МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт информационных технологий

Кафедра/департамент «Информационные системы»

**Пояснительная записка**

к курсовому проекту

по дисциплине «Практикум по цифровому проектированию»

на тему “Разработка сайта для кондитерской «Figma»”

Выполнил: обучающиеся 3 курса

группы: ИС/б-20-2-о

Направления подготовки

09.03.02

Информационные системы и технологии

Хроменко Д.А.

Семко В.В

« \_\_« \_\_\_\_\_\_\_20 22 г.

Научный руководитель:

«\_\_ « \_\_\_\_\_\_\_\_20 22 г.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« \_\_«\_\_\_\_\_\_\_\_\_ 20 22 г.

Севастополь 2022

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc122685156)

[1. ЗАПУСК ПРОЕКТА 4](#_Toc122685157)

[1.1. Проблема, решаемая в проекте 4](#_Toc122685158)

[1.2. Пользователи и другие вовлеченные стороны 4](#_Toc122685159)

[1.3. Требования к решению 6](#_Toc122685160)

[1.4. Предлагаемое решение 6](#_Toc122685161)

[1.5. Аналоги разрабатываемого решения 6](#_Toc122685162)

[2. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ЕЕ БИЗНЕС-ПРОЦЕССОВ 9](#_Toc122685163)

[2.1. Формирование начальных гипотез 9](#_Toc122685164)

[2.2. AS IS сценарии, описывающие бизнес-процессы в предметной области 10](#_Toc122685165)

[3. РАЗРАБОТКА РЕШЕНИЯ 12](#_Toc122685166)

[3.2. Разработка решения по циклу Деминга 12](#_Toc122685167)

[ЗАКЛЮЧЕНИЕ 14](#_Toc122685168)

[Приложение А 15](#_Toc122685169)

# **ВВЕДЕНИЕ**

В рамках настоящей работы ведется разработка сайта для кондитерской «Figma», позволяющего удобно заказывать кондитерские изделия для клиентов. Сайт разработан как инициативный проект в рамках предметной области малого и среднего бизнеса, предполагающего в своих бизнес-процессах непосредственную связь клиента и исполнителя заказа.

Целью курсового проектирования является выработка и закрепление на практике навыков разработки цифровых проектов, включающих в себя навыки по запуску проекта, исследованию предметной области, разработке решения и упаковке решения, а также углубление практических навыков разработки программных продуктов.

Для достижения поставленной цели на разных этапах курсового проектирования должны быть решены следующие задачи:

* Запуск проекта
* Исследование предметной области и ее бизнес-процессов
* Разработка решения
* Упаковка проекта

1. **ЗАПУСК ПРОЕКТА**
   1. **Проблема, решаемая в проекте**

Нашими целевыми пользователями являются представители малого и среднего бизнеса, нуждающиеся в удобном способе продажи своих продуктов для клиентов. Такими клиентами могут являться любые люди, желающие приобрести кондитерские изделия, которые будут удовлетворять их потребностям. Данная модель отношений «Клиент» – «Исполнитель» сейчас осуществляется не самым приятным образом, так как зачастую аналогичные кондитерские не предоставляют такого сервиса, как им хотелось бы.

* 1. **Пользователи и другие вовлеченные стороны**

Для дальнейшего проектирования были определены пользователи и другие вовлеченные стороны, так или иначе взаимодействующие с проектом. Результат анализа групп пользователей и других вовлеченных сторон представлен в таблице 1.

Таблица 1 – Пользователи и вовлеченные стороны

|  |  |  |  |
| --- | --- | --- | --- |
| Кто? | Требования к решению | Как и когда связываться | Что необходимо узнать/спросить |
| CEO бизнеса | Удобный интерфейс учета заказов, выполняемых всеми исполнителями; возможность быстрого подсчета прибыли без учета затрат | По телефону/email, при личной встрече | 1. Уточнить информацию по предметной области. 2. Получить дополнительные сведения о сроках выполнения проекта. 3. Уточнить особые пожелания к проекту |
| Исполнители(работники) | Удобный интерфейс для организации собственных поступающих заказов, просмотра контактной информации с клиентом, автоматизация базовых моментов работы с клиентом | С помощью чата, созданного под руководством CEO для разработки приложения; по телефону/email | 1. Узнать об особых случаях, которые необходимо рассмотреть при разработке 2. Первоначально определить критерии удобства использования |
| Клиенты | Интерфейс, предоставляющий достаточную информацию по заказу, возможность выхода на контакт с исполнителем в случае форс-мажорных ситуаций. | С помощью голосования в официальной группе вк; с помощью отзывов; через исполнителей | 1. Критерии удобства использования 2. Наиболее интересующая информация о исполнителе/заказе |

Указанные в таблице 1 данные позволяют здраво оценить требования к разрабатываемому проектному решению для корректировки представления о нем у разработчика, специфицировать возможные сценарии взаимодействия пользователя с решением.

* 1. **Требования к решению**

Для формирования предлагаемого решения необходимо определить требования к решению со стороны пользователей и вовлеченных сторон. В результате анализа были получены следующие требования от указанных в таблице 1 групп пользователей:

* CEO бизнеса: Удобный интерфейс учета заказов, выполняемых всеми исполнителями; возможность быстрого подсчета прибыли без учета затрат.
* Исполнители(работники): Удобный интерфейс для организации собственных поступающих заказов, просмотра контактной информации с клиентом, автоматизация базовых моментов работы с клиентом.
* Клиенты: Интерфейс, предоставляющий достаточную информацию по заказу, возможность выхода на контакт с исполнителем в случае форс-мажорных ситуаций.
  1. **Предлагаемое решение**

Для пользователей наше решение будет предоставлять полный функционал просмотра информации о заказах, редактировании и анализа данной информации, упрощения кооперации между исполнителем и клиентом. В отличие от указанных ниже альтернатив, наше приложение также будет предоставлять функционал для кооперации касаемого заказа для клиента в то время, как аналоги предоставляют схожий функционал только для менеджеров внутри компании.

* 1. **Аналоги разрабатываемого решения**

Для выявления слабых и сильных сторон предлагаемого решения был выполнен анализ аналогов разрабатываемого решения. Результаты проведенного анализа представлены в таблице 2.

Таблица 2 - Аналоги

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Ссылка | Преимущества  (с точки зрения пользователя) | Недостатки  (с точки зрения пользователя) | Отличия  (от вашего решения) |
| Полигон Про | https://pbprog.ru/docs/pro/workwithorded.html | Обширный функционал для контроля над заказом. | Отсутствие интерфейса для клиентов, отсутствия портативной версии | Данный аналог предоставляет только функционал органайзера проектов |
| FreshOffice | https://www.freshoffice.ru/projects | Удобный просчет экономики заказов | Отсутствие интерфейса для клиентов | Данный аналог делает уклон в менеджмент, нежели во взаимодействии с клиентом |
| RemOnline | https://remonline.app/features/order-management/ | Заявки с разных каналов в единой базе  Четкие сроки выполнения | Отсутствие автоматизированного взаимодействия с клиентом, что приводит к необходимости использования кол-центров | Отсутствие интерфейса взаимодействия в формате чат-бота |

Сильная сторона разрабатываемого решения на фоне существующих аналогов заключается в том, что для пользователя наше решение будет предоставлять простой и не перегруженный функционалом интерфейс для учета и взаимодействия с заказами и в отличие от аналогов, будет иметь интерфейс работы через чат бота, что избавит клиента от необходимости устанавливать отдельное приложение.

Проведенный анализ на данном этапе позволяет перейти к следующему этапу разработки проекта.

1. **ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ЕЕ БИЗНЕС-ПРОЦЕССОВ**
   1. **Формирование начальных гипотез**

Для разработки всего необходимого функционала и достижения баланса в разрабатываемом функционале сформирована HADI-таблица, представленная в таблице 3.

Таблица 3 – HADI таблица на момент этапа исследования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **H - hypothesis - гипотеза / вопрос** | **A - actions - действия** | **D - data - данные** | | **I - insight - выводы** |
| Какую гипотезу проверяем /  на какой вопрос хотим найти ответ | Что будем делать, чтобы проверить гипотезу? | **Ожидания** Какие ожидаем результаты?  Что гипотезу опровергнет, а что «подтвердит»? | **Реальность** Какие данные получились по факту | Какие выводы мы делаем из полученных данных |
| Как клиент узнает о состоянии заказа? | Проведем опрос среди мастеров | 20% - Узнают о состоянии придя к мастеру  80% - Узнают через звонок мастеру | 30% - узнают о состоянии придя к мастеру  10% - не узнают о состоянии заказа, так как мастер был недоступен по телефону и не находился в мастерской  60% - узнают информацию позвонив | 1. Оказывается, есть доля клиентов, не получающая необходимой информации 2. Была получена реальная картина распределения способов получения информации, которая отличается от ожидаемой. |
| Как клиент вносит правки касаемо сроков выполнения заказа | Проведем опрос среди мастеров | Клиент изменяет сроки выполнения заказа | Сроки выполнения заказов могут не подлежать изменению, потому что:   1. Мастер перегружен 2. Клиент запросил невыполнимые сроки 3. Мастер готов изменить сроки, но за отдельную доплату | 1. Оказывается, что сфере обслуживания клиента, клиент может запросить невозможное 2. Мастер также может выдвинуть встречные условия |

Благодаря формированию HADI-таблицы было получено четкое представление, о разрабатываемом программном продукте на основе чего можно описать AS IS сценарии бизнес-процессов в предметной области.

* 1. **AS IS сценарии, описывающие бизнес-процессы в предметной области**

На основе HADI-таблицы были составлены основные AS IS сценарии, реализуемые в предметной области, внедрение в приложение и, следовательно автоматизация которых, сильно влияет на работу всей системы взаимодействий клиент-исполнитель. Сформированные сценарии описаны ниже.

Акторы: клиент, мастер.

Предусловия: клиент хочет узнать информацию о заказе.

Ожидаемый результат: клиент получает информацию о заказе.

Шаги сценария:

1. Клиент пытается узнать о состоянии заказа по телефону или очно
2. Проблема: мастер не может предоставить данные
3. Клиент ожидает
4. Продолжение с шага 1

Акторы: клиент, мастер

Предусловия: клиент хочет запросить уменьшение времени выполнения заказа

Ожидаемый результат: получение клиентом подтверждения или отказа по уменьшению срока

Шаги сценария:

1. Клиент просит об ускорении выполнения заказа
2. Мастер рассматривает возможность
3. Проблема: мастер не может ускорить выполнение заказа
4. Клиент ожидает выполнение заказа

Варианты:

3а. Мастер может ускорить выполнения заказа без дополнительных условий

3а1. Мастер меняет очередь заказов, чтобы удовлетворить запросу клиента

Продолжение с шага 4

3б. Мастер может ускорить выполнение заказа, но за доп. плату

3б1. Мастер предлагает клиенту свою новую цену

3б2. Клиент соглашается на новую цену

3б3. Мастер меняет очередь заказов

Продолжение с шага 4

Варианты для 3б2:

3б2а. Клиент не согласен с ценой

3б2а1. Мастер принимает ответ от клиента и не меняет очередь

Продолжение с шага 4

1. **РАЗРАБОТКА РЕШЕНИЯ**
   1. **Разработка решения по циклу Деминга**

Была начата разработка решения. Был использован цикл Дёминга, описанный ниже.

1-я итерация цикла Дёрминга.

**План.** Создать необходимый минимум для работы приложения с базой данных заказов. Создать интерфейс клиента.

**Разработка.** Реализована работа приложения с базой данных, вёрстка интерфейса клиента.

**Тестирование.** Во время тестирования у предполагаемых клиентов проблем обнаружено не было.

**Вывод.** На основании последнего тестирования было установлено, что разработка интерфейса пользователя движется в верном направлении. Также тестировщиками был выделен положительный опыт использования кнопок клавиатуры как быстрого и пользователь-дружелюбного способа взаимодействия с ботом.

2-я итерация цикла Дёминга.

**План.** Разграничить права доступа пользователей к бд, разработать пользовательский интерфейс для не-администраторов системы, создать многоступенчатые сценарии взаимодействия пользователей с ботом.

**Разработка.** Разграничение прав доступа к бд в приложении, обеспечение запоминания промежуточных этапов общения пользователя с ботом, вёрстка интерфейса пользователя.

**Тестирование.** Тестирование проводилось на работниках. Было отмечено, что бот отвечает правильно не на все сообщения пользователя внутри сценария, что приводит к сохранению неполной либо некорректной информации. Выделен способ выбора пользователя через текстовый ввод как неудобный и не интуитивно понятный.

**Вывод.** На основании тестирования была установлена необходимость переработки пользовательского интерфейса, перепроверки сценариев и сообщений в них. На основании замечания к способу выбора пользователя было выдвинуто предложение в следующей итерации цикла сделать интерфейс выбора пользователя на основе клавиатуры с кнопками.

3-я итерация цикла Дёминга.

**План.** Изменить способ выбора пользователя с ввода Telegram Username пользователя на встроенную в сообщение клавиатуру, разработать пользовательский интерфейс для администраторов системы, обеспечить возможность создания требований для клиентов и ответа на них для работников.

**Разработка.** Создание возможности формирования и изменения требований на уровне приложения и базы данных, вёрстка пользовательского интерфейса для администраторов системы, вёрстка интерфейса выбора пользователя.

**Тестирование.** Тестирование проводилось на всех группах: клиенты, работники и администраторы. Было выделено, что ввод даты не удобен. Других недочётов не было выявлено.

**Вывод.** На основании тестирования установлена необходимость изменения способа ввода даты на другую, более понятную и удобную пользователю. Так как в целом реализация необходимого функционала завершена, в дальнейшем планируется проведение рефакторинга кода и оптимизация некоторых процессов для повышения производительности.

В результате итерирования по циклю Дёминга был получен качественный продукт, который был презентован заказчику.

**ЗАКЛЮЧЕНИЕ**

В соответствии с поставленной перед собой темой инициативного проекта был разработан сайт для взаимодействия клиентов и исполнителей, предоставляющий удобный и лаконичный интерфейс пользователя, предоставляющий функции запроса состояния заказа, запроса изменения сроков выполнения заказа. Для исполнителя предоставляется интерфейс изменения статуса заказа, добавления новых заказов. Для подсистемы CEO/администратора был разработан интерфейс доступа для всего описанного ранее функционала, а также интерфейс добавления работников, клиентов, редактирования данных всех сущностей. На этапе запуска проекта были изучены стороны, взаимодействующие с полученным решением, аналоги разрабатываемого решения и сильных черт решения на фоне аналогов. На этапе исследования была составлена таблица HADI, которая была использована для описания AS IS сценариев.

Таким образом, была достигнута цель курсового проектирования – в результате прохождения всех этапов цифрового проектирования были выработаны и закреплены на практике знания о цифровом проектировании, усовершенствованы навыки создания программных продуктов.

**Приложение А**

**Частичный код решения**

Листинг 1 — Код интерфейса DatabaseConnectable

package com.undefinedgroup.epsilonbot.model.database;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public interface DatabaseConnectable

extends AutoCloseable

{

PreparedStatement createPreparedStatement (String sql)

throws SQLException;

}

Листинг 2 — Код класса SQLiteDBConnection

package com.undefinedgroup.epsilonbot.model.database;

import java.io.File;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class SQLiteDBConnection

implements DatabaseConnectable

{

private final File dbFile;

private Connection connection;

public SQLiteDBConnection (File file)

{

this.dbFile = file;

this.connection = null;

}

@Override

public PreparedStatement createPreparedStatement (String sql)

throws SQLException

{

if (connection == null)

{

connection = DriverManager.getConnection(String.format("jdbc:sqlite:%s", dbFile.toString()));

}

return connection.prepareStatement(sql);

}

@Override

public void close ()

throws Exception

{

final var conn = connection;

connection = null;

conn.close();

}

}

Листинг 3 — Код интерфейса ICustomerRepository

package com.undefinedgroup.epsilonbot.model.repository;

import java.util.List;

public interface ICustomerRepository<T>

{

T add (T data);

T update (T newData);

T findByTelephoneNumber (String telephoneNumber);

T findByUsername (String username);

List<T> getAll ();

}

Листинг 4 — Код интерфейса IOrderRepository

package com.undefinedgroup.epsilonbot.model.repository;

import com.undefinedgroup.epsilonbot.model.data.CustomerInfo;

import com.undefinedgroup.epsilonbot.model.data.WorkerInfo;

import java.util.List;

public interface IOrderRepository<T, K>

{

T add (T data);

T findById (Integer orderId);

List<T> findAllByCustomerInfo (CustomerInfo customerInfo);

List<T> findAllByWorkerInfo (WorkerInfo workerInfo);

T updateState (T order, K newState);

}

Листинг 5 — Код интерфейса IWorkerRepository

package com.undefinedgroup.epsilonbot.model.repository;

import java.util.List;

public interface IWorkerRepository<T>

{

T add (T data);

T update (T newData);

T findByTelephoneNumber (String telephoneNumber);

T findByUsername (String username);

List<T> getAll ();

}

Листинг 6 — Код класса SQLiteCustomerRepository

package com.undefinedgroup.epsilonbot.model.repository;

import com.undefinedgroup.epsilonbot.model.data.CustomerInfo;

import com.undefinedgroup.epsilonbot.model.database.DatabaseConnectable;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.LinkedList;

import java.util.List;

public class SQLiteCustomerRepository

implements ICustomerRepository<CustomerInfo>

{

private final DatabaseConnectable connection;

public SQLiteCustomerRepository (DatabaseConnectable connection)

{

this.connection = connection;

}

@Override

public CustomerInfo add (CustomerInfo data)

{

try (final var stm = connection.createPreparedStatement(insertNewCustomerPattern))

{

stm.setString(1, data.firstName);

stm.setString(2, data.lastName);

stm.setString(3, data.telephoneNumber);

stm.setString(4, data.tgUsername);

stm.setLong(5, data.tgChatId);

stm.execute();

return findByTelephoneNumber(data.telephoneNumber);

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public CustomerInfo update (CustomerInfo newData)

{

try (final var stm = connection.createPreparedStatement(updateWorkerInfoPattern))

{

stm.setString(1, newData.firstName);

stm.setString(2, newData.lastName);

stm.setString(3, newData.telephoneNumber);

stm.setString(4, newData.tgUsername);

stm.setLong(5, newData.tgChatId);

stm.setInt(6, newData.id);

stm.execute();

return newData;

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public CustomerInfo findByTelephoneNumber (String telephoneNumber)

{

try (final var stm = connection.createPreparedStatement(selectCustomerByTelephoneNumberPattern))

{

stm.setString(1, telephoneNumber);

try (final var result = stm.executeQuery())

{

result.next();

return createCustomerInfoFromResultSet(result);

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public CustomerInfo findByUsername (String username)

{

try (final var stm = connection.createPreparedStatement(selectCustomerByTgUsernamePattern))

{

stm.setString(1, username);

try (final var result = stm.executeQuery())

{

result.next();

return createCustomerInfoFromResultSet(result);

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public List<CustomerInfo> getAll ()

{

try (

final var stm = connection.createPreparedStatement(selectAllCustomers);

final var results = stm.executeQuery()

)

{

final List<CustomerInfo> customers = new LinkedList<>();

while (results.next())

{

customers.add(createCustomerInfoFromResultSet(results));

}

return customers;

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

private CustomerInfo createCustomerInfoFromResultSet (ResultSet result)

throws SQLException

{

final Integer id = result.getInt("id");

final String firstName = result.getString("first\_name");

final String lastName = result.getString("last\_name");

final String telephoneNumber = result.getString("tel\_number");

final String tgUsername = result.getString("tg\_username");

final Long tgChatId = result.getLong("tg\_chat\_id");

return new CustomerInfo(id, firstName, lastName, telephoneNumber, tgUsername, tgChatId);

}

private static final String insertNewCustomerPattern = """

INSERT INTO "customer" ("first\_name", "last\_name", "tel\_number", "tg\_username", "tg\_chat\_id")

VALUES (?, ?, ?, ?, ?);

""", selectCustomerByTelephoneNumberPattern = """

SELECT \* FROM "customer" WHERE "tel\_number" = ?;

""", selectCustomerByTgUsernamePattern = """

SELECT \* FROM "customer" WHERE "tg\_username" = ?;

""", selectAllCustomers = """

SELECT \* FROM "customer";

""", updateWorkerInfoPattern = """

UPDATE "customer"

SET "first\_name" = ?,

"last\_name" = ?,

"tel\_number" = ?,

"tg\_username" = ?,

"tg\_chat\_id" = ?

WHERE "id" = ?;

""";

}

Листинг 7 — Код класса SQLiteOrderRepository

package com.undefinedgroup.epsilonbot.model.repository;

import com.undefinedgroup.epsilonbot.model.data.CustomerInfo;

import com.undefinedgroup.epsilonbot.model.data.OrderInfo;

import com.undefinedgroup.epsilonbot.model.data.WorkerInfo;

import com.undefinedgroup.epsilonbot.model.database.DatabaseConnectable;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.time.Instant;

import java.time.format.DateTimeFormatter;

import java.util.LinkedList;

import java.util.List;

public class SQLiteOrderRepository

implements IOrderRepository<OrderInfo, OrderInfo.State>

{

private final DatabaseConnectable connection;

public SQLiteOrderRepository (DatabaseConnectable connection)

{

this.connection = connection;

}

private OrderInfo createOrderFromResultSet (ResultSet result)

throws SQLException

{

final Integer id = result.getInt("id");

final Integer customerId = result.getInt("customer\_id");

final Integer workerId = result.getInt("worker\_id");

final String description = result.getString("description");

final Integer price = result.getInt("price");

final String beginTime = result.getString("begin\_time");

final String lastUpdateTime = result.getString("last\_state\_update\_time");

final String end\_time = result.getString("end\_time");

final OrderInfo.State state = switch (result.getString("state"))

{

case "READY" -> OrderInfo.State.READY;

case "COMPLETE" -> OrderInfo.State.COMPLETE;

default -> OrderInfo.State.IN\_PROGRESS;

};

return new OrderInfo(id,

customerId,

workerId,

description,

price,

state,

Instant.from(DateTimeFormatter.ISO\_INSTANT.parse(beginTime)),

Instant.from(DateTimeFormatter.ISO\_INSTANT.parse(lastUpdateTime)),

Instant.from(DateTimeFormatter.ISO\_INSTANT.parse(end\_time))

);

}

private static final String insertOrderPattern = """

INSERT INTO "order" ("customer\_id", "worker\_id", "description", "price", "state", "end\_time") VALUES (?, ?, ?, ?, ?, ?);

""", updateOrderStatePattern = """

UPDATE "order"

SET "state" = ?, "last\_state\_update\_time" = (strftime('%Y-%m-%dT%H:%M:%SZ', 'now'))

WHERE "id" = ?;

""", selectOrderByIdPattern = """

SELECT \* FROM "order"

WHERE "id" = ?;

""", selectOrdersByCustomerIdPattern = """

SELECT \* FROM "order"

WHERE "customer\_id" = ?;

""", selectOrdersByWorkerIdPattern = """

SELECT \* FROM "order"

WHERE "worker\_id" = ?;

""", selectOrdersBySpecificDataPattern = """

SELECT \* FROM "order"

WHERE "customer\_id" = ?

AND "worker\_id" = ?

AND "description" = ?

AND "price" = ?

ORDER BY "id" DESC;

""";

@Override

public OrderInfo add (OrderInfo data)

{

try (var stm = connection.createPreparedStatement(insertOrderPattern))

{

stm.setInt(1, data.customerId);

stm.setInt(2, data.workerId);

stm.setString(3, data.description);

stm.setInt(4, data.price);

stm.setString(5, data.currentState.sqlName);

stm.setString(6, DateTimeFormatter.ISO\_INSTANT.format(data.endTime));

stm.execute();

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

try (var stm = connection.createPreparedStatement(selectOrdersBySpecificDataPattern))

{

stm.setInt(1, data.customerId);

stm.setInt(2, data.workerId);

stm.setString(3, data.description);

stm.setInt(4, data.price);

try (var result = stm.executeQuery())

{

result.next();

return createOrderFromResultSet(result);

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public OrderInfo findById (Integer orderId)

{

try (final var stm = connection.createPreparedStatement(selectOrderByIdPattern))

{

stm.setInt(1, orderId);

try (final var result = stm.executeQuery())

{

result.next();

return createOrderFromResultSet(result);

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public List<OrderInfo> findAllByCustomerInfo (CustomerInfo customerInfo)

{

try (final var stm = connection.createPreparedStatement(selectOrdersByCustomerIdPattern))

{

stm.setInt(1, customerInfo.id);

try (final var queryResult = stm.executeQuery())

{

final List<OrderInfo> ret = new LinkedList<>();

while (queryResult.next())

{

ret.add(createOrderFromResultSet(queryResult));

}

return ret;

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public List<OrderInfo> findAllByWorkerInfo (WorkerInfo workerInfo)

{

try (final var stm = connection.createPreparedStatement(selectOrdersByWorkerIdPattern))

{

stm.setInt(1, workerInfo.id);

try (final var queryResult = stm.executeQuery())

{

final List<OrderInfo> ret = new LinkedList<>();

while (queryResult.next())

{

ret.add(createOrderFromResultSet(queryResult));

}

return ret;

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public OrderInfo updateState (OrderInfo order, OrderInfo.State newState)

{

try (final var stm = connection.createPreparedStatement(updateOrderStatePattern))

{

stm.setString(1, order.currentState.sqlName);

stm.setInt(2, order.id);

stm.execute();

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

return findById(order.id);

}

}

Листинг 8 — Код класса SQLiteWorkerRepository

package com.undefinedgroup.epsilonbot.model.repository;

import com.undefinedgroup.epsilonbot.model.data.WorkerInfo;

import com.undefinedgroup.epsilonbot.model.database.DatabaseConnectable;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.LinkedList;

import java.util.List;

public class SQLiteWorkerRepository

implements IWorkerRepository<WorkerInfo>

{

final DatabaseConnectable connection;

public SQLiteWorkerRepository (DatabaseConnectable connection)

{

this.connection = connection;

}

@Override

public WorkerInfo add (WorkerInfo data)

{

try (final var stm = connection.createPreparedStatement(insertWorkerPattern))

{

stm.setString(1, data.role.sqlName);

stm.setString(2, data.firstName);

stm.setString(3, data.lastName);

stm.setString(4, data.telephoneNumber);

stm.setString(5, data.tgUsername);

stm.setLong(6, data.tgChatId);

stm.execute();

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

try (final var stm = connection.createPreparedStatement(selectWorkerBySpecificFieldsPattern))

{

stm.setString(1, data.telephoneNumber);

try (final var resultSet = stm.executeQuery())

{

resultSet.next();

return createWorkerFromResultSet(resultSet);

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public WorkerInfo update (WorkerInfo newData)

{

try (final var stm = connection.createPreparedStatement(updateWorkerInfoPattern))

{

stm.setString(1, newData.role.sqlName);

stm.setString(2, newData.firstName);

stm.setString(3, newData.lastName);

stm.setString(4, newData.telephoneNumber);

stm.setString(5, newData.tgUsername);

stm.setLong(6, newData.tgChatId);

stm.setInt(7, newData.id);

stm.execute();

return newData;

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public WorkerInfo findByTelephoneNumber (String telephoneNumber)

{

try (final var stm = connection.createPreparedStatement(selectWorkerByTelephoneNumberPattern))

{

stm.setString(1, telephoneNumber);

try (final var resultSet = stm.executeQuery())

{

resultSet.next();

return createWorkerFromResultSet(resultSet);

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public WorkerInfo findByUsername (String username)

{

try (final var stm = connection.createPreparedStatement(selectWorkerByUsernamePattern))

{

stm.setString(1, username);

try (final var resultSet = stm.executeQuery())

{

resultSet.next();

return createWorkerFromResultSet(resultSet);

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public List<WorkerInfo> getAll ()

{

try (

final var stm = connection.createPreparedStatement(selectAllWorkers);

final var resultSet = stm.executeQuery()

)

{

final List<WorkerInfo> workers = new LinkedList<>();

while (resultSet.next())

{

workers.add(createWorkerFromResultSet(resultSet));

}

return workers;

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

private WorkerInfo createWorkerFromResultSet (ResultSet result)

throws SQLException

{

final Integer id = result.getInt("id");

final String firstName = result.getString("first\_name");

final String lastName = result.getString("last\_name");

final String telephoneNumber = result.getString("tel\_number");

final String tgUsername = result.getString("tg\_username");

final Long tgChatId = result.getLong("tg\_chat\_id");

final WorkerInfo.Role role = switch (result.getString("role"))

{

case "ADMIN" -> WorkerInfo.Role.ADMIN;

default -> WorkerInfo.Role.PLAIN;

};

return new WorkerInfo(id, role, firstName, lastName, telephoneNumber, tgUsername, tgChatId);

}

private static final String insertWorkerPattern = """

INSERT INTO "worker" ("role", "first\_name", "last\_name", "tel\_number", "tg\_username", "tg\_chat\_id")

VALUES (?, ?, ?, ?, ?, ?);

""", selectWorkerByTelephoneNumberPattern = """

SELECT \* FROM "worker" WHERE "tel\_number" = ?;

""", selectWorkerByUsernamePattern = """

SELECT \* FROM "worker" WHERE "tg\_username" = ?;

""", selectAllWorkers = """

SELECT \* FROM "worker";

""", selectWorkerBySpecificFieldsPattern = """

SELECT \* FROM "worker"

WHERE "tel\_number" = ?;

""", updateWorkerInfoPattern = """

UPDATE "worker"

SET "role" = ?,

"first\_name" = ?,

"last\_name" = ?,

"tel\_number" = ?,

"tg\_username" = ?,

"tg\_chat\_id" = ?

WHERE "id" = ?;

""";

}

Листинг 9 — Код класса IAdditionalRequirementRepository

package com.undefinedgroup.epsilonbot.model.repository;

public interface IAdditionalRequirementRepository<T>

{

T add (T x);

T getByOrderId (Integer x);

T update (T x);

}

Листинг 10 — Код класса SQLiteAdditionalRequirementRepository

package com.undefinedgroup.epsilonbot.model.repository;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.time.Instant;

import java.time.format.DateTimeFormatter;

import com.undefinedgroup.epsilonbot.model.data.AdditionalRequirement;

import com.undefinedgroup.epsilonbot.model.database.DatabaseConnectable;

public class SQLiteAdditionalRequirementRepository implements IAdditionalRequirementRepository<AdditionalRequirement>

{

private final DatabaseConnectable connection;

public SQLiteAdditionalRequirementRepository (DatabaseConnectable connection)

{

this.connection = connection;

}

@Override

public AdditionalRequirement add (AdditionalRequirement x)

{

try

{

try (final var stm = connection.createPreparedStatement(insertInfoPattern))

{

stm.setInt(1, x.orderId);

stm.setString(2, x.description);

stm.setInt(3, x.priceDelta);

stm.execute();

}

return getByOrderId(x.orderId);

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public AdditionalRequirement getByOrderId (Integer x)

{

try (final var stm = connection.createPreparedStatement(selectInfoByIdPattern))

{

stm.setInt(1, x);

try (final var resultSet = stm.executeQuery())

{

resultSet.next();

return buildFromResult(resultSet);

}

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

@Override

public AdditionalRequirement update (AdditionalRequirement x)

{

try (final var stm = connection.createPreparedStatement(updateInfoPattern))

{

stm.setString(1, x.description);

stm.setString(2, x.state.sqlName);

stm.setInt(3, x.priceDelta);

stm.setInt(4, x.orderId);

stm.execute();

return x;

}

catch (Exception ex)

{

throw new RuntimeException(ex);

}

}

private AdditionalRequirement buildFromResult (ResultSet result) throws SQLException

{

try

{

final var orderId = result.getInt("order\_id");

final var instant = Instant.from(DateTimeFormatter.ISO\_INSTANT.parse(result.getString("instant")));

final var type = result.getString("type");

final var description = result.getString("description");

final var state = AdditionalRequirement.State.fromString(result.getString("state"));

final var priceDelta = result.getInt("price\_delta");

return new AdditionalRequirement(orderId, instant, type, description, state, priceDelta);

}

catch (NullPointerException ex)

{

throw new NonexistentRecordException(ex);

}

}

private static final String insertInfoPattern = """

INSERT INTO "additional\_requirement"

("order\_id", "description", "price\_delta")

VALUES (?, ?, ?);

""", selectInfoByIdPattern = "SELECT \* FROM \"additional\_requirement\" WHERE \"order\_id\" = ?;",

updateInfoPattern = """

UPDATE "additional\_requirement"

SET "description" = ?,

"state" = ?,

"price\_delta" = ?

WHERE "order\_id" = ?;

""";

}