системы является графическим интерфейсом для взаимодействия с сервером. Она реализована с помощью стека технологий jQuery/Pjax.

Третьим компонентом является поисковой сервер Sphinx, который предназначен для взаимодействия пользователя с поиском мест, новостей или скидок. В сервисе используется система полнотекстового поиска Sphinx, отличительной особенностью которой является высокая скорость индексации и поиска, а также интеграция с существующими СУБД и API для распространённых языков веб-программирования.

ЛИТЕРАТУРА

1. Wikipedia. [Электронный ресурс] / Wikipedia.org – Режим доступа: https://wikipedia.org. Дата доступа: 16.04.2019.

2. Github.com. [Электронный ресурс] / Github.com – Режим доступа: https://github.com/yiisoft/yii2. Дата доступа: 16.04.2019.

УДК 004.042

Студ. А. Д. Русакович
Науч. рук. ст. преп. А. С. Наркевич
(кафедра программной инженерии, БГТУ)

ПРИМЕНЕНИЕ ФРЕЙМВОРКА PRISM ПРИ СОЗДАНИИ НАСТОЛЬНЫХ ПРИЛОЖЕНИЙ, ОСНОВАННЫХ НА ТЕХНОЛОГИИ WPF

Составные приложения обычно имеют богатое взаимодействие с пользователем и визуализацию данных, которые воплощают значительную бизнес-логику. Эти приложения обычно взаимодействуют с несколькими серверными системами и службами, используя многоуровневую архитектуру, могут быть физически развернуты на нескольких уровнях. Ожидается, что приложение будет значительно развиваться в течение всего срока службы в ответ на новые требования и возможности для бизнеса. Короче говоря, эти приложения "построены до последнего" и "построены для изменений". Приложения, которые не требуют этих характеристик, не могут извлечь выгоду из использования Prism.

Проектирование и создание приложений в монолитном стиле может привести к решению, которое очень сложно и неэффективно поддерживать. В этом случае "монолитный" относится к приложению, в котором компоненты очень плотно соединены и нет четкого разделения между ними. Как правило, приложения, разработанные и по-

строенные таким образом, страдают от проблем, которые делают жизнь разработчика трудной. Трудно добавить новые функции в систему или заменить существующие функции, трудно устранить ошибки, не нарушая другие части системы, трудно протестировать и развернуть.

Большинство корпоративных приложений сложны и требуют для разработки целую команду программистов и дизайнеров. То, как разрабатывать приложения таким образом, чтобы несколько разработчиков или команд могли эффективно работать над различными частями приложения независимо и быть уверенным, что части приложения гладко соединятся при интеграции, может оказаться серьёзной проблемой.

Эффективным средством решения этих проблем является разделение приложения на ряд дискретных, слабо связанных, полунезависимых компонентов, которые затем могут быть легко интегрированы вместе в "оболочку" приложения для формирования согласованного решения. Приложения, разработанные и построенные таким образом, часто известны как составные приложения. Составные приложения хорошо подходят для различных сценариев клиентских приложений

На рисунке 1 показан пример составного приложения с несколькими серверными системам:

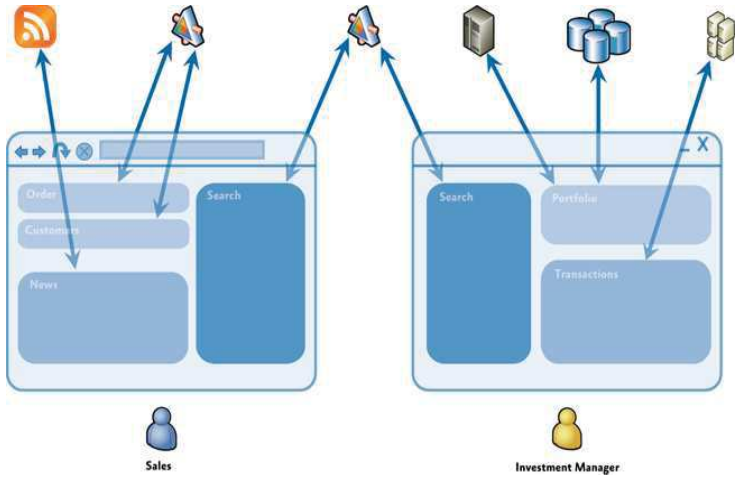

Рисунок 1 –Схема взаимодействия частей приложения с серверными системами.

Prism предоставляет возможности и шаблоны проектирования, которые могут быть вам незнакомы, особенно если вы новичок в разработке шаблонов и составных приложений. Этот раздел содержит краткий обзор основных концепций Prism и определяет некоторые термины, которые будут использоваться в документации и коде:

- **Модули.** Модули — это пакеты функций, которые могут быть независимо разработаны, протестированы и (необязательно) раз-

вернуты. Во многих ситуациях модули разрабатываются и обслуживаются отдельными группами. Типичное приложение Prism построено из нескольких модулей.

- **Shell**. Shell -это хост-приложение, в которое загружаются модули. Shell определяет общую компоновку и структуру приложения, но, как правило, не знает точных модулей, которые она будет размещать. Обычно он реализует общие службы и инфраструктуру приложений, но большая часть функциональности и содержимого приложения реализована в модулях.
- **Представления**. Представления — это элементы управления пользовательского интерфейса, инкапсулирующие пользовательский интерфейс для определенной функции или функциональной области приложения. Представления используются в сочетании с шаблоном MVVM, который используется для обеспечения четкого разделения проблем между пользовательским интерфейсом и логикой представления и данными приложения.
- **Модель представления**. Модели представлений — это классы, инкапсулирующие логику и состояние представления приложения. Они являются частью шаблона MVVM.
- **Модели**. Классы моделей инкапсулируют данные приложения и бизнес-логику. Они используются как часть шаблона MVVM. Модели инкапсулируют данные и любые связанные с ними проверки и бизнес-правила для обеспечения согласованности и целостности данных.
- **Команды**. Команды используются для инкапсуляции функциональных возможностей приложения таким образом, чтобы их можно было определять и тестировать независимо от пользовательского интерфейса приложения. Они могут быть определены как объекты команд или как методы команд в модели представления. Prism предоставляет класс DelegateCommand и класс CompositeCommand. Последний используется для представления коллекции команд, которые вызываются вместе.
- **Регионы**. Регионы — это логические заполнители, определенные в пользовательском интерфейсе приложения (в оболочке или в представлениях), в которые отображаются представления. Регионы позволяют обновлять макет пользовательского интерфейса приложения без внесения изменений в логику приложения.
- **EventAggregator**. Компоненты в составном приложении часто должны взаимодействовать с другими компонентами и службами приложения слабосвязанным образом. Для поддержки этого Prism предоставляет компонент EventAggregator, который реализует меха-

низм событий pub-sub, тем самым позволяя компонентам публиковать события и другим компонентам подписываться на эти события без необходимости ссылки на другие. EventAggregator часто используется, чтобы позволить компонентам, определенным в разных модулях, взаимодействовать друг с другом.

- **Контейнера внедрения зависимостей**. Шаблон инъекции зависимостей (DI) используется в Prism для управления зависимостями между компонентами. Инъекция зависимостей позволяет выполнять зависимости компонентов во время выполнения и поддерживает расширяемость и тестируемость. Prism предназначен для работы с Unity или MEF или с любыми другими контейнерами для инъекций зависимостей через ServiceLocator.
- **Bootstrapper**. Компонент Bootstrapper используется приложением для инициализации различных компонентов и служб Prism. Он используется для инициализации контейнера инъекции зависимостей для регистрации любых компонентов и служб уровня приложения. Он также используется для настройки и инициализации каталога модулей и представления оболочки и модели представления или презентатора.

На рисунке 2 показана типичная архитектура приложения Prism и показано, как все различные возможности Prism могут работать вместе в составном многомодульном приложении:

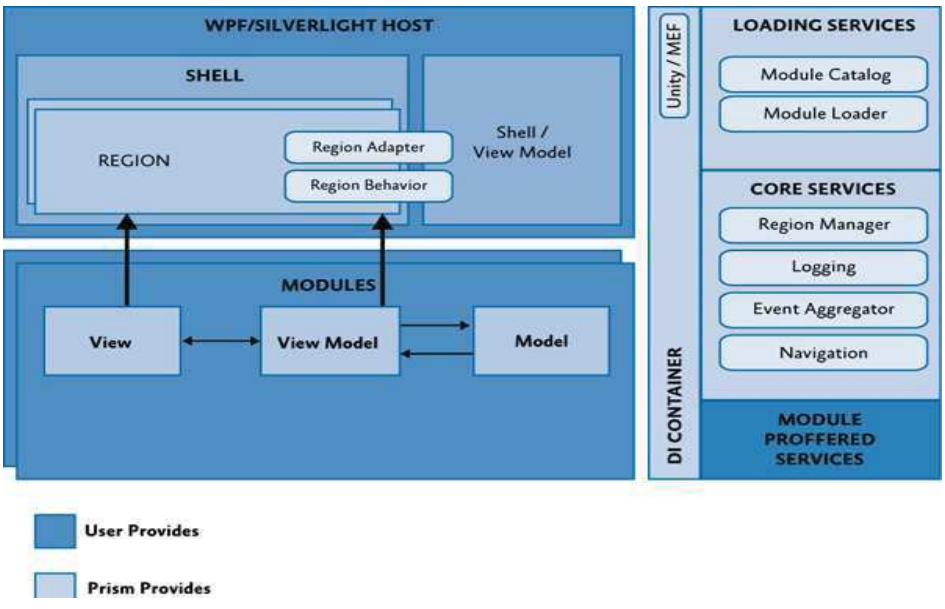

Рисунок 2 –Архитектура приложения, использующего Prism.

Prism разработан таким образом, что вы можете использовать любую из предыдущих возможностей и шаблонов проектирования по отдельности или все вместе, в зависимости от ваших требований и сценария приложения.

ЛИТЕРАТУРА

1. Prismlibrarydocumentation.[Электронный ресурс]. – Электронные данные. – Режим доступа:
http://prismlibrary.github.io/docs/index.html

2. Руководство разработчика Prism. [Электронный ресурс]. – Электронные данные. – Режим доступа:
https://habr.com/ru/post/176851/

УДК004.042

Студ. Д. С. Кунец
Науч. рук. ст. преп.А. С. Наркевич
(кафедра программной инженерии, БГТУ)

ПРОГРЕССИВНЫЕ ВЕБ-ПРИЛОЖЕНИЯ

Цель работы – разработать веб-приложение для создания и прохождения обучающих курсов на языке программирования Scratch.

Решаемые задачи данного приложения:

- составление, изменение и удаление уроков;
- возможность создавать класс и приглашать в него студентов;
- возможность ведения статистики по урокам;
- возможность поиска урока;
- поддержка режима без доступа к сети интернет, который основан на технологии PWA.

Термин «*Progressive Web App*» введен Алексом Расселом и Франсесом Берриманом. По словам Алекса, прогрессивные веб-приложения — это всего лишь веб-сайты, которые принимали правильные витамины. То, что лежит в основе *PWA*, нельзя назвать новым фреймворком или новой технологией. Это, фактически, набор передовых методов разработки, которые позволяют сделать так, чтобы поведение веб-приложения оказалось бы очень похожим на поведение классических настольных или мобильных приложений.

PWA дают пользователю новые удобные возможности посредством прогрессивных улучшений приложений. В целом, это означает, что *PWA*, на мобильных устройствах разных поколений, будут работать примерно одинаково. Конечно, некоторые возможности обычных телефонных приложений могут быть им недоступны, но такие приложения обычно работают на разных устройствах так, как они и должны на них работать.

PWA — это веб приложение, созданное с использованием определенных технологий для достижения заданных целевых показателей, которые представлены на рисунке 1.