Національний технічний університет України
«Київський політехнічний інститут»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Розрахунково-графічна робота

по курсу

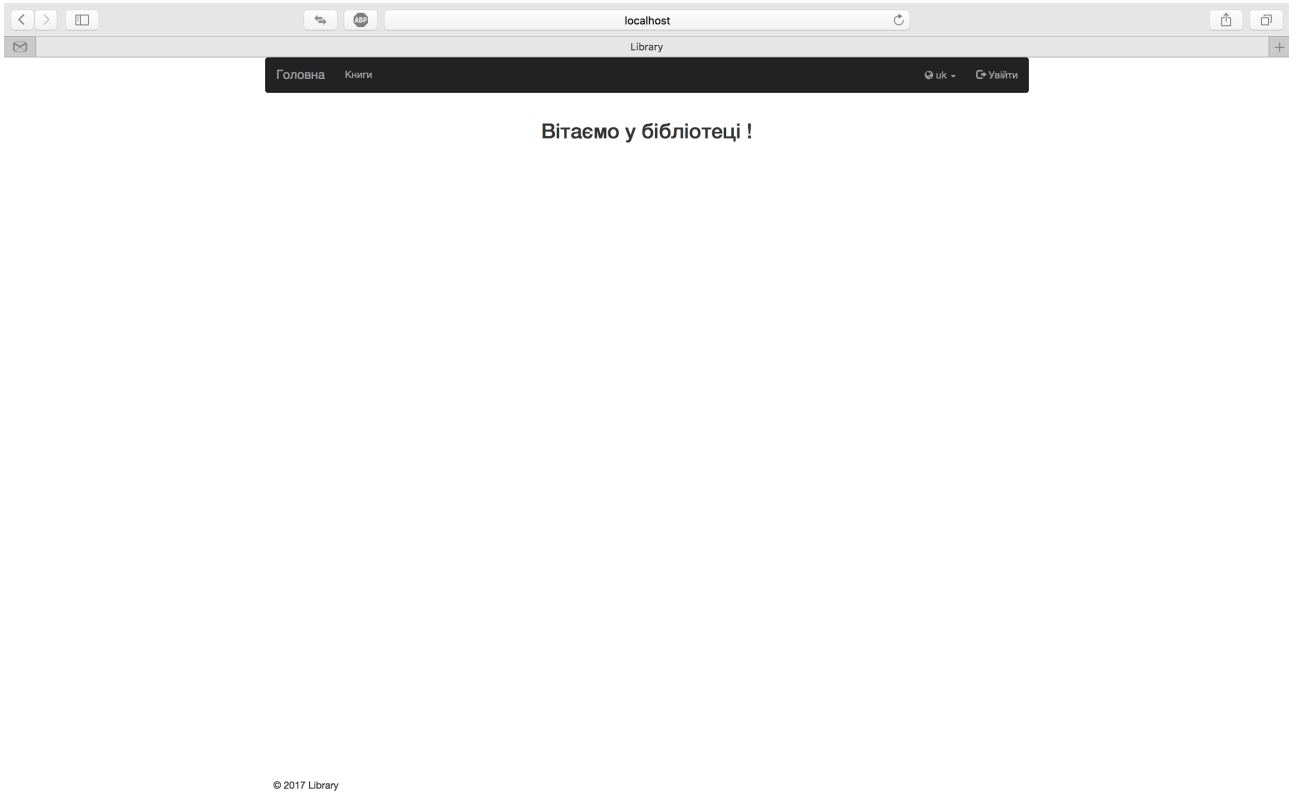
«Інтеграційні програмні системи »

Виконали: студенти 4курсу
ФІОТ гр. ІО-43
в складі:
Кушмирук Р.
Дьомін В.
Рижановский А.
Ковтун А.

Київ 2017

1. Опис проекту

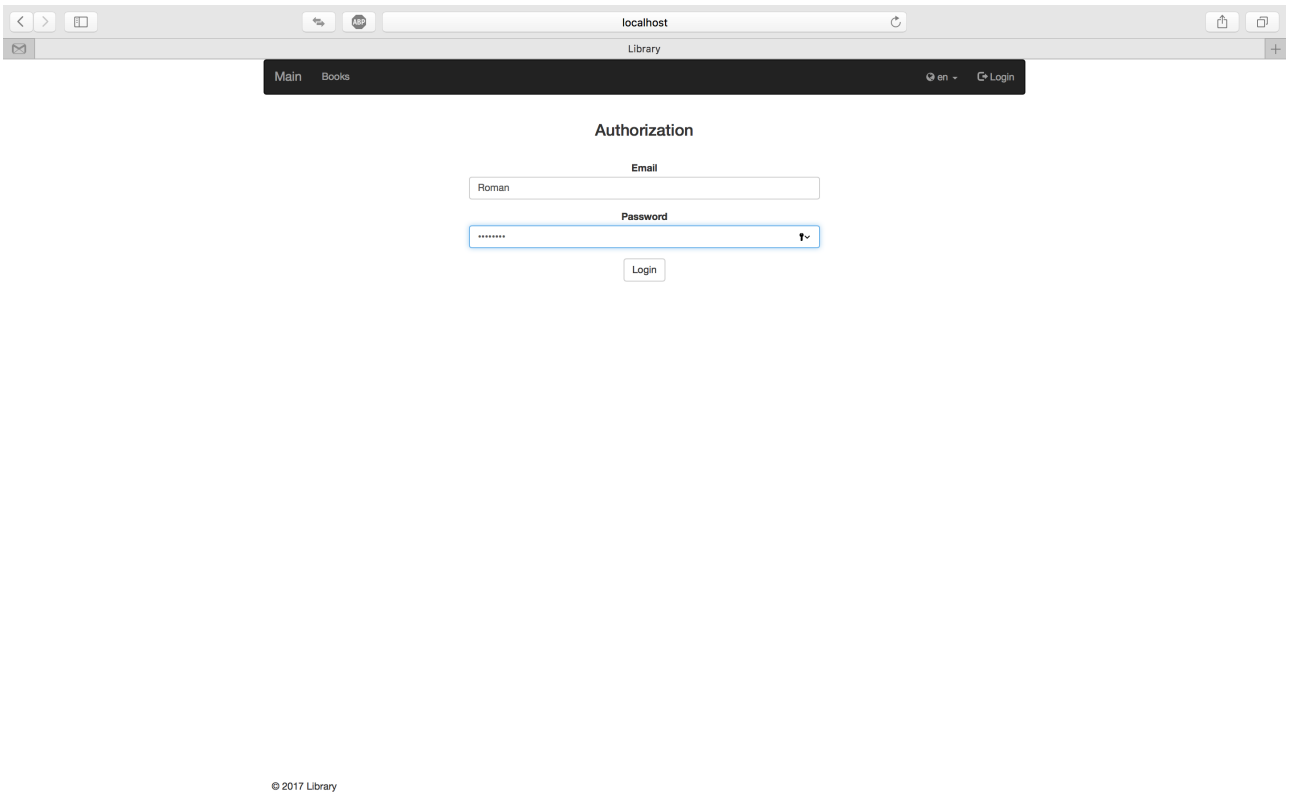
Розроблений нами проект це Система Бібліотека: Читач має можливість здійснювати пошук і замовлення Книг в каталозі. Бібліотекар видає Читачеві Книгу на абонемент або в читальнийзал. Книга може бути присутнім в Бібліотеці в одному або несколькіхекземплярах.



Скріншот роботи

Серверна частина сервісу розроблена за допомогою засобів Java EE. В основі архітектури лежить паттерн MVC, який реалізовано за допомогою Spring MVC та Hibernate. Під час запуску сервера відбувається очистка embedded бази і заповнення її актуальними записами.

Також присутня авторизація користувачів



2.Unit-тести і Mock

функціональне тестування

Проведення функціонального тестування, як правило, пов'язано зі створенням спеціальної групи фахівців, що займаються тестуванням. На цьому етапі додаток розгортається в чинному оточенні і перевіряється його відповідність ТЗ (Технічному Завданням) і пред'явленим функціональним вимогам. Команда тестувальників використовує комплекс автоматизованих і ручних тестів.

Автоматизувати процес функціонального тестування можна, якщо програма включає API (Application Programming Interface) - інтерфейс прикладного програмування, на якому воно побудовано. Однак наявність інтерфейсу в додатку (desktop, web) істотно знижує можливості повної автоматизації даного процесу.

інтеграційне тестування

Стратегія інтеграційного тестування грунтується на перевірці прикладного коду в оточенні, близькому до фактичного оточенню, але не є ним. Головна мета даної стратегії - переконатися в правильності взаємодії коду з зовнішніми ресурсами та взаємодії різних технологій в додатку між собою.

В інтеграційному тестуванні не потрібно використовувати фіктивні дані, як при модульному тестуванні. Замість цього в інтеграційних тестах часто використовуються бази даних, що знаходяться в пам'яті, які легко можна створювати і знищувати під час виконання тестів. База даних в пам'яті - це справжнісінька база даних, що дає можливість перевірити правильність роботи сутностей JPA. Але все ж ця база даних не зовсім справжня - вона лише імітує справжню базу даних для цілей інтеграційного тестування.

модульне тестування

Метою модульного тестування є перевірка роботи прикладної логіки всього програми або окремих його частин при різних вихідних даних, і аналіз правильності отриманих результатів. Незважаючи на те, що мета модульного тестування виглядає простою і зрозумілою, реалізація цього типу тестування може виявитися дуже складною і заплутаною справою, особливо при наявності «старого» коду. Основні прийоми проведення модульного тестування спираються на такі базові принципи:

зовнішні ресурси не використовуються, тобто неприпустимо підключення до баз даних, веб-службам і т.п .;

кожен клас має свій тест;

тестуються лише загальнодоступні методи або інтерфейси, а внутрішній код тестується за рахунок зміни вхідних даних;

для отримання даних, необхідних тестируемой логіці, повинні створюватися фіктивні залежності.

При проведенні модульного тестування для створення фіктивних класів-залежностей можна використовувати простий, але потужний фреймворк Mockito спільно з JUnit.

```
@RunWith(SpringRunner.class)
@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE_EACH_TEST_METHOD)
@ContextConfiguration("META-INF/spring/app-context-xml.xml")
public class AuthorTest {

    AuthorRepository authorRepository;

    @Autowired
    AuthorService authorService;

    @Before
    public void init() { authorRepository = mock(AuthorRepository.class); }

    @Test
    public void getAllAuthorsTest() {
        Author author = new Author("Roman", "Kushmyrul", "Ukraine");
        List<Author> authors = new ArrayList<>();
        authors.add(authorService.save(author));
        authors.add(authorService.save(author));
        authors.add(authorService.save(author));

        Author otherAuthor = new Author("Dima", "Chaliy", "Estonia");
        authors.add(authorService.save(otherAuthor));

        when(authorRepository.findAll()).thenReturn(authors);
        List<Author> result = Lists.newArrayList(authorRepository.findAll());
        verify(authorRepository, atLeastOnce()).findAll();

        long expectedValue = 4L;
        long actualValue = result.size();
        assertThat(expectedValue, is(actualValue));
    }
```

3. Система атоматичної збірки. Maven

Maven - це засіб автоматизації роботи з програмними проектами, який спочатку використовувався для Java проектів. Використовується для управління (management) та складання (build) програм. Має простіший вигляд щодо build-налаштувань, яке надається в форматі XML. XML-файл описує проект, його зв'язки з зовнішніми модулями і компонентами, порядок будування (build), папки та необхідні плагіни. Сервер із додатковими модулями та додатковими бібліотеками розміщується на серверах. Раніше Maven, де він був частиною *Jakarta Project*.

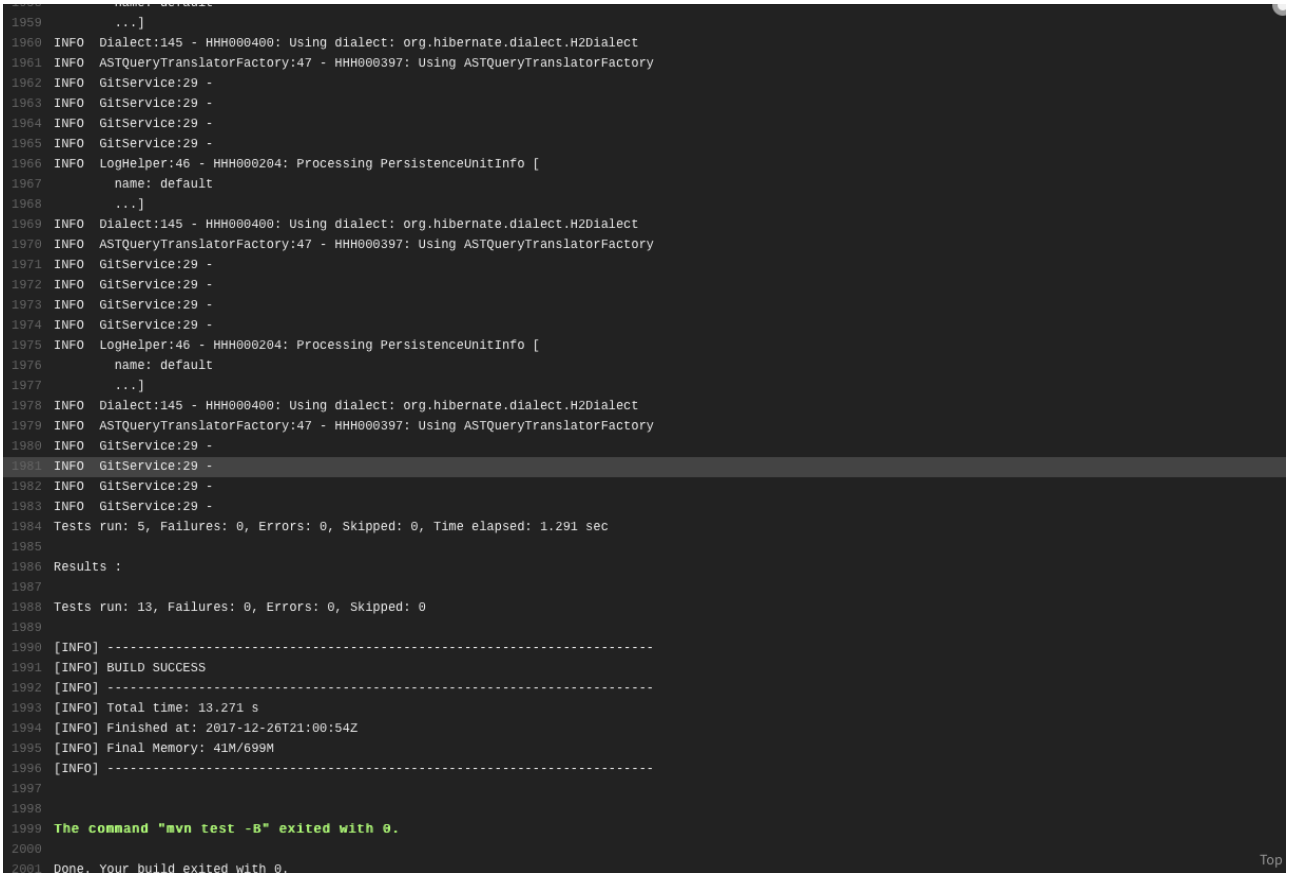
Для опису програмного проекту який потрібно побудувати (*build*), Maven використовує конструкцію відому як Project Object Model (POM), залежності від зовнішніх модулів, компонентів та порядку побудови. Виконання певних, чітко визначених задач - таких, як компіляція коду та пакетування відбувається шляхом досягнення заздалегідь визначених цілей (targets).

Ключовою особливістю Maven є його мережева готовність (network-ready).

Двигун ядра може динамічно завантажувати плагіни з репозиторію, того самого репозиторію, що забезпечує доступ до багатьох версій різних Java-проектів з відкритим кодом, від Apache та інших організацій та окремих розробників. Цей репозиторій та його реорганізований наступник, - Maven 2 репозиторій, - намагається бути де-факто механізмом для дистрибуції Java програм, але прийняття його в такій якості йде повільно.

Maven забезпечує підтримку побудови не просто перебираючи файли з цього репозиторію, але й завантажуючи назад артефакти у кінці побудови. Локальний кеш звантажених артефактів діє як первісний засіб синхронізації виходу проектів на локальній системі.

Результат виконання maven compile для збірки проекту:



4. Сервер безперервної інтеграції. Travis-ci

Принцип досить простий: на окремій машині працює якась служба, в обов'язки якої входить отримання вихідного коду проекту, його збірка, тестування, логування, а також можливість надати для аналізу дані виконання перерахованих операцій.

Travis-сі підтримує безліч мов програмування серед. Почати користуватися сервісом дуже просто. Потрібно всього лише зробити кілька кроків, які детально описані у власному Гайд проекту. Я лише опишу процес в цілому.

Короткий перелік багів, які шукає утиліта:

- Проблеми продуктивності, пов'язані з розміткою інтерфейсу
- Невикористані ресурси
- Невідповідності розмірів масивів (коли масиви визначені у множинних конфігураціях).
- Проблеми доступності та інтернаціоналізації («магічні» рядки, відсутність атрибуту contentDescription і т.д)
- Проблеми з іконками (невідповідності розмірів, порушення DRY)
- Проблеми зручності використання (Наприклад, не зазначений спосіб введення для текстового поля)
- Помилки в маніфесті

4. Експоненціальна витримка

Для вирішення задачі раптового зникнення з’єднання з базою даних було вирішено задачу експоненціальної витримки (для того, щоб не потрібно було виконувати перезапуск сервера). Для цього ми задіяли технологію SpringRetry. Конфігурація представлена нижче:

```
@EnableRetry
public interface GenericService<T>  {
    @Retryable(
        value = CannotCreateTransactionException.class,
        backoff = @Backoff(delay = Delay.DELAY,
            maxDelay = Delay.MAX_DELAY,
            multiplier = Delay.MULTIPLIER))
    List<T> findAll();

    @Retryable(
        value = CannotCreateTransactionException.class,
        backoff = @Backoff(delay = Delay.DELAY,
            maxDelay = Delay.MAX_DELAY,
            multiplier = Delay.MULTIPLIER))
    T findById(Long id);

    @Retryable(
        value = CannotCreateTransactionException.class,
        backoff = @Backoff(delay = Delay.DELAY,
            maxDelay = Delay.MAX_DELAY,
            multiplier = Delay.MULTIPLIER))
    T save(T entity);

    @Retryable(
        value = CannotCreateTransactionException.class,
        backoff = @Backoff(delay = Delay.DELAY,
            maxDelay = Delay.MAX_DELAY,
            multiplier = Delay.MULTIPLIER))
    void delete(Long id);

}
```

Формула витримки складетеся лише з двох параметрів. Мінімальний час у степені кількості спроб. Отриманий графік можна переглянути нижче