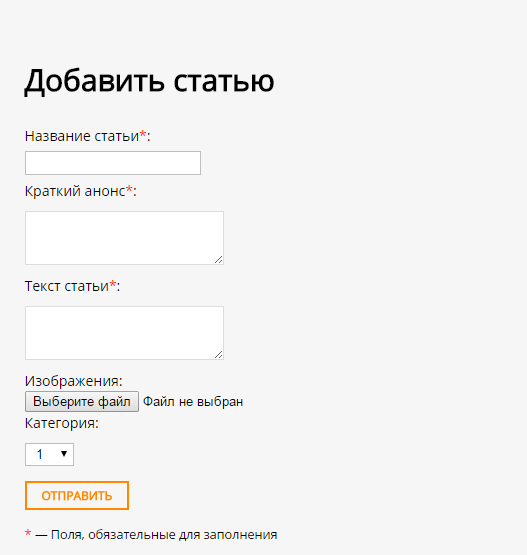
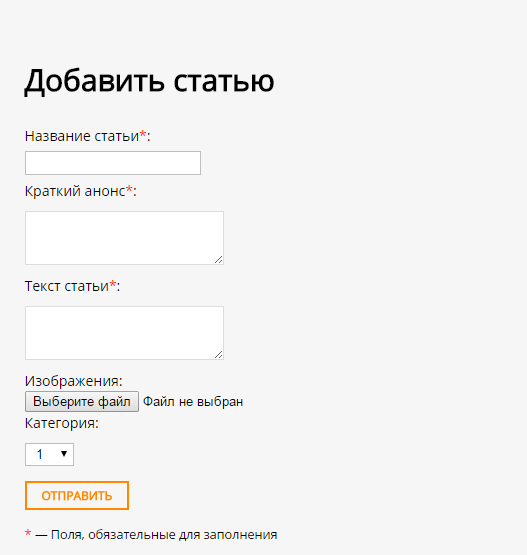


Рисунок 3 - форма добавления статьи

# 

# ЗАКЛЮЧЕНИЕ

В ходе производственной практики были выполнены цели и задачи индивидуального задания. Были изучены следующие технологии: Play framework, Scala, Squeryl ORM, Liquibase, Postgresql. И на основании перечисленных технологий была разработана онлайн база знаний для хранения статей.

Технологии, которые предлагалось изучить в ходе индивидуального задания, оказались легки в освоении и очень полезны в применении. Благодаря именно такому подбору технологий получилось быстро и качественно разработать приложение по заданию.

Знания, практики, умения, полученные за период практики, явились отличным стимулом для активной работы в освоении будущей специальности, позволили получить профессиональный опыт работы и сформировать общее представление о специфики деятельности ООО “Центр разработки”.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Play (фреймворк[)](https://ru.wikipedia.org/wiki/Play_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)) // [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Play\_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) (дата обращения: 04.05.2021).

2. Scala Documentation // [Электронный ресурс]. URL: https://docs.scala-lang.org (дата обращения: 04.05.2021).

3. Версионирование структуры БД с помощью Liquibase // [Электронный ресурс]. URL: http://easy-code.ru/lesson/database-versioning-liquibase (дата обращения: 25.04.2021).

4. Squeryl Documentation // [Электронный ресурс]. URL: https://www.squeryl.org (дата обращения: 04.05.2021).

5. PostgreSQL — объектно-реляционная система управления базами данных // [Электронный ресурс]. URL: https://web-creator.ru/articles/postgresql (дата обращения: 04.05.2021).

# 

# ПРИЛОЖЕНИЕ А

Файл ArticleService.scala.

package services

import models.\_

import org.squeryl.PrimitiveTypeMode.{\_\_thisDsl, select, transaction}

import org.squeryl.customtypes.CustomTypesMode.createConstantNodeOfScalarStringType

import org.squeryl.PrimitiveTypeMode.\_

import org.squeryl.\_

trait ArticleService {

def list(): List[Article]

def findArticle(id:String): Option[Article]

def deleteArticle(id: String):Unit

def updateArticle(id:String, shortName: String, fullName: String, text: String, upperChapterId: String):Unit

def createArticle(newArticle: Article):Unit

def deleteAllArticles(): Unit

}

class DbArticleService() extends ArticleService {

override def list(): List[Article] = {

transaction {

from(AppSchema.articles)(a => select(a)).toList

}

}

override def findArticle(id: String): Option[Article]= {

transaction {

AppSchema.articles.lookup(id)

}

}

override def deleteArticle(id: String): Unit = {

transaction {

AppSchema.articles.lookup(id) match {

case Some(value) => AppSchema.articles.delete(id)

case None =>

}

}

}

override def deleteAllArticles(): Unit = {

transaction {

AppSchema.articles.deleteWhere(a => 1 === 1)

}

}

override def updateArticle(id: String, shortName: String, fullName: String, text: String, upperChapterId: String): Unit = {

transaction {

AppSchema.articles.lookup(id) match {

case Some(value) => AppSchema.articles.delete(id)

AppSchema.articles.insert(Article(id, shortName, fullName, text, upperChapterId))

case None => println("not")

}

}

}

override def createArticle(newArticle: Article): Unit = {

transaction {

AppSchema.articles.insert(newArticle)

}

}

}

Файл ChapterService.scala.

package services

import models.\_

import org.squeryl.PrimitiveTypeMode.{\_\_thisDsl, select, transaction, where}

import org.squeryl.customtypes.CustomTypesMode.createConstantNodeOfScalarStringType

import org.squeryl.PrimitiveTypeMode.\_

trait ChapterService {

def list(): List[Chapter]

def findChapter(id:String): Option[Chapter]

def deleteChapter(id: String):Unit

def updateChapter(id:String, shortName: String, fullName: String, upperChapterId: String):Unit

def createChapter(newChapter: Chapter):Unit

def deleteAllChapters(): Unit

}

class DbChapterService() extends ChapterService {

override def list(): List[Chapter] = {

transaction {

from(AppSchema.chapters)(c => select(c)).toList

}

}

override def findChapter(id: String): Option[Chapter] = {

transaction {

AppSchema.chapters.lookup(id)

}

}

override def deleteChapter(id: String): Unit = {

transaction {

AppSchema.chapters.lookup(id) match {

case Some(value) => AppSchema.chapters.delete(id)

case None =>

}

}

}

override def deleteAllChapters(): Unit = {

transaction {

AppSchema.chapters.deleteWhere(c => 1 === 1)

}

}

override def updateChapter(id: String, shortName: String, fullName: String, upperChapterId: String): Unit = {

transaction {

AppSchema.chapters.lookup(id) match {

case Some(value) => AppSchema.chapters.delete(id)

AppSchema.chapters.insert(Chapter(id, shortName, fullName, upperChapterId))

case None => println("not")

}

}

}

override def createChapter(newChapter: Chapter): Unit = {

transaction {

AppSchema.chapters.insert(newChapter)

}

}

}

Файл ArticleController.scala.

package controllers

import javax.inject.Inject

import play.api.mvc.{AbstractController, ControllerComponents}

import play.api.data.Form

import play.api.data.Forms.{mapping, nonEmptyText}

import play.api.i18n.MessagesApi

import services.\_

import models.\_

class ArticleController @Inject() (cc: ControllerComponents)(messages: MessagesApi) extends AbstractController(cc) {

val articleService = new DbArticleService

val chapterService = new DbChapterService

val articleDTO = new ArticleDTO

val chapterDTO = new ChapterDTO

val articleForm: Form[ArticleWithoutId] = Form {

mapping(

"short\_name" -> nonEmptyText,

"full\_name" -> nonEmptyText,

"text" -> nonEmptyText

)(ArticleWithoutId.apply)(ArticleWithoutId.unapply)

}

def get(id: String) = Action { implicit request =>

articleService.findArticle(id) match {

case Some(value) => Ok(views.html.articlePage(value, articleForm)(request, msg = messagesApi.preferred(request)))

case None => NotFound

}

}

def delete(id: String) = Action { implicit request =>

articleService.deleteArticle(id)

Redirect(routes.MainController.index())

}

def create(id: String) = Action { implicit request =>

articleForm.bindFromRequest.fold(

formWithErrors => BadRequest("Not good"),

formData => {

articleService.createArticle(Article(

java.util.UUID.randomUUID().toString,

formData.shortName,

formData.fullName,

formData.text,

id))

Redirect(routes.MainController.index())

}

)

}

def update(id: String, upperChapterId: String) = Action { implicit request =>

articleForm.bindFromRequest.fold(

formWithErrors => BadRequest("Not good"),

formData => {

articleService.updateArticle(id,

formData.shortName,

formData.fullName,

formData.text,

upperChapterId)

Redirect(routes.MainController.index())

}

)

}

}

Файл ChapterController..scala.

package controllers

import javax.inject.Inject

import play.api.mvc.{AbstractController, ControllerComponents}

import models.\_

import play.api.data.Form

import play.api.data.Forms.{mapping, nonEmptyText}

import play.api.i18n.MessagesApi

import services.\_

class ChapterController @Inject() (cc: ControllerComponents)(messages: MessagesApi) extends AbstractController(cc) {

val chapterService: ChapterService = new DbChapterService()

val chapterDTO: ChapterDTO = new ChapterDTO

val chapterForm: Form[ChapterWithoutId] = Form {

mapping(

"short\_name" -> nonEmptyText,

"full\_name" -> nonEmptyText

)(ChapterWithoutId.apply)(ChapterWithoutId.unapply)

}

val articleForm: Form[ArticleWithoutId] = Form {

mapping(

"short\_name" -> nonEmptyText,

"full\_name" -> nonEmptyText,

"text" -> nonEmptyText

)(ArticleWithoutId.apply)(ArticleWithoutId.unapply)

}

def get(id: String) = Action { implicit request =>

chapterService.findChapter(id) match {

case Some(value) => Ok(views.html.chapterPage(value, chapterForm, articleForm)(request, msg = messagesApi.preferred(request)))

case None => NotFound

}

}

def delete(id: String) = Action { implicit request =>

chapterService.deleteChapter(id)

Redirect(routes.MainController.index())

}

def create(id: String) = Action { implicit request =>

chapterForm.bindFromRequest.fold(

formWithErrors => BadRequest("Not good"),

formData => {

chapterService.createChapter(Chapter(

java.util.UUID.randomUUID().toString,

formData.shortName,

formData.fullName,

id))

Redirect(routes.MainController.index())

}

)

}

def update(id: String, upperChapterId: String) = Action { implicit request =>

chapterForm.bindFromRequest.fold(

formWithErrors => BadRequest("Not good"),

formData => {

chapterService.updateChapter(id, formData.shortName, formData.fullName, upperChapterId)

Redirect(routes.MainController.index())

}

)

}

def list() = Action { implicit request =>

Ok(chapterDTO.getListChapterJson(chapterService.list()))

}

}

Файл MainController.scala.

package controllers

import akka.shapeless.HList

import akka.shapeless.HList.ListCompat

import com.typesafe.config.ConfigFactory

import javax.inject.Inject

import play.api.mvc.{AbstractController, ControllerComponents}

import play.api.data.Form

import play.api.data.Forms.{mapping, nonEmptyText}

import play.api.i18n.MessagesApi

import play.api.libs.json.Json

import services.\_

import models.\_

import scala.io.Source

class MainController @Inject() (cc: ControllerComponents)(messages: MessagesApi) extends AbstractController(cc) {

val articleService = new DbArticleService

val chapterService = new DbChapterService

val articleDTO = new ArticleDTO

val chapterDTO = new ChapterDTO

val articleForm: Form[ArticleWithoutId] = Form {

mapping(

"short\_name" -> nonEmptyText,

"full\_name" -> nonEmptyText,

"text" -> nonEmptyText

)(ArticleWithoutId.apply)(ArticleWithoutId.unapply)

}

val chapterForm: Form[ChapterWithoutId] = Form {

mapping(

"short\_name" -> nonEmptyText,

"full\_name" -> nonEmptyText

)(ChapterWithoutId.apply)(ChapterWithoutId.unapply)

}

def listChapter() = Action { implicit request =>

Ok(chapterDTO.getListChapterJson(chapterService.list()))

}

def listArticle() = Action { implicit request =>

val name = "Stefan"

var str=

"""

[

{

"shortName": """

str += """"""" + name + """""""

str += """},"""

str += """]"""

val r = Json.parse(str.dropRight(2) + "]").toString()

println(r)

println(Json.parse(r))

Ok(articleDTO.getListArticleJson(articleService.list()))

}

def list() = Action { implicit request =>

def recursiveWriter(upperChapterId: String): List[ChapterReader] ={

val chaptersList: List[ChapterReader] = List[ChapterReader]

chapterService.list().foreach( chapter => {

if (chapter.upperChapterId == upperChapterId) {

val articlesList : Seq[ArticleReader] => List[ArticleReader] = List[ArticleReader]

articlesList.->()

articleService.list().foreach(article =>

if (article.upperChapterId == chapter.id) {

ArticleReader(article.shortName, article.fullName, article.text) :: articlesList

}

)

ChapterReader(

chapter.shortName,

chapter.fullName,

Option(articlesList),

Option(recursiveWriter(chapter.id))

) :: chaptersList

}

})

chaptersList

}

println(chapterDTO.getRecursiveListChapterJson(recursiveWriter("0")))

Ok(chapterDTO.getRecursiveListChapterJson(recursiveWriter("0")))

}

def loadFromJson = Action { implicit request =>

val config = ConfigFactory.load("application.conf").getConfig("importJson")

val fileName = config.getString("path")

val source = Source.fromFile(fileName).getLines.mkString

val json = Json.parse(source)

val jsResult = json.validate[List[ChapterReader]](chapterDTO.ChaptersRead)

articleService.deleteAllArticles()

chapterService.deleteAllChapters()

def recursiveReader(upperChapterId: String, chapterReader: ChapterReader): Unit ={

val newChapterId = java.util.UUID.randomUUID().toString

chapterService.createChapter(Chapter(newChapterId, chapterReader.shortName, chapterReader.fullName, upperChapterId))

chapterReader.articles match {

case Some(value) => value.foreach(article =>

articleService.createArticle(Article(java.util.UUID.randomUUID().toString,

article.shortName,

article.fullName,

article.text,

newChapterId)))

case None =>

}

chapterReader.children match {

case Some(value) =>

value.foreach( children =>

recursiveReader(newChapterId, children))

case None =>

}

}

jsResult.get.foreach( chapter => recursiveReader("0", chapter))

Redirect(routes.MainController.index())

}

def index = Action { implicit request =>

Ok(views.html.index(articleForm, chapterForm)(request, msg = messagesApi.preferred(request)))

}

}

Файл articlePage.scala.html.

@import helper.\_

@(

article: Article,

articleForm: Form[ArticleWithoutId]

)(

implicit request: RequestHeader,

msg: Messages

)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>@article.shortName</title>

<link rel="stylesheet" media="screen" href="@routes.Assets.versioned("stylesheets/main.css")">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script src="@routes.Assets.versioned("javascripts/articlePage.js")" type="text/javascript"></script>

<script src="@routes.Assets.versioned("javascripts/modal.js")" type="text/javascript"></script>

</head>

<body>

<a href="/">

<div class="backBtn">

Back

</div>

</a>

<h2>@article.fullName</h2>

<p>@article.text</p>

<button id="articleDeleteBtn">Delete article</button>

<button id="articleUpdateBtn">Update article</button>

<div id="myModal" class='modal'>

<div id="articleDelete" class='modal-content'>

<span class='close-modal'>×</span>

@form(action = routes.ArticleController.delete(article.id)) {

<input class="button" type="submit" value="Delete article">

}

</div>

<div id="articleUpdate" class='modal-content'>

<span class='close-modal'>×</span>

<form method="get" action="@routes.ArticleController.update(article.id, article.upperChapterId)">

<label for="short\_name">Short Name</label>

<input type="text" name="short\_name" value="@article.shortName">

<label for="full\_name">Full Name</label>

<input type="text" name="full\_name" value="@article.fullName">

<label for="text">Text</label>

<textarea type="textarea" name="text">@article.text</textarea>

<input class="button" type="submit" value="Update article">

</form>

</div>

</div>

</body>

</html>

Файл index.scala.html.

@\*

\* This template is called from the `index` template. This template

\* handles the rendering of the page header and body tags. It takes

\* two arguments, a `String` for the title of the page and an `Html`

\* object to insert into the body of the page.

\*@

@import helper.\_

@(articleForm: Form[models.ArticleWithoutId],

chapterForm: Form[models.ChapterWithoutId]

)(

implicit request: RequestHeader,

msg: Messages

)

<!DOCTYPE html>

<html lang="en">

<head>

<title>Sandbox</title>

<link rel="stylesheet" media="screen" href="@routes.Assets.versioned("stylesheets/main.css")">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script src="@routes.Assets.versioned("javascripts/modal.js")" type="text/javascript"></script>

<script src="@routes.Assets.versioned("javascripts/addChaptersAndArticlesIndex.js")" type="text/javascript"></script>

</head>

<body>

<ol id = "contents" type="1">

</ol>

@form(action = CSRF(routes.MainController.loadFromJson)) {

<input class="button" type="submit" value="Add db">

}

<button id="chapterCreateBtn">Create chapter</button>

<div id="myModal" class='modal'>

<div id="chapterCreate" class='modal-content'>

<span class='close-modal'>×</span>

@form(action = CSRF(routes.ChapterController.create("0"))) {

@inputText(chapterForm("short\_name"))

@inputText(chapterForm("full\_name"))

<input class="button" type="submit" value="Add chapter">

}

</div>

</div>

</body>

</html>

Файл routes.

# Routes

# This file defines all application routes (Higher priority routes first)

# ~~~~

# An example controller showing a sample home page

GET / controllers.MainController.index

GET /api/list controllers.MainController.list

GET /api/listChapter controllers.MainController.listChapter

GET /api/listArticle controllers.MainController.listArticle

GET /api/loadFromJson controllers.MainController.loadFromJson

GET /api/article/get/:id controllers.ArticleController.get(id: String)

GET /api/article/delete/:id controllers.ArticleController.delete(id: String)

POST /api/article/create/:id controllers.ArticleController.create(id: String)

GET /api/article/update/:id/:upperChapterId controllers.ArticleController.update(id: String, upperChapterId: String)

GET /api/chapter/get/:id controllers.ChapterController.get(id: String)

GET /api/chapter/delete/:id controllers.ChapterController.delete(id: String)

POST /api/chapter/create/:id controllers.ChapterController.create(id: String)

GET /api/chapter/update/:id/:upperChapterId controllers.ChapterController.update(id: String, upperChapterId: String)

# Map static resources from the /public folder to the /assets URL path

GET /assets/\*file controllers.Assets.versioned(path="/public", file: Asset)