Міністерство освіти і науки України
Національний технічний університет України
«Київський Політехнічний Інститут»
Кафедра автоматики та управління в технічних системах

Система контролю щоденних завдань «Life Tracker»

IT41.090418.001

Курсовий проект

З дисципліни «Сучасні технології програмування - 2»

Керівник:
Букасов М.М.

«Допущений до захисту»

_____
(особистий підпис керівника)

«____» _____ 2017 р.

Виконавець:
ст. Довгопола Н.Ю.
зал. книжка № IT41-09
гр. IT-41

_____
(особистий підпис студента)

Захищено с оцінкою

_____
(оцінка)
«____» _____ 2017 р.

Члени комісії:

_____ _____
(особистий підпис) (П.І.Б.)

_____ _____
(особистий підпис) (П.І.Б.)

Київ 2017 р.

Номер рядка	Формат	Позначення	Найменування	Кіл. листів	Номер экзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4		Анотація	1		
6						
7	А4	ІТ41. 090518.001 ТЗ	Технічне завдання	3		
8						
9	А4	ІТ41. 090518.001 ТП	Відомість технічного	1		
10			проекту			
11						
12	А4	ІТ41. 090518.001 ПЗ	Пояснювальна записка	26		
13						
14	А4	ІТ41. 090518.001	Специфікація	1		
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
33						

Зм.	Лист	№ докум.	Підп.	Дата	ІТ41.090418.001 ОП		
Розроб.		Довгопола Н.			Система контролю щоденних завдань Опис	Літ.	Лист	Листів
Перевірив		Букасов М.М.				Т		1
Н. контр.						НТУУ ”КПІ” ФІОТ Група ІТ-41		
Затв.								

Анотація

Змістом курсового проекту було створення системи для контроля щоденних завдань, яка б дозволила користувачу створювати записи про свої рутинні справи, відмічати їх як виконані та відстежувати свій прогрес. Для такої задачі був розроблений сервіс з трирівневою архітектурою: база даних, сервер, що оброблює бізнес-логіку, та клієнти.

У курсовому проекті розглянуто рівень бізнес-логіки і рівня бази даних. Система представлена у вигляді веб-сайту і мобільного додатку, користувач взаємодіє з нею, використовуючи браузер, або пристрій Android.

Серверна частина сервісу реалізована з використанням Java і паттерну проектування Singleton, СУБД MySql. Під час розробки проекту застосовувалися сучасні програмні продукти та технології.

Summary

The content of the course project was to create a system for monitoring daily tasks that would allow the user to create a record of their routine business, mark them as done and track your progress. For such a task was designed service with three-tiered architecture: the database server that handles business logic and clients.

The system is represented as a website and mobile app, the user interacts with it using the browser or device Android.

The server component service is implemented using Java and MySql. During the development of the project used modern software products and technologies.

Номер рядка	Формат	Позначення	Найменування	Кіл. листів	Номер экзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІТ41.080200.001ПЗ	Пояснювальна записка	26		
6						
7	А4	ІТ41.080200.001	Специфікація	1		
8						
9	А3	ІТ41.080200.001 Д1.1	Схема бази даних	1		
10						
11	А3	ІТ41.080200.001 Д1.2	Діаграма прецедентів	1		
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
33						

Зм.	Лист	№ докум.	Підп.	Дата	ІТ41.090418.001 ПЗ			
Розроб.		Довгопола Н.			Система контролю щоденних завдань Пояснювальна записка	Літ.	Лист	Листів
Перевірив		Букасов М.М.				Т		34
Н. контр.						НТУУ "КПІ" ФІОТ Група ІТ-41		
Затв.								

ЗМІСТ

ВСТУП

Як відомо, звичка - друга натура. Чомусь саме слово «звичка», в усякому разі, в українській мові, має якусь слабонегативную конотацію. Про що ви думаєте, коли чуєте його? Сподіваюся, що про ранкові пробіжки в парку, щирих побажаннях доброго дня і здорову їжу. Але найчастіше саме поняття звички зводиться до п'ятничних загулів і випливають звідти наслідки, які ніяк не назвеш сприятливими для організму.

Як це працює? Створіть «звичку», наприклад, ранкову зарядку. Вибираєте потрібні інтервали (кожен день треба зарядку робити, ви чого!) І визначаєтесь, чи потрібні вам нагадування від програми, або власної мотивації буде цілком достатньо. Зрозуміло, як і більшість фітнес-трекерів наш розрахований на самоповагу користувача, який не буде обманювати себе.

Якщо ви дійсно зробили зарядку, прибирання або подзвонили мамі, натисніть кнопку підтвердження і отримаєте бали. Використовуючи систему регулярно, можна розраховувати на отримання нових рівнів, що є додатковою мотивацією.

Система контролю звичок і завдань – це потужний і зручний інструмент для роботи над собою, він стане у нагоді тим, хто прагне до саморозвитку і самодисципліни, або просто допоможе із щоденними справами.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

1.1 Програма є серверною частиною системи контролю за щоденними завданнями і звичками «Life Tracker».

1.2 Програма дає можливість взаємодії з сайтом і мобільним додатком про протоколу https, є проміжною ланкою між клієнтською частиною і БД MySql.

2 Підстава для розробки

2.1 Навчальний план спеціальності «Програмна інженерія».

2.2 Завдання на курсове проектування, видане керівником.

3 Мета та функції системи

3.1 Метою є створення серверної частини для веб-ресурсу і мобільного додатку з трирівневою архітектурою, що забезпечує клієнт-серверну взаємодію і доступ до бази даних.

3.2 Функції серверної частини:

- зв’язування сайту і мобільного додатку;
- взаємодія з сайтом і мобільним додатком про протоколу https;
- є проміжною ланкою між клієнтською частиною і БД MySql.

4 Джерела розробки

4.1 Методичні вказівки до виконання курсових проектів для студентів спеціальності «Програмна інженерія».

4.2 Довідкова, навчальна і наукова література, тематичні веб-сайти.

5 Технічні вимоги

Вимоги до сервера та БД:

– операційна система: Windows 7, 8, 10;

– оперативна пам’ять: не менше 1 ГБ.

6 Стадії та етапи розробки

6.1	Отримання завдання	15.02.2017
6.2	Аналіз наявних рішень	18.02.2017
6.3.	Розроблення архітектури системи	28.02.2017
6.4	Розроблення схеми бази даних	08.03.2017
6.5	Розроблення діаграми класів	15.03.2017
6.6	Реалізація бізнес-логіки системи	30.04.2017
6.7	Тестування системи	13.05.2017
6.8	Оформлення текстової документації	15.05.2017
6.9	Оформлення графічної документації	20.05.2017
6.10	Представлення курсового проекту до захисту	07.06.2017

7 Характер розробки

Текстові і графічні документи повинні бути виконані на рівні технічного проекту та мати літеру «Т».

8 Порядок контролю та прийому

Оформлений курсовий проект підписується виконавцем, перевіряться і підписується керівником і представляється до захисту комісії, яка складається з викладачів кафедри.

2 ТЕХНІЧНА ХАРАКТЕРИСТИКА СИСТЕМИ

2.1 Формування вимог до майбутньої системи

Технічні характеристики:

– операційна система: Windows 7, 8,10;

– оперативна пам’ять: 1 ГБ.

Вимоги до функціоналу:

– забезпечити можливість взаємодії з клієнтською частиною: веб-сайтом і мобільним додатком;

– перевірка на правильність вхідних даних;

– взаємодія з базою даних MySql за допомогою JDBC;

– В MySql створити таблиці і зв’язки між ними.

2.2 Вибір платформи розробки

Для розробки даної системи були обрані такі технології реалізації, як мова програмування Java, СУБД MySQL,

2.2.1 Мова програмування Java

Java — об'єктно-орієнтована мова програмування. Програми на Java транслюються в байт-код, що виконується віртуальною машиною Java (JVM) — програмою, що переробляє байтовий код і передає інструкції обладнанню як інтерпретатор.[1]

Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і обладнання, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина.

Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), негайне переривання.

2.2.2 Система управління базами даних MySql

СУБД MySQL — це система управління базами даних. У реляційній базі даних дані зберігаються не горою, а в окремих таблицях, завдяки чому досягається виграш в швидкості і гнучкості. Таблиці зв'язуються між собою за допомогою відношень, завдяки чому забезпечується можливість об'єднувати при виконанні запиту дані з декількох таблиць. SQL як частина системи MySQL можна охарактеризувати як мову структурованих запитів плюс

найбільш поширена стандартна мова, яка використовується для доступу до баз даних. [2]

Програмне забезпечення MySQL — це ПЗ з відкритим кодом. ПЗ з відкритим кодом застосовувати і модифікувати може будь-хто.

2.2.3 Паттерн Singleton

Singleton — шаблон проектування, відноситься до класу твірних шаблонів. Гарантує, що клас матиме тільки один екземпляр, і забезпечує глобальну точку доступу до цього екземпляра. ля деяких класів важливо, щоб існував тільки один екземпляр. Наприклад, хоча у системі може існувати декілька принтерів, може бути тільки один спулер. Повинна бути тільки одна файлова система та тільки один активний віконний менеджер.[3]

Глобальна змінна не вирішує такої проблеми, бо не забороняє створити інші екземпляри класу.

Рішення полягає в тому, щоб сам клас контролював свою «унікальність», забороняючи створення нових екземплярів, та сам забезпечував єдину точку доступу. Це є призначенням шаблону Singleton.

3 АНАЛІЗ НАЯВНИХ РІШЕНЬ

Для формування вимог до системи, що розробляється необхідно проаналізувати зразки подібних систем. Детальний аналіз допоможе сформувати перелік необхідного функціоналу, виявити переваги та недоліки подібних систем.

3.1 HabitBull - Habit Tracker

HabitBull - Habit Tracker— мобільний додаток для контролю за своїми звичками. Користувачі можуть додавати нові заняття, дивитись статистику пройдених. Проте, є тільки мобільний додаток.

Інтерфейс зображений на рисунку 3.1.

Рисунок 3.1 — додаток HabitBull - Habit Tracker

3.2 Habitica.com

Habitica — це відеогра, з допомогою якої ви можете поліпшити свої звички у реальному житті. Вона „ігрофікує“ ваше життя, перетворюючи всі ваші завдання (звички, щоденні справи та обов'язки) на маленьких потвор, яких вам потрібно побороти. Чим ліпше вам це вдаватиметься, тим більше ви просуватиметеся грою. Кожна ваша помилка у реальному житті відкидатиме назад вашого персонажа у грі. Портал зображено на рисунку 3.2.

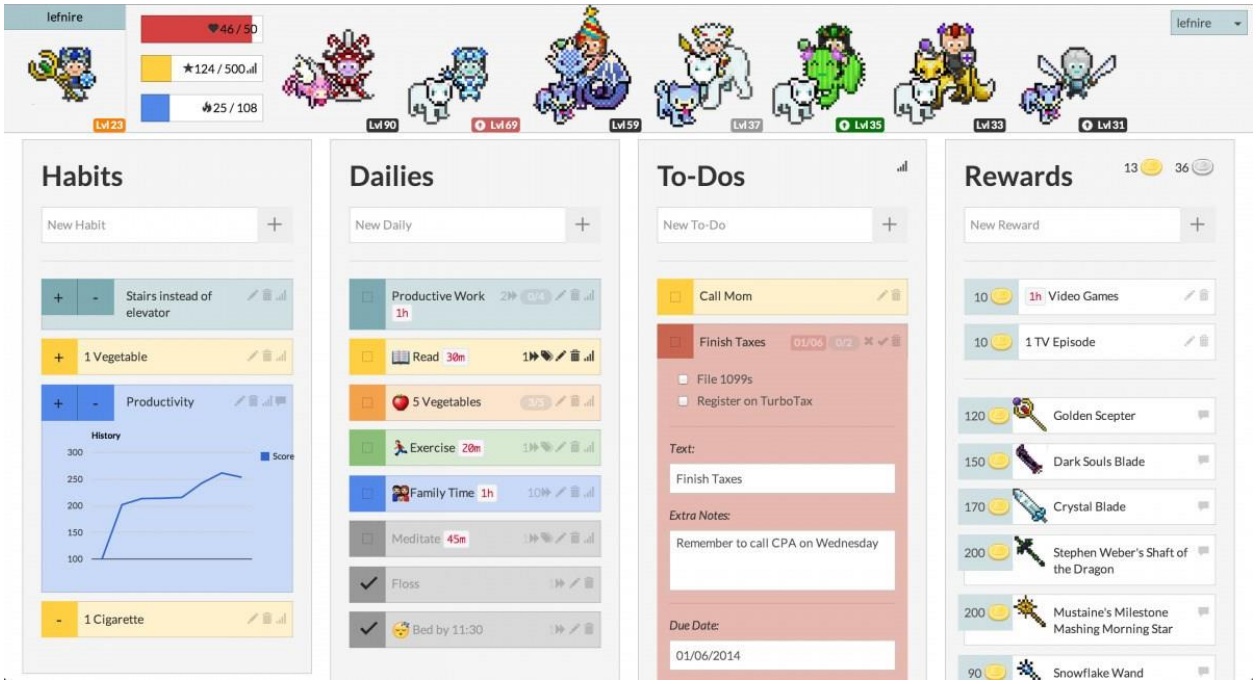

Рисунок 3.2 — Портал Habitica.com

4 АРХІТЕКТУРА СИСТЕМИ

4.1 Трирівнева архітектура проектування системи

У комп'ютерних технологіях трирівнева архітектура (англ. three-tier або Multitier architecture) передбачає наявність наступних компонентів програми: клієнтський застосунок (зазвичай говорять «тонкий клієнт» або термінал), підключений до сервера застосунків, який в свою чергу підключений до серверу бази даних.[4]

Клієнт — це інтерфейсний (зазвичай графічний) компонент, який представляє перший рівень, власне застосунок для кінцевого користувача. Перший рівень не повинен мати прямих зв'язків з базою даних (за вимогами безпеки), не повинен бути навантаженим основною бізнес-логікою (за вимогами масштабованості) і зберігати стан програми (за вимогами надійності). На перший рівень може бути винесена і зазвичай виноситься найпростіша бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка значень, що вводяться, на допустимість і відповідність формату, нескладні операції (сортування, групування, підрахунок значень) з даними, вже завантаженими на термінал.

Сервер застосунків розташовується на другому рівні. На другому рівні зосереджена більша частина бізнес-логіки. Поза ним залишаються фрагменти, що експортуються на термінали, а також розміщені в третьому рівні збережені процедури і тригери.

Сервер бази даних забезпечує зберігання даних і виноситься на третій рівень. Зазвичай це стандартна реляційна або об'єктно-орієнтована СУБД. Якщо третій рівень являє собою базу даних разом з збереженими процедурами, тригерами і схемою, яка описує застосунок в термінах реляційної моделі, то другий рівень будується як програмний інтерфейс, що зв'язує клієнтські компоненти з прикладною логікою бази даних.

У правильній з точки зору безпеки, надійності і масштабування конфігурації, сервер бази даних міститься на відділеному комп'ютері (або кластері), до якого по мережі підключені один або кілька серверів застосунків, до яких, в свою чергу, по мережі підключаються термінали [4]. Схему трирівневої архітектури зображено на рисунку 5.1.

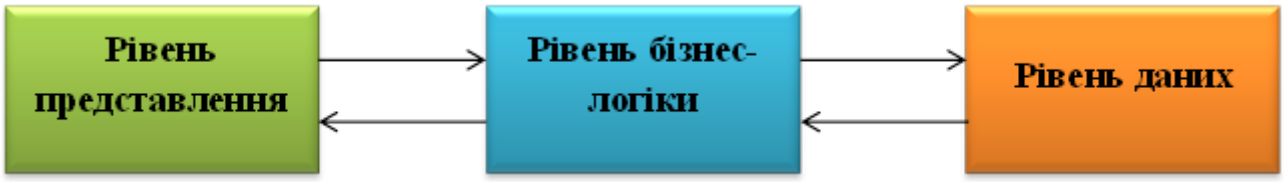

Рисунок 5.1 — Схема трирівневої архітектури

4.2 Архітектура серверної частини

Клас DatabaseController реалізований з використанням паттерну проектування Singleton. Шаблон Singleton накладає обмеження на створення екземпляра класу і гарантує, що в JVM (віртуальної джава машині) існує тільки один екземпляр даного класу. Клас Singleton повинен мати глобальну точку доступу для отримання екземпляру класу. Шаблон використовують для логування, об'єктів драйверів, кешування і наборів ниток.

Існують кілька різних підходів, реалізації шаблону Singleton, але всі вони мають загальні принципи.

- Private конструктор - для заборони ініціалізації екземпляра класу з іншого класу через конструктор;
- Private static змінну того ж класу, яка і буде єдиним екземпляром цього класу;
- Public static метод, який повертає екземпляр класу. Це - глобальна точка доступу для зовнішнього світу дозволяє отримати екземпляр класу Singleton.

DatabaseController реалізований способом Public static методу.

4.3 Проектування моделі бази даних

Модель бази даних відображається у вигляді ER-діаграми (entity-relationship diagram).

Для побудови ER-моделі необхідно визначити всі сутності, їх атрибути та зв’язки між сутностями для візуалізації моделі. Перелік сутностей та атрибутів подано у Таблиці 5.1.

Таблиця 5.1 — Перелік сутностей та атрибутів

Сутність	Атрибути
Користувач	Ім’я, пароль, рівень, рейтинг
Досягнення	Назва, бали до рейтингу, опис
Заняття	Назва, тип, мінімум, максимум
Перевірка	Дата початку, дата кінця, значення

ER-модель зображено на рисунку 5.2.

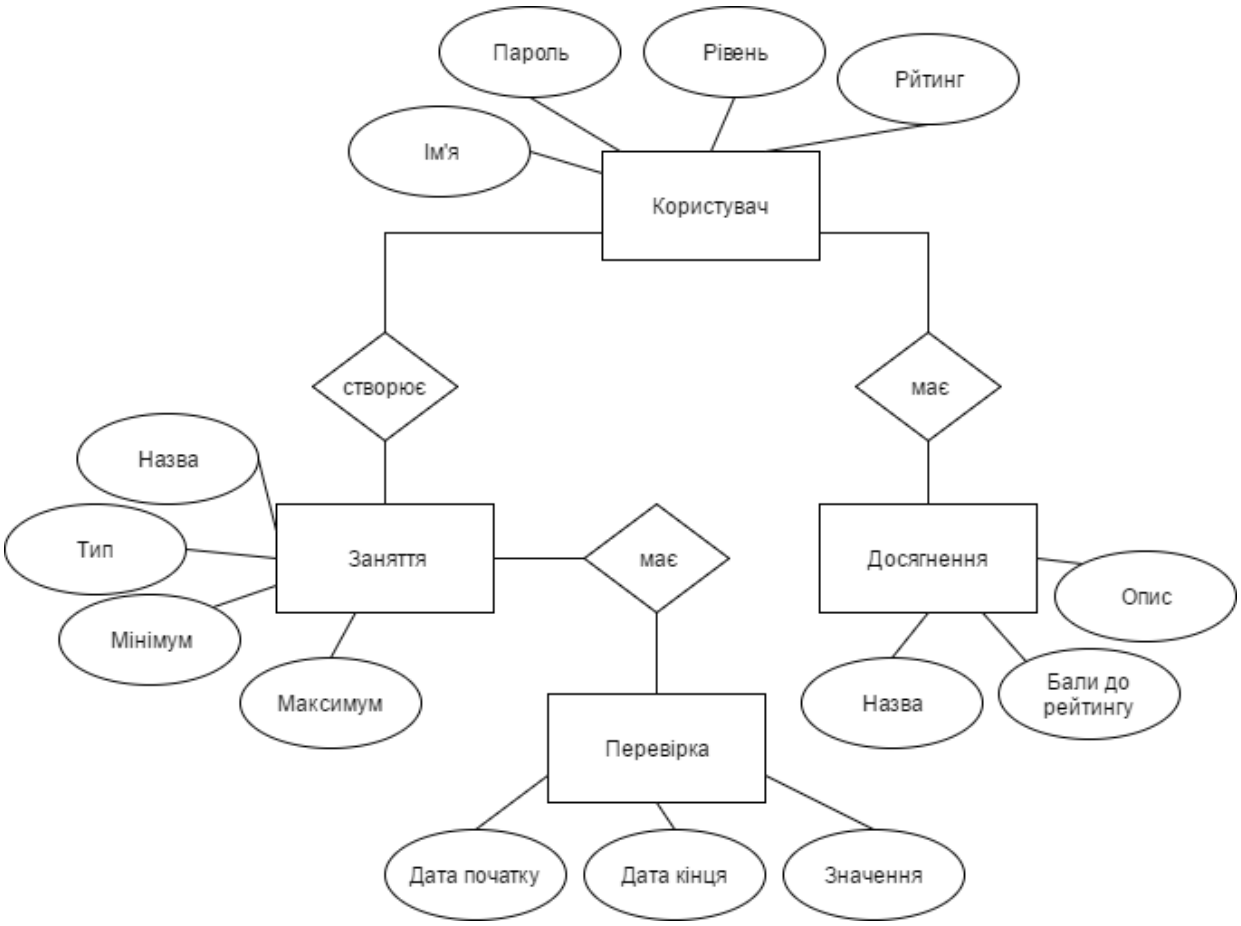

Рисунок 5.2

5 РЕАЛІЗАЦІЯ СПРОЕКТОВАНИХ ПІДСИСТЕМ

5.1 Реалізація моделі бази даних

В даний час найбільшого поширення набули реляційні бази даних. Коротко особливості реляційної бази даних можна сформулювати наступним чином:

- дані зберігаються в таблицях, що складаються із стовпців (атрибутів) і рядків (записів);
- на перетині кожного стовпця і рядка стоїть в точності одне значення;
- у кожного стовпчика є своє ім'я, яке служить його назвою, і всі значення в одному стовпці мають один тип;
- кожна таблиця має унікальний первинний ключ.

База даних системи нормалізована, знаходиться у третій нормальній формі. Опис таблиць, які входять в базу даних системи представлено в Таблицях 6.1− 6.6:

Таблиця 6.1 — Користувач (User)

Назва поля	Тип	Опис
id (Primary key)	BIGINT	Ідентифікатор
Pass	VARCHAR	Пароль
Name	VARCHAR	Ім’я
Lvl	INT	Рівень
Xp	INT	Рейтинг

Таблиця 6.2 — Підписники (Followers)

Назва поля	Тип	Опис
id INT (Primary key)	BIGINT	Ідентифікатор
id_UserWatcher	BIGINT	Ідентифікатор
id_UserWatching	BIGINT	Ідентифікатор

Таблиця 6.3 — Діяльність (Activity)

Назва поля	Тип	Опис
id (Primary key)	BIGINT	Ідентифікатор
Private	BOOLEAN	Приватність заняття
Name	VARCHAR	Назва
Type	BOOLEAN	Тип
Min	INT	Мінімум виконання
Max	INT	Максимум виконання
id_User	BIGINT	Ідентифікатор

Таблиця 6.4 — Перевірка (Check)

Назва поля	Тип	Опис
id (Primary key)	BIGINT	Ідентифікатор
Date	DATE	Початкова дата
EndDate	DATE	Кінцева дата
Value	INT	Значення виконання
id_Activity	BIGINT	Ідентифікатор

Таблиця 6.5 — Досягнення користувача (UserBadges)

Назва поля	Тип	Опис
id (Primary key)	BIGINT	Ідентифікатор
id_User	BIGINT	Ідентифікатор
id_Badge	BIGINT	Ідентифікатор
Date	DATE	Дата отримання

Таблиця 6.6 — Досягнення (Badge)

Назва поля	Тип	Опис
id (Primary key)	BIGINT	Ідентифікатор
Name	Тип	Назва

Назва поля	Тип	Опис
Points	INT	Бали за досягнення
Description	VARCHAR	Опис

5.2 Реалізація бізнес-логіки

У склад підсистеми створення записів входить шість класів-сутностей бази даних і контролер бази даних. Опис функціоналу структурних елементів системи наведений у Таблицях 6.7.

Таблиця 6.7 — Опис класів-сутностей бази даних

Назва	Опис
Activity	Клас Звичка. Використовуэться для створення користувачем його завдань.
Auth	Для входу користувача у систему.
Badge	Досягнення корисувача.
Check	Для відслідковування виконання завдань.
User	Дані про користувача для входу в систему та додаткова інформація.
Identified	Присвєння токена сесії.

Клас DatabaseController реалізований з використанням паттерну Сінглтон. Має методи для підключення до СУБД MySql, додавання, редагування та видалення даних з бази.

ВИСНОВКИ

В ході виконання курсового проекту було проведено дослідження предметної області, виділено головні ролі системи та бізнес-процеси. Проаналізовано вимоги до системи в цілому, вимоги до функцій системи, програмного і технічного забезпечення.

Було проведено дослідження технологій для побудови розподілених додатків. В результаті дослідження був обраний наступний стек засобів: Java, MySQL, та налаштувати взаємодію з клієнтською частиною додатку.

Використання принципу тришарової архітектури додатку, тобто поділу на рівень представлення, рівень бізнес-логіки та рівень данних дало можливість розробити гнучку та ефективну систему, адже на кожному шарі відбувається вирішення окремих локальних задач, що позитивно відображається на надійності системи.

Результатом роботи над курсовим проектом стало створення сервісу слідкування за звичками та завданнями, що став би корисним ресурсом для людей будь-якого віку. Система є сучасною та зручною, задовольняє всі вимоги з точки зору функціональності, юзабіліті, дизайну та безпеки даних.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Derek C. Ashmore The Java EE Architect's Handbook, Second Edition: How to be a successful application architect for Java EE applications [Текст] / Derek C. Ashmore - M.: DVT Press, 2014. – 268 ст.
2. Обеспечение высокой доступности систем на основе MySQL [Текст] / Талманн Л., Киндал М., Белл Ч. - 2012. – 624 с.
3. Шаблон проектування Singleton [Електронний ресурс] – Режим доступу: http://info.javarush.ru/translation/2013/09/14/Шаблон-проектирования-Singleton-одиночка-наиболее-рациональные-реализации-в-примерах-.html
4. Триярусна архітектура [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Триярусна_архітектура

ДОДАТОК А. ЗАВДАННЯ НА КП

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Кафедра автоматики та управління в технічних системах
Дисципліна: Сучасні технології програмування-2
Спеціальність: Програмна інженерія
Курс 3 Група ІТ-41 Семестр 6

ЗАВДАННЯ

на курсову роботу студента

Довгополої Наталії Юріївни

(прізвище, ім’я, по батькові)

1. **Тема роботи** Система контролю щоденних завдань “Life Tracker”
2. **Термін здачі** студентом закінченої роботи 07.06.2017
3. **Вхідні дані до роботи**: технічне завдання, мова програмування Java, БД – MySQL, паттерн Singleton.
4. **Зміст розрахунково-пояснювальної записки** (перелік питань, які підлягають розробці): розробка інформаційного забезпечення, розробка програмного забезпечення, розробка технічного забезпечення.
5. **Перелік графічного матеріалу**:
Графічний опис бізнес-процесів, схема бази даних, діаграми з проектування програмного забезпечення;
6. **Дата видачі завдання** “15” лютого 2017р.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМИ

DatabaseController.java

```
package Controller;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;

import Model.*;

public class DatabaseController {
    //Db provides data, main controller turns into json
    //GET
    private static volatile DatabaseController instance;

    static final String url = "jdbc:mysql://localhost:3306/LifeTracker";
    static final String user = "root";
    static final String password = "";
    // JDBC variables for opening and managing connection
    static Connection con;
    static Statement stmt;
    static ResultSet rs;
    static int rsIUD;
    static DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
    public static String query = "";

    public static DatabaseController getInstance() {
        DatabaseController localInstance = instance;
        if (localInstance == null) {
            synchronized (DatabaseController.class) {
                localInstance = instance;
                if (localInstance == null) {
                    instance = localInstance = new DatabaseController();
                    instance.DatabaseConnect();
                }
            }
        }
        return localInstance;
    }

    public void DatabaseConnect(){
        // opening database connection to MySQL server
        try {
            con = DriverManager.getConnection(url, user, password);
            stmt = con.createStatement();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        // getting Statement object to execute query

    }

    //всякие трайкэтчи и проверки на адекватность параметров должны быть в MainController, а не
здесь
    public  Auth GetAuths(String userid){
        Auth auth = new Auth();
        query = "SELECT Pass FROM User WHERE id="+userid+";";
        try {
            rs = stmt.executeQuery(query);while(rs.next()){
                auth.Password = rs.getString("Pass");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return auth;
    }

    public  Auth[] GetAuths() {
        Auth res = new Auth();
```

```
        ArrayList<Auth> auts = new ArrayList<Auth>();
        query = "SELECT Pass FROM Auth;";
        try {
            rs = stmt.executeQuery(query);
            while(rs.next()){
                res.Password = rs.getString("Pass");
                auts.add(res);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Auth[] a = new Auth[auts.size()];
        return auts.toArray(a);
    }

    public  User GetUser(String userid) {
        User res = new User("Silvia");
        res.FriendsId = new Long[] {123L, 358L};
        return res;
    } //yep, like this
    public  Activity GetActivity(String activityid) {
        Activity res = new Activity();
        query = "SELECT Name FROM Activity WHERE id="+activityid+";";
        try{
            rs = stmt.executeQuery(query);
            while(rs.next()){
                res.Name = rs.getString("Name");
                res.Id = Long.parseLong(activityid);
            }
        }
        catch(SQLException e){
            e.printStackTrace();
        }
        return res;
    }
    public  Activity[] GetActivities(String userid) {
        Activity res = new Activity();
        ArrayList<Activity> acts = new ArrayList<Activity>();
        query = "SELECT Name FROM Activity WHERE id_User="+userid+";";
        try {
            rs = stmt.executeQuery(query);
            while(rs.next()){
                res.Name = rs.getString("Name");
                res.UserId = Long.parseLong(userid);
                acts.add(res);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Activity[] a = new Activity[acts.size()];
        return acts.toArray(a);
    }
    public  Check GetCheck(String checkid) {
        Check res = new Check(Long.parseLong(checkid));
        query = "SELECT Date FROM Check WHERE id="+checkid+";";
        try {
            rs = stmt.executeQuery(query);
            while(rs.next()){
                res.Date = LocalDateTime.parse(rs.getString("Date"), formatter);
            }
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
        return res;
    }
    public  Check[] GetChecks(String activityid) {
        Check res = new Check(Long.parseLong(activityid));
        ArrayList<Check> chks = new ArrayList<Check>();
        query = "SELECT Date FROM Check WHERE id_Activity="+activityid+";";
        try {
            rs = stmt.executeQuery(query);
            while(rs.next()){
                res.Date = LocalDateTime.parse(rs.getString("Date"), formatter);
                res.ActivityId = Long.parseLong(activityid);
                chks.add(res);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        Check[] c = new Check[chks.size()];
        return chks.toArray(c);
    }
    public  Badge[] GetBadges() {
        Badge res = new Badge(" ",0);
        ArrayList<Badge> badges = new ArrayList<Badge>();
        query = "SELECT Name, Points FROM Badge;";
       try {
            rs = stmt.executeQuery(query);
           while(rs.next()){
               res.Name = rs.getString("Name");
               res.Points = rs.getInt("Points");
               badges.add(res);
           }
       }
       catch (SQLException e) {
           e.printStackTrace();
       }
        Badge[] b = new Badge[badges.size()];
        return badges.toArray(b);
    }

    //POST (new)
    public  void PostAuth(Auth gotten) {
        query = "INSERT INTO User (id, Name, Pass) VALUES ("+ gotten.Id+", "+gotten.Name+ ",
"+gotten.Password+");";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public  void PostUser(User user) {
        query = "INSERT INTO User (Lvl, Xp) VALUES ("+ user.Level+", "+user.Experience+ ") WHERE
id ="+user.Id+";";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public  void PostActivity(Activity activity) {
        query = "INSERT INTO Activity (id, id_User, Name) VALUES ("+ activity.Id+",
"+activity.UserId+ ", "+activity.Name+");";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public  void PostCheck(Check check) {
        query = "INSERT INTO Check (id, id_Activity, Date) VALUES ("+ check.Id+",
"+check.ActivityId+ ", "+check.Date+");";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public  void PostFriend(Long id, Long friendId) {
        query = "INSERT INTO Followers (id, id_UserWatcher, id_UserWatching) VALUES ("+ id+",
"+friendId+ ");";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //PUT (update)
    public  void PutAuth(Auth auth) {
        query = "UPDATE User SET (Name, Pass) VALUES ( "+auth.Name+ ", "+auth.Password+") WHERE
id="+ auth.Id+";";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

```
    public  void PutUser(User user) {
        query = "";//!
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public  void PutActivity(Activity activity) {
        query = "UPDATE Activity SET (id_User, Name) VALUES ("+ activity.Id+", "+activity.UserId+
") WHERE id="+ activity.UserId+";";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //DELETE
    public  void DeleteUser(String userid) {
        query = "DELETE FROM User WHERE id="+ userid+";";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public  void DeleteActivity(String activityid) {
        query = "DELETE FROM Activity WHERE id="+ activityid+";";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public  void DeleteFriend(Long id, Long friendId) {
        query = "DELETE FROM Followers WHERE id_UserWatcher="+ id+", id_UserWatching="+friendId+
";";
        try {
            rsIUD = stmt.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public  void CloseConnection(){
        try { con.close(); } catch(SQLException se) { /*can't do anything */ }
        try { stmt.close(); } catch(SQLException se) { /*can't do anything */ }
    }
}
```

Activity.java

```
package Model;

public class Activity extends Identified {
    public Long UserId;
    public String Name;

    public Activity() {
        GenerateId();
    }

    public Activity(String name, Long userId) {
        GenerateId();
        UserId = userId;
        Name = name;
    }
}
```

Auth.java

```
package Model;

import Controller.DatabaseController;

public class Auth extends Identified{
```

```
    public String Name;
    public String Password;

    public Auth() {
        GenerateId();
    }

    public Auth(String name, String password) {
        GenerateId();
        Name = name;
        Password = password;
    }

    public Auth(Long id, String password) {
        Id = id;
        Password = password;
        DatabaseController db = DatabaseController.getInstance();
        User user = db.GetUser(Id.toString());
        Name = user.Name;
    }
}
```

Badge.java

```
package Model;

import Controller.DatabaseController;

public class Badge {
    public String Name;
    public int Points;

    public static Badge[] Badges;

    public Badge(String name, int points) {
        Name = name;
        Points = points;
    }

    public static void LoadBadges()
    {
        DatabaseController db = DatabaseController.getInstance();
        Badges = db.GetBadges();
    }
}
```

Check.java

```
package Model;

import java.time.LocalDateTime;

public class Check extends Identified {
    public Long ActivityId;
    public LocalDateTime Date;

    public Check(Long activityId) {
        GenerateId();
        ActivityId = activityId;
        Date = LocalDateTime.now();
    }
}
```

Identified.java

```
package Model;

import java.util.UUID;

public class Identified {
    public Long Id;

    public void GenerateId()
    {
        Id = UUID.randomUUID().getMostSignificantBits() & Long.MAX_VALUE;
```

```
    }
}
```

User.java

```
package Model;

import java.time.LocalDateTime;

public class User extends Identified{
    //idgaf about a smell of this
    public String Name;

    public int Level = 1;
    public int Experience = 0;
    public LocalDateTime[] Badges = new LocalDateTime[Badge.Badges.length];

    public Long[] FriendsId;

    public User(String name) {
        GenerateId();
        Name = name;
    }
}
```

ДОДАТОК В. ЗАПИТ СТВОРЕННЯ БД MYSQL

```
CREATE TABLE `User` (
	`id` bigint(20) NOT NULL,
	`Name` varchar(128),
	`Pass` varchar(128),
	`Lvl` int(11),
	`Xp` int(11),
	PRIMARY KEY (`id`)
);

CREATE TABLE `Activity` (
	`id` bigint(20) NOT NULL,
	`Private` bool,
	`Name` varchar(128),
	`Type` bool,
	`Min` int(11),
	`Max` int(11),
	`id_User` int(11) NOT NULL,
	PRIMARY KEY (`id`)
);

CREATE TABLE `Check` (
	`id` bigint(20) NOT NULL,
	`Date` DATE,
	`EndDate` DATE,
	`Value` int(11),
	`id_Activity` int(11),
	PRIMARY KEY (`id`)
);

CREATE TABLE `Badge` (
	`id` bigint(20) NOT NULL,
	`Name` varchar(128),
	`Points` int(11),
	`Description` varchar(1000),
	PRIMARY KEY (`id`)
);

CREATE TABLE `Panel` (
	`id` bigint NOT NULL,
	`Autocommit` bool,
	`Private` bool,
	`Name` varchar(128),
	`Image` varchar(128),
	`Color` varchar(6),
	PRIMARY KEY (`id`)
);

CREATE TABLE `PanelActivites` (
	`id` bigint(20) NOT NULL,
```

```
	`id_Panel` bigint(20) NOT NULL,
	`id_Activity` bigint(20) NOT NULL,
	PRIMARY KEY (`id`)
);

CREATE TABLE `PanelTags` (
	`id` bigint(20) NOT NULL,
	`id_Panel` bigint(20) NOT NULL,
	`id_Tag` bigint(20) NOT NULL,
	PRIMARY KEY (`id`)
);

CREATE TABLE `Tag` (
	`id` bigint(20) NOT NULL,
	`Tag` varchar(128),
	`id_User` bigint(20) NOT NULL,
	PRIMARY KEY (`id`)
);

CREATE TABLE `UserBadges` (
	`id` bigint(20) NOT NULL,
	`id_User` bigint(20) NOT NULL,
	`id_Badge` bigint(20) NOT NULL,
	`Date` DATE,
	PRIMARY KEY (`id`)
);

ALTER TABLE `Activity` ADD CONSTRAINT `Activity_fk0` FOREIGN KEY (`id_User`) REFERENCES `User`(`id`);

ALTER TABLE `Check` ADD CONSTRAINT `Check_fk0` FOREIGN KEY (`id_Activity`) REFERENCES `Activity`(`id`);

ALTER TABLE `PanelActivites` ADD CONSTRAINT `PanelActivites_fk0` FOREIGN KEY (`id_Panel`) REFERENCES `Panel`(`id`);

ALTER TABLE `PanelActivites` ADD CONSTRAINT `PanelActivites_fk1` FOREIGN KEY (`id_Activity`) REFERENCES `Activity`(`id`);

ALTER TABLE `PanelTags` ADD CONSTRAINT `PanelTags_fk0` FOREIGN KEY (`id_Panel`) REFERENCES `Panel`(`id`);

ALTER TABLE `PanelTags` ADD CONSTRAINT `PanelTags_fk1` FOREIGN KEY (`id_Tag`) REFERENCES `Tag`(`id`);

ALTER TABLE `Tag` ADD CONSTRAINT `Tag_fk0` FOREIGN KEY (`id_User`) REFERENCES `User`(`id`);

ALTER TABLE `UserBadges` ADD CONSTRAINT `UserBadges_fk0` FOREIGN KEY (`id_User`) REFERENCES `User`(`id`);

ALTER TABLE `UserBadges` ADD CONSTRAINT `UserBadges_fk1` FOREIGN KEY (`id_Badge`) REFERENCES `Badge`(`id`);
```

ДОДАТОК Г. СПИСОК ВИКОРИСТАНИХ СКОРОЧЕНЬ ТА ТЕРМІНІВ

1. REST - Representational State Transfer, «передача репрезентативного стану»
2. API - Application Programming Interface, Прикладни́й програ́мний інтерфе́йс
3. JSON - JavaScript Object Notation, об'єктний запис JavaScript
4. GSON – Бібліотека, що дозволяє конвертувати об'єкти JSON в Java-об'єкти і навпаки.
5. Singleton – паттерн Одинак.