Лабораторная работа №3

Ход работы:

Часть 1. Проектирование архитектуры

Для разрабатываемой системы на высоком уровне абстракций:

1. Тип приложения: веб-приложение для выполнения преимущественно на сервере. Реализуется клиент-серверная архитектура.
2. Стратегия распределенного развертывания
3. Используются следующие технологии: Django, JS, Bootstrap.
   Выбор пал на них по следующим причинам:
   a. Имеется опыт работы с большинством из них.
   b. Разработаны для создания клиент-серверных веб-приложений.
   c. Позволяют создавать гибкие приложения не используя грязных хаков.
4. Показатели качества: безопасность, централизованный доступ к данным, простота обслуживания, надёжность, тестируемость.
5. Пути реализации сквозной функциональности.
   a. Аутентификация и авторизация — обеспечение конфиденциальности пользовательских данных.
   b. Кэширование — снижение нагрузки на сервер.
6. “Архитектура To Be”
   Диаграммы компонентов и развёртывания представлены на рисунках.

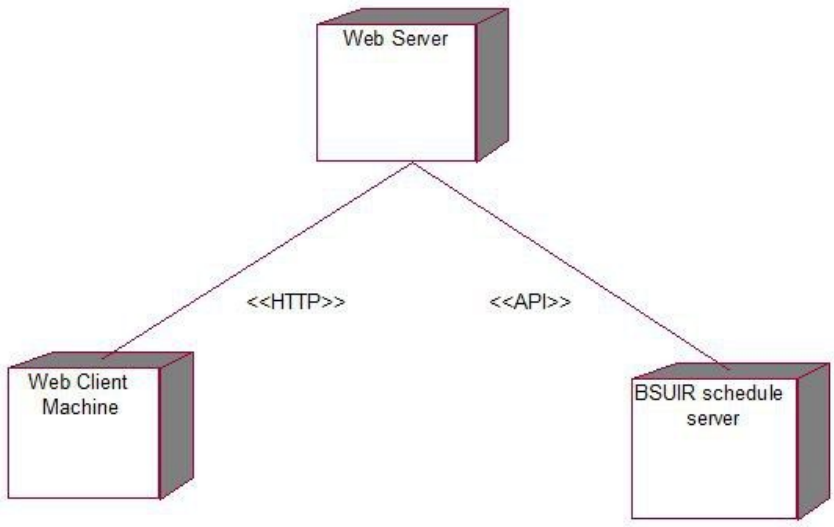

Рисунок 1. Диаграмма развертывания

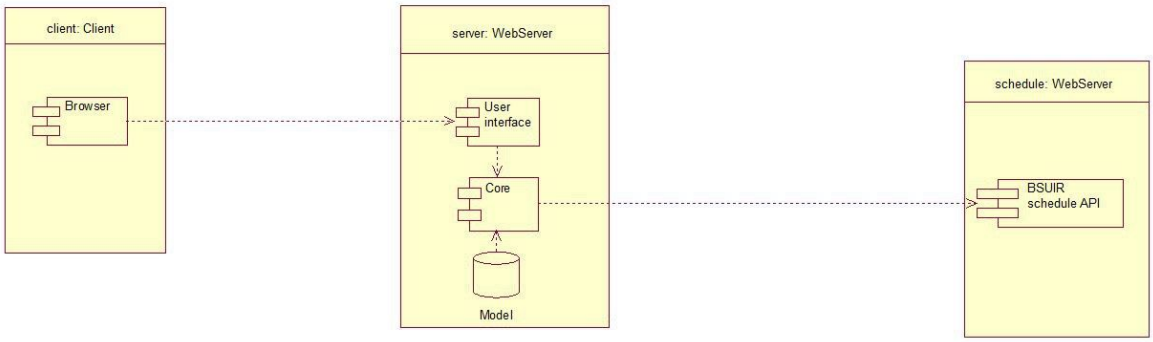

Рисунок 2. Диаграмма компонентов

Часть 2.

1. Анализ архитектуры разрабатываемого приложения
    a. Архитектура — клиент-сервер.
    b. Стратегия развёртывания — нераспределённое развёртывание.

2. Обобщённое представление архитектуры идентично приведённой в предыдущей части.
3. Архитектура «As is»

Диаграмма классов приведена на рисунке

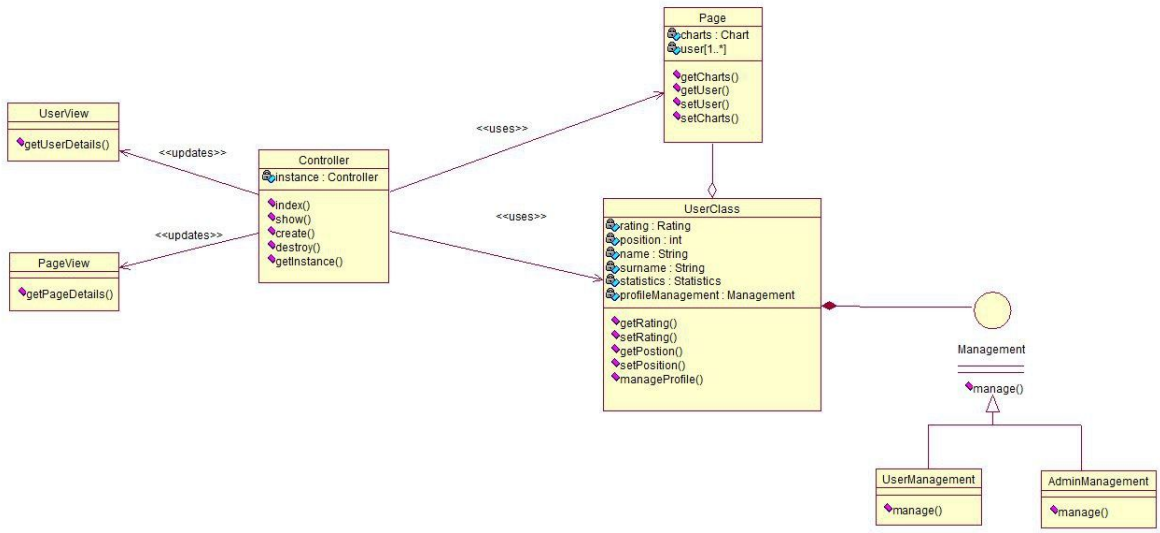

Рисунок 3. Диаграмма классов

Часть 3. Сравнение и рефакторинг

1. Сравнение архитектур «As is» и «To be», отличия и анализ их причин:

Разработчики придерживаются клиент-серверной архитектуры, при этом было решено перенести часть обработки данных(такое как построение графиков) на сервер, чтобы устройства на всех платформах могли их отображать быстро и корректно.

2. Пути улучшения архитектуры
    a. Избежание дублирования функциональности — со временем “раздувает” систему и делает ее поведение непредсказуемой.

b. Избежание дублирования кода (принцип DRY) — замедляет скорость разработки, и, особенно, дебага, ухудшает расширяемость кода.
c. Акцентирование внимания на Single Responsibility Principle при возникновении желания создать парочку классов, которые будут делать всё.
d. Следование принципу KISS