МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

Тема: Описание предполагаемого способа решения

Студент гр. 4304 _____ Михайлов А.А.

Руководитель _____

Санкт-Петербург

2019

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 2
1. СОЗДАНИЕ ПРОТОТИПА «ПЕСОЧНИЦЫ» 3
1.1. Определение понятия «песочница» 3
1.2. Реализация прототипа 3
2. АРХИТЕКТУРА ПРИЛОЖЕНИЯ 5
2.1. Описание используемых технологий 5
2.2. Архитектура приложения 7
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 11

1. СОЗДАНИЕ ПРОТОТИПА «ПЕСОЧНИЦЫ»

1.1. Определение понятия «песочница»

Песочница (sandbox) – в компьютерной безопасности это механизм для безопасного исполнения программ.

Обычно песочница – это жестко контролируемый набор ресурсов, предназначенный для исполнения гостевой программы, например место в памяти или на диске. Доступ к сети, возможность сообщения с главной ОС или считывание информации с устройства ввода зачастую либо эмулируют, либо ограничивают. По сути – песочница это пример виртуализации. Часто песочницы используют при разработке ПО для запуска еще «сырого» кода, который возможно может повредить систему либо сложную конфигурацию. Песочницы копируют самые основные элементы среды, для которой был написан код, и позволяют разработчикам безопасно экспериментировать.

В нашем случае песочница – это симуляция браузера внутри браузера, которая позволит пользователю на странице веб-приложения писать HTML, CSS и JS-код, который будет отображаться на этой же странице веб-приложение в специальной области, которая и будет симулировать поведение браузера.

1.2. Реализация прототипа

На рисунке 1.1 представлен прототип песочницы.

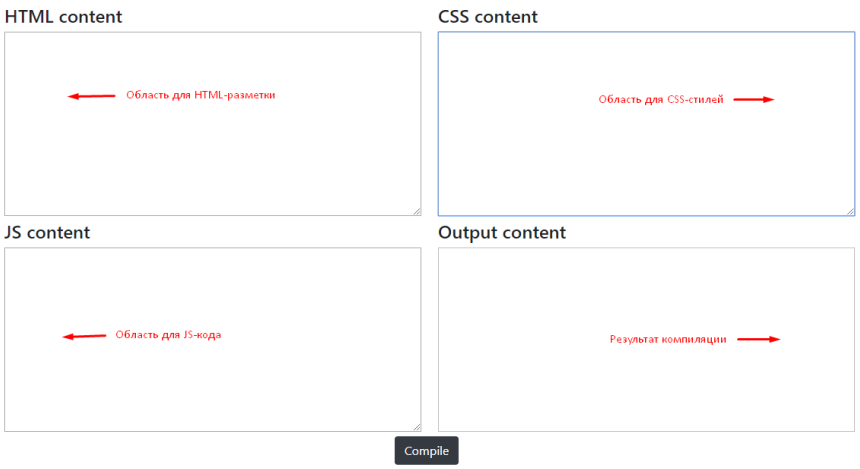

Рисунок 1.1. Прототип песочницы

На рисунке 1.2 представлен прототип песочницы с содержимым внутри блоков «HTML content», «CSS content» и «JS content».

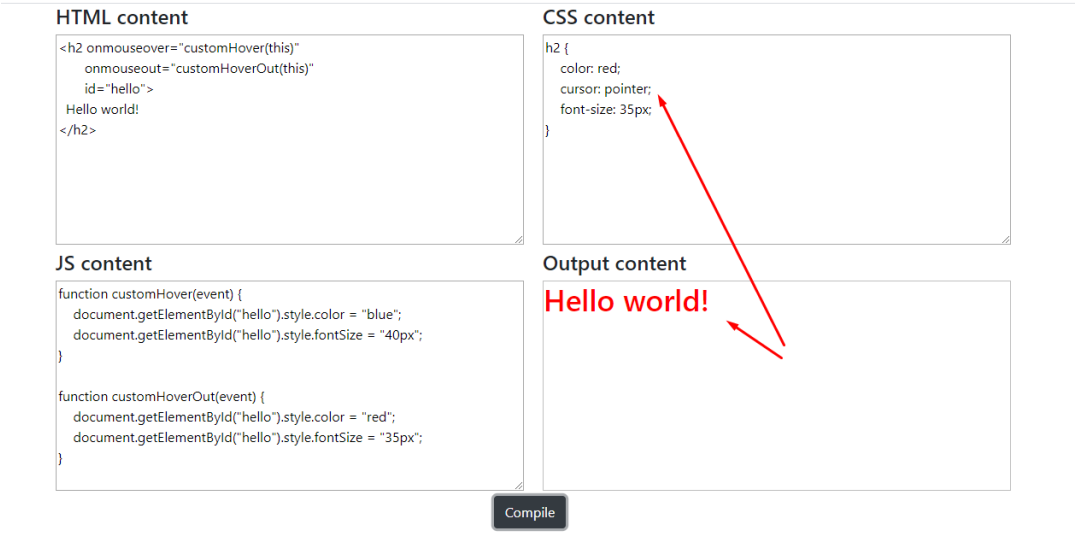

Рисунок 1.2. Прототип песочницы с содержимым внутри блоков

На рисунке 1.2 в блоке «Output content» отображается разметка из блока «HTML content», к котором применен стиль из блока «CSS content». Именно поэтому текст имеет красный текст. К данному тексту привязан JS-скрипт из блока «JS content», который исполнится при наведении на него курсора мыши.

На рисунке 1.3 представлен прототип песочницы, когда на него наведен курсор мыши (использован эффект hover).

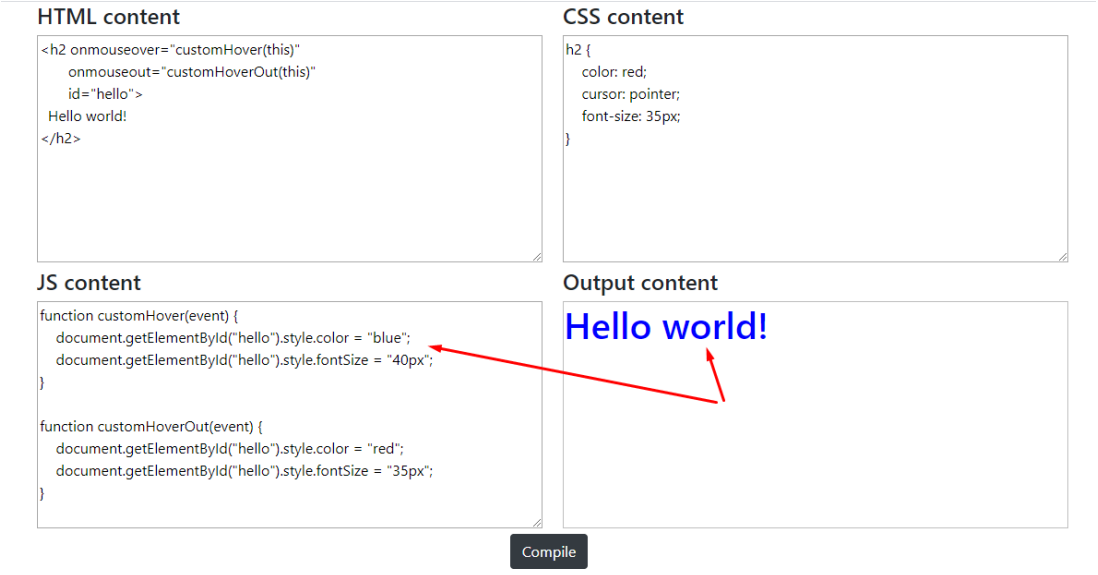

Рисунок 1.3. Прототип песочницы с примененным hover-эффектом

2. АРХИТЕКТУРА ПРИЛОЖЕНИЯ

2.1. Описание используемых технологий

Разрабатываемый инструмент представляет собой веб-приложение. В качестве языка программирования (ЯП) был выбран язык Java. Было принято решение использовать язык Java. Этот выбор довольно просто обосновать, потому что Java имеет действительно большое число различных преимуществ[2, 3]:

- Java – это объектно-ориентированный язык программирова-ния. Это позволяет с легкостью моделироваться системы любой сложности и расширять уже существующие;
- Данный язык программирования имеет довольно низкий порог вхождения;
- Just-In-Time компилятор Java позволяет добиться действитель-но высокой производительности;
- Данный язык очень распространен в мире ИТ, что позволяет без особых проблем привлечь к разработке новых специали-стов. Абсолютное большинство всех известных проблем, с ко-торыми можно встретиться в процессе разработки, уже решены и решения эти с легкостью можно найти в сети Интернет;
- Обширный стек технологий;
- Пожалуй, одно из самых больших преимуществ языка Java над остальными – платформонезависимость. Любая Java-программа компилируется в байт-код, который может выпол-няться на любой машине независимо от ее платформы с помо-щью Java Virtual Machine (JVM);

После того, как сам язык был выбран, необходимо определиться с остальным стеком технологий, которые будут использоваться в процессе

разработки. Для реализуемого программного продукта выбор СУБД не критично важен, поэтому остановиться можно на любой из них. Выбор пал на MongoDB, так как имеет внутреннюю поддержку для реактивного стиля программирования, который будет использоваться при реализации.

Сборщиком проектов был выбран Maven. Выбор обусловлен тем, что это, де-факто, стандартный сборщик в мире Java Enterprise-приложений.

В качестве интегрированной среды разработки была выбрана Intellij IDEA от компании JetBrains, так как это самая удобная среда разработки.

В качестве основного фреймворка для разработки был выбран Spring Framework по нескольким причинам:

- Он содержит всю необходимую инфраструктуру для создания современного и высоконагруженного Enterprise-приложения;
- Гибкость конфигурации;
- Использование Spring позволяет масштабировать систему в будущем с минимальным количеством затрачиваемых усилий;
- Spring поощряет подход использования слабой связанности между компонентами. Как следствие:
    - Упрощение инициализации и настройки компонентов;
    - Упрощение модульного тестирования;
    - Упрощение разработки и дальнейшей поддержи приложения в целом;

Помимо всего этого, стоит также отметить, что Spring Framework – это, пожалуй, самое популярное решение в плане разработки enterprise-приложений на языке Java и Kotlin. Оно позволяет конфигурировать и разрабатывать приложения абсолютно любой сложности. У Spring Framework одно из самых больших сообществ разработчиков в мире, это по большей части связано с тем, что данный фреймворк является open-source проектом, что также является значительным плюсом. Spring обладает весьма развитой экосистемой, большим числом независимых модулей, которые используются по мере необходимости[4]. Однако

есть и ряд существенных минусов. Основные из них:

- Относительно высокий порог вхождения;
- Сложность конфигурации. Цена за предоставляемую гибкость. Поэтому Spring Framework нецелесообразно использовать для небольших проектов. Это также одна из причин высокого порога вхождения для новичков.

Ознакомиться с инфраструктурой Spring Framework можно на рисунке 2.1.

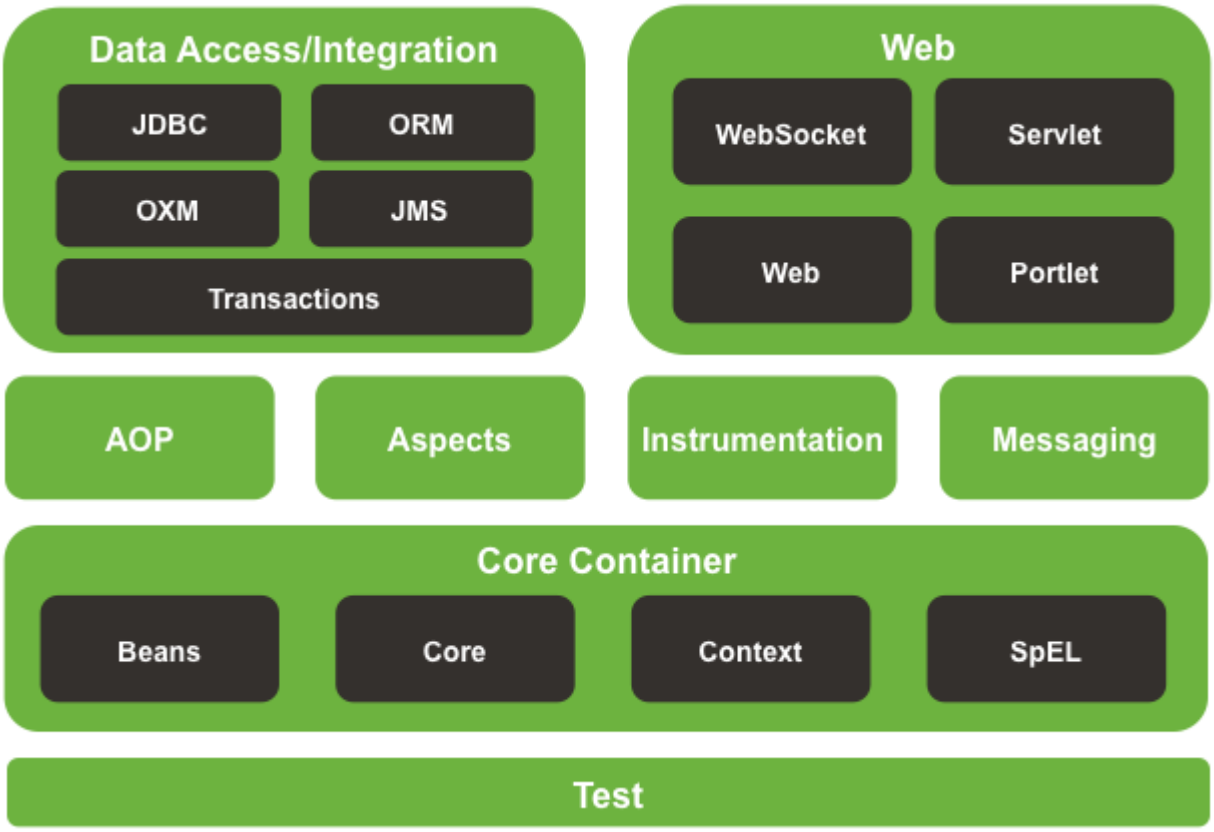

Рисунок 2.1. – Инфраструктура Spring Framework

В качестве контейнера сервлетов был выбран встроенный в Spring Jetty.

2.2. Архитектура приложения

Один из ключевых и самых основных принципов проектирования для достижения таких характеристик – разделение ответственности. Он базируется на том, что каждый отдельно взятый компонент или модуль должен быть ответственен только за одно конкретное свойство или функцию в приложении.

Это относится как к приложению целиком, так и к отдельным частям. В дальнейшем для реализации приложения будет использоваться микросервисная архитектура, где каждый сервис будет представлять из себя цельную с семантической точки зрения единицу. Каждый сервис будет спроектирован по принципу многоуровневой архитектуры.

Многоуровневая архитектура имеет следующие достоинства: абстракция, изоляция, более прозрачная организация, масштабируемость отдельных модулей, независимое тестирование функциональности отдельных модулей, возможность переиспользования модулей.

Чаще всего используются трехуровневая модель. Она состоит из уровня представления, уровня бизнес-логики и уровня доступа к данным. Задача уровня представления заключается в коммуникации с пользователем: преобразование данных для визуализации, предоставление интерфейса взаимодействия и последующая обработка действий пользователя. Бизнес-уровень содержит все функциональные алгоритмы, выступает в роли посредника между уровнем представления и уровнем доступа к данным. Последний же представляет из себя хранилище данных и набор унифицированных методов доступа к этим данным, известных как CRUD (Create, Update, Delete).

Более наглядно ознакомиться с данной моделью многоуровневой архитектуры можно на рисунке 2.2.

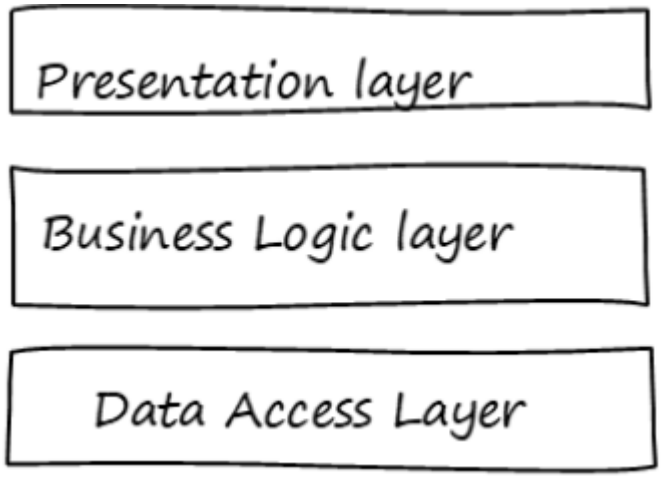

Рисунок 2.2. – Трехуровневая модель архитектуры

В реализуемом приложении используется именно трехуровневая модель

архитектуры.

Для реализации уровня доступа к данным используется паттерн проектирования DAO (Data Access Object). DAO – это объект, который предоставляет абстрактный интерфейс для работы с БД или каким-либо другим хранилищем данных. Использование данного паттерна предоставляет ряд возможностей работы с данными независимо от того, какой именно механизм хранения данных используется.

Уровень бизнес-логики реализован с помощью сервисов – набора интерфейсов, который используются уровнем представления для совершения манипуляций над данными.

Для реализации уровня представления был использован паттерн проектирования, известный как MVC (Model-View-Controller). Наглядно он продемонстрирован на рисунке 2.3.

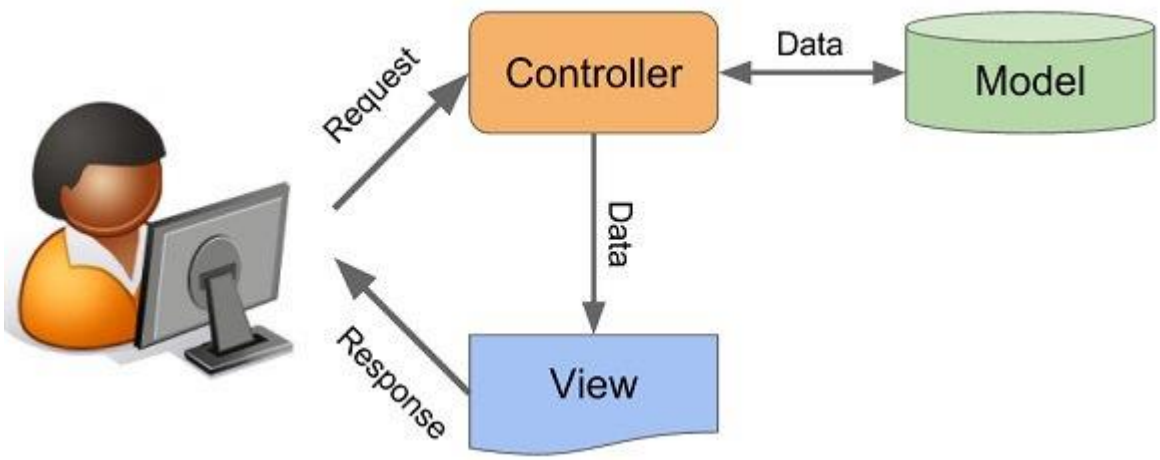

Рисунок 2.3. – Паттерн Model-View-Controller

Представление (View) отвечает за отображение данных модели пользователю. Пользователь взаимодействует с предоставляемым ему графическим интерфейсом, тем самым изменяя модель. Такие изменения обрабатываются контроллером.

Контроллер (Controller) – это интерпретатор действий пользователя, который с помощью интерфейса взаимодействия с уровнем бизнес-логики оповещает модель о необходимости изменения.

Модель (Model), в свою очередь, представляет из себя набор данных,

которые могут видоизменяться, реагируя на команды контроллера[1].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Свободная энциклопедия Википедия. [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Model-View-Controller (дата обращения: 25.12.2019).
2. Б. Эккель Философия Java СПб.: Питер, 2003. 976 с.
3. Г. Шилдт Java 8. Полное руководство М.: Вильямс, 2015. 1376 с.
4. К. Уоллс Спринг в действии. 3-е полное изд. СПб: ДМК Пресс, 2013. 721 с.