```
#!/bin/sh

#
# Copyright © 2015-2021 the original authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

##############################################################################
#
#   Gradle start up script for POSIX generated by Gradle.
#
#   Important for running:
#
#   (1) You need a POSIX-compliant shell to run this script. If your /bin/sh is
#       noncompliant, but you have some other compliant shell such as ksh or
#       bash, then to run this script, type that shell name before the whole
#       command line, like:
#
#           ksh Gradle
#
#       Busybox and similar reduced shells will NOT work, because this script
#       requires all of these POSIX shell features:
#         * functions;
#         * expansions «$var», «${var}», «${var:-default}», «${var+SET}»,
#           «${var#prefix}», «${var%suffix}», and «$( cmd )»;
#         * compound commands having a testable exit status, especially «case»;
#         * various built-in commands including «command», «set», and «ulimit».
#
#   Important for patching:
#
#   (2) This script targets any POSIX shell, so it avoids extensions provided
#       by Bash, Ksh, etc; in particular arrays are avoided.
#
#       The "traditional" practice of packing multiple parameters into a
#       space-separated string is a well documented source of bugs and security
#       problems, so this is (mostly) avoided, by progressively accumulating
#       options in "$@", and eventually passing that to Java.
#
#       Where the inherited environment variables (DEFAULT_JVM_OPTS, JAVA_OPTS,
#       and GRADLE_OPTS) rely on word-splitting, this is performed explicitly;
#       see the in-line comments for details.
#
#       There are tweaks for specific operating systems such as AIX, CygWin,
#       Darwin, MinGW, and NonStop.
#
#   (3) This script is generated from the Groovy template
#       https://github.com/gradle/gradle/blob/master/subprojects/plugins/src/main/resources/org/gradle/a
#       within the Gradle project.
#
#       You can find Gradle at https://github.com/gradle/gradle/.
#
```

```
##############################################################################

# Attempt to set APP_HOME

# Resolve links: $0 may be a link
app_path=$0

# Need this for daisy-chained symlinks.
while
    APP_HOME=${app_path%"${app_path##*/}"}  # leaves a trailing /; empty if no leading path
    [ -h "$app_path" ]
do
    ls=$( ls -ld "$app_path" )
    link=${ls#*' -> '}
    case $link in             #(
      /*)   app_path=$link ;; #(
      *)    app_path=$APP_HOME$link ;;
    esac
done

APP_HOME=$( cd "${APP_HOME:-./}" && pwd -P ) || exit

APP_NAME="Gradle"
APP_BASE_NAME=${0##*/}

# Add default JVM options here. You can also use JAVA_OPTS and GRADLE_OPTS to pass JVM options to this s
DEFAULT_JVM_OPTS='"-Xmx64m" "-Xms64m"'

# Use the maximum available, or set MAX_FD != -1 to use that value.
MAX_FD=maximum

warn () {
    echo "$*"
} >&2

die () {
    echo
    echo "$*"
    echo
    exit 1
} >&2

# OS specific support (must be 'true' or 'false').
cygwin=false
msys=false
darwin=false
nonstop=false
case "$( uname )" in                #(
  CYGWIN* )         cygwin=true  ;; #(
  Darwin* )         darwin=true  ;; #(
  MSYS* | MINGW* )  msys=true    ;; #(
  NONSTOP* )        nonstop=true ;;
esac

CLASSPATH=$APP_HOME/gradle/wrapper/gradle-wrapper.jar


# Determine the Java command to use to start the JVM.
if [ -n "$JAVA_HOME" ] ; then
    if [ -x "$JAVA_HOME/jre/sh/java" ] ; then
        # IBM's JDK on AIX uses strange locations for the executables
        JAVACMD=$JAVA_HOME/jre/sh/java
```

```
    else
        JAVACMD=$JAVA_HOME/bin/java
    fi
    if [ ! -x "$JAVACMD" ] ; then
        die "ERROR: JAVA_HOME is set to an invalid directory: $JAVA_HOME

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
    fi
else
    JAVACMD=java
    which java >/dev/null 2>&1 || die "ERROR: JAVA_HOME is not set and no 'java' command could be found

Please set the JAVA_HOME variable in your environment to match the
location of your Java installation."
fi

# Increase the maximum file descriptors if we can.
if ! "$cygwin" && ! "$darwin" && ! "$nonstop" ; then
    case $MAX_FD in #(
      max*)
        MAX_FD=$( ulimit -H -n ) ||
            warn "Could not query maximum file descriptor limit"
    esac
    case $MAX_FD in  #(
      '' | soft) :;; #(
      *)
        ulimit -n "$MAX_FD" ||
            warn "Could not set maximum file descriptor limit to $MAX_FD"
    esac
fi

# Collect all arguments for the java command, stacking in reverse order:
#   * args from the command line
#   * the main class name
#   * -classpath
#   * -D...appname settings
#   * --module-path (only if needed)
#   * DEFAULT_JVM_OPTS, JAVA_OPTS, and GRADLE_OPTS environment variables.

# For Cygwin or MSYS, switch paths to Windows format before running java
if "$cygwin" || "$msys" ; then
    APP_HOME=$( cygpath --path --mixed "$APP_HOME" )
    CLASSPATH=$( cygpath --path --mixed "$CLASSPATH" )

    JAVACMD=$( cygpath --unix "$JAVACMD" )

    # Now convert the arguments - kludge to limit ourselves to /bin/sh
    for arg do
        if
            case $arg in                                #(
              -*)   false ;;                            # don't mess with options #(
              /?*)  t=${arg#/} t=/${t%%/*}              # looks like a POSIX filepath
                    [ -e "$t" ] ;;                      #(
              *)    false ;;
            esac
        then
            arg=$( cygpath --path --ignore --mixed "$arg" )
        fi
        # Roll the args list around exactly as many times as the number of
        # args, so each arg winds up back in the position where it started, but
        # possibly modified.
```

```
        #
        # NB: a `for` loop captures its iteration list before it begins, so
        # changing the positional parameters here affects neither the number of
        # iterations, nor the values presented in `arg`.
        shift                   # remove old arg
        set -- "$@" "$arg"      # push replacement arg
    done
fi

# Collect all arguments for the java command;
#   * $DEFAULT_JVM_OPTS, $JAVA_OPTS, and $GRADLE_OPTS can contain fragments of
#     shell script including quotes and variable substitutions, so put them in
#     double quotes to make sure that they get re-expanded; and
#   * put everything else in single quotes, so that it's not re-expanded.

set -- \
        "-Dorg.gradle.appname=$APP_BASE_NAME" \
        -classpath "$CLASSPATH" \
        org.gradle.wrapper.GradleWrapperMain \
        "$@"

# Use "xargs" to parse quoted args.
#
# With -n1 it outputs one arg per line, with the quotes and backslashes removed.
#
# In Bash we could simply go:
#
#   readarray ARGS < <( xargs -n1 <<<"$var" ) &&
#   set -- "${ARGS[@]}" "$@"
#
# but POSIX shell has neither arrays nor command substitution, so instead we
# post-process each arg (as a line of input to sed) to backslash-escape any
# character that might be a shell metacharacter, then use eval to reverse
# that process (while maintaining the separation between arguments), and wrap
# the whole thing up as a single "set" statement.
#
# This will of course break if any of these variables contains a newline or
# an unmatched quote.
#

eval "set -- $(
        printf '%s\n' "$DEFAULT_JVM_OPTS $JAVA_OPTS $GRADLE_OPTS" |
        xargs -n1 |
        sed ' s~[^-[:alnum:]+,./:=@_]~\\&~g; ' |
        tr '\n' ' '
    )" '"$@"'

exec "$JAVACMD" "$@"
```

```
# formatted as key=value
PORT=8080
DB_USERNAME="onelenyk"
DB_PASSWORD="ZvENwF0TzcsirUIC"
DB_CONNECTION="@cluster0.gmqkpcg.mongodb.net/?retryWrites=true&w=majority"
```

```
// Define variables for project configurations
val projectGroup = "dev.onelenyk"
val appId = "crudfather"
val mainAppClassName = "$projectGroup.$appId.ApplicationKt"

val ktorVersion: String by project
val kotlinVersion: String by project
val logbackVersion: String by project

val projectVersion: String by project

plugins {
    application // Apply the application plugin to add support for building a CLI application in Java.
    kotlin("jvm") version "2.0.0"
    id("io.ktor.plugin") version "2.3.3"

    kotlin("plugin.serialization") version "2.0.0-RC1"
    id("org.jlleitschuh.gradle.ktlint") version "12.1.0"
    id("org.jetbrains.dokka") version "1.9.10"
}

group = projectGroup
version = projectVersion

repositories {
    mavenCentral()
    maven(url = "https://jitpack.io")
}

dependencies {
    // Align versions of all Kotlin components
    implementation(platform("org.jetbrains.kotlin:kotlin-bom"))

    implementation(kotlin("stdlib"))

    // Ktor dependencies
    implementation("io.ktor:ktor-server-core:$ktorVersion")
    implementation("io.ktor:ktor-server-netty:$ktorVersion")
    implementation("io.ktor:ktor-client-cio:$ktorVersion")
    implementation("io.ktor:ktor-server-html-builder:$ktorVersion")
    implementation("io.ktor:ktor-server-host-common:$ktorVersion")
    implementation("io.ktor:ktor-server-call-logging:$ktorVersion")
    implementation("io.ktor:ktor-serialization-kotlinx-json:$ktorVersion")
    implementation("io.ktor:ktor-server-content-negotiation:$ktorVersion")
    implementation("io.ktor:ktor-server-auth:$ktorVersion")
    implementation("io.ktor:ktor-server-auth-jwt:$ktorVersion")

    implementation("org.mongodb:mongodb-driver-kotlin-coroutine:4.10.1")

    implementation("io.ktor:ktor-server-request-validation:$ktorVersion")

    // Koin
    implementation("io.insert-koin:koin-core:3.4.3")
    implementation("io.insert-koin:koin-ktor:3.4.3")
    implementation("io.insert-koin:koin-logger-slf4j:3.4.3")

    implementation("io.github.cdimascio:dotenv-kotlin:6.4.1")
    // Logging
    implementation("ch.qos.logback:logback-classic:$logbackVersion")
    // Serialization
    implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.7.1")
    // Test dependencies
```

```
    testImplementation("io.ktor:ktor-server-test-host:$ktorVersion")
    testImplementation("org.jetbrains.kotlin:kotlin-test")
    testImplementation("org.jetbrains.kotlin:kotlin-test-junit")
}

application {
    mainClass.set(mainAppClassName)
}

ktor {
    fatJar {
        archiveFileName.set("fat.jar")
    }
}

tasks.shadowJar {
    manifest {
        attributes["Implementation-Title"] = project.name
        attributes["Implementation-Version"] = project.version
        attributes["Main-Class"] = mainAppClassName
    }
}

// dokka

tasks.withType<org.jetbrains.dokka.gradle.DokkaTask>().configureEach {
    outputDirectory.set(buildDir.resolve("dokka"))
}
// Custom task to copy Dokka output to resources
tasks.register<Copy>("copyDokkaToResources") {
    dependsOn(tasks.dokkaHtml)
    from(tasks.dokkaHtml.flatMap { it.outputDirectory })
    into("src/main/resources/dokka")
}

// Preparation task that generates Dokka documentation and copies it to resources
tasks.register("prepare") {
    dependsOn(tasks.dokkaHtml)
    dependsOn("copyDokkaToResources")
}

tasks.getByName("build") {
    dependsOn("prepare")
}

// Main run task that includes preparation
tasks.getByName("run") {
    dependsOn("prepare")
}

// Ensure Dokka runs during build and copies to resources
tasks.named("processResources") {
    dependsOn("copyDokkaToResources")
}

tasks {
    create("stage").dependsOn("installDist")
}

tasks.register<Jar>("dokkaHtmlJar") {
    dependsOn(tasks.dokkaHtml)
```

```
    from(tasks.dokkaHtml.flatMap { it.outputDirectory })
    archiveClassifier.set("javadoc")
}

tasks.register<Jar>("dokkaJavadocJar") {
    dependsOn(tasks.dokkaJavadoc)
    from(tasks.dokkaJavadoc.flatMap { it.outputDirectory })
    archiveClassifier.set("javadoc")
}
```

```
web: ./build/install/crudfather/bin/crudfather
```

```
rootProject.name = "crudfather"
```

```
.gradle
build/
src/main/resources/dokka
!gradle/wrapper/gradle-wrapper.jar
!**/src/main/**/build/
!**/src/test/**/build/

### IntelliJ IDEA ###
.idea/modules.xml
.idea/jarRepositories.xml
.idea/compiler.xml
.idea/libraries/
*.iws
*.iml
*.ipr
out/
!**/src/main/**/out/
!**/src/test/**/out/

### Eclipse ###
.apt_generated
.classpath
.factorypath
.project
.settings
.springBeans
.sts4-cache
bin/
!**/src/main/**/bin/
!**/src/test/**/bin/

### NetBeans ###
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/

### VS Code ###
.vscode/

### Mac OS ###
.DS_Store
```

```
#Tue Jul 16 16:53:20 EEST 2024
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-8.2-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

```
root = true

[*.{kt,kts}]
ktlint_standard_no-wildcard-imports = disabled
```

```
kotlin.code.style=official
projectVersion=0.0.6
ktorVersion=2.3.3
kotlinVersion=1.9.10
logbackVersion=1.2.11
```

```
import dev.onelenyk.crudfather.app.Server
import io.ktor.client.request.*
import io.ktor.client.statement.*
import io.ktor.http.*
import io.ktor.server.application.*
import io.ktor.server.testing.*
import kotlin.test.*

class ApplicationTest {
    @Test
    fun testRoot() {
        val server = Server()

        testApplication {
            application {
                server.module(this)
            }
            client.get("/live").apply {
                assertEquals(HttpStatusCode.OK, status)
            }
        }
    }
}
```

```kotlin
package dev.onelenyk.crudfather.repository

import com.mongodb.client.model.Filters
import com.mongodb.client.model.UpdateOptions
import com.mongodb.kotlin.client.coroutine.MongoCollection
import dev.onelenyk.crudfather.data.scheme.ModelScheme
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.flow.firstOrNull
import kotlinx.coroutines.flow.toList
import org.bson.conversions.Bson
import java.util.*

class ModelSchemeRepository(private val collection: MongoCollection<ModelScheme>) {
    suspend fun create(model: ModelScheme): ModelScheme {
        val insertOne = collection.insertOne(model)
        val filter = Filters.eq("_id", insertOne.insertedId)
        return collection.find(filter).first()
    }

    suspend fun getById(id: UUID): ModelScheme? {
        val filter = Filters.eq("_id", id)
        return collection.find(filter).firstOrNull()
    }

    suspend fun delete(id: UUID): Boolean {
        val filter = Filters.eq("_id", id)
        val result = collection.deleteOne(filter)
        return result.deletedCount > 0
    }

    suspend fun modelExistsByName(modelName: String): Boolean {
        val filter = Filters.eq("definition.modelName", modelName)
        return collection.find(filter).firstOrNull() != null
    }

    suspend fun getModelSchemeByDefinitionName(modelName: String): ModelScheme? {
        val filter = Filters.eq("definition.modelName", modelName)
        return collection.find(filter).firstOrNull()
    }

    suspend fun readAll(): List<ModelScheme> {
        return collection.find().toList()
    }

    suspend fun update(
        id: UUID,
        document: Bson,
    ): ModelScheme? {
        val filter = Filters.eq("_id", id)
        collection.updateOne(filter, document, UpdateOptions().upsert(true))
        return getById(id)
    }
}
```

```kotlin
package dev.onelenyk.crudfather.repository

import com.mongodb.client.model.Filters
import com.mongodb.client.model.FindOneAndUpdateOptions
import com.mongodb.client.model.ReturnDocument
import com.mongodb.kotlin.client.coroutine.MongoCollection
import com.mongodb.kotlin.client.coroutine.MongoDatabase
import kotlinx.coroutines.flow.firstOrNull
import kotlinx.coroutines.flow.toList
import kotlinx.serialization.json.JsonObject
import org.bson.Document

class DynamicRepository(private val database: MongoDatabase) {
    private fun getCollection(collectionName: String): MongoCollection<Document> {
        return database.getCollection(collectionName)
    }

    suspend fun createDocument(
        collectionName: String,
        json: JsonObject,
    ): Document {
        val collection = getCollection(collectionName)
        val document = Document.parse(json.toString())
        collection.insertOne(document)
        return document
    }

    suspend fun getAllDocuments(collectionName: String): List<Document> {
        val collection = getCollection(collectionName)
        return collection.find().toList()
    }

    suspend fun getDocumentById(
        collectionName: String,
        id: String,
    ): Document? {
        val collection = getCollection(collectionName)
        return collection.find(Document("_id", id)).firstOrNull()
    }

    suspend fun updateDocument(
        collectionName: String,
        id: String,
        json: JsonObject,
    ): Document? {
        val collection = getCollection(collectionName)
        val options =
            FindOneAndUpdateOptions()
                .upsert(true)
                .returnDocument(ReturnDocument.AFTER)

        // Parse the JSON string into a Document
        val updateDoc = Document("\$set", Document.parse(json.toString()))

        // Perform the update operation
        collection.findOneAndUpdate(
            filter = Filters.eq("_id", id),
            update = updateDoc,
            options = options,
        )

        return getDocumentById(collectionName, id)
```

```
    }

    suspend fun deleteDocument(
        collectionName: String,
        id: String,
    ): Boolean {
        val collection = getCollection(collectionName)
        val result = collection.deleteOne(Document("_id", id))
        return result.deletedCount > 0
    }
}
```

```
package dev.onelenyk.crudfather.di

import com.mongodb.kotlin.client.coroutine.MongoDatabase
import dev.onelenyk.crudfather.app.routing.ServerRouting
import dev.onelenyk.crudfather.data.scheme.ModelScheme
import dev.onelenyk.crudfather.db.MongoDBManager
import dev.onelenyk.crudfather.repository.DynamicRepository
import dev.onelenyk.crudfather.repository.ModelSchemeRepository
import io.github.cdimascio.dotenv.Dotenv
import io.github.cdimascio.dotenv.dotenv
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.DelicateCoroutinesApi
import kotlinx.coroutines.GlobalScope
import org.koin.dsl.module

val koinModule =
    module {
        single { dotenv() }
        single { modelSchemeRepository(get()) }
        single { database(get()) }
        single { DynamicRepository(get()) }
        single { ServerRouting(get(), get()) }
        single { provideCoroutineScope() }
        single { provideDbCredentials(get()) }
        single { MongoDBManager(get()) }
    }

@OptIn(DelicateCoroutinesApi::class)
private fun provideCoroutineScope(): CoroutineScope {
    return GlobalScope
}

fun modelSchemeRepository(mongoDBManager: MongoDBManager): ModelSchemeRepository {
    val collection = mongoDBManager.getCollection("models", ModelScheme::class.java)
    return ModelSchemeRepository(collection)
}

fun database(mongoDBManager: MongoDBManager): MongoDatabase {
    val database = mongoDBManager.getDatabase()
    return database
}

fun provideDbCredentials(dotenv: Dotenv): DbCredentials {
    val username = dotenv["DB_USERNAME"]
    val pass = dotenv["DB_PASSWORD"]
    val connection = dotenv["DB_CONNECTION"]
    return DbCredentials(username, pass, connection)
}

fun provideServerPort(dotenv: Dotenv): Int {
    val port = dotenv["PORT"].toIntOrNull()
    return port ?: 8080
}

data class DbCredentials(val username: String, val password: String, val connection: String)
```

```
package dev.onelenyk.crudfather.app.routing

import dev.onelenyk.crudfather.repository.DynamicRepository
import dev.onelenyk.crudfather.repository.ModelSchemeRepository
import io.ktor.http.*
import io.ktor.server.application.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import io.ktor.util.pipeline.*
import kotlinx.coroutines.runBlocking
import org.bson.Document
import org.bson.types.ObjectId

class ServerRouting(
    private val repository: ModelSchemeRepository,
    private val dynamicRepository: DynamicRepository,
) {
    private val modelSchemeRoutes = ModelSchemeRoutes(repository)
    private val dynamicRoutes = DynamicRoutes(repository, dynamicRepository)

    fun registerRoutes(routing: Routing) {
        modelSchemeRoutes.registerRoutes(routing)
        dynamicRoutes.registerRoutes(routing)
        routing.routesInfo()
    }
}

suspend fun PipelineContext<Unit, ApplicationCall>.checkModelExists(
    repository: ModelSchemeRepository,
    modelName: String,
): Boolean {
    if (modelName.isBlank()) {
        call.respond(HttpStatusCode.BadRequest, "Model name is required")
        return false
    }

    val modelExists = runBlocking { repository.modelExistsByName(modelName) }
    if (!modelExists) {
        call.respond(HttpStatusCode.NotFound, "Model $modelName not found")
        return false
    }

    return true
}

suspend fun PipelineContext<Unit, ApplicationCall>.receiveJsonDocumentWithId(): Document {
    val json = call.receiveText()
    val document = Document.parse(json)

    // Check if the _id field is present, if not, generate one
    if (!document.containsKey("_id")) {
        document["_id"] = ObjectId()
    } else {
        // Ensure the _id field follows the required rules
        val id = document["_id"]
        if (id !is ObjectId && id !is String) {
            throw IllegalArgumentException("Invalid _id field type")
        }
        // Convert _id to ObjectId if it is a valid string representation of ObjectId
        if (id is String && ObjectId.isValid(id)) {
            document["_id"] = ObjectId(id)
```

```
        }
    }

    return document
}
```

```
package dev.onelenyk.crudfather.app.routing

import io.ktor.http.*
import io.ktor.server.application.*
import io.ktor.server.http.content.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import io.ktor.util.pipeline.*

fun Routing.routesInfo() {
    get("/routes") {
        val routesList = this@routesInfo.getAllRoutes()
        call.respond(routesList)
    }
    staticResources("/doc", "dokka")
    staticResources("/", "dokka")
    get("/live") {
        call.respond(HttpStatusCode.OK)
    }
}

private fun Route.getAllRoutes(): List<String> {
    val routesList = mutableListOf<String>()
    this.children.forEach { route ->
        route.toRouteList(routesList)
    }
    return routesList
}

private fun Route.toRouteList(
    routesList: MutableList<String>,
    parentPath: String = "",
) {
    val currentPath =
        if (this.selector is RootRouteSelector) {
            parentPath
        } else {
            val segment = this.selector.toString()
            if (parentPath.isEmpty()) segment else "$parentPath/$segment"
        }

    if (this.children.isEmpty()) {
        routesList.add(currentPath)
    } else {
        this.children.forEach { child ->
            child.toRouteList(routesList, currentPath)
        }
    }
}
```

```
package dev.onelenyk.crudfather.app.routing

import com.mongodb.client.model.Updates
import dev.onelenyk.crudfather.data.scheme.DynamicModelManager.generateModelDefinition
import dev.onelenyk.crudfather.data.scheme.ModelScheme
import dev.onelenyk.crudfather.repository.ModelSchemeRepository
import io.ktor.http.*
import io.ktor.server.application.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import java.util.*

class ModelSchemeRoutes(private val repository: ModelSchemeRepository) {
    fun registerRoutes(routing: Routing) {
        routing.route("/models") {
            post {
                val modelName = call.parameters["name"].orEmpty()

                if (modelName.isBlank()) {
                    call.respond(HttpStatusCode.BadRequest, "Model name is required")
                    return@post
                }

                try {
                    val json = call.receiveText()
                    val modelDefinition = generateModelDefinition(modelName, json)
                    val modelScheme = ModelScheme(definition = modelDefinition)
                    repository.create(modelScheme)
                    call.respond(HttpStatusCode.Created, modelScheme)
                } catch (e: Exception) {
                    call.respond(HttpStatusCode.BadRequest, "Invalid JSON: ${e.message}")
                }
            }

            get {
                try {
                    val list = repository.readAll()
                    call.respond(HttpStatusCode.OK, list)
                } catch (e: Exception) {
                    call.respond(HttpStatusCode.BadRequest, "Error fetching models: ${e.message}")
                }
            }

            get("/{id}") {
                val id = call.parameters["id"]
                if (id == null) {
                    call.respond(HttpStatusCode.BadRequest, "Invalid model ID")
                    return@get
                }

                try {
                    val model = repository.getById(UUID.fromString(id))
                    if (model != null) {
                        call.respond(HttpStatusCode.OK, model)
                    } else {
                        call.respond(HttpStatusCode.NotFound, "Model not found")
                    }
                } catch (e: Exception) {
                    call.respond(HttpStatusCode.BadRequest, "Error fetching model: ${e.message}")
                }
            }
```

```
            put("/{id}") {
                val id = call.parameters["id"]
                if (id == null) {
                    call.respond(HttpStatusCode.BadRequest, "Invalid model ID")
                    return@put
                }

                try {
                    val json = call.receiveText()

                    val model = repository.getById(UUID.fromString(id))

                    if (model == null) {
                        call.respond(HttpStatusCode.NotFound, "Model not found")
                        return@put
                    }

                    val modelDefinition = generateModelDefinition(model.definition.modelName, json)

                    val updates =
                        Updates.combine(
                            Updates.set(ModelScheme::definition.name, modelDefinition),
                        )

                    val updatedModel = repository.update(UUID.fromString(id), updates)
                    if (updatedModel != null) {
                        call.respond(HttpStatusCode.OK, updatedModel)
                    } else {
                        call.respond(HttpStatusCode.NotFound, "Model not found")
                    }
                } catch (e: Exception) {
                    call.respond(HttpStatusCode.BadRequest, "Error updating model: ${e.message}")
                }
            }

            delete("/{id}") {
                val id = call.parameters["id"]
                if (id == null) {
                    call.respond(HttpStatusCode.BadRequest, "Invalid model ID")
                    return@delete
                }

                try {
                    val deleted = repository.delete(UUID.fromString(id))
                    if (deleted) {
                        call.respond(HttpStatusCode.OK, "Model deleted")
                    } else {
                        call.respond(HttpStatusCode.NotFound, "Model not found")
                    }
                } catch (e: Exception) {
                    call.respond(HttpStatusCode.BadRequest, "Error deleting model: ${e.message}")
                }
            }
        }
    }
}
```

```
package dev.onelenyk.crudfather.app.routing

import dev.onelenyk.crudfather.data.dynamic.DynamicModel
import dev.onelenyk.crudfather.data.scheme.DynamicModelManager.validateDynamicModel
import dev.onelenyk.crudfather.repository.DynamicRepository
import dev.onelenyk.crudfather.repository.ModelSchemeRepository
import io.ktor.http.*
import io.ktor.server.application.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*

class DynamicRoutes(private val repository: ModelSchemeRepository, private val dynamicRepository: Dynami
    fun registerRoutes(routing: Routing) {
        routing.route("dynamic/{model}") {
            post {
                try {
                    val modelName = call.parameters["model"].orEmpty()
                    val json = call.receiveText()

                    val dynamicModel =
                        DynamicModel.fromString(
                            modelName = modelName,
                            json = json,
                        )

                    val inputModel = dynamicModel.toJsonInput()

                    if (!checkModelExists(repository, modelName)) return@post

                    val modelScheme = repository.getModelSchemeByDefinitionName(modelName)
                    if (modelScheme == null) {
                        call.respond(HttpStatusCode.NotFound, "Model definition for $modelName not found
                        return@post
                    }

                    if (!validateDynamicModel(modelScheme.definition, dynamicModel)) {
                        call.respond(HttpStatusCode.BadRequest, "Invalid JSON: does not match the model
                        return@post
                    }

                    val created = dynamicRepository.createDocument(modelName, inputModel)
                    val result =
                        DynamicModel.fromDocument(modelName, created)
                            .toJsonOutput()
                    call.respond(HttpStatusCode.Created, result)
                } catch (e: Exception) {
                    call.respond(HttpStatusCode.BadRequest, "Error creating document: ${e.message}")
                }
            }

            get {
                val modelName = call.parameters["model"].orEmpty()
                if (!checkModelExists(repository, modelName)) return@get

                try {
                    val output =
                        dynamicRepository.getAllDocuments(modelName)
                            .map {
                                DynamicModel.fromDocument(modelName, it)
                            }
                            .map {
```

```
                    it.toJsonOutput()
                }
            call.respond(HttpStatusCode.Found, output)
        } catch (e: Exception) {
            call.respond(HttpStatusCode.InternalServerError, "Error fetching documents: ${e.mess
        }
    }

    get("/{id}") {
        val modelName = call.parameters["model"].orEmpty()
        val id = call.parameters["id"].orEmpty()

        if (!checkModelExists(repository, modelName)) return@get

        try {
            val document = dynamicRepository.getDocumentById(modelName, id)
            if (document != null) {
                val result =
                    DynamicModel.fromDocument(modelName, document)
                        .toJsonOutput()
                call.respond(HttpStatusCode.Found, result)
            } else {
                call.respond(HttpStatusCode.NotFound, "Document not found")
            }
        } catch (e: Exception) {
            call.respond(HttpStatusCode.InternalServerError, "Error fetching document: ${e.messa
        }
    }

    put("/{id}") {
        val modelName = call.parameters["model"].orEmpty()
        val id = call.parameters["id"].orEmpty()

        try {
            val json = call.receiveText()

            val dynamicModel =
                DynamicModel.fromString(
                    modelName = modelName,
                    json = json,
                )

            val inputModel = dynamicModel.toJsonInput()

            if (!checkModelExists(repository, modelName)) return@put

            val modelScheme = repository.getModelSchemeByDefinitionName(modelName)
            if (modelScheme == null) {
                call.respond(HttpStatusCode.NotFound, "Model definition for $modelName not found
                return@put
            }

            if (!validateDynamicModel(modelScheme.definition, dynamicModel)) {
                call.respond(HttpStatusCode.BadRequest, "Invalid JSON: does not match the model
                return@put
            }

            val updatedDocument = dynamicRepository.updateDocument(modelName, id, inputModel)

            if (updatedDocument != null) {
                val result =
                    DynamicModel.fromDocument(modelName, updatedDocument)
```

```
                        .toJsonOutput()
                    call.respond(HttpStatusCode.OK, result)
                } else {
                    call.respond(HttpStatusCode.NotFound, "Document not found")
                }
            } catch (e: Exception) {
                call.respond(HttpStatusCode.BadRequest, "Error updating document: ${e.message}")
            }
        }

        delete("/{id}") {
            val modelName = call.parameters["model"].orEmpty()
            val id = call.parameters["id"].orEmpty()
            if (!checkModelExists(repository, modelName)) return@delete

            try {
                val deleted = dynamicRepository.deleteDocument(modelName, id)
                if (deleted) {
                    call.respond(HttpStatusCode.OK, "Document deleted")
                } else {
                    call.respond(HttpStatusCode.NotFound, "Document not found")
                }
            } catch (e: Exception) {
                call.respond(HttpStatusCode.InternalServerError, "Error deleting document: ${e.messa
            }
        }
    }
}
}
```

```
package dev.onelenyk.crudfather.app

import dev.onelenyk.crudfather.app.routing.ServerRouting
import dev.onelenyk.crudfather.di.koinModule
import dev.onelenyk.crudfather.di.provideServerPort
import io.github.cdimascio.dotenv.dotenv
import io.ktor.serialization.kotlinx.json.*
import io.ktor.server.application.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*
import io.ktor.server.plugins.callloging.*
import io.ktor.server.plugins.contentnegotiation.*
import io.ktor.server.plugins.requestvalidation.*
import io.ktor.server.routing.*
import kotlinx.serialization.json.Json
import org.koin.core.component.KoinComponent
import org.koin.core.component.inject
import org.koin.ktor.ext.inject
import org.koin.ktor.plugin.Koin
import org.koin.logger.slf4jLogger


class Server {
    fun start(): NettyApplicationEngine {
        val server = embeddedServer(
            Netty,
            port = provideServerPort(dotenv = dotenv()) ?: 8080
        ) {
            module(this)
        }
        server.start(wait = false)
        return server
    }

    fun module(application: Application) = application.apply {
        install(Koin) {
            slf4jLogger()
            modules(koinModule)
        }
        install(CallLogging)
        install(RequestValidation)
        configureSerialization()

        configureRouting()
    }

    private fun Application.configureSerialization() =
        install(ContentNegotiation) {
            json(
                Json {
                    ignoreUnknownKeys = true
                },
            )
        }

    private fun Application.configureRouting() {
        val router: ServerRouting by inject()

        routing {
            router.registerRoutes(this)
        }
    }
```

```
}
```

```
package dev.onelenyk.crudfather

import dev.onelenyk.crudfather.app.Server

fun main(args: Array<String>): Unit {
    val server = Server()
    server.start()
    return
}
```

```kotlin
package dev.onelenyk.crudfather.utils

import kotlinx.serialization.KSerializer
import kotlinx.serialization.descriptors.PrimitiveKind
import kotlinx.serialization.descriptors.PrimitiveSerialDescriptor
import kotlinx.serialization.encoding.Decoder
import kotlinx.serialization.encoding.Encoder
import java.util.*

object UUIDSerializer : KSerializer<UUID> {
    override val descriptor = PrimitiveSerialDescriptor("UUID", PrimitiveKind.STRING)

    override fun deserialize(decoder: Decoder): UUID {
        return UUID.fromString(decoder.decodeString())
    }

    override fun serialize(
        encoder: Encoder,
        value: UUID,
    ) {
        encoder.encodeString(value.toString())
    }
}
```

```
package dev.onelenyk.crudfather.db

import com.mongodb.ConnectionString
import com.mongodb.MongoClientSettings
import com.mongodb.ServerApi
import com.mongodb.ServerApiVersion
import com.mongodb.kotlin.client.coroutine.MongoClient
import com.mongodb.kotlin.client.coroutine.MongoCollection
import com.mongodb.kotlin.client.coroutine.MongoDatabase
import dev.onelenyk.crudfather.di.DbCredentials
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import org.bson.Document
import org.bson.UuidRepresentation

class MongoDBManager(dbCredentials: DbCredentials) {
    private val connectionString =
        "mongodb+srv://${dbCredentials.username}:${dbCredentials.password}${dbCredentials.connection}"

    private val serverApi = ServerApi.builder().version(ServerApiVersion.V1).build()
    private val mongoClientSettings =
        MongoClientSettings.builder()
            .uuidRepresentation(UuidRepresentation.STANDARD) // Specify the desired UUID representation
            .applyConnectionString(ConnectionString(connectionString)).serverApi(serverApi).build()

    private val mongoClient = MongoClient.create(mongoClientSettings)

    private val database: MongoDatabase = mongoClient.getDatabase("Cluster0")

    fun <T : Any> getCollection(
        collectionName: String,
        clazz: Class<T>,
    ): MongoCollection<T> {
        return database.getCollection(collectionName, clazz)
    }

    fun getDatabase(): MongoDatabase {
        return database
    }

    suspend fun pingDatabase(): String {
        return withContext(Dispatchers.IO) {
            val pingResult = database.runCommand(Document("ping", 1))
            pingResult.toJson()
        }
    }

    fun close() {
        mongoClient.close()
    }
}
```

```
package dev.onelenyk.crudfather.api

import io.ktor.client.*
import io.ktor.client.engine.cio.*
import io.ktor.client.request.*
import io.ktor.client.statement.*
import io.ktor.http.*
import kotlinx.coroutines.delay
import kotlinx.serialization.json.Json
import kotlinx.serialization.json.jsonObject
import kotlinx.serialization.json.jsonPrimitive
import java.util.*

class TestApiClient(private val client: HttpClient,
                    private val baseUrl: String) {

    val dynamic = "$baseUrl/dynamic"
    suspend fun createModel(
        modelName: String,
        modelJson: String,
    ): HttpResponse {
        val response =
            client.post("$baseUrl/models?name=$modelName") {
                contentType(ContentType.Application.Json)
                setBody(modelJson)
            }

        return response
    }

    suspend fun getModels() {
        val response = client.get("$baseUrl/models")
        println("Get Models Response: ${response.bodyAsText()}")
    }

    suspend fun getModelById(id: String) {
        val response = client.get("$baseUrl/models/$id")
        println("Get Model By ID Response: ${response.bodyAsText()}")
    }

    suspend fun updateModel(
        id: String,
        updates: String,
    ) {
        val response =
            client.put("$baseUrl/models/$id") {
                contentType(ContentType.Application.Json)
                setBody(updates)
            }
        println("Update Model Response: ${response.bodyAsText()}")
    }

    suspend fun deleteModel(id: String) {
        val response = client.delete("$baseUrl/models/$id")
        println("Delete Model Response: ${response.bodyAsText()}")
    }

    suspend fun createDynamicDocument(
        modelName: String,
        documentJson: String,
    ): HttpResponse {
        val response =
```

```
            client.post("$dynamic/$modelName") {
                contentType(ContentType.Application.Json)
                setBody(documentJson)
            }
        return response
        println("Create Dynamic Document Response: ${response.bodyAsText()}")
    }

    suspend fun getAllDynamicDocuments(modelName: String) {
        val response = client.get("$dynamic/$modelName")
        println("Get All Dynamic Documents Response: ${response.bodyAsText()}")
    }

    suspend fun getDynamicDocumentById(
        modelName: String,
        id: String,
    ) {
        val response = client.get("$dynamic/$modelName/$id")
        println("Get Dynamic Document By ID Response: ${response.bodyAsText()}")
    }

    suspend fun updateDynamicDocument(
        modelName: String,
        id: String,
        updates: String,
    ) {
        val response =
            client.put("$dynamic/$modelName/$id") {
                contentType(ContentType.Application.Json)
                setBody(updates)
            }
        println("Update Dynamic Document Response: ${response.bodyAsText()}")
    }

    suspend fun deleteDynamicDocument(
        modelName: String,
        id: String,
    ) {
        val response = client.delete("$dynamic/$modelName/$id")
        println("Delete Dynamic Document Response: ${response.bodyAsText()}")
    }
}

suspend fun main() {
    val client =
        HttpClient(CIO) {
            followRedirects = false
        }
    val apiClient = TestApiClient(client, "http://localhost:9999")

    val documentJson = createDocumentJson()
    val documentJsonNext = createDocumentJsonNext()

    testModelSchemeRoutes(apiClient, documentJson)
    testDynamicRoutes(apiClient, documentJson, documentJsonNext)

    client.close()
}

fun createDocumentJson() =
    """
    {
```

```
        "id": "${UUID.randomUUID()}",
        "name": "Jane Doe",
        "age": 25,
        "email": "jane.doe@example.com",
        "isActive": true
    }
    """.trimIndent()

fun createDocumentJsonNext() =
    """
    {
        "name": "Jane Doe NEXT",
        "age": 25,
        "email": "jane.doe@example.com NEXT",
        "isActive": true
    }
    """.trimIndent()

suspend fun testModelSchemeRoutes(
    apiClient: TestApiClient,
    modelJson: String,
) {
    val response = apiClient.createModel("projects", modelJson)
    val body = response.bodyAsText()
    val objextId = Json.parseToJsonElement(body).jsonObject.getValue("id").jsonPrimitive.content

    println("Create Model Response: $body")

    apiClient.getModels()
    delay(3000L)
    apiClient.getModelById(objextId)
//    apiClient.deleteModel(objextId)
}

suspend fun testDynamicRoutes(
    apiClient: TestApiClient,
    documentJson: String,
    documentJsonNext: String,
) {
    val response = apiClient.createDynamicDocument("projects", documentJson)
    val body = response.bodyAsText()

    println("Create Dynamic Document Response: $body")

    if (!response.status.isSuccess()) return

    val objextId = Json.parseToJsonElement(body).jsonObject.getValue("id").jsonPrimitive.content

    apiClient.getDynamicDocumentById("projects", objextId)
    apiClient.updateDynamicDocument("projects", objextId, documentJsonNext)
    //  apiClient.deleteDynamicDocument("users", objextId)
}
```

```
package dev.onelenyk.crudfather.data.scheme

import dev.onelenyk.crudfather.data.dynamic.DynamicModel
import kotlinx.serialization.json.*

object DynamicModelManager {
    fun generateModelDefinition(
        modelName: String,
        json: String,
    ): ModelDefinition {
        val jsonObject = Json.parseToJsonElement(json).jsonObject
        val fields = parseJsonObject(jsonObject)
        return ModelDefinition(modelName, fields)
    }

    private fun parseJsonObject(jsonObject: JsonObject): List<FieldDefinition> {
        val fields = mutableListOf<FieldDefinition>()
        for ((key, value) in jsonObject) {
            val fieldType = determineFieldType(value)
            val fieldDefinition =
                when (fieldType) {
                    FieldType.OBJECT ->
                        FieldDefinition(
                            name = key,
                            type = fieldType,
                            nestedFields = parseJsonObject(value.jsonObject),
                        )

                    FieldType.ARRAY ->
                        FieldDefinition(
                            name = key,
                            type = fieldType,
                            elementType = determineArrayElementType(value.jsonArray),
                        )

                    else -> FieldDefinition(name = key, type = fieldType)
                }
            fields.add(fieldDefinition)
        }
        return fields
    }

    fun validateDynamicModel(
        modelDefinition: ModelDefinition,
        dynamicModel: DynamicModel,
    ): Boolean {
        return validateJsonModel(modelDefinition = modelDefinition, dynamicModel.toJsonInput())
    }

    private fun validateJsonModel(
        modelDefinition: ModelDefinition,
        jsonData: JsonObject,
    ): Boolean {
        for (field in modelDefinition.fields) {
            val jsonElement = jsonData[field.name] ?: return false

            when (field.type) {
                FieldType.STRING -> if (!jsonElement.jsonPrimitive.isString) return false
                FieldType.INTEGER -> if (jsonElement.jsonPrimitive.intOrNull == null) return false
                FieldType.BOOLEAN -> if (jsonElement.jsonPrimitive.booleanOrNull == null) return false
                FieldType.OBJECT ->
                    if (!validateJsonModel(
```

```
                            ModelDefinition(field.name, field.nestedFields ?: emptyList()),
                            jsonElement.jsonObject,
                        )
                    ) {
                        return false
                    }

                FieldType.ARRAY -> if (!validateJsonArray(field, jsonElement.jsonArray)) return false
            }
        }
        return true
    }

    private fun validateJsonArray(
        field: FieldDefinition,
        jsonArray: JsonArray,
    ): Boolean {
        val elementType = field.elementType ?: return false
        for (element in jsonArray) {
            when (elementType) {
                FieldType.STRING -> if (!element.jsonPrimitive.isString) return false
                FieldType.INTEGER -> if (element.jsonPrimitive.intOrNull == null) return false
                FieldType.BOOLEAN -> if (element.jsonPrimitive.booleanOrNull == null) return false
                FieldType.OBJECT ->
                    if (!validateJsonModel(
                            ModelDefinition(field.name, field.nestedFields ?: emptyList()),
                            element.jsonObject,
                        )
                    ) {
                        return false
                    }

                FieldType.ARRAY -> return false // Nested arrays are not supported
            }
        }
        return true
    }

    private fun determineFieldType(jsonElement: JsonElement): FieldType {
        return when (jsonElement) {
            is JsonPrimitive ->
                when {
                    jsonElement.isString -> FieldType.STRING
                    jsonElement.booleanOrNull != null -> FieldType.BOOLEAN
                    jsonElement.intOrNull != null -> FieldType.INTEGER
                    else -> throw IllegalArgumentException("Unknown primitive type")
                }

            is JsonObject -> FieldType.OBJECT
            is JsonArray -> FieldType.ARRAY
            else -> throw IllegalArgumentException("Unknown field type")
        }
    }

    private fun determineArrayElementType(jsonArray: JsonArray): FieldType {
        if (jsonArray.isEmpty()) {
            throw IllegalArgumentException("Array cannot be empty")
        }
        val firstElement = jsonArray.first()
        return determineFieldType(firstElement)
    }
}
```

```
package dev.onelenyk.crudfather.data.scheme

import dev.onelenyk.crudfather.data.scheme.DynamicModelManager.generateModelDefinition
import kotlinx.serialization.json.*
import kotlin.random.Random

class Sample {
    private fun generateSampleJson(modelDefinition: ModelDefinition): JsonObject {
        val jsonObject =
            buildJsonObject {
                modelDefinition.fields.forEach { field ->
                    put(field.name, generateSampleValue(field))
                }
            }
        return jsonObject
    }

    private fun generateSampleValue(field: FieldDefinition): JsonElement {
        return when (field.type) {
            FieldType.STRING -> JsonPrimitive(generateRandomString())
            FieldType.INTEGER -> JsonPrimitive(generateRandomInt())
            FieldType.BOOLEAN -> JsonPrimitive(generateRandomBoolean())
            FieldType.OBJECT -> generateSampleJson(ModelDefinition(field.name, field.nestedFields ?: emp
            FieldType.ARRAY ->
                buildJsonArray {
                    repeat(3) {
                        when (field.elementType) {
                            FieldType.STRING -> add(JsonPrimitive(generateRandomString()))
                            FieldType.INTEGER -> add(JsonPrimitive(generateRandomInt()))
                            FieldType.BOOLEAN -> add(JsonPrimitive(generateRandomBoolean()))
                            FieldType.OBJECT ->
                                add(
                                    generateSampleJson(
                                        ModelDefinition(
                                            field.name,
                                            field.nestedFields ?: emptyList(),
                                        ),
                                    ),
                                )

                            else -> throw IllegalArgumentException("Unsupported array element type")
                        }
                    }
                }
        }
    }

    private fun generateRandomString(): String {
        val chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
        return (1..10)
            .map { chars.random() }
            .joinToString("")
    }

    private fun generateRandomInt(): Int {
        return Random.nextInt(0, 100)
    }

    private fun generateRandomBoolean(): Boolean {
        return Random.nextBoolean()
    }
```

```
    fun main() {
        val json = """
{
    "id":"null",
    "name":"nazar",
    "age":"0",
    "items":[
      {"value": "Open", "onclick": "OpenDoc()"},
        {"value": "Open", "onclick": "OpenDoc()"}
    ]
}
    """

        val modelName = "User"
        val modelDefinition = generateModelDefinition(modelName, json)

        println(modelDefinition)

        val sampleJson = generateSampleJson(modelDefinition)
        println(Json.encodeToString(JsonObject.serializer(), sampleJson))
    }
}
```

```
package dev.onelenyk.crudfather.data.scheme

import kotlinx.serialization.Serializable
import kotlinx.serialization.json.*

@Serializable
enum class FieldType {
    STRING,
    INTEGER,
    BOOLEAN,
    OBJECT,
    ARRAY,
}

@Serializable
data class FieldDefinition(
    val name: String,
    val type: FieldType,
    val required: Boolean = true,
    val nestedFields: List<FieldDefinition>? = null,
    val elementType: FieldType? = null,
)

@Serializable
data class ModelDefinition(
    val modelName: String,
    val fields: List<FieldDefinition>,
)
```

```
package dev.onelenyk.crudfather.data.scheme

import dev.onelenyk.crudfather.utils.UUIDSerializer
import kotlinx.serialization.Serializable
import org.bson.codecs.pojo.annotations.BsonId
import java.util.UUID

@Serializable
data class ModelScheme(
    @Serializable(with = UUIDSerializer::class) @BsonId val id: UUID = UUID.randomUUID(),
    val definition: ModelDefinition,
)
```

```
package dev.onelenyk.crudfather.data.dynamic

import kotlinx.serialization.Serializable
import kotlinx.serialization.json.Json
import kotlinx.serialization.json.JsonObject
import kotlinx.serialization.json.jsonObject
import org.bson.Document

@Serializable
data class DynamicModel(
    val modelName: String,
    val data: JsonObject,
) {
    companion object {
        fun fromString(
            modelName: String,
            json: String,
        ): DynamicModel {
            val jsonData = Json.parseToJsonElement(json).jsonObject

            val dynamicModel =
                DynamicModel(
                    modelName = modelName,
                    data = jsonData,
                )

            return dynamicModel
        }

        fun fromDocument(
            modelName: String,
            document: Document,
        ): DynamicModel {
            val json = document.toJson()
            val jsonData = Json.parseToJsonElement(json).jsonObject

            val dynamicModel =
                DynamicModel(
                    modelName = modelName,
                    data = jsonData,
                )

            return dynamicModel
        }
    }

    fun toJsonInput(): JsonObject {
        return data.changeIdToMongoDbId()
    }

    fun toJsonOutput(): JsonObject {
        return data.changeMongoDbIdToId()
    }

    private fun JsonObject.changeIdToMongoDbId(): JsonObject {
        val mutableMap = toMutableMap()
        val id = mutableMap.remove("id") ?: return this // Remove "id" and get its value
        mutableMap["_id"] = id // Add "_id" with the same value
        return JsonObject(mutableMap)
    }

    private fun JsonObject.changeMongoDbIdToId(): JsonObject {
```

```
        val mutableMap = toMutableMap()
        val objectId = mutableMap.remove("_id") ?: return this // Remove "_id" and get its value
        mutableMap["id"] = objectId // Add "id" with the same value
        return JsonObject(mutableMap)
    }
}
```