Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка навчально-тренінгової програми для підприємства «Кей Логістикс»»

Студентки 4 курсу, 7 групи, спеціальності 121 «Інженерія програмного забезпечення» _____ підпис студента — Романченко Лінда Анаезіонву

Науковий керівник кандидат технічних наук, доцент _____ підпис керівника — Савченко Тетяна Віталіївна

Гарант освітньої програми кандидат технічних наук, доцент _____ підпис керівника — Цензура Микола Олександрович

КИЇВ – 2020

ЗМІСТ

ВСТУП

Актуальність: На сьогоднішній день одними з найбільш широко використовуваних компонентів стека технологій, що використовуються при розробці веб-додатків, є Spring Framework. Знання даного фреймворка, і розуміння принципів і патернів, що лежать в його основі широко затребувані на ринку праці. Хоча у відкритому доступі є велика кількість матеріалів за даними фреймворками, досить складних опенсорсний проектів з їх використанням вкрай мало, в зв'язку з чим їх практичне вивчення ускладнене.

Об'єкт дослідження: навчально-тренінгова програма.

Предмет дослідження: розробка пропозиції щодо навчально-тренінгової програми для підприємства «Кей Логістикс».

Мета роботи: розробка добре документованого, багаторівневого веб-додатка, в стек технологій якого входить Spring Framework.

Методи і технології розробки: Аналіз технологій для створення веб-додатків, вибір підходящої мови програмування, аналіз існуючих підходів до розробки веб-додатків та ентерпрайз систем, вибір технологій та кращих алгоритмів для доступу до баз даних, технологій для відображення даних у графічному вигляді з використанням браузерів.

Результат роботи: Створення модуля доступу до бази даних веб-додатку навчально-тренінгової програми.

					КНТЕУ 121 06-23.БР		
Зм.	Аркуш	№ докум	Підпис	Дата			
Зав. кафедри		Криворучко О.В.			Розробка навчально-тренінгової програми для підприємства «Кей Логістикс»	Стадія	Аркуш	Аркушів
Керівник		Савченко Т. В.				В	4	52
Гарант		Цензура М.О.				Факультет інформаційних технологій, 4 курс, 7 група		
Розроб.		Романченко Л.А.			Вступ			

РОЗДІЛ 1. ОСНОВИ РОБОТИ ВЕБ-ДОДАТКІВ ТА ДІЯЛЬНОСТІ ПІДПРИЄМСТВА «КЕЙ ЛОГІСТИКС»

1.1. Фреймворк Spring

Spring являє собою легкий, багатофункціональний java фреймворк для побудови складних додатків з багаторівневою архітектурою, який найчастіше використовують при розробці веб-додатків [1]. Фундаментом даного фреймворка є його реалізація IoC контейнера, про який мова піде в наступних підпунктах.

Типовий додаток на java складається з безлічі об'єктів, що взаємодіють між собою, і тим самим формуючи залежності між собою. Платформа java надає величезний функціонал при розробці ПЗ, проте в ній відсутні способи організації компонентів додатка в єдине ціле. І хоча можна використовувати дизайн-патерни, такі як Builder, Factory, і Service Locator для композиції об'єктів, які формують додаток, ці патерни є не більше ніж описаної практикою в програмуванні, і їх ще необхідно реалізувати. IoC контейнер Spring вирішує дану проблему, надає формалізовані способи складання різних компонентів в працюючий додаток, готовий до використання.

Розглянутий фреймворк дуже великий і має модульну структуру, найбільш часто використовувані елементи якої представлені в таблиці 1.1.

Таблиця 1.1. – Найчастіше використовувані елементи

Група	Модуль	Опис
Кореневий контейнер	**spring-core, spring-beans**	Фундамент фреймворка, що надає функціонал IoC І впровадження залежностей.
	spring-context	Надбудова над попередніми модулями, що формалізує доступ до компонентів додатку подібно реєстру JNDI.
	spring-context-support	Підтримка інтеграції сторонніх бібліотек до Spring.
	spring-expression	Реалізація мови виразівSpring (SpEL).
	spring-tx	Підтримка декларативних транзакцій.
Доступ до даних	**spring-orm**	Шар інтеграції з API для об’єктно-реляційного відображення
Web	**spring-webmvc**	Реалізація парадигми MVC для веб-додатків, а також REST веб-сервісів.

1.2. IoC контейнер для фреймворку Spring

Spring реалізує принцип інверсії управління (IoC), також відомий як впровадження залежностей (Dependency Injection, DI). Згідно з цим принципом, компоненти програми визначають свої залежності, тобто інші об'єкти, з якими вони працюють тільки через аргументи конструктора, аргументи до factory-методам, або в своїх властивостях, які встановлюються після створення об'єкта або повертаються з factoryметода. Контейнер потім впроваджує ці залежності в момент створення компонента. Метадані, необхідні контейнеру для створення екземплярів компонентів можуть поставлятися у вигляді конфігугаціонних xml-файлів, у вигляді java-анотацій, або у вигляді java-класу конфігурації. Нижче представлені три підходи до конфігурації контейнера.

xml-конфігурація:

```
<? Xml version = "1.0" encoding = "UTF-8"?>
<Beans xmlns = "http://www.springframework.org/schema/beans"
 xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns: context = "http://www.springframework.org/schema/context"
xsi: schemaLocation = "http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/springbeans.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/springcontext.xsd ">
 <! - Опис компонента sampleRepository! ->
 <Bean class = "example.SampleRepository">
 <! - Блок залежностей компонента! ->
 </ Bean>
 <! - Опис компонента sampleService! ->
 <Bean class = "example.SampleService">
 <! - Блок залежностей компонента! ->
 </ Bean>
</ Beans>
```

annotation-конфігурація:

```
// Клас конфігурації, що включає сканування пакету "example-
// package ". Для класів в цьому пакеті, проаннотірованних
// @ Component будуть згенеровані оголошення компонентів
@Configuration
@ComponentScan ("example-package")
public class BeanConfiguration {
 // Додаткові оголошення компонентів
}
```

java-конфігурація:

```
// Клас конфігурації, що містить factory-методи для створення
// примірників компонентів
@Configuration
public class BeanConfiguration {
 // factory-метод для компонента sampleRepository
 @Bean
 public SampleRepository sampleRepository () {...}
 // factory-метод для компонента sampleService
 @Bean
 public SampleService sampleService () {...}
}
```

Всі три конфігурації приведуть до одного результату – оголошенню компонентів програми з ідентифікаторами sampleRepository і sampleService. Однак слід зауважити, що java-конфігурація найбільш гнучка, тому що вона дозволяє описати будь-який процес створення компонента. Слід також зазначити, що annotation-конфігурація буде ефективною, тільки якщо класи компонентів будуть проаннотіровани як @Component. Після надання необхідних метаданих контейнер можна використовувати явно, за допомогою наданих реалізацій інтерфейсів ApplicationContext і BeanFactory.

```
// створюємо контекст
ApplicationContext context =
 new ClassPathXmlApplicationContext (new String []
{ "Repositories.xml", "services.xml"});
// отримуємо екземпляр компонента
SampleRepository repository =
context.getBean ("sampleRepository", SampleRepository.class);
// використовуємо конфігурований примірник
Data someData = service.getSomeData ();
```

1.3. Конфігурація контейнера

Слід зазначити, що хоча в якості прикладів роботи з контейнером часто призводять явне звернення до реалізації ApplicationContext або BeanFactory, в реальних додатках використовується декларативний підхід до використання контейнера. У середовищі, що підтримує версію Servlet 3.0 допустимо розширювати абстрактний клас AbstractAnnotationConfigDispatcher ServletInitializer для початку процесу конфігурування контейнера. У середовищі, що підтримує версію Servlet 3.0 допустимо розширювати абстрактний клас AbstractAnnotationConfigDispatcherServletInitializer для початку процесу конфігурування контейнера:

```
// Ініціалізатор додатки
public class ApplicationInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {
 // Визначаємо класи конфігурації кореневого контексту
 @Override
 protected Class <?> [] getRootConfigClasses () {
 return new Class [] {RootConfig.class} ;;
 }
 // Визначаємо класи конфігурації веб-контексту
 @Override
 protected Class <?> [] getServletConfigClasses () {
 return new Class [] {MyWebConfig.class};
 }

 // Визначаємо шляху реєстрації DispatcherServlet
 @Override
 protected String [] getServletMappings () {
 return new String [] { "/"};
```

```
    }
}
```

В цьому класі RootConfig і WebConfig - класи конфігурації, проанотовані @Configuration, і містять оголошення компонентів кореневого і веб-контексту відповідно, і можуть оголошувати компоненти додатка за допомогою factory-методів, анотованих @Bean, з допомогою сканування компонентів @ComponentScan, або імпортуючи конфігурацію ззовні за допомогою @Import. Повернення методу getServletMappings () визначає, на які шляхи буде призначений сервлет DispatcherServlet, мова про який піде далі. Такий підхід до конфігурації дозволяє повністю уникнути xml-конфігурації (і в тому числі наявності дескриптора розгортання web.xml), якщо це необхідно.

Доступ до даних / інтеграція

Рівень доступу до даних / інтеграції складається з модулів JDBC, ORM, OXM, JMS і Transaction, подробиці яких наступні:

Модуль JDBC надає рівень абстракції JDBC, який усуває необхідність в осоружному кодуванні, пов'язаному з JDBC.

Модуль ORM надає шари інтеграції для популярних API об'єктно-реляційного відображення, включаючи JPA, JDO, Hibernate і iBatis.

Модуль OXM надає рівень абстракції, який підтримує реалізації відображення об'єктів / XML для JAXB, Castor, XMLBeans, JiBX і XStream.

Модуль JMS Java Messaging Service містить функції для створення і споживання повідомлень.

Модуль Transaction підтримує програмне і декларативне управління транзакціями для класів, які реалізують спеціальні інтерфейси, і для всіх ваших POJO.

Модуль JDBC надає рівень абстракції JDBC, який усуває необхідність в осоружному кодуванні, пов'язаному з JDBC.

Модуль ORM надає шари інтеграції для популярних API об'єктно-реляційного відображення, включаючи JPA, JDO, Hibernate і iBatis.

Модуль OXM надає рівень абстракції, який підтримує реалізації відображення об'єктів / XML для JAXB, Castor, XMLBeans, JiBX і XStream.

Модуль JMS Java Messaging Service містить функції для створення і споживання повідомлень.

Модуль Transaction підтримує програмне і декларативне управління транзакціями для класів, які реалізують спеціальні інтерфейси, і для всіх ваших POJO.

Web

Веб-шар складається з модулів Web, Web-MVC, Web-Socket і Web-Portlet, подробиці яких наведені нижче:

Веб- модуль забезпечує базові функції веб-інтеграції, такі як функція багатоетапної завантаження файлів і ініціалізація контейнера IoC з використанням Прослуховувач сервлетів і контексту веб-орієнтованого додатки.

Модуль Web-MVC містить реалізацію Spring-Model-View-Controller (MVC) для веб-додатків.

Модуль WebSocket забезпечує підтримку двостороннього зв'язку на основі WebSocket між клієнтом і сервером в веб-додатках.

Модуль Web- портлету надає реалізацію MVC для використання в середовищі портлету і відображає функціональність модуля Web-Servlet.

Веб- модуль забезпечує базові функції веб-інтеграції, такі як функція багатоетапної завантаження файлів і ініціалізація контейнера IoC з використанням Прослуховувач сервлетів і контексту веб-орієнтованого додатки.

Модуль Web-MVC містить реалізацію Spring-Model-View-Controller (MVC) для веб-додатків.

Модуль WebSocket забезпечує підтримку двостороннього зв'язку на основі WebSocket між клієнтом і сервером в веб-додатках.

Модуль Web- портлету надає реалізацію MVC для використання в середовищі портлету і відображає функціональність модуля Web-Servlet.

1.4. WebMVC модуль

Spring WebMVC реалізує парадигму MVC для веб додатків. Для роботи з даним модулем необхідно оголосити сервлет DispatcherServlet, який грає роль передового контролера (front controller), отримуючи всі запити до сервлет-контейнеру. Після отримання запиту, як правило, відбувається наступне:

1. Сервлет передає отриманий запит компоненту HandlerMapping, який в залежності від запитуваної шляху, HTTPметода, і ряду інших параметрів віддає назад екземпляр компонента Controller, який відповідає цьому запиту.

2. Сервлет передає запит отриманому раніше контролера, і отримує назад модель і ім'я виду, який буде використовуватися для відображення даних моделі. Ці дані вміщені в об'єкті ModelAndView

3. Сервлет передає ім'я виду компоненту ViewResolver, який по імені виду надає екземпляр компонента View, який буде Відповідальний за відображення даних моделі.

4. Сервлет передає модель виду, отриманого на попередньому кроці, отримуючи назад сформований відповідь сервера.

Примірники компонентів HandlerMapping і Controller створюються і конфігуруються контейнером відповідно до анотацій класів, проанотованих @Controller. Мова про них піде далі. Примірники компонентів ViewResolver створюються і конфігурується контейнером відповідно до їх оголошеннями в конфігурації програми. Приклад такого оголошення представлений далі. Взаємодія згаданих компонентів проілюстровано на рисунку 1.1.

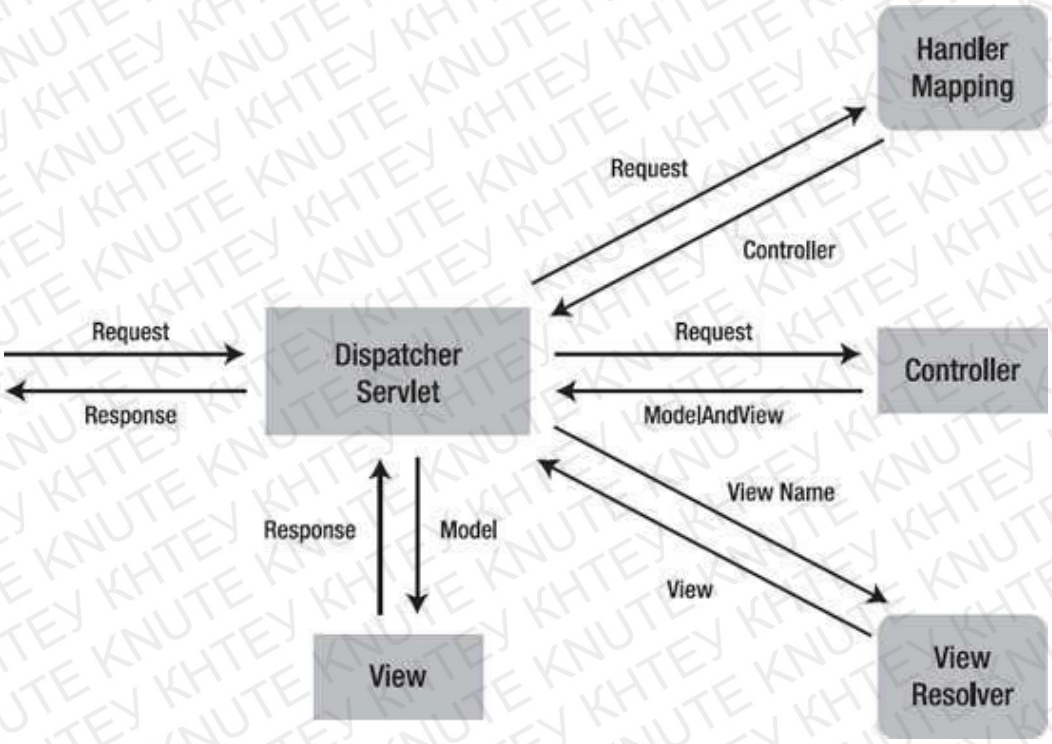

Рисунок 1.1. Взаємодія компонентів

Головною метою MVC є поділ об'єктів, бізнес-логіки й зовнішнього вигляду програми. Всі ці компоненти слабо пов'язані між собою і при бажанні ми можемо змінити, наприклад, зовнішній вигляд програми, не вносячи суттєві зміни в інші два компонента.

Модель (Model)

Цей блок инкапсулирует дані програми. На практиці це POJO-класи (Plain Old Java Objects - Прості старі Java-об'єкти).

Подання (View)

Модуль уявлення відповідає за виведення даних користувачеві. Зазвичай це JSP файл, який може бути пізнаний і інтерпретований браузером на користувальницької машині.

Контролер (Controller)

Контролер відповідає за обробку запитів користувачів і передачу даних модулю View для обробки.

У ASP.NET MVC модулі представляють спеціальні класи, що реалізують інтерфейс System.Web.IHttpModule . Їх ініціалізація відбувається після створення

об'єкта додатка. За допомогою модулів ми можемо управляти обробкою запиту в додатку.

Модуль Core забезпечує ключові частини фреймворка, включаючи властивості IoC і DI.

Context побудований на основі Beans і Core і дозволяє отримати доступ до будь-якого об'єкту, який визначений в настройках. Ключовим елементом модуля Context є інтерфейс ApplicationContext.

Модуль SpEL забезпечує потужний мову виразів для маніпулювання об'єктами під час виконання.

Контейнер Data Access / Integration складається з JDBC, ORM, OXM, JMS і модуля Transatcions.

JDBC забезпечує абстрактний шар JDBC і позбавляє розробника від необхідності вручну прописувати монотонний код, пов'язаний з з'єднанням з БД.

ORM забезпечує інтеграцію з такими популярними ORM, як Hibernate, JDO, JPA і т.д.

Модуль OXM відповідає за зв'язок Об'єкт / XML - XMLBeans, JAXB і т.д.

Модуль JMS (Java Messaging Service) відповідає за створення, передачу та отримання повідомлень

Transactions підтримує управління транзакціями для класів, які реалізують певні методи.

Фреймворк ASP.NET MVC вже використовує ряд вбудованих модулів для різних завдань:

AnonymousIdentification : представлений класом System.Web.Security.AnonymousIdentificationModule. Відповідає за ідентифікацію запитів навіть у тих випадках, коли користувач не аутентифікований - тобто для анонімних запитів

DefaultAuthentication : представлений класом System.Web.Security.DefaultAuthenticationModule. Відповідає за установку властивості User у об'єкта HttpContext

FileAuthorization : представлений класом System.Web.Security.FileAuthorizationModule, який використовується при аутентифікації Windows

FormsAuthentication : представлений класом System.Web.Security.FormsAuthenticationModule. Даний модуль використовується при аутентифікації форм і встановлює значення властивості HttpContext.User, за допомогою якого ми в додатку можемо отримати аутентифицированного користувача

OutputCache : представлений класом System.Web.Caching.OutputCacheModule. Відповідає за кешування відповіді клієнту

PageInspectorHttpModule : цей модуль підтримує функціональність Page Inspector, яка є в Visual Studio і за допомогою якої здійснюється налагодження HTML і CSS

Profile : представляє клас System.Web.Profile.ProfileModule, який пов'язує дані профілю користувача з даними в запиті

RoleManager : представлений класом System.Web.Security.RoleManagerModule, який управляє призначенням ролей

ScriptModule-4.0 : представлений класом System.Web.Handlers.ScriptModule. Призначений для підтримки Ajax-запитів

ServiceModel-4.0 : представлений класом System.ServiceModel.Activation.ServiceHttpModule. Цей модуль використовується веб-службами ASP.NET

Session : реалізований класом System.Web.SessionState.SessionStateModule. Призначений для зв'язку даних сесії з запитами

UrlAuthorization : представлений класом System.Web.Security.UrlAuthorizationModule, який забезпечує авторизований доступ до ресурсів

UrlMappingsModule : представлений класом System.Web.UrlMappingsModule. Призначений для підтримки функції URL Mappings, яка в ASP.NET MVC не застосовується

UrlRoutingModule-4.0 : реалізований класом System.Web.Routing.UrlRoutingModule, який використовується системою маршрутизації

WebPageHttpModule : даний модуль зіставляє запити з файлами уявлень

WindowsAuthentication : представляє клас System.Web.Security.WindowsAuthenticationModule. Відповідає за аутентифікацію Windows

Ключовою абстракцією даного модуля є контролер. Саме контролери при використанні SpringMVC грають роль точки входу для усієї логіки додатка. Для реєстрації класу-контролера, його необхідно проанотувати як @Controller. Приклад класу-контролера представлений далі.

```
// Приклад класу-контролера
@Controller
public class HelloController {
 // Метод-обробник запитів на шлях "/ hello" с методом GET
 @RequestMapping ("/ hello")
 public String hello (Model model) {
 model.addAttribute ("msg", "Hello");
 return "hello";
 }
}
```

Даний контролер буде обробляти запити надходять на шлях "/ Hello" щодо шляху, на який зареєстрований DispatcherServlet, формувати модель з єдиним атрибутом msg, і передавати цю модель азом з ім'ям виду "hello" сервлету DispatcherServlet. За допомогою анотацій java можна описати безліч аспектів поведінки контролера. Найбільш часто використовувані анотації, використовувані при описі контролерів представлені в таблиці 1.2.

Таблиця 1.2. – Найбільш використовувані анотації

Приклад анотації	Коментарі
@Controller	Перед ім'ям класу сигналізує про те, що цей клас - контролер, і повинен бути відповідним чином оброблений контейнером.
@RestController	Перед ім'ям класу сигналізує про те, що цей клас - контролер, і крім цього – що повернення методів даного контролера повинен інтерпретуватися як тіла відповіді, а не ім'я моделі.
@RequestMapping(“/foo”)	Перед методом контролера сигналізує про тому, що це метод-обробник, який необхідно зареєструвати по шляху «/ foo»
@RequestParam(“foo”)	Перед аргументом методу контролера сигналізує про те, що в цим аргументом треба передати значення параметра запиту з ім'ям «foo»
@PathVariable(“foo”)	Перед аргументом методу сигналізує про те, що в цей аргументом треба передати значення змінної шляху запиту з ім'ям «foo»
@ModelAttibute(“foo”)	Перед аргументом методу сигналізує про те, що в цей аргументом треба десереалізувати атрибут моделі з ім'ям «foo»

@RequestBody	Перед аргументом методу сигналізує про те, що в цей аргументом треба десереалізувати тіло запиту
@ResponseBody	Перед методом сигналізує про те, що повернення необхідно трактувати як тіло відповіді, а не назву моделі.
@ResponseStatus	Перед методом вказує, який HTTP-статус використовувати, в разі успішної обробки запиту.

1.5. Особливості роботи підприємства «Кей Логістикс»

Ухвалення рішення про проведення корпоративного навчання не відноситься до компетенції тренера, але менеджер по персоналу або керівник компанії теж не завжди здатні самостійно прийняти зважене і збалансоване рішення. В такому випадку тренер може запропонувати керівникам спільно провести аналіз результативності діяльності співробітників.

Для цього спочатку необхідно вибрати підрозділу або відібрати групу працівників, що показують незадовільні результати (або нижче запланованих). Потім потрібно зробити оцінку професійних знань, умінь і навичок - і мотивації (бажань, установок) цих працівників: наскільки вони відповідають корпоративним стандартам, необхідним для успішного виконання роботи. Оцінку можна зробити, провівши атестацію або опитування (інтерв'ювання, анкетування) них самих та їхніх безпосередніх керівників.

Кожен співробітник, відповідно до отриманими оцінками, потрапляє в один з чотирьох квадрантів :

Квадрант А (мотивація). Працівник має достатній рівень знань, умінь і навичок, але низький рівень мотивації. Підвищити його результативність шляхом додаткового навчання неможливо , це мотиваційна проблема . Краще зосередитися на підвищенні значущості ділянки робіт, за який він відповідає; розробити додаткові стимули, заохочення та нагороди для самого працівника (або включити його в наявну корпоративну систему винагород).

Квадрант B (ресурси / процеси / середа). Працівник має достатній рівень як знань, умінь, навичок, так і мотивації. Низькі показники результативності пов'язані з проблемами, які він не може контролювати, швидше за все - з браком часу, неергономічними робочого місця і т. Д. Підвищити його результативність шляхом додаткового навчання неможливо . Краще проаналізувати якість лінійного менеджменту і усунути недоліки в плануванні та організації діяльності.

Квадрант C (відбір). Працівник має як низький рівень знань, умінь і навичок, так і низький рівень мотивації. Низький рівень результативності, швидше за все, пов'язаний з неправильним призначенням, підбором або просуванням працівника. Підвищити його результативність шляхом додаткового навчання неможливо . Краще перевести людину в інший підрозділ, на більш відповідне місце або звільнити.

Квадрант D (тренінг). Працівник має низький рівень знань, умінь і навичок, але при цьому високий рівень мотивації. Підвищити результативність діяльності можна шляхом додаткового навчання (тренінг, навчання на робочому місці, наставництво і т. Д.).

Матриця аналізу результативності - дуже корисний інструмент, він допомагає зрозуміти причини неуспішності працівників і прийняти правильні рішення для виправлення ситуації. Як видно з аналізу результатів оцінки, тренінг «показаний» далеко не кожному співробітнику, що демонструє низькі результати в роботі, в більшості випадків потрібні управлінські рішення. Без такого попереднього відбору учасників ефективність тренінгу буде набагато нижче. Більш того, типова програма навчання «для всіх» часто може навіть погіршити бізнес-результати.

Коли за результатами проведеного аналізу оцінки працівників відібрано групу, для якої проведення тренінгу необхідно, тренер може перейти до етапу розробки програми . Хорошу тренінгову програму можна написати «з нагоди», її розробці повинен передувати серйозний аналіз і планування. Потрібно, щоб тренер

ясно уявляв собі, які бізнес-цілі ставить перед собою компанія, які матеріали необхідно буде вивчити і які потреби учасників. В іншому випадку можливі серйозні помилки, які загрожують тренеру повним провалом:

- невірна фокусування програми , в результаті чого курс не відповідає бізнес-потребам компанії;
- занадто легкий / важкий курс для конкретної групи учасників, який або фрустрирует їх, або змушує нудьгувати;
- розробка некоректної програми курсу : неповної, перевантаженій або містить невідповідні матеріали.

У багатьох країнах тренери і фахівці з розробки навчальних програм використовують стандартизовані процедури - свого роду професійні стандарти для тренерів. Наприклад, стандарт Instructional System Development (ISD) складається з п'яти етапів (рис. 2): 1) аналіз; 2) проектування; 3) розробка; 4) реалізація; 5) оцінка.

1.6. Автоматизація діяльності підприємства «Кей Логістикс»

Коли чітко сформульовані цілі (в конкретних показниках, якщо це можливо), яких повинні досягти учасники на своїх робочих місцях, і визначено вихідний рівень їх компетентності, можна визначити цілі і завдання навчання (які навички, знання і установки повинні бути засвоєні і сформовані під час тренінгу), а також способи вимірювання успішності навчання .

Правильне формулювання цілей має критичне значення. Якщо мета сформульована так: «Учасники повинні розуміти тему (предмет, теорію, процес і т. Д.)», То яким чином «на виході» можна буде виміряти ступінь її розуміння? Як ми дізнаємося, чи зможуть вони застосувати нові знання в роботі? Тому цілі навчання повинні формулюватися в термінах здійснення дій або вчинків, які можна спостерігати і вимірювати: учасник тренінгу має бути в змозі продемонструвати виконання конкретних дій і процедур, володіння певними навичками або

застосування отриманих знань. Крім того, необхідно розробити методи оцінки зміни реального поведінки учасників на робочому місці.

При розробці програми тренінгу ключовим є питання: яким чином допомогти учасникам просунутися з уже наявного рівня знань і умінь на рівень майстерного володіння новим матеріалом, застосування нових навичок? Спочатку є сенс описати загальний дизайн майбутнього курсу: основні змістовні блоки (теми), структуру і стратегії викладу матеріалу.

Існує безліч способів організації та подання змісту навчального курсу. Наприклад, можна використовувати такі стратегії викладу матеріалу:

покрокову;

від цілісної картини - до складових частин;

від загального - до специфічного;

від відомого - до невідомого;

від невідомого - до відомого і т. п.

Дуже важливо також правильно вибудувати послідовність викладу матеріалу. Для цього спочатку кожну з завдань навчання потрібно зіставити з відповідними теоретичними знаннями і практичними вправами, різними видами активності і методами навчання. Всі ці матеріали будуть об'єднані однією спільною темою . Згрупувавши кілька логічно пов'язаних тематичних блоків, ми отримаємо навчальний модуль (див. Додаток). Послідовність викладу окремих модулів утворює структуру курсу . До кожного модулю потрібно розробити перелік всіх необхідних роздавальних і демонстраційних матеріалів , спланувати їх підготовку, оформлення та тиражування.

Матеріалами можуть служити слайди (для презентації та для проектора), робочий конспект тренінгу (Workbook), плани роботи, шаблони і типові форми різних документів, тести, аудіо- та відеозаписи, іграшки, форми для отримання зворотного зв'язку від учасників та їх керівників, матеріали для практичних завдань

і експериментів, анкети для учасників (самооцінка навичок і самоперевірка знань), плакати і т. п.

На цьому ж етапі планується, яке обладнання знадобиться для реалізації всіх видів активності: комп'ютери, відеопроектор, фліпчарти, відеокамери, самоклеючі листочки і т. П.

Дуже важливо залучати до оцінки програми тренінгу (і на етапі її розробки, і на етапі реалізації, і після її закінчення) менеджерів по персоналу і безпосередніх керівників учасників програми навчання, враховувати їх зауваження і пропозиції.

Розробляючи програму тренінгового курсу і вибираючи методи навчання, потрібно враховувати необхідність інтеграції в тренінгу теорії і практики, а також фокусування його на досягненні поставлених цілей. Крім того, обов'язково потрібно враховувати закономірності навчання дорослих, які відображені в відомою схемою Д. Киркпатріка (Donald Kirkpatrick):

Фаза 1. Отримання практичного досвіду. Вибираються важливі практичні проблеми, конкретні ситуації. Ця інформація вимагає оцінки з боку тренера.

Використовувані методи: задавання питань, аналіз контекстних ситуацій, групове рішення проблем, розбір конкретних ситуацій, рольові ігри, робота в малих групах.

Фаза 2. Осмислення досвіду. Аналіз наявного практичного досвіду та інформації, отриманої в першій фазі навчання.

Використовувані методи: аналіз отриманої інформації, обговорення в малих групах і общегрупповие дискусії, індивідуальні виступи учасників, звіт про роботу малих груп.

Фаза 3. Узагальнення досвіду. Уже можна підбити підсумки дискусій, інтерпретуються результати, отримані в другій фазі, відбувається кристалізація нових знань.

Використовувані методи: узагальнюючі групові дискусії, підсумковий огляд або міні-лекція.

Фаза 4. Практичне застосування нових знань. Вони закріплюються в ході виконання практичних завдань; встановлюються зв'язки між новими знаннями і вимогами робочих ситуацій.

Використовувані методи: складання плану дій, дискусії, практичне відпрацювання нових навичок, виїзні практичні заняття.

Для отримання, а головне - закріплення ефективних результатів тренінгу члени групи повинні пройти через всі фази циклу.

1.7. Висновки до розділу 1

В даному розділі описано концепцію інверсії контролю (IoC, Inversion of control), його переваги у порівнянні з ручним управлінням життєвого циклу об'єктів, описано spring framework, та яке місце в ньому займає контейнер інверсії контролю, а саме його реалізацію у вигляді впровадження залежностей (dependency injection) та як він працює. Також було описано основні компоненти бібліотек spring та їх взаємодію, велику увагу приділено бібліотеці spring mvc (Model-view-controller), що надлаштовується на бібліотеку spring core, та відповідає за передачу даних з мови програмування Java на зовнішній вигляд, що буде відображатися користувачу у браузері у вигляді html сторінок, або у текстовому вигляді у різних форматах, як, наприклад, json, hateoas, xml та інші.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ

2.1. Схема побудови Enterprise-додатку

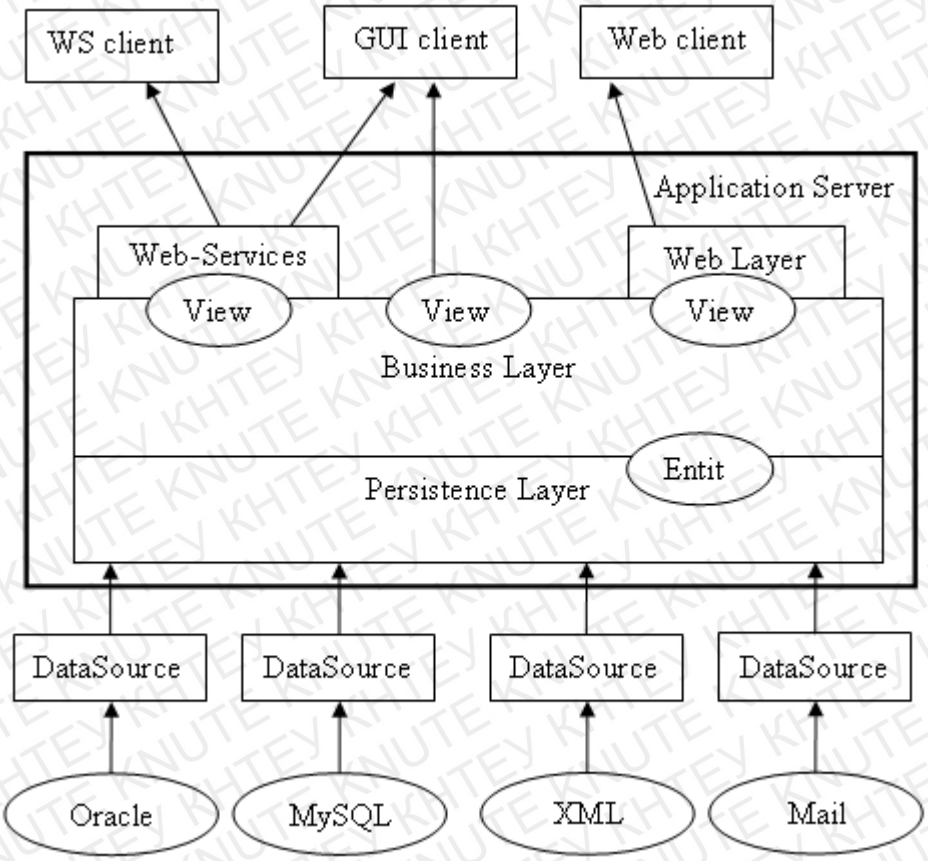

Рисунок 2.1. Схема побудови Enterprise-додатку

Дана схема призначена для побудови систем без використання головної складової СОА - ESB (Enterprise Service Bus). Якщо ж ви використовуєте СОА, то схема буде виглядати не зовсім так. Хоча якась частина даної схеми прекрасно може бути використана і у випадку з СОА.

2.2. Рівень зберігання даних

Зв'язка DataSource і Persistance Layer призначена для роботи з даними.

Причому дані можуть зберігатися не тільки в базі даних. Це може бути XML-файл або просто текстовий файл. Це може бути поштовий сервер. Або Excel-файл. Загалом це може бути все що завгодно - аби це було більш-менш постійним сховищем.

Для того, щоб можна було абстрагуватися від конкретного сховища даних застосовується таке поняття, як DataSource - джерело даних, за яким ховається зазвичай щось більш-менш відчутне - конкретна база даних, файл або ще щось. Поняття DataSource виступає в ролі такого собі моста між Persistance Layer і реальним сховищем. Насправді є навіть спеціальний клас - javax.sql.DataSource. Він исползуется для соданія конекту до бази даних - почитайте про нього, є сенс. Але крім цієї «абстракції» існує ще дуже важливий шаблон проектування - Data Access Object (DAO). Його поява пояснюється декількома причинами:

- Джерела даних можуть бути різними.
- Для доступу безпосередньо до конкретного сховища може використовуватися різний API
- Включення в логіку коду, який обмежує зміну сховища не є гарною ідеєю.

Яким же чином DAO допомагає вирішити ці проблеми? Основна ідея - визначити для Business Layer інтерфейс, який він може використовувати для доступу до даних. А вже безпосередня реалізація цього інтерфейсу буде залежати від сховища.

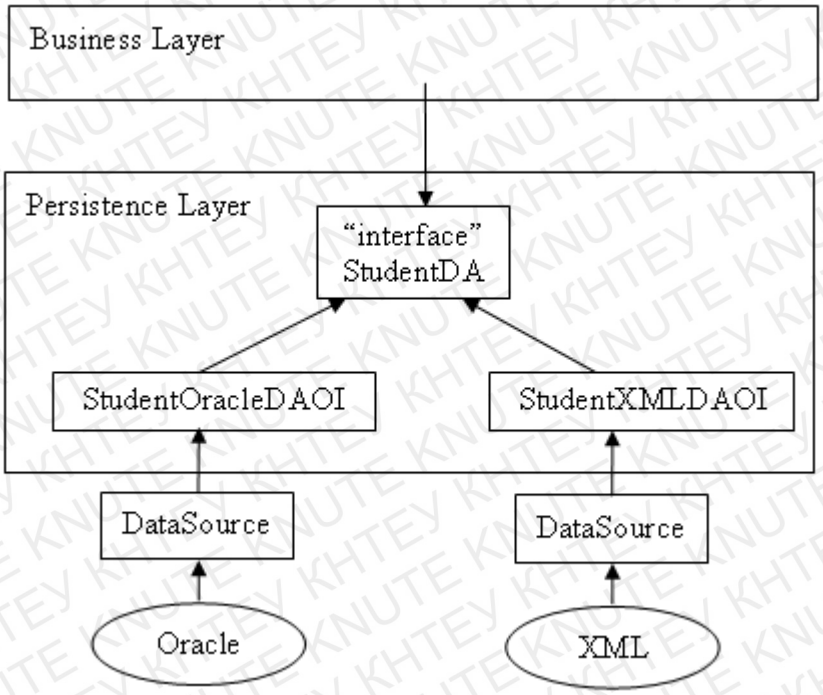

Рисунок 2.2. Схема визначення інтерфейсу Business Layer

Рсунок показує, що Business Layer ніяк не залежить від реалізації сховища. Йому абсолютно байдуже, чи буде це база даних або файл. Або просто інша база даних. У нього є интерфес, який він використовує. Якщо раптом буде потрібно з якоїсь причини використовувати той же MySQL замість Oracle - вам треба тільки реалізувати інтерфейс StudentDAO для MySQL і все. Що безсумнівно підвищує гнучкість і переносимість системи.

До об'єктів бази даних належать:

Таблиці – це набір даних з однієї визначеної теми. Наприклад, дані про студентів, їх прізвища, адреси, телефони. Рядки двовимірних таблиць називають записами, стовпчики – полями. У термінах реляційних СКБД подібні таблиці називаються відношеннями, їх записи – кортежі відношень, поля – атрибути відношень. Записи відрізняються своїм номером, а поля – своїм ім¢ям. Основні умови щодо змісту таблиць такі:

однакові записи забороняються;

всі записи повинні мати однакову кількість полів;

значення полів атомарні, тобто таблиця не може мати своїми компонентами інші таблиці.

Форми – використовуються для введення нових даних і перегляду існуючих у вказаному форматі. Можна використовувати кнопкову форму для відкриття інших форм або звітів. Форми – основний засіб побудови інтерфейсу користувача, що забезпечує найбільш зручний спосіб перегляду та редагування даних, а також контроль за ходом виконання прикладної програми. Таким чином, форми будуються для:

виведення та редагування даних. Це найбільш поширений спосіб використання форм. Вони забезпечують виведення на екран монітора даних у необхідному вигляді. Використання таких форм значно спрощує редагування даних, їх введення та видалення з бази даних. При цьому деякі дані можна зробити доступними лише для перегляду, а деякі – зовсім не демонструвати. Крім цього, є можливість у таких формах забезпечити процес обчислення полів в залежності від параметрів, які задає користувач;

керування ходом виконання прикладної програми. Сучасні прикладні програми, як правило, мають оболонку для надання користувачеві можливості виконувати ті чи інші дії у певній послідовності. Таку оболонку можна побудувати за допомогою форм, а саме: кнопкових форм. Такі форми містять кнопки, що викликають при їх виборі дію певного макроса або процедури VBA. Ці кнопки називаються командними. Командні кнопки можуть використовуватись і для виклику іншої форми, а також багатьох інших дій, що пов'язані з ходом виконання прикладної програми, а саме: виконувати запити, команди меню, фільтрувати дані, друкувати звіти тощо;

введення даних. Можна побудувати форми, які призначені лише для введення в базу даних нових значень, які автоматизують виконання програми за певним алгоритмом;

виведення повідомлень. Такі форми забезпечують інформацією події, що пов'язані з виконанням прикладної програми, наприклад, повідомлення про різноманітні помилки;

друк інформації. Як правило, для друку призначені звіти, але надрукувати інформацію можна і за допомогою форм. Очевидно для введення та виведення даних діють різні параметри, тому у таких формах інструментально підтримується їх подвійна роль.

Запити – вимоги на відбір даних, що зберігаються у таблицях, або вимога на виконання певних дій з даними. За допомогою запитів дані упорядковують, фільтрують, об'єднують, аналізують.

Звіти – об'єкти бази даних Microsoft Access, призначені для відображення даних, організованих і відформатованих відповідно до вимог користувача. За допомогою звітів складаються комерційні відомості, списки телефонів або списки розсилання. Звіти використовуються для відображення інформації у друкованому вигляді.

Макроси – містять одну або декілька макрокоманд, які використовуються для автоматичного виконання деяких операцій.

Модулі – містять програми на VISUAL BASIC. Дозволяють розширити можливості системи, якщо написати для цього необхідні програми.

Принципова заміна сховища даних відбувається вкрай рідко. Наприклад заміна Oracle на MS SQL - таке я зустрічав. Про зміну бази даних на інше джерело (наприклад LDAP) - я таке не бачив. Але тим не менше про це шаблоні треба думати - в кінці кінців при якомусь злиття або навпаки, нове керівництво може в наказовому порядку змусити всіх змінити сховища даних. Наприклад, раніше дані приходили з 1С, а сьогодні керівництво купило систему «Парус» або взагалі SAP. І якщо ви будете до цього готові, то честь і хвала вам як професіоналу своєї справи.

Не можу не згадати, що в моїй практиці нерідко траплялися ситуації, коли такий варіант проектування не був потрібен - в цьому випадку я думаю, що можна

дозволити собі не використовувати інтерфейс StudentDAO, а проектувати відразу клас StudentDAO. Щоб не плодити зайві файли. Але тут вже вибирати вам.

Що ще хотілося б відзначити - не думайте, що для всього програми треба використовувати тільки один DAO. Якщо у вас кілька сотень таблиць і для кожної потрібно як мінімум 3-4 методу - то кількість DAO може (та й має) бути більше одного. Поділ може бути різним - це і рішення зробити для кожної сутності свій DAO, і поділ по деякому функціональною ознакою (наприклад все, що пов'язано з довідниками, все, що пов'язано з рахунками і т.д.) або ще якось. Тут складно дати якісь поради. Фантазуйте, аналізуйте, пробуйте.

ORM - Object Relation Mapping. Попереджаю одразу - я не буду розглядати випадки використання будь-яких інших сховищ, крім реляційних баз даних. Але навіть в цьому випадку дані треба отримати з сховища і представити у вигляді, який зручно обробляти. Вже досить давно існує термін ORM - Object Relation Mapping. Я б перевів це як «об'єктне відображення реляційних таблиць». Хоч в оригіналі слово «таблиці» і не зустрічаються, але так буде точніше відображений сенс. Він позначає технології, які на рівні додатків дозволяють розглядати записи в таблицях, як об'єкти. Кожен рядок в таблиці - це об'єкт. Просто властивості цього класу відображаються на колонки в таблиці. Такі класи називаються Entity - Сутність. До речі, ми майже впритул підійшли до цього - ми вводили класу «Група» і «Студент», які, по суті, виконували дуже просту функцію - зберігали дані з таблиці. Дані системи беруть на себе досить нудні, але в той же час дуже важливі функції - реалізують функції CRUD (Create, Read, Update, Delete).

Пошук даних за допомогою фільтра. Фільтр використовується для того, щоб відобразити лише ті значення, які визначені заданими критеріями. Існують чотири способи, які використовуються для вибору записів за допомогою фільтрів:

Фільтр за виділеним фрагментом, потрібно використовувати, якщо можна легко знайти і вибрати у таблиці те значення, яке повинне містити записи, що вибираються.

Звичайний фільтр використовується для вибору шуканих значень із списку без перегляду всіх записів у таблиці.

Поле «Фільтр для» використовується для введення конкретного значення або виразу, введеного як умова вибору в контекстне меню.

Розширений фільтр використовується для створення складних фільтрів, якщо необхідно задати ряд критеріїв вибору записів.

Для використання фільтра служить команда меню ЗАПИСИ/ ФИЛЬТР.

Запити та їх застосування. Для користувача основним режимом роботи з базою даних є режим отримання інформації. Причому нарівні з інформаційним пошуком у базі даних потрібно здійснювати перетворення (редагування, упорядкування, групування та елементарні обчислення) даних для видачі користувачеві різних довідок, звітів і т.п. Це виконується за допомогою запитів. Запити дозволяють переглядати, аналізувати і змінювати дані з декількох таблиць.

Основна подібність між запитами на вибірку і фільтрами полягає у тому, що в них проводиться вилучення підмножини записів із базової таблиці або запиту.

Фільтр, як правило, використовують під час роботи у режимі Форми або у режимі Таблиці для перегляду або зміни підмножини записів. На відміну від фільтра запит дозволяє отримати більш змістовний результат. Перш за все, це пояснюється тим, що фільтр дає інформацію для перегляду (друку), але на відміну від запиту автоматично не зберігається як окремий об¢єкт бази даних. Запити, маючи таку властивість, дозволяють динамічно поновлювати інформацію у своїх таблицях, якщо у таблицях бази даних виникла зміна інформації. Крім цього, запит має і зворотну дію: якщо змінювати інформацію у його таблицях, то таблиці бази

даних, на основі яких побудований запит, будуть адекватно змінювати свою інформацію.

Запити використовують для виконання таких дій:

Перегляд підмножини записів без попереднього відкриття конкретної таблиці або форми.

Вибір таблиць, що містять записи, з можливістю подальшого додавання інших таблиць.

Виконання обчислень над значеннями полів.

При виконанні запиту опитується активна база даних. Результат переглядається на екрані або роздруковується. У той самий час інформація, що не відповідає заданому критерію, звичайно не витягується, хоч вона і зберігається у базі даних.

Типи запитів:

Запит на вибирання, забезпечує вибір даних із зв'язаних таблиць і таблиць, побудованих під час реалізації інших запитів. (Наприклад, запит на вибирання записів про робітників, які мають вищу освіту, запит на вибирання записів про робітників, які працюють менше трьох років, і т.д.). Запити на вибирання створюють тимчасові результуючі таблиці. Базові таблиці при цьому не змінюються. Запити на вибирання можна також використовувати для групування записів і обчислення сум, середніх значень, підрахунку кількості записів і отримання інших типів підсумкових значень.

Запит з параметрами відображає одне або декілька визначених діалогових вікон, які виводять запрошення користувачу ввести умови вибирання. При запуску такого запиту можна ввести критерії вибирання записів або значення для вставки у поле.

Перехресний запит відображає результати статистичних розрахунків (такі, як суми, кількість записів і середні значення), виконаних за даними з одного поля таблиці. Ці результати групуються у вигляді таблиці по двох наборах даних, один з яких визначає заголовки стовпчиків, а інший – заголовки рядків. Перехресні

запити використовують для розрахунків і подання даних у структурі, яка полегшує їх аналіз.

Запит на внесення змін за одну операцію вносить зміни або переміщує декілька записів. Існує чотири типи таких запитів:

Запит на створення таблиці – ґрунтується на запиті на вибирання і забезпечує формування і створення нової таблиці на основі усіх або частини даних із однієї чи декількох таблиць.

Запит на оновлення – дає змогу вносити зміни у групу записів, які вибираються за допомогою запиту на вибирання. (Підвищити заробітну плату деякій категорії робітників, наприклад, водіям або робітникам, які мають ставку, меншу від зазначеної суми).

Запит на знищення – забезпечує виключення записів з однієї або кількох зв’язаних таблиць (наприклад, ліквідувати на підприємстві деяку посаду).

Запит на додавання записів. Даний запит додає групу записів із однієї або декількох таблиць у кінець однієї або декількох таблиць.

Для однієї і тієї самої таблиці можна створювати різні запити, кожен з яких може вилучати з таблиці певну частину інформації, яка в даний момент необхідна.

Не доводиться писати однотипний код при вставці нового запису або при отриманні даних з таблиці, SQL-запити з параметрами, писали код щодо заповнення параметрів в SQL-запиті, писали код для запам'ятовування отриманих даних з ResultSetв колекції об'єктів певного типу. Цілком природно, що це незручність було помічено вже давно і були зроблені спроби якось спростити. Так з'явилися деякі framework, про які ви вже чули. Ось найбільш відомі:

1. Hibernate
2. TopLink
3. JPA – Java Persistence API
4. JDO – Java Data Object

5. iBatis

Швидше за все, існують і будуть створені ще якісь, але я не обізнаний. Що найважливіше і що є головною метою цього рівня - дати простий і зрозумілий механізм, який дозволяє з бази даних отримати ці самі дані і представити їх у вигляді об'єктів. Зворотній мета: зберегти стан об'єктів в базі даних у вигляді записів в таблицях (якщо ми говоримо про реляційних базах даних). Все інше - бантики. Наприклад, до появи анотацій для Hibernate доводилося писати спеціальні описатели у вигляді XML. С появою анотацій від цих файлів тепер можна відмовитися, що зробило опис ще зручніше.

Сенс Persistence Layer в цілому один - з одного боку цього рівня знаходиться будь-яке сховище даних (найчастіше реляційна база даних), а з іншого боку розробник може працювати з класами-Entity. Смію вас запевнити - працювати з об'єктами набагато зручніше, ніж з тими ж масивами даних або ResultSet. А вже якщо вам треба використовувати несколкьо видів сховищ - то без Persistnece Layer обійтися вкрай складно. Та й не треба - раз вже стільки розумних голів його радять (автор ні в якому разі не не претендує чи не це звання).

2.3. Рівень бізнес-логіки

На цьому рівні з одного боку все дуже просто, з іншого боку - складно. Просто тому, що його ідея і функціональне призначення дійсно дуже просте - він виконує обробку даних відповідно до якоїсь логікою - змінює, додає, робить вибірку. І більше нічого. З іншого боку існує кілька складнощів. Перша - необхідно організувати роботу з даними у вигляді цілісних транзакцій (сподіваюся, що це таке ви знаєте). А друга - система може мати складну ієрархію класів, яку треба налаштовувати (ініціалізацію), що теж є не завжди тривіальним завданням.

Перша проблема досить добре вирішується декількома способами:

1. EJB - Enterprise Java Beans. По суті, кожен клас (методу), який виконує будь-які дії можна привласнити транзакцію, якою управляє Application Server (не забудьте, що Application Server повинен підтримувати роботу

з EJB). Вона починається при заході в метод і закінчується при виході з методу. Остання версія EJB 3.0 (якщо знову ж сервер таке підтримує) досить непогано справляється з потрібними завданнями. виходить.

2. Самому чесно починати транзакції і ловити результат SQL-запитів. Цим звичайно краще не користуватися - через дерев лісу не побачите. Деяким полегшенням роботи може стати використання аспектів. Основна ідея аспектів - це щось на зразок вставки коду в різні місця програми. Наприклад, можна визначити якийсь набір методів і для них визначити дії в їх початку і наприкінці. І тоді при виклику методу буде спочатку виконуватися код аспекту і тільки після цього сам метод. Більш детально можна подивитися - Аспектно-орієнтоване програмування
3. Використовувати framework Spring. Містить можливість працювати з транзакціями різних видів - звичайні JDBC-транзакції, транзакції для Hibernate, TopLink і навіть використовувати транзакції для Application Server. Річ дійсно зручна, і що дуже приємно, не вимагає Application Server - можна працювати зі звичайним web-сервером Tomcat. Що важливо відзначити, для реалізації цього Spring використовує AOP, яким володіє. Крім цього щоб ще хотілося відзначити - зверніть увагу, що між Persistence Layer і Business Layer ми передаємо Entity, а на інші рівні (UI рівні) BusinessLayer віддає / отримує View.

По-перше - це зручно. Треба віддавати UI то, що він буде показувати. А показувати він буде не завжди точну структуру відносин між Entity. Значить задуматися над цим доведеться - отже, простіше відразу передбачити таку можливість. Може бути, що деякі View будуть за своєю структурою навіть збігатися з сутностями. Але найчастіше так не буває.

Є ще один момент, який додає ваги для використання View - стандартні способи передачі параметрів по HTTP (див. Трохи нижче розділ «Web-client»). Для

перетворення класів з Java в якесь текстове представлення часто використовується механізм reflection (можливість отримання інформації про клас - його полях, методах і ін). Так ось при перетворенні пакети, отримуючи клас, намагаються розібрати всі його поля. Якщо якесь поле посилається на інший об'єкт, пакет намагається отримати доступ до всіх полів цього об'єкта. І так до нескінченності. Ви напевно вже здогадалися, чим це може загрожувати, якщо два об'єкти посилаються один на одного. Або в загальному випадку є якась замкнута ланцюжок об'єктів. Таке перетворення просто увійде в нескінченний цикл. Можна звичайно писати кожен раз свій перетворювач. Але чи не простіше написати клас для View.

Виходячи з цих міркувань і виходить - зручно зробити окремі класи для видачі на рівень UI.

2.4. Рівень обробки HTTP-запитів

Чомусь на цей рівень часто покладають управління бізнес-логікою - особливо це стосується сервлетів. На мій погляд, це не зовсім правильно.

По-перше - змішувати перетворення даних с логікою - це всі методи зібрати в одну купу і зробити смітник. А ви будете змушені це робити - адже приходять вам аж ніяк не Java-об'єкти, а звичайний HTTP. Так, звичайно, є способи це автоматизувати, але тим не менше буде це виглядати жахливо.

По-друге - хто вам сказав, що будуть ТІЛЬКИ Web-клієнти? А якщо це будуть Web-Sevices для демонів або звичайні «товсті» GUI-клієнти, які можуть спілкуватися з додатком по RMI або через ті ж самі Web-Services. Викликатимете логіку з сервлетів? В принципі можна, але це явно збільшить «помоечние» вашого коду.

Тому і народилося рішення покласти на рівень Web просту задачу перетворення даних між HTTP і Java. Єдине, що ще можна покласти на Web-рівень - це логіку зміни екранів. Той самий шаблон проектування MVC (Model-View-Controller). Але це має не дуже сильну зв'язок саме з бізнес-логікою. Що стосується технологій і framework, які можна використовувати на цьому рівні, то їх чимало:

- JSP + JSTL
- JSF
- Struts
- Spring

2.5. Об'єктно-реляційне відображення простих об'єктів

JPA - це технологія, що забезпечує об'єктно-реляційне відображення простих JAVA об'єктів і надає API для збереження, отримання та управління такими об'єктами.

JPA - це специфікація (документ, затверджений як стандарт, що описує всі аспекти технології), частина EJB3 специфікації.

Сам JPA не вміє ні зберігати, ні управляти об'єктами, JPA тільки визначає правила гри: як щось буде діяти. JPA також визначає інтерфейси, які повинні будуть бути реалізовані провайдерами. Плюс до цього JPA визначає правила про те, як повинні описуватися метадані відображення і про те, як повинні працювати провайдери. Далі, кожен провайдер, реалізуючи JPA визначає отримання, збереження і управління об'єктами. У кожного провайдера реалізація різна.

У JPA існують різні реалізації:

- Hibernate
- Oracle TopLink
- Apache OpenJPA

EJB описує об'єкти, як вони повинні зберігатися в базу - там це дуже складно. А hibernate як ORM реалізація завоювала дуже велику популярність. JPA - це в основному ідеї hibernate і JDO.

Структура JPA:

1. API - інтерфейси в пакеті javax.persistance. Набір інтерфейсів, які дозволяють організувати взаємодію з ORM провайдером.

2. JPQL - об'єктна мова запитів. Дуже схожий на SQL, але запити виконуються до об'єктів.
3. Metadata - анотації над об'єктами. Набір анотацій, якими ми описуємо метадані відображення. Тоді вже JPA знає який об'єкт в яку таблицю потрібно зберегти. Метадані можна описувати двома способами: XML файл можна через анотації.

Основні інтерфейси:

1. Інтерфейс Persistence - по суті це клас. У ньому практично немає методів, крім пари статичних. Один з них CreateEntityManagerFactory, тобто він створює EntityManagerFactory. Що він приймає на вхід: persistence unit. По суті, він шукає percistance (конфігурацію), зчитує її і створює EntityManagerFactory на основі конфігурації. Тут ми визначаємо до якої базі даних будемо з'єднуватися, логін, пароль і найголовніше, в конфігурації визначається провайдер (яка конкретна реалізація JPA буде використовуватися). EntityManagerFactory - це інтерфейс, реалізація якого і є провайдер.
2. EntityManagerFactory - це фабрика, яка містить connection до бази, всю конфігурацію, кеш об'єкти ітд. Суть фабрики - створювати EntityManager.
3. EntityManager - це менеджер сутностей. Він визначає сеанс взаємодії з базою даних (один сеанс). Коли, наприклад, потрібно виконати запит, то ми створюємо EntityManager, виконуємо потрібну дію, закриваємо EntityManager. Об'єкти EntityManager дуже леговесние. Можна створювати їх скільки завгодно. По суті, тут просто відкривається підключення до бази днних. Може навіть з'єднання вже буде відкрито і він приєднатися до поточного.
4. EntityManagerFactory - фабрика і вона створюється довго (ініцілізація всіх сутностей, скануються всі класи, зчитуються анотації,

визначаються зв'язку ітд). Тому фабрика повинна бути одна (в основному це так). На одну базу одна фабрика.

5. Для чого Соза EntityManager? У ньому є один об'єкт, який називається EntityTransaction (це інтерфейс). Він визначає три методи. Begin (стартує транзакцію), commit, rollback. Також EntityManager може містити кілька об'єктів Query, по суті, він їх створює.
6. EntityManager містить кілька Entity. Все Entity, які завантажуються з бази даних, наприклад get name, будуть висіти постійно в пам'яті, тобто EntityManager містить кеш цих маленьких об'єктів і поки EntityManager ми не закрили, наступне звернення (get name) не полізе в базу даних.
7. Якщо розглядати hibernate, то там є два рівня кешу: перший - EntityManager, другий кеш - це EntityManagerFactory.

Послідовність виклику методів:

1. Persistance, створюємо createEntityManagerFactory, передаючи параметри Unit. На виході маємо фабрику або нічого.
2. Звертаємося до фабрики і говоримо "Дай мені EntityManager"
3. Потім до EntityManager "Дай мені transaction"
4. У transaction викликаємо метод begin
5. Звертаємося до EntityManager. викликаємо query
6. Query. ResultList
7. Transaction. закриваємо
8. EntityManager. закриваємо
9. Фабрику. Закриваємо

Вимоги до об'єктів домену

Щоб JPA міг нормально працювати, так як не кожен об'єкт може бути збережений в базу даних, потрібно стежити за наступними вимогами:

1. POJO або JavaBean об'єкт
2. Клас повинен бути не final, тобто його ніхто не повинен наслідувати

3. Наявність конструктора за замовчуванням (без параметрів)
4. Implements Serializable, щоб об'єкти могли зберігатися в кеші, в серіалізовані вигляді
5. Наявність полів ідентифікації, тобто жоден об'єкт не може зберігатися в БД, якщо у нього немає primary key.
6. Атрибути - колекції обов'язково оголошені в термінах інтерфейсів колекції, а не конкретних реалізацій. Якщо використовуються будь-які списки, набори, map-и, вони повинні бути класами List, Map, Set ітд., Тобто повинні відповідати інтерфейсів з Java Collection. JPA і hibernate стежать за колекціями.
7. У getters необхідно повертати конкретно посилання на колекцію, а не її копію (не створювати нові колекції). Тому не рекомендується використовувати Array, так як коли ми повертаємо масив треба завжди робити сору, щоб хтось ззовні не міг якщо змінити. Також від масиву складно успадковуватися.

2.6. Вибір первинних ключів

Ідентичні об'єкти - посилання на об'єкти рівні. Порівняння оператором.

Однакові об'єкти - однакові з логічної точки зору, але при цьому мають різні області пам'яті. Порівняння методом equals.

Ідентичні в термінах БД - об'єкти посилаються на одну і ту ж запис однієї таблиці з однаковим primary key.

Бажано використовувати сурагатний первинний ключ - той, який ніяк не відповідає бізнес логіці.

Наприклад, ми зчитуємо з бази один і той же ряд, в підсумку ми отримуємо один об'єкт і другий, тобто, з двох різних запитів ми напівавто два різних об'єкта. Уявімо собі, що в одному з об'єктів ви змінили інформацію: поміняли телефон у користувача. І поки ще нічого не зберігали. Виникає питання: чи вважати ці два об'єкти однаковими чи ні? Якщо їх порівнювати двома одно, то об'єкти виходити різними (різні області пам'яті). Метод equals - дивлячись на те як ми його

реалізуємо. Якщо почнемо порівнювати логічну структуру об'єктів, то вони знову будуть різними, так як у них змінилося одне поле. Питання: як реалізувати порівняння цих двох об'єктів? Відповідь: сурогатний ключ. Ніякого відношення до об'єкту цей ключ не повинен мати, його буде використовувати тільки база даних. Тоді метод equals ми реалізуємо таким чином щоб він порівнював тільки цей ключ. Виходить, що якщо ключ дорівнює, то ці об'єкти однакові. Навіть якщо в об'єктах будуть змінюватися дані відносяться до бізнес логіці, наприклад телефон, то при порівнянні методом equals ці об'єкти будуть рівними.

Управління первинними ключами краще надати JPA.

@GeneratedValue (strategy = GenerationType.

AUTO - автоматичний вибір

IDENTITY - алгоритм ідентичності (зростаючий порядок, можливо з порожніми місцями). Часто використовується. Тут id унікальні в межах однієї таблиці.

SEQUENCE - послідовний алгоритм

TABLE - використовується додаткова таблиця для зберігання останнього призначеного ключа.

)

2.6. Висновки до розділу 2

В даному розділі описано основні концепції та підходи до створення веб-додатків з використанням модулів spring framework, основні підходи до архітектури ентерпрайз систем, модулів (рівнів архітектури). Також описано рівень доступу до даних з використанням специфікації Java Persistence API (JPA) та його реалізації у фреймворках spring data, hibernate. Описано основні принципи об'єктно-реляційного зв'язування (ORM, object-relational mapping) фреймворків приведено приклади, як за допомогою них набагато швидше розробляти модулі доступу до баз даних, ніж з використанням стандартного інтерфейсу JDBC (Java database connectivity). Також описано, як працюють контейнери сервлетів та

протокол HTTP з відображенням даних з мови програмування Java до кінцевого користувача системи.

РОЗДІЛ 3. ОГЛЯД ІНТЕРФЕЙСУ РОЗРОБЛЕНОГО ВЕБ-ДОДАТКУ

3.1. Можливості неавторизованого користувача.

При вході на сайт неавторизованому користувачу доступна лише основна інформація, що включає в себе інформацію про підприємство. При відкритті головного вікна буде видно стартову сторінку з вікном авторизації(рис 3.1).

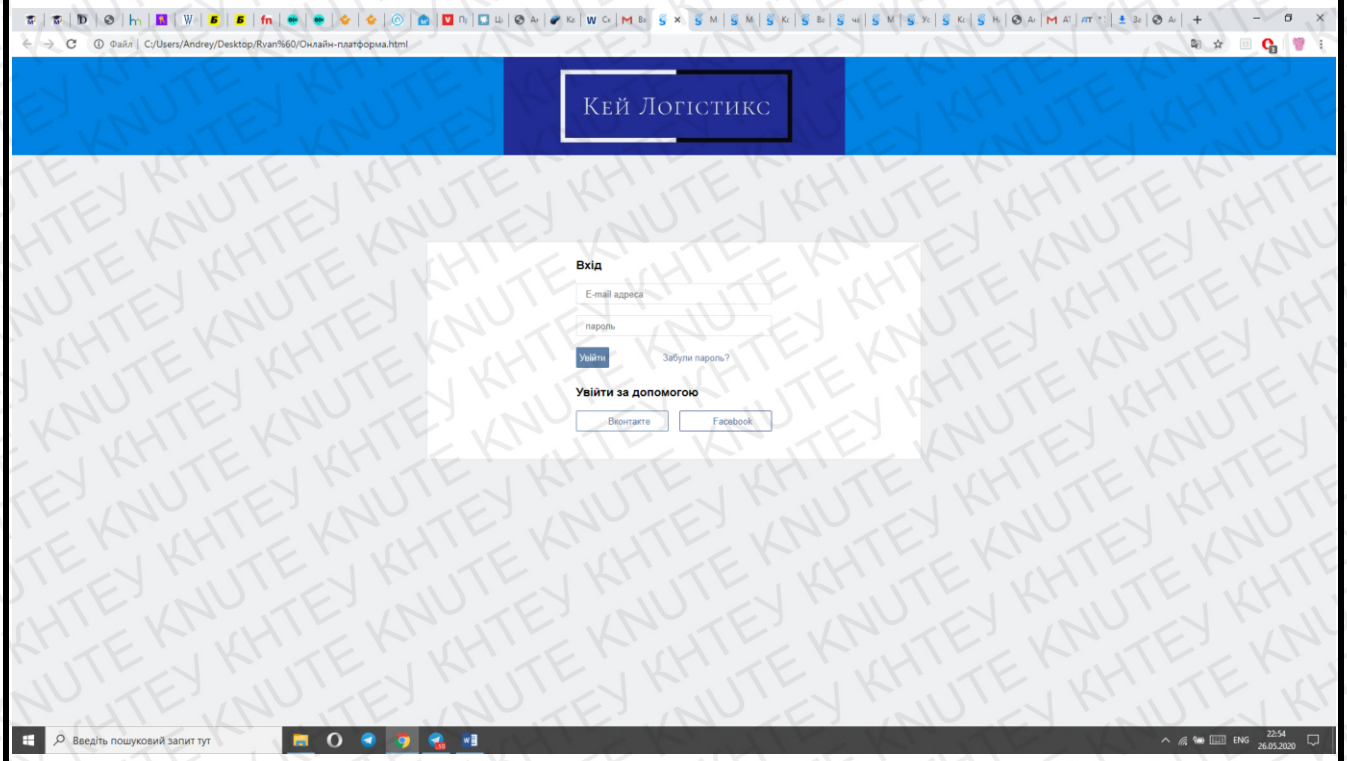

Рисунок 3.1. Стартова сторінка

3.2. Можливості працівника, який проходить навчання чи тренінг

Працівник окрім основної інформації дивитися ще свої оцінки та інформацію про свою групу, для цього йому потрібно авторизуватися під своїм логіном та паролем. Для авторизації потрібно перейти на вкладку авторизації. Для цього потрібно натиснути на кнопку увійти у правому верхньому куті, після чого відкриється сторінка авторизації (рис. 3.2).

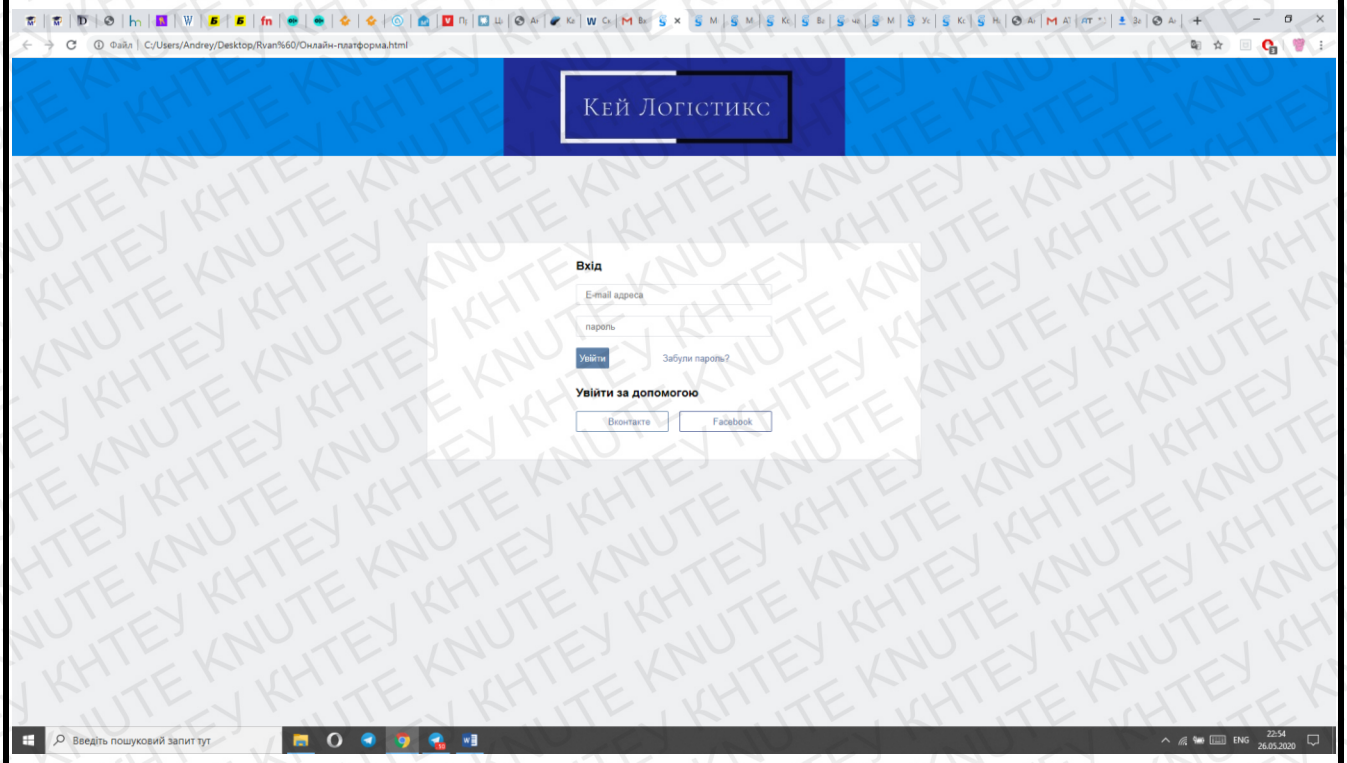

Рисунок 3.2. Вікно «Сторінка авторизації»

Після введення паролю потрібно натиснути кнопку увійти. У випадку невірного логіну чи паролю система сповістить про це, як показано на рисунку 3.3.

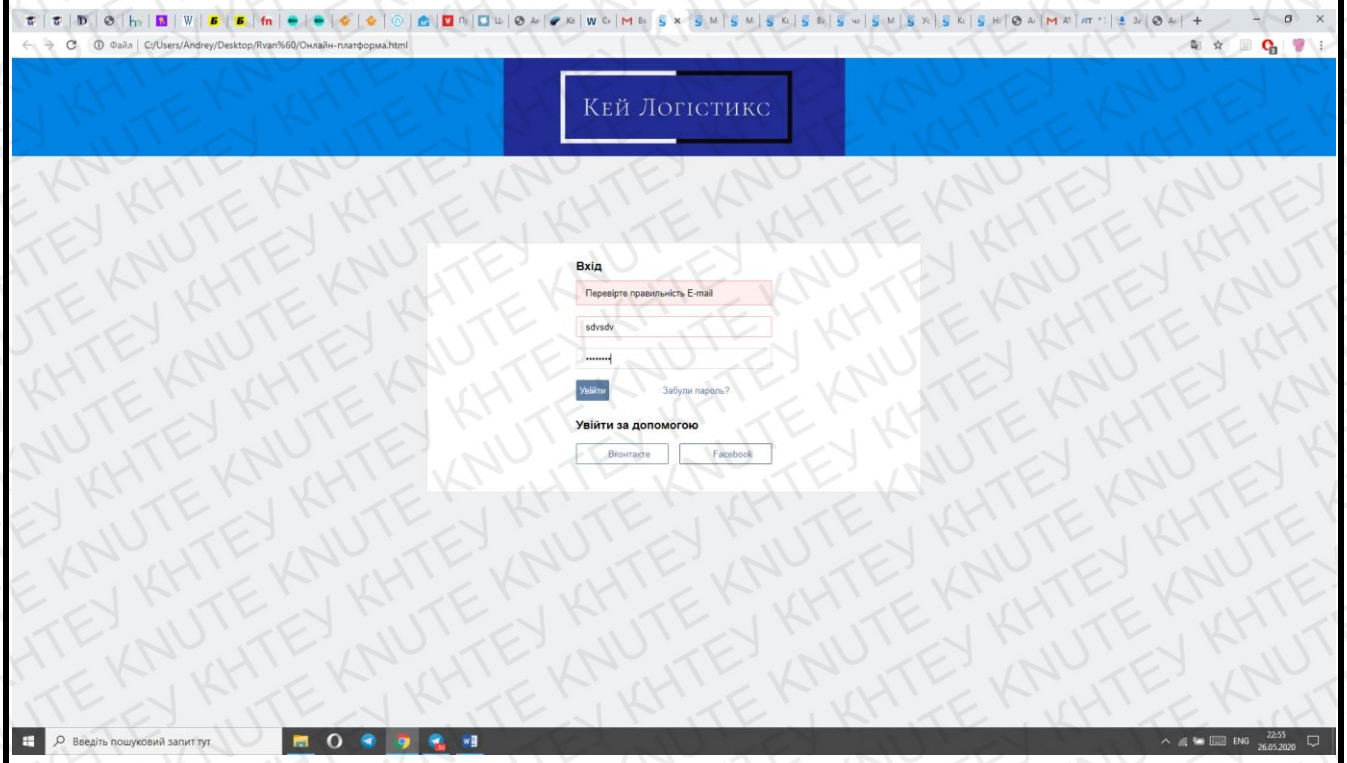

Рисунок 3.7. Попередження про некоретне введення паролю

Після успішного вводу логіна та паролю користувача перекине на головну сторінку, але з’являться курси, якы відкриті для користувача.

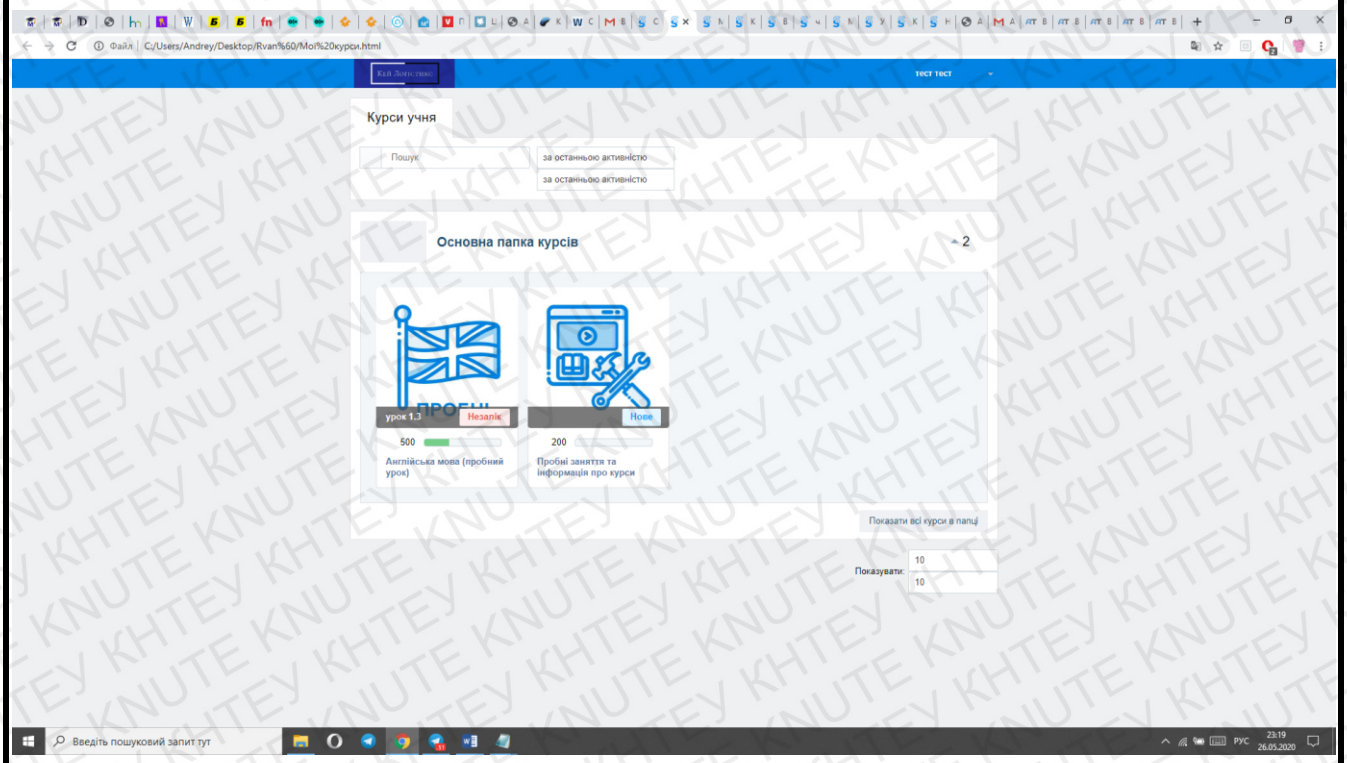

Рисунок 3.3. Вікно «Курси учня»

При виборі курса відкриється сторінка з завданнями для працівника, що зайшов у систему (рис 3.4).

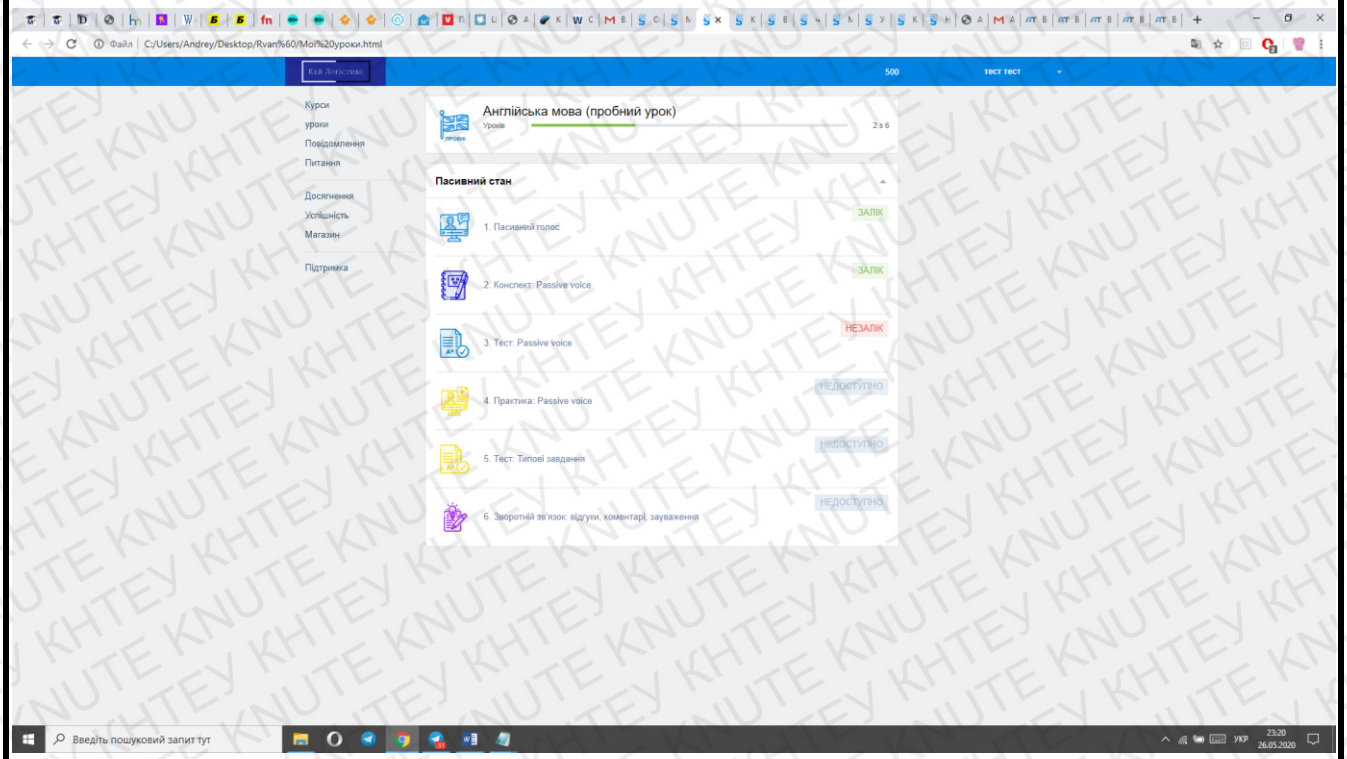

Рисунок 3.5. Вікно з завданнями для працівника

При завдання, працівник має змогу побачити інформацію по курсу (рис 3.5).

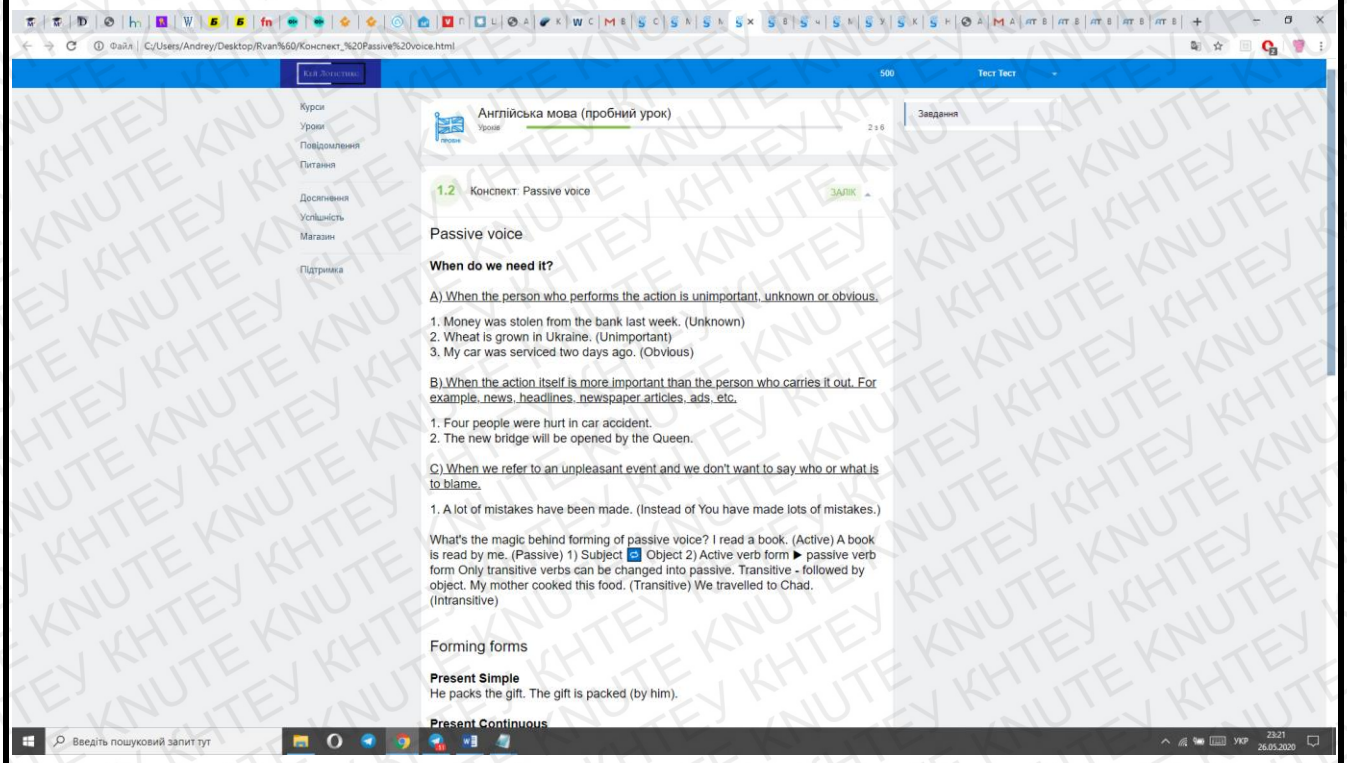

Рисунок 3.5. Інформація для вивчення

Після вивчення теми, працівник повинен пройти тестування по вивченому матеріалу(рис 3.6.)

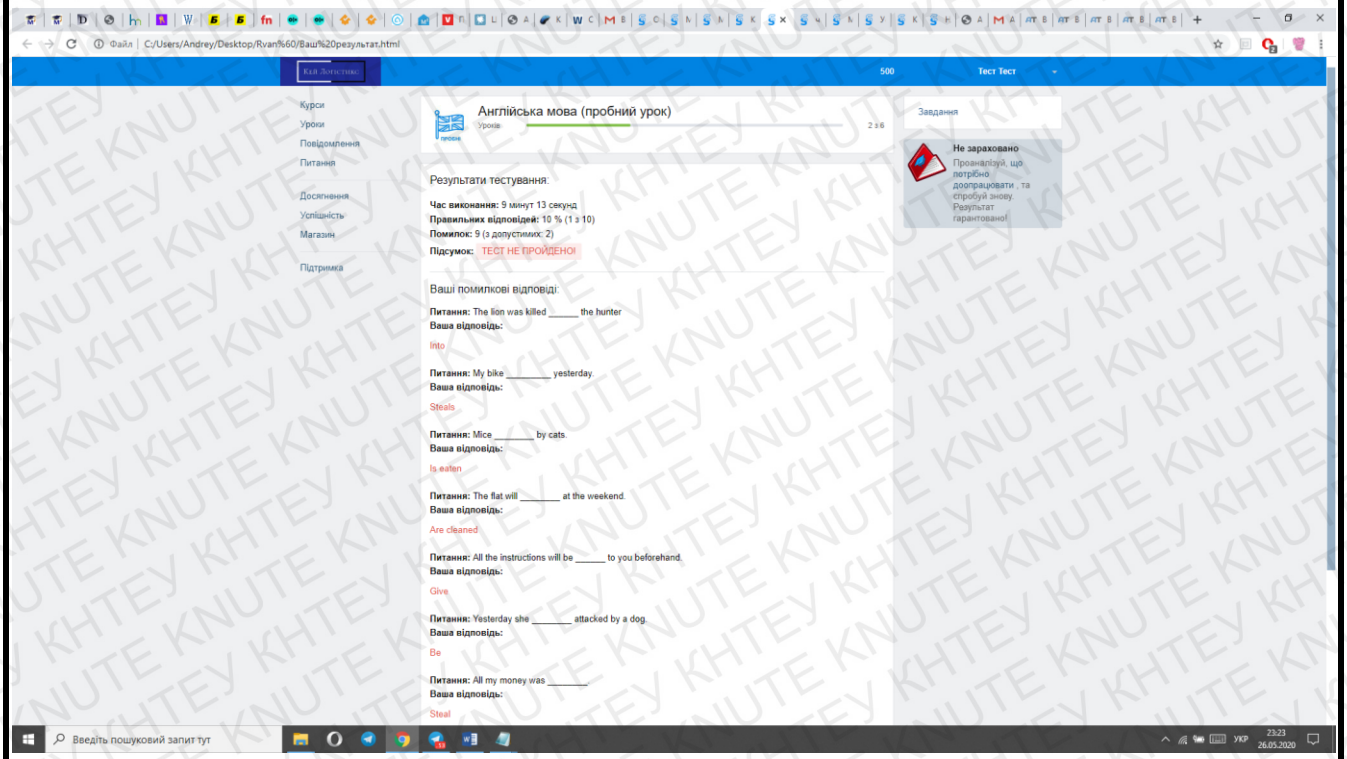

Рисунок 3.6. Тестування по вивченому матеріалу

Також працівник має змогу спілкуватися з наставником по курсу, а також іншими працівниками, які проходять даний курс. Для цього йому необхідно в меню праворуч обрати пункт повідомлення. Після чого на екрані відкриється вікно з діалогами працівника (рис. 3.7.)

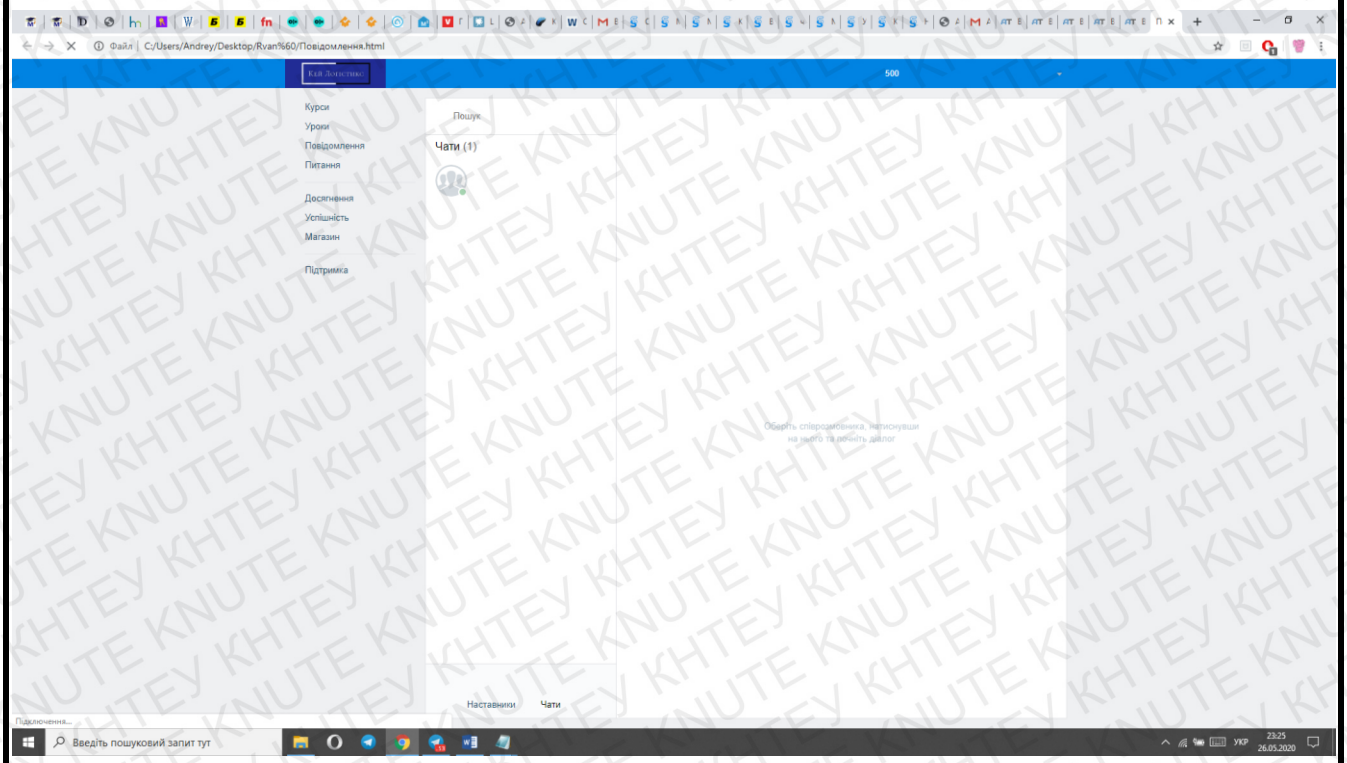

Рисунок 3.7. Вікно «Діалоги працівника»

Для вирішення питань по системі навчання у кожного працівника є пункт меню «Питання», відкривши який, працівник має змогу знайти відповідь на найпоширеніші питання по системі навчання (рис. 3.8.)

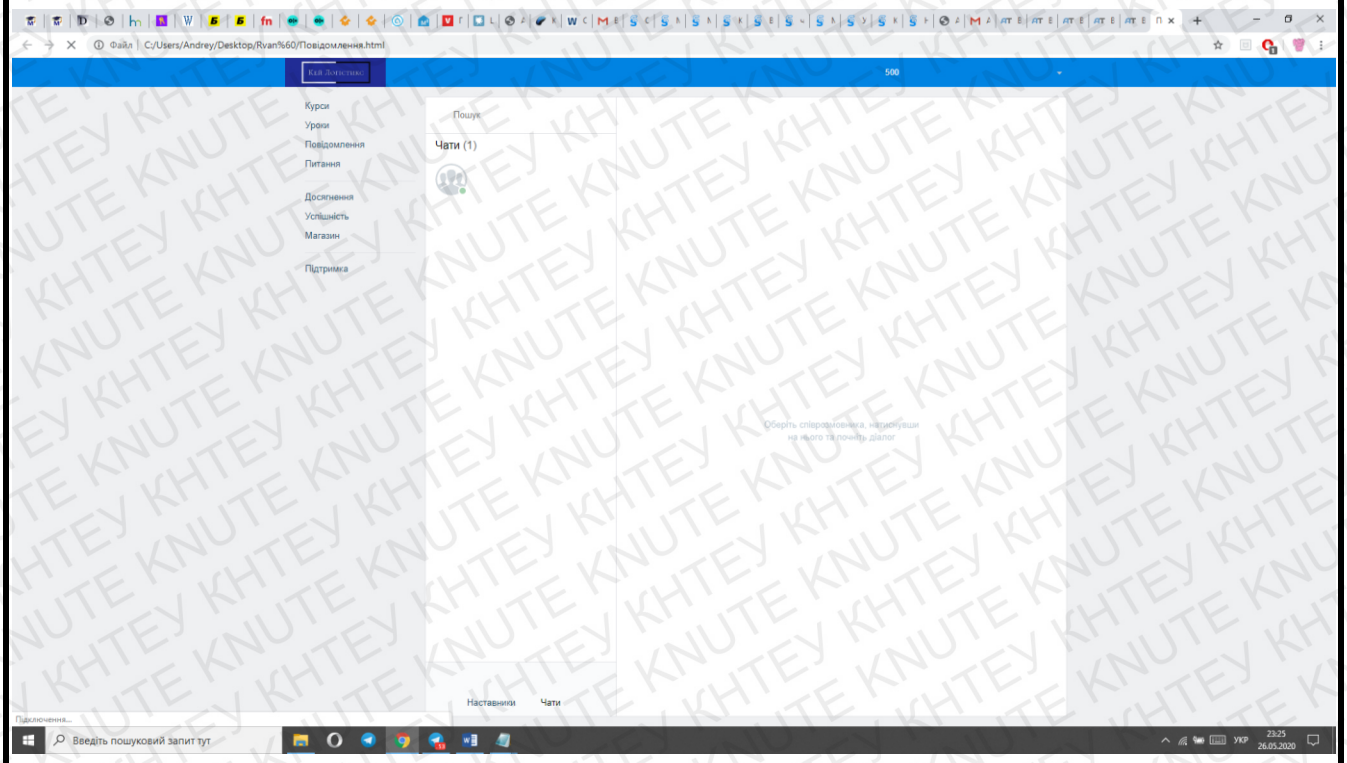

Рисунок 3.8. Вікно «Питання»

Під час навчання кожен працівник отримує досягнення (рис.3.9).

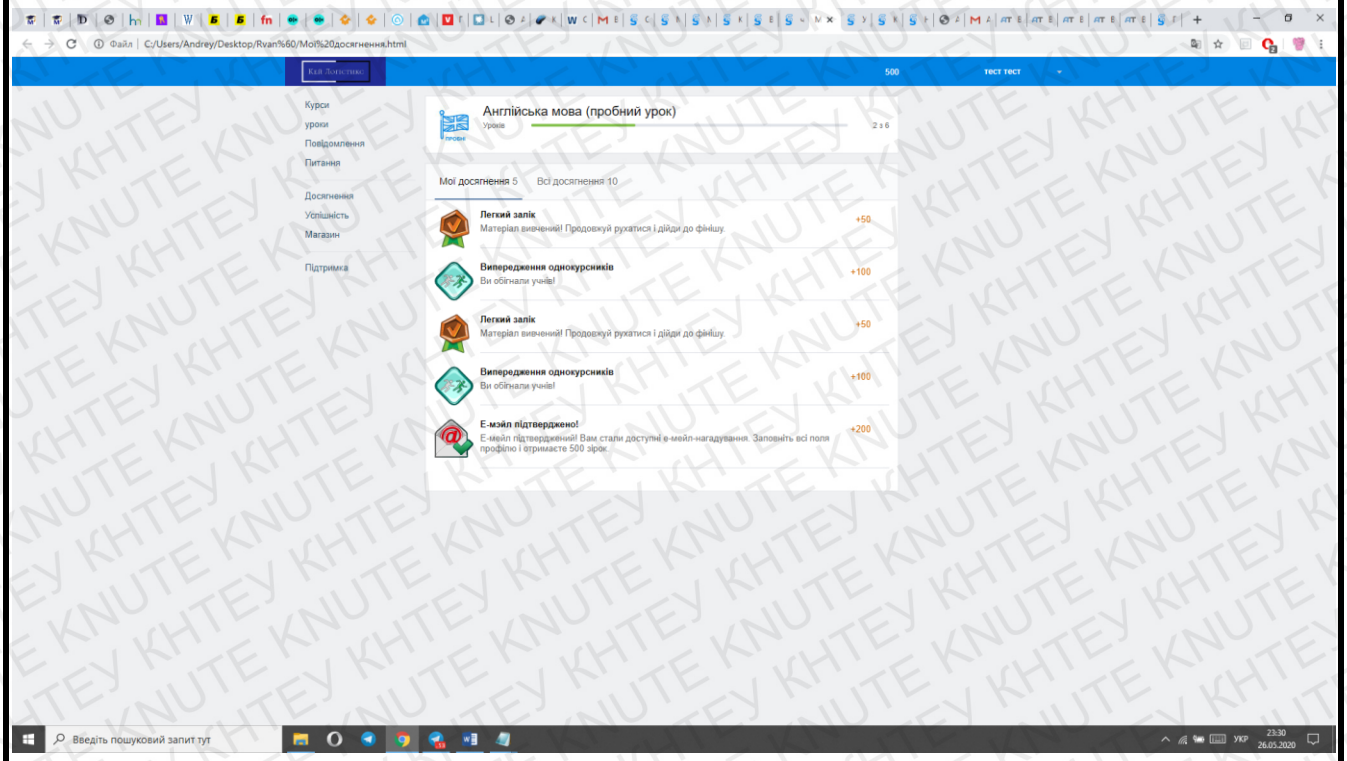

Рисунок 3.9. Вікно «Досягнення»

Також для самоконтролю кожен працівник може переглядати свою успішність (рис 3.10).

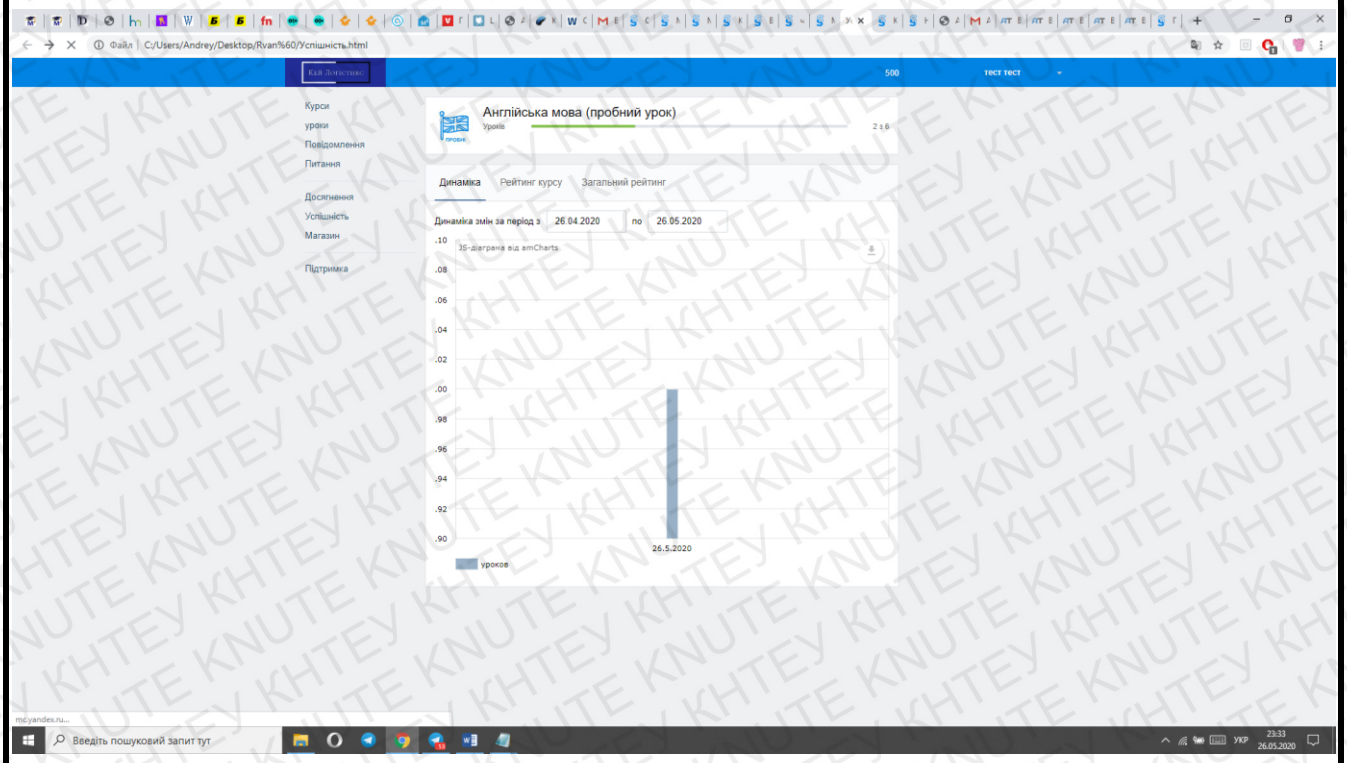

Рисунок 3.10. Вікно «Успішність»

Натискаючи кнопку «підтримка» працівник отримує доступ до контактів служби підтримки, в разі технічних збоїв системи навчання (рис. 3.11.)

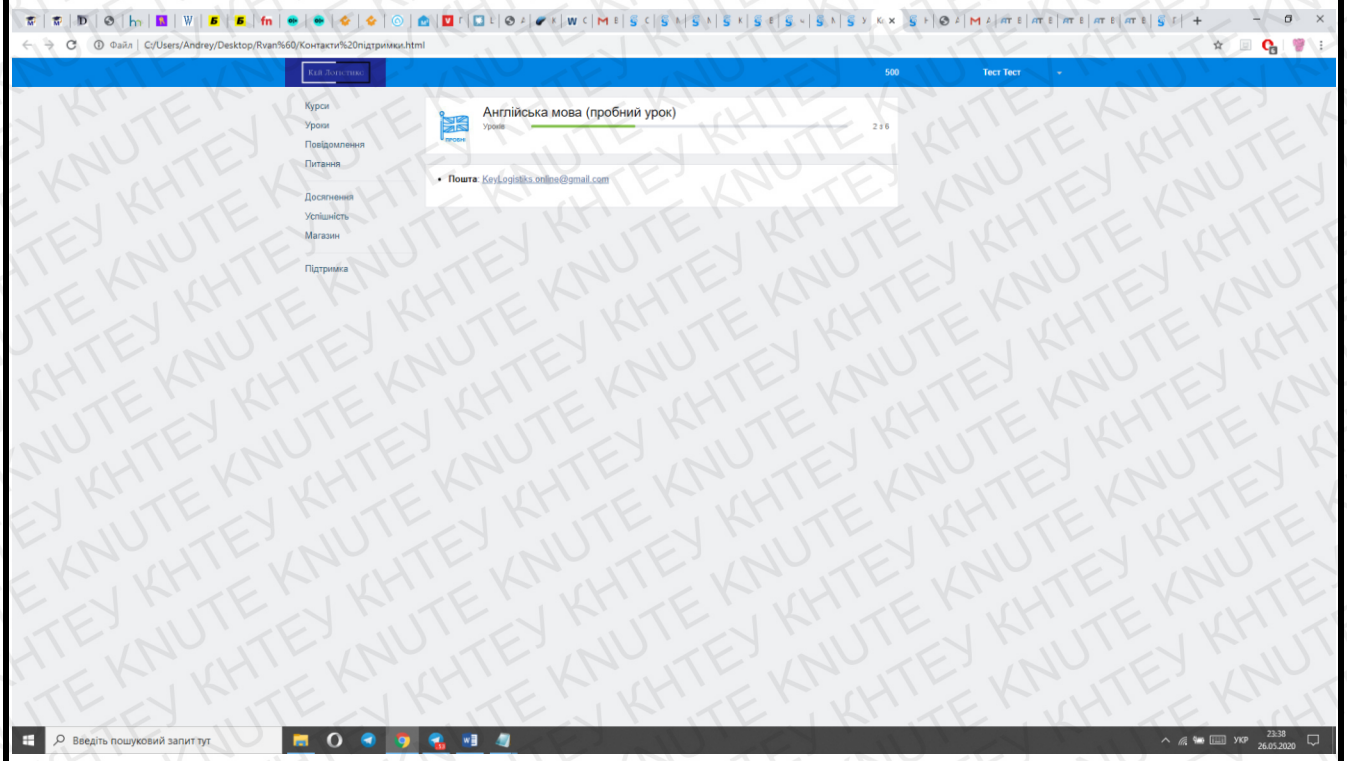

Рисунок 3.11. Вікно з контактами технічної підтримки

3.3. Можливості викладача

Викладач, за допомогою даної системи може дивитися свої предмети та виставляти оцінки працівникам. При авторизації викладача перекине на основне меню з такою кнопкою у верхній панелі, як мої предмети, при натисканні на яку відкриється меню з предметами викладача.

Для того щоб виставити оцінку по предмету потрібно знайти потрібний предмет за групою та натиснути кнопку виставити, після цього відкриється вікно із працівниками, щоб виставити їм оцінки.

Для того щоб виставити оцінку потрібно у полі вводу модульної оцінки, або оцінки за залік або екзамен виставити відповідну оцінку та натиснути кнопку змінити. За замовчуванням, якщо оцінка не стоїть, то буде стояти 0. При натисканні кнопки змінити сторінка оновиться з новими даними про оцінку.

3.4. Можливості адміністратора

Найбільше привілегій в системі має, звичайно, адмиіністратор,

Про авторизації з роллю адміністратора, людина побачить у контекстному меню зверху основні функції, що може робити адміністратор (рис 3.12):

- Редагування курсів (кнопка «Курси»)
- Перегляд та налаштування домашніх завдань для працівників (кнопка «Домашні завдання»)
- Перегляд та налаштування правил формування успішності (кнопка «Щоденник успіху»)
- Налаштування та створення уроків для працівників (кнопка «Уроки»)
- Додавання, оновлення, редагування та видалення як всього списку, так і кожного працівника (учня) окремо (кнопка «Учні»)
- Додавання, оновлення, редагування та видалення як всього списку, так і кожного наставника окремо (кнопка «Наставники»)
- Моніторинг процесу навчання (кнопка «Аналітика»)
- Налаштування каналів інформування (кнопка «Повідомлення»)

- Електронні листи від працівників (учнів) та наставників (кнопка «E-mail повідомлення»
- Створення та редагувааня відповідей на часті питання по системі навчання (кнопка «Часті питання»)
- Адміністрування групових чатів працівників (кнопка «Групові чати»)
- Налаштування курсу (кнопка «Налаштування курсу»)
- Редагування ролей та прав працівників (кнопка «Права студента»)
- Налаштування інших параметрів (кнопка «Автоматизація»)

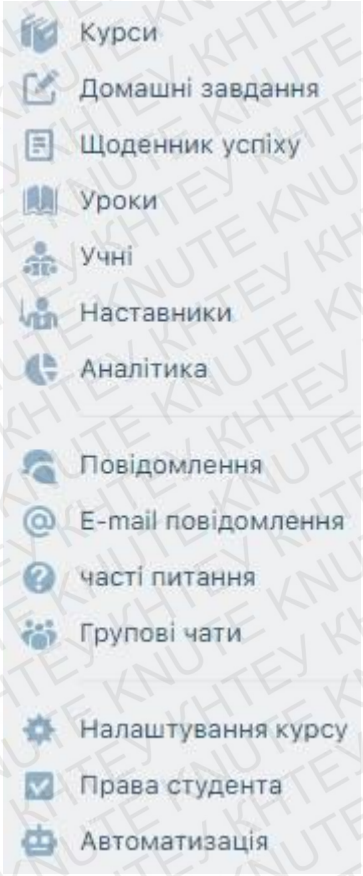

Рисунок 3.11. Можливості адміністратора

Також адміністратору необхідно створювати особисті картки кожного працівника, які містять інформацію про нього. Для цього реалізовано інтерфейс додавання працівника в базу даних (рис.3.12.)

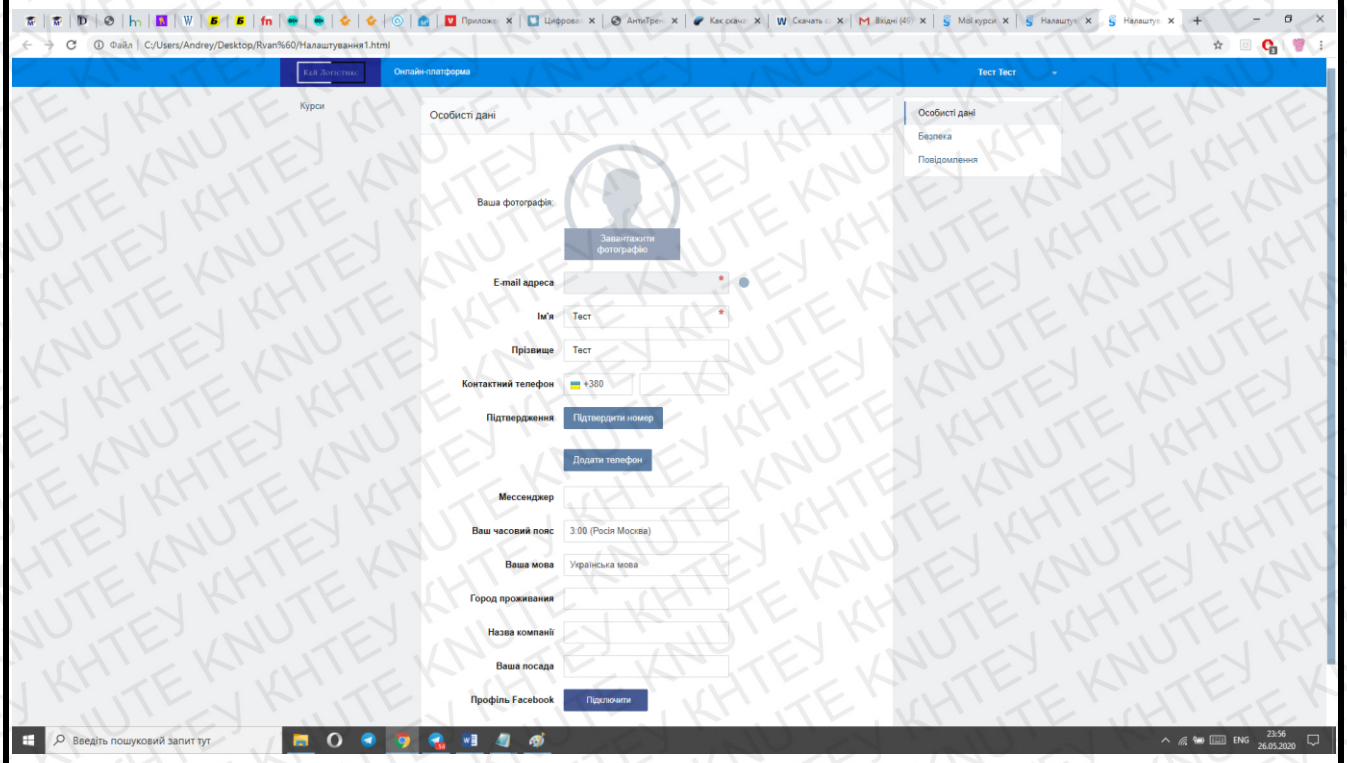

Рисунок 3.12. Вікно додавання нового працівника в систему навчання

При необхідності адміністратор також може додати новий навчальний предмет, та призначити викладача, який буде проводити заняття. Інтерфейс додавання предмету .

При необхідності додати до бази даних нового наставника є можливість зробити це у вікні додавання викладача.

Також реалізовано можливість змінювати чи видаляти дані про працівника. Під час видалення працівника зявляється вікно для підтвердження видалення, для запобігання випадкового видалення.

У веб-додатку реалізовано можливість виконання звітів, які можна переглянути у відповідному вікні або ж зберегти у вигляді файлу з розширенням «.xls» для подальшої зручної роботи зі звітом поза межами програмного комплексу.

Також у адміністратора є всі права доступу, як у викладачів та працівників.

3.5. Висновки до розділу 3

В 3 розділі було розроблено веб-додаток, який дозволяє отримувати зручний доступ до бази даних. Редагувати, додавати та видаляти записи в БД. Також фонкціонал розробленого веб-додатку залежить від того, хто саме користується системою: працівник, викладач чи адміністратор. Під час створення додатку реалізовано звернення до бази даних за допомогою захищених транзакцій, що значно збільшили безпеку використання, та знизило вірогідність небажаного втручання. Головними перевагами розробленого веб-додатку є захищеність, модульність, що дозволить у майбутньому змінювати чи додавати функціонал без внесення змін в ядро програми, швидкодія, а також зручний інтерфейс користувача.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Найбільш складним завданням при роботі з дорослою аудиторією є формування нових установок або зміна наявних. Навчання (особливо групове) часто призводить до зміни поглядів людини на існуючі проблеми і звичні шляхи їх вирішення, до переосмислення власного досвіду і поведінкових стереотипів. Тренер повинен організувати діяльність групи таким чином, щоб участь в ній стимулювало формування у її членів нових підходів, поглядів, оцінок, відносин і установок, які допоможуть змінити їх звичне поведінку на робочому місці. З іншого боку, він нічого не може нав'язатидорослим людям: спроба запропонувати - нехай і ефективний, але єдино вірний погляд - викликає у аудиторії тільки негативну реакцію. Результати навчання тут можна виміряти лише побічно, спостерігаючи за поведінкою учнів.

Найчастіше в тренінгах використовуються наступні методи: лекція, семінар, розбір конкретної ситуації (case study), гра, навчальний кіно-і відеофільм, психогімнастика, майстерня (workshop), групова дискусія (див. Таблицю).

Вирішено покласти на рівень веб просту задачу перетворення даних між HTTP і Java. Єдине, що ще можна покласти на веб-рівень - це логіку зміни екранів. Той самий шаблон проектування MVC (Model-View-Controller).

Описано рівень доступу до даних з використанням специфікації Java Persistence API (JPA) та його реалізації у фреймворках spring data, hibernate. Описано основні принципи об'єктно-реляційного зв'язування (ORM, object-relational mapping) фреймворків приведено приклади, як за допомогою них набагато швидше розробляти модулі доступу до баз даних, ніж з використанням стандартного інтерфейсу JDBC (Java database connectivity).

Також було розроблено веб-додаток . Під час створення додатку реалізовано звернення до бази даних за допомогою захищених транзакцій, що значно збільшили безпеку використання, та знизило вірогідність небажаного втручання. Головними перевагами розробленого веб-додатку є захищеність, модульність, що дозволить у майбутньому змінювати чи додавати функціонал без внесення змін в ядро програми, швидкодія, а також зручний інтерфейс користувача.

Список використаних джерел

1. Лазарев А.А. Решение NP-трудной задачи теории расписаний минимизации суммарного запаздывания // Журнал Вычислительной математики и математической физики – 2007. том 47, N.6. – С. 1087– 1098
2. Лазарев А.А., Гафаров Е.Р. Теория расписаний. Минимизация суммарного запаздывания для одного прибора. // Научное издание, М.: Вычислительный центр им. А.А. Дороницына РАН, 2006. - 134 с.
3. Лазарев А.А., Гафаров Е.Р. Теория расписаний. Исследование задач с отношениями предшествования и ресурсными ограничениями. // Научное издание, М.: Вычислительный центр им. А.А. Дороницына РАН, 2007. – 80 с.
4. Лазарев А.А., Гафаров Е.Р. Теория расписаний задачи и алгоритмы. М.: Наука, 2009.
5. Танаев В.С., Шкурба В.В. Введение в теорию расписаний. М.: Наука, 1975.
6. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы. М.: Наука. Гл. ред. физ.-мат. лит., 1984. 384 с.
7. Танаев В.С., Сотсков Ю.Н., Струсевич В.А. Теория расписаний. Многостадийные системы. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 328 с.
8. Танаев В.С., Ковалев М.Я., Шафранский Я.М. Теория расписаний. Групповыетехнологии. Минск: Институт технической кибернетики НАН Беларуси, 1998. 290 с.
9. Alon N., Woeginger G.J., Yadid T. Approximation schemes for scheduling on parallel machines // J. of Scheduling.– 1998.– V. 1.– P. 55 – 66.
10. Van de Akker J.M., Hoogeveen J.A., van de Velde S.L. Parallel machine scheduling by column generation // Oper. Res.– 1999.– V. 47, N 6.– P. 862 – 872.
11. Van de Akker J.M., Hurkens C.A.J., Savelsbergh M.W.P. Timeindexed formulations for single-machine scheduling problems: column generation // INFORMS J. on Computing.– 2000.– V. 12, N 2.– P. 111 – 124.
12. Baptiste Ph., Le Pape C., Nuijten W. Constraint-based scheduling: applying constraint programming to scheduling problems // Kluwer Academic Publishers, 2001.– 198 p.
13. Brooks G.N., White C.R. An algorithm for finding optimal or near – optimal solutions to the production scheduling problem // J. Ind. Eng.– 1965.– V. 16, N 1.– P. 34 – 40.
14. Brucker P. Scheduling Algorithms. Germany: Springer-Verlag 2001. 365 p.

15. Brucker P., Knust S. Complex scheduling Springer-Verlag Berlin, Heidelberg, Germany, 2006.Oracle – Hardware and Software docs. – Режим доступу: www.oracle.com/. - Дата доступу: 19.05.2016.
16. Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm) . - Дата доступу: 08.05.2016.
17. Noureddin Sadawi. – Режим доступу: https://www.youtube.com/channel/UCNYv4HA3WjV3gZGLfBehRWQ. - Дата доступу: 27.05.2016.
18. SiteMarker.Ru Интернет технологии. – Режим доступу: http://sitemaker.ru/technologies/database/mysqlvspostgresql/. - Дата доступу: 19.05.2016.
19. Spring Framework Overview. – Режим доступу: http://www.tutorialspoint.com/spring/spring_overview.htm. - Дата доступу: 02.06.2016.
20. Принцип MVC в веб программировании. – Режим доступу: http://folkprog.net/printsip-mvc-u-web-programmirovanii/.
21. Spring security. – Режим доступу: http://projects.spring.io/spring-security/.
22. Hibernate (framework). – Режим доступу: https://en.wikipedia.org/wiki/Hibernate_(framework).

Додатки

Файл домену

```
package com.knute.domain.entity;

import javax.persistence.*;
import java.util.Date;
import java.util.Objects;

@Entity
@Table(name = "previous_place_of_study", schema = "public", catalog = "kei_logist")
public class PreviousPlaceOfStudy {
    private int prevStudyId;
    private String studyDocument;
    private String studyPlace;
    private String orderOfEnrollment;
    private Date orderOfEnrollmentDateFrom;
    private Date dateOfAdmission;
    private String enrollmentCondition;
    private String enrollmentFinancing;
    private String specialEnrollmentCondition;
    private String employmentHistoryBook;
    private String idCode;
    private String documentType;
    private String series;
    private String docNumber;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "prev_study_id", nullable = false)
```

```
public int getPrevStudyId() {
    return prevStudyId;
}

public void setPrevStudyId(int prevStudyId) {
    this.prevStudyId = prevStudyId;
}

@Basic
@Column(name = "study_document", nullable = false, length = -1)
public String getStudyDocument() {
    return studyDocument;
}

public void setStudyDocument(String studyDocument) {
    this.studyDocument = studyDocument;
}

@Basic
@Column(name = "study_place", nullable = false, length = -1)
public String getStudyPlace() {
    return studyPlace;
}

public void setStudyPlace(String studyPlace) {
    this.studyPlace = studyPlace;
}

@Basic
@Column(name = "order_of_enrollment", nullable = false, length = -1)
```

```
public String getOrderOfEnrollment() {
   return orderOfEnrollment;
}

public void setOrderOfEnrollment(String orderOfEnrollment) {
   this.orderOfEnrollment = orderOfEnrollment;
}

@Basic
@Column(name = "order_of_enrollment_date_from", nullable = false)
public Date getOrderOfEnrollmentDateFrom() {
   return orderOfEnrollmentDateFrom;
}

public void setOrderOfEnrollmentDateFrom(Date orderOfEnrollmentDateFrom) {
   this.orderOfEnrollmentDateFrom = orderOfEnrollmentDateFrom;
}

@Basic
@Column(name = "date_of_admission", nullable = false)
public Date getDateOfAdmission() {
   return dateOfAdmission;
}

public void setDateOfAdmission(Date dateOfAdmission) {
   this.dateOfAdmission = dateOfAdmission;
}

@Basic
@Column(name = "enrollment_condition", nullable = false, length = -1)
```

```
public String getEnrollmentCondition() {
    return enrollmentCondition;
}

public void setEnrollmentCondition(String enrollmentCondition) {
    this.enrollmentCondition = enrollmentCondition;
}

@Basic
@Column(name = "enrollment_financing", nullable = false, length = -1)
public String getEnrollmentFinancing() {
    return enrollmentFinancing;
}

public void setEnrollmentFinancing(String enrollmentFinancing) {
    this.enrollmentFinancing = enrollmentFinancing;
}

@Basic
@Column(name = "special_enrollment_condition", nullable = true, length = -1)
public String getSpecialEnrollmentCondition() {
    return specialEnrollmentCondition;
}

public void setSpecialEnrollmentCondition(String specialEnrollmentCondition) {
    this.specialEnrollmentCondition = specialEnrollmentCondition;
}

@Basic
@Column(name = "employment_history_book", nullable = true, length = -1)
```

```
public String getEmploymentHistoryBook() {
    return employmentHistoryBook;
}

public void setEmploymentHistoryBook(String employmentHistoryBook) {
    this.employmentHistoryBook = employmentHistoryBook;
}

@Basic
@Column(name = "id_code", nullable = false, length = -1)
public String getIdCode() {
    return idCode;
}

public void setIdCode(String idCode) {
    this.idCode = idCode;
}

@Basic
@Column(name = "document_type", nullable = false, length = -1)
public String getDocumentType() {
    return documentType;
}

public void setDocumentType(String documentType) {
    this.documentType = documentType;
}

@Basic
@Column(name = "series", nullable = false, length = -1)
```

```
    public String getSeries() {
        return series;
    }

    public void setSeries(String series) {
        this.series = series;
    }

    @Basic
    @Column(name = "doc_number", nullable = false, length = -1)
    public String getDocNumber() {
        return docNumber;
    }

    public void setDocNumber(String docNumber) {
        this.docNumber = docNumber;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        PreviousPlaceOfStudy that = (PreviousPlaceOfStudy) o;
        return prevStudyId == that.prevStudyId &&
                Objects.equals(studyDocument, that.studyDocument) &&
                Objects.equals(studyPlace, that.studyPlace) &&
                Objects.equals(orderOfEnrollment, that.orderOfEnrollment) &&
                Objects.equals(orderOfEnrollmentDateFrom,
that.orderOfEnrollmentDateFrom) &&
                Objects.equals(dateOfAdmission, that.dateOfAdmission) &&
```

```
            Objects.equals(enrollmentCondition, that.enrollmentCondition) &&
            Objects.equals(enrollmentFinancing, that.enrollmentFinancing) &&
            Objects.equals(specialEnrollmentCondition,
that.specialEnrollmentCondition) &&
            Objects.equals(employmentHistoryBook,      that.employmentHistoryBook)
&&
            Objects.equals(idCode, that.idCode) &&
            Objects.equals(documentType, that.documentType) &&
            Objects.equals(series, that.series) &&
            Objects.equals(docNumber, that.docNumber);
    }

    @Override
    public int hashCode() {

        return Objects.hash(prevStudyId, studyDocument, studyPlace,
orderOfEnrollment, orderOfEnrollmentDateFrom, dateOfAdmission,
enrollmentCondition, enrollmentFinancing, specialEnrollmentCondition,
employmentHistoryBook, idCode, documentType, series, docNumber);
    }

    public String toString() {
        return "PreviousPlaceOfStudy{" +
            "prevStudyId=" + prevStudyId +
            ", studyDocument='" + studyDocument + '\'' +
            ", studyPlace='" + studyPlace + '\'' +
            ", orderOfEnrollment='" + orderOfEnrollment + '\'' +
            ", orderOfEnrollmentDateFrom=" + orderOfEnrollmentDateFrom +
            ", dateOfAdmission=" + dateOfAdmission +
            ", enrollmentCondition='" + enrollmentCondition + '\'' +
```

```
            ", enrollmentFinancing='" + enrollmentFinancing + '\'' +
            ", specialEnrollmentCondition='" + specialEnrollmentCondition + '\'' +
            ", employmentHistoryBook='" + employmentHistoryBook + '\'' +
            ", idCode='" + idCode + '\'' +
            ", documentType='" + documentType + '\'' +
            ", series='" + series + '\'' +
            ", docNumber='" + docNumber + '\'' +
            '}';
    }
}
```

Файл pom.xml

```
    <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://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.dec</groupId>
    <artifactId>dec</artifactId>
    <packaging>war</packaging>
    <version>0.1</version>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <spring.version>4.3.10.RELEASE</spring.version>
        <spring.security.version>4.2.3.RELEASE</spring.security.version>
        <spring.data.version>1.11.6.RELEASE</spring.data.version>
        <hibernate.version>5.2.10.Final</hibernate.version>
        <hibernate.validator.version>6.0.2.Final</hibernate.validator.version>
        <postgres.driver.version>42.1.4</postgres.driver.version>
```

```
    </properties>


    <dependencies>
   <!-- spring core and web/webmvc -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- spring security -->
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-
security-taglibs -->
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-taglibs</artifactId>
      <version>4.2.3.RELEASE</version>
    </dependency>
```

```
<!-- spring transactions -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
  <version>${spring.version}</version>
</dependency>

<!-- spring data -->
        <dependency>
                <groupId>org.springframework.data</groupId>
                <artifactId>spring-data-jpa</artifactId>
                <version>${spring.data.version}</version>
        </dependency>

<!-- postgres driver -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
                <version>${postgres.driver.version}</version>
                <exclusions>
    <exclusion>
      <artifactId>slf4j-api</artifactId>
      <groupId>org.slf4j</groupId>
    </exclusion>
    <exclusion>
      <artifactId>slf4j-simple</artifactId>
      <groupId>org.slf4j</groupId>
    </exclusion>
  </exclusions>
```

```
</dependency>

<!-- hibernate core and validator -->
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>${hibernate.version}</version>
  <exclusions>
    <exclusion>
      <artifactId>org.jboss.logging</artifactId>
      <groupId>jboss-logging</groupId>
    </exclusion>
  </exclusions>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-jpamodelgen</artifactId>
        <version>${hibernate.version}</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>${hibernate.version}</version>
        <exclusions>
            <exclusion>
                <artifactId>org.jboss.logging</artifactId>
                <groupId>jboss-logging</groupId>
            </exclusion>
        </exclusions>
```

```
        </dependency>
        <dependency>
                <groupId>org.hibernate.javax.persistence</groupId>
                <artifactId>hibernate-jpa-2.1-api</artifactId>
                <version>1.0.0.Final</version>
        </dependency>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-validator</artifactId>
                <version>${hibernate.validator.version}</version>
    <exclusions>
      <exclusion>
        <artifactId>org.jboss.logging</artifactId>
        <groupId>jboss-logging</groupId>
      </exclusion>
    </exclusions>
        </dependency>

<!-- slf4j and logback -->
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>jcl-over-slf4j</artifactId>
   <version>1.7.22</version>
</dependency>

<dependency>
   <groupId>ch.qos.logback</groupId>
   <artifactId>logback-classic</artifactId>
   <version>1.1.9</version>
</dependency>
```

```
<!-- jstl and servlet-api -->
        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>jstl</artifactId>
                <version>1.2</version>
        </dependency>
        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>javax.servlet-api</artifactId>
                <version>3.1.0</version>
        </dependency>

        <dependency>
                <groupId>org.apache.poi</groupId>
                <artifactId>poi-ooxml</artifactId>
                <version>3.17</version>
        </dependency>

    </dependencies>

    <build>
        <finalName>dec</finalName>
        <plugins>
                <plugin>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <version>3.0</version>
                        <configuration>
                                <source>1.8</source>
                                <target>1.8</target>
```

```
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.6</version>
                <configuration>

    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.bsc.maven</groupId>
                <artifactId>maven-processor-plugin</artifactId>
                <executions>
                    <execution>
                        <id>process</id>
                        <goals>
                            <goal>process</goal>
                        </goals>
                        <phase>generate-sources</phase>
                        <configuration>
                            <processors>

    <processor>org.hibernate.jpamodelgen.JPAMetaModelEntityProcessor</proce
ssor>
                            </processors>
                        </configuration>
                    </execution>
                </executions>
```

```
                </plugin>
            </plugins>
        </build>
</project>
```

Файл конфігурації

```
package com.knute.config;

import org.hibernate.jpa.HibernatePersistenceProvider;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.LocalEntityManagerFactoryBean;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import javax.sql.DataSource;
import java.util.Properties;

@Configuration
@EnableTransactionManagement
@EnableJpaRepositories("com.knute")
public class DataConfig {
   private static final String PROP_HIBERNATE_DIALECT = "hibernate.dialect";
   private static final String PROP_HIBERNATE_SHOW_SQL =
"hibernate.show_sql";
```

```
@Value("${jdbc.driver}")
private String driver;

@Value("${jdbc.url}")
private String url;

@Value("${jdbc.username}")
private String username;

@Value("${jdbc.password}")
private String password;

@Value("${hibernate.dialect}")
private String dialect;

@Value("${hibernate.show_sql}")
private String showSql;

@Bean
public DataSource dataSource() {
   DriverManagerDataSource dataSource = new DriverManagerDataSource();

   dataSource.setDriverClassName(driver);
   dataSource.setUrl(url);
   dataSource.setUsername(username);
   dataSource.setPassword(password);

   return dataSource;
}
```

```
    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
        LocalContainerEntityManagerFactoryBean entityManagerFactoryBean = new
LocalContainerEntityManagerFactoryBean();
        entityManagerFactoryBean.setDataSource(dataSource());

entityManagerFactoryBean.setPersistenceProviderClass(HibernatePersistenceProvide
r.class);
        entityManagerFactoryBean.setPackagesToScan("com.knute");
        entityManagerFactoryBean.setJpaProperties(getHibernateProperties());

        return entityManagerFactoryBean;
    }

    @Bean
    public JpaTransactionManager transactionManager() {
        JpaTransactionManager transactionManager = new JpaTransactionManager();

transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());

        return transactionManager;
    }

    private Properties getHibernateProperties() {
        Properties properties = new Properties();
        properties.put(PROP_HIBERNATE_DIALECT, dialect);
        properties.put(PROP_HIBERNATE_SHOW_SQL, showSql);

        return properties;
    }
```

}

Файл контроллер

```
package com.knute.controller;

import com.knute.domain.dto.SessionResultDTO;
import com.knute.domain.entity.*;
import com.knute.repository.PositionRepository;
import com.knute.repository.ScientificDegreeRepository;
import com.knute.repository.ScientificRankRepository;
import com.knute.service.DepartmentService;
import com.knute.service.ResultService;
import com.knute.service.StudentService;
import com.knute.service.TeacherService;
import com.knute.util.UserDetailsUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.*;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Controller
@RequestMapping(value = "/teacher")
```

```
public class TeacherController {
    private final TeacherService teacherService;
    private final StudentService studentService;
    private final ResultService resultService;
    private final ScientificDegreeRepository degreeRepository;
    private final ScientificRankRepository rankRepository;
    private final DepartmentService departmentService;
    private final PositionRepository positionRepository;

    @Autowired
    public TeacherController(TeacherService teacherService,
                             StudentService studentService,
                             ResultService resultService,
                             ScientificDegreeRepository degreeRepository,
                             ScientificRankRepository rankRepository,
                             DepartmentService departmentService,
                             PositionRepository positionRepository) {
        this.teacherService = teacherService;
        this.studentService = studentService;
        this.resultService = resultService;
        this.degreeRepository = degreeRepository;
        this.rankRepository = rankRepository;
        this.departmentService = departmentService;
        this.positionRepository = positionRepository;
    }

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.registerCustomEditor(Date.class, new CustomDateEditor(new
SimpleDateFormat("yyyy-MM-dd"), true, 10));
```

```
    }


    @RequestMapping
    public String getMainTeacherInfo(Model model) {
        model.addAttribute("teacher",
teacherService.getTeacherByLogin(UserDetailsUtil.getUsernameFromContext(Secur
ityContextHolder.getContext())));
        return "teacher";
    }

    @RequestMapping(value = "/set/{subject}", method = RequestMethod.GET)
    public String setMarks(@PathVariable("subject") Integer subjectId, Model model)
{
        System.out.println(subjectId + " " +
UserDetailsUtil.getUsernameFromContext(SecurityContextHolder.getContext()));
        List<SessionResultDTO> list =
resultService.getBySubjectIdAndTeacherLogin(subjectId,
UserDetailsUtil.getUsernameFromContext(SecurityContextHolder.getContext()));
        model.addAttribute("results", list);
        return "setMarks";
    }

    @RequestMapping(value = "/set/{type}/{result}/{subject}", method =
RequestMethod.POST)
    public String setMark(@PathVariable("type") String type, @PathVariable("result")
Integer resultId,
                          @PathVariable("subject") Integer subjectId,
@RequestParam("mark") Integer mark) {
        resultService.updateMark(resultId, type, mark);
```

```
        return "redirect:/teacher/set/" + subjectId;
    }

    @RequestMapping(value = "/add", method = RequestMethod.GET)
    public String addTeacher(Model model) {
        model.addAttribute("degrees", degreesToMap(degreeRepository.findAll()));
        model.addAttribute("positions", positionsToMap(positionRepository.findAll()));
        model.addAttribute("departments",
departmentsToMap(departmentService.getAll()));
        model.addAttribute("ranks", ranksToMap(rankRepository.findAll()));
        model.addAttribute("teacher", new Teacher());
        return "addTeacher";
    }

    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public String addTeacher(@ModelAttribute("teacher") Teacher teacher){
        System.out.println(teacher);
        teacherService.save(teacher);
        return "redirect:/teacher/all";
    }

    @RequestMapping(value = "/all", method = RequestMethod.GET)
    public String getAll(Model model) {
        model.addAttribute("teachers", teacherService.getAll());
        return "teachers";
    }

    private Map<Integer, String> degreesToMap(List<ScientificDegree> degrees) {
```

```
        return
degrees.stream().collect(Collectors.toMap(ScientificDegree::getScientificDegreeId,
ScientificDegree::getDegreeName));
    }

    private Map<Integer, String> positionsToMap(List<PositionEntity> positions) {
        return  positions.stream().collect(Collectors.toMap(PositionEntity::getPositionId,
PositionEntity::getPositionName));
    }

    private Map<Integer, String> ranksToMap(List<ScientificRank> ranks) {
        return
ranks.stream().collect(Collectors.toMap(ScientificRank::getScientificRankId,
ScientificRank::getRankName));
    }

    private Map<Integer, String> departmentsToMap(List<Department> departments)
{
        return
departments.stream().collect(Collectors.toMap(Department::getDepartmentId,
Department::getDepartmentName));
    }
}
```

Файл репозиторію

```
package com.knute.repository;

import com.knute.domain.dto.SessionResultDTO;
import com.knute.domain.entity.StudentResult;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
```

```
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface ResultRepository extends JpaRepository<StudentResult, Integer> {
    String SELECT_QUERY = "select new com.knute.domain.dto.SessionResultDTO(sub.subjectId, r.resultId, s.firstName, s.lastName, s.secondName, sub.assesmentType, " +
            "n.subjectDesc, r.moduleMark, r.finalMark, sub.term, g.year) " +
            "from StudentResult r " +
            "inner join r.studentByStudentId s " +
            "inner join r.subjectBySubjectId sub " +
            "inner join s.decGroupByDecGroupId g " +
            "inner join sub.subjectNameBySubjectNameId n ";

    @Query(SELECT_QUERY +
            "inner join s.decUserByUserId u " +
            "where s.decGroupByDecGroupId.decGroupId = :group and sub.term = :term and u.login = :id " +
            "order by sub.term desc")
    List<SessionResultDTO> getByGroupAndTermByStudentId(@Param("group") int groupId, @Param("term") int term, @Param("id") String id);

    @Query(SELECT_QUERY)
    List<SessionResultDTO> getAll();

//      @Query("select new com.knute.domain.dto.SessionResultDTO(sub.subjectId, r.resultId, s.firstName, s.lastName, s.secondName, sub.assesmentType, " +
//            "n.subjectDesc, r.moduleMark, r.finalMark, sub.term, g.year) " +
```

```
//          "from StudentResult r " +
//          "inner join r.studentByStudentId s " +
//          "inner join r.subjectBySubjectId sub " +
//          "inner join s.decGroupByDecGroupId g " +
//          "inner join sub.subjectNameBySubjectNameId n " +
//          "inner join sub.teacherByLaborantId t1 " +
//          "inner join sub.teacherByLectorId t2 " +
//          "inner join t1.decUserByUserId u1 " +
//          "inner join t2.decUserByUserId u2 " +
//           "where (u1.login = :username or u2.login = :username) and sub.subjectId =
:subjectId " +
//          "order by s.lastName")

   @Query(SELECT_QUERY +
          "inner join sub.teacherByLaborantId t1 " +
                "inner join sub.teacherByLectorId t2 " +
                "inner join t1.decUserByUserId u1 " +
                "inner join t2.decUserByUserId u2 " +
                "where (u1.login = :username or u2.login = :username) and sub.subjectId
= :subjectId " +
                "order by s.lastName")
   List<SessionResultDTO>  getBySubjectIdAndTeacherLogin(@Param("subjectId")
Integer subjectId, @Param("username") String username);

   @Query(SELECT_QUERY + "where g.decGroupId = :groupId")
   List<SessionResultDTO> getByGroupId(@Param("groupId") Integer groupId);

   @Query(SELECT_QUERY + "where g.specialtyBySpecialtyId.specialtyId =
:specialtyId")
```

```
List<SessionResultDTO> getBySpecialtyId(@Param("specialtyId") Integer specialtyId);

@Query(SELECT_QUERY + "where s.studentId = :studentId")
List<SessionResultDTO> getByStudentId(@Param("studentId") Integer studentId);

@Query(SELECT_QUERY +
        "inner join s.decUserByUserId u " +
        "where u.login = :username " +
        "order by sub.term desc")
List<SessionResultDTO> getByStudentUserName(@Param("username") String username);

@Modifying
@Query("update StudentResult s set s.moduleMark = :mark where s.resultId = :resultId")
void updateModuleMark(@Param("resultId") Integer resultId, @Param("mark") Integer mark);

@Modifying
@Query("update StudentResult s set s.finalMark = :mark where s.resultId = :resultId")
void updateFinalMark(@Param("resultId") Integer resultId, @Param("mark") Integer mark);

@Query(SELECT_QUERY + "where g.decGroupId = :groupId and sub.term = :term " +
        "order by sub.term")
List<SessionResultDTO> getByGroupId(@Param("groupId") Integer groupId, @Param("term") Integer term);
```

```
    @Query(SELECT_QUERY + "where g.decGroupId = :groupId")
    List<SessionResultDTO> getAllByGroupId(@Param("groupId") Integer groupId);
}
```

Файл сервісу

```
package com.knute.service;

import com.knute.domain.entity.Teacher;

import java.util.List;

public interface TeacherService {
    List<Teacher> getAll();
    Teacher save(Teacher teacher);
    Teacher getTeacherByLogin(String login);
}
package com.knute.service.impl;

import com.knute.domain.entity.DecGroup;
import com.knute.domain.entity.Department;
import com.knute.domain.entity.Specialty;
import com.knute.repository.DecGroupRepository;
import com.knute.repository.DepartmentRepository;
import com.knute.repository.SpecialtyRepository;
import com.knute.service.DecGroupService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
```

```
@Service
public class DecGroupServiceImpl implements DecGroupService {
   private final DecGroupRepository groupRepository;
   private final DepartmentRepository departmentRepository;
   private final SpecialtyRepository specialtyRepository;

   @Autowired
   public DecGroupServiceImpl(DecGroupRepository groupRepository,
                              DepartmentRepository departmentRepository,
                              SpecialtyRepository specialtyRepository) {
      this.groupRepository = groupRepository;
      this.departmentRepository = departmentRepository;
      this.specialtyRepository = specialtyRepository;
   }

   @Override
   public List<DecGroup> getAll() {
      return groupRepository.findAll();
   }

   @Override
   public DecGroup getById(Integer groupId) {
      return groupRepository.findOne(groupId);
   }

   @Override
   @Transactional
   public DecGroup save(DecGroup group) {
```

```
        Department dep = departmentRepository.findOne(group.getDepartmentByDepartmentId().getDepartmentId());
        group.setDepartmentByDepartmentId(dep);
        Specialty specialty = specialtyRepository.findOne(group.getSpecialtyBySpecialtyId().getSpecialtyId());
        group.setSpecialtyBySpecialtyId(specialty);
        return groupRepository.save(group);
    }

    @Override
    public DecGroup getByUsernameId(String username) {
        return groupRepository.getByUsernameId(username);
    }
}
```