УДК 621.9.048.7

О РАЗРАБОТКЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ ПОИСКА ОБЪЕКТОВ НЕДВИЖИМОСТИ, ВАКАНСИЙ И СООБЩЕСТВ

[1]Рудикова-Фронхёфер Л. В., [2]Романчук С. А.
[1]УО «Гродненский государственный университет имени Янки Купалы», Гродно, Беларусь, lada.rudikowa@gmail.com
[2]УО «Гродненский государственный университет имени Янки Купалы», Гродно, Беларусь, sergcom1998@gmail.com

В работе описываются технологии и методы при разработке архитектуры релевантного поиска для задач обработки информации пользователей с последующим выявлением закономерностей и предоставления релевантного результата на поисковый запрос. Формулируются необходимые требования к созданию специализированного программного обеспечения такого рода. Приводится пример реализации.

Обработка данных запроса и предоставление релевантного результата. Процесс получения релевантной информации происходит при использовании многомерного анализа данных в совокупности с нейросетью в конкретной выборке на основе запроса и детального портрета пользователя. Так как сбор подобных данных будет осуществляться в самой информационной системе, то и данные будут добываться изначально в заранее согласованном виде и согласованных форматах хранилищ данных, что упростит сбор, обработку и анализ данных.

Нейронная сеть – математическая модель, а также ее программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. После разработки алгоритмов обучения получаемые модели стали использовать в практических целях: в задачах прогнозирования, для распознавания образов, в задачах управления.

Нейронная сеть представляет собой систему соединенных и взаимодействующих между собой простых процессоров (искусственных нейронов). Такие процессоры обычно довольно просты (особенно в сравнении с процессорами, используемыми в

персональных компьютерах). Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам. И, тем не менее, будучи соединенными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые процессоры вместе способны выполнять довольно сложные задачи.

Нейронные сети не программируются в привычном смысле этого слова, они обучаются. Возможность обучения – одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение. Это значит, что в случае успешного обучения сеть сможет вернуть верный результат на основании данных, которые отсутствовали в обучающей выборке, а также неполных и «зашумленных», частично искаженных данных.

Нейронные сети используют для решения задач связанных с принятием решений и управлением, кластеризацией, прогнозированием и т. д.

Принятие решений и управление. Эта задача близка к задаче классификации. Классификации подлежат ситуации, характеристики которых поступают на вход нейронной сети. На выходе сети при этом должен появиться признак решения, которое она приняла. При этом в качестве входных сигналов используются различные критерии описания состояния управляемой системы.

Кластерный анализ. Под кластеризацией понимается разбиение множества входных сигналов на классы, при том, что ни количество, ни признаки классов заранее не известны. После обучения такая сеть способна определять, к какому классу относится входной сигнал. Сеть также может сигнализировать о том, что входной сигнал не относится ни к одному из выделенных классов – это является признаком новых, отсутствующих в обучающей выборке, данных. Таким образом, подобная сеть может выявлять новые, неизвестные ранее классы сигналов. Соответствие между классами, выделенными сетью, и классами, существующими в предметной области, устанавливается человеком.

Прогнозирование. Способности нейронной сети к прогнозированию напрямую следуют из ее способности к обобщению и выделению скрытых зависимостей между входными и выходными данными. После обучения сеть способна предсказать будущее значение некой

последовательности на основе нескольких предыдущих значений и (или) каких-то существующих в настоящий момент факторов. Прогнозирование возможно только тогда, когда предыдущие изменения действительно в какой-то степени предопределяют будущие. Например, прогнозирование котировок акций на основе котировок за прошлую неделю может оказаться успешным.

Исходя из вышеперечисленного можно сделать вывод, что нейросеть отлично подходит для решения задач поставленных при разработке описываемой информационной системы

Разрабатываемая информационная система представляет собой набор независимых сервисов реализованных с использованием различных технологических решений, в том числе и сторонних.

В системе за релевантный поиск будут отвечать сервисы «AI» (рисунок 1) и «Analyze», как показано на рисунке 1, при взаимодействии которых результатом должны быть исключительно релевантные рекомендации для конкретного пользователя. Логический подход к созданию сервиса «AI» нейронной сети основан на моделировании рассуждений. Теоретической основой служит логика, такой подход может быть проиллюстрирован применением для этих целей языка и системы логического программирования Prolog. Программы, записанные на языке Prolog, представляют наборы фактов и правил логического вывода без жесткого задания алгоритма как последовательности действий, приводящих к необходимому результату. Улучшение результатов поиска и рекомендаций при обработке запросов достигается за счет более точной интерпретации намерений пользователя.

Для реализации задачи связанной с релевантными результатами для предоставления пользователю информации об объектах недвижимости, вакансиях и сообществах будет служить библиотека TensorFlow. TensorFlow – открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. RankBrain (Ранжирующий интеллект) – самообучающаяся система с искусственным интеллектом основанная на TensorFlow, использование которого было утверждено поисковой системой Google (рисунок 1). Информационная система будет построена на сервис-ориентированной архитектуре. Сервис-ориентированная архитектура – это модульный подход к разработке программного обеспечения, основанный на

использовании распределенных, слабо связанных заменяемых компонентов, оснащенных стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам [1].

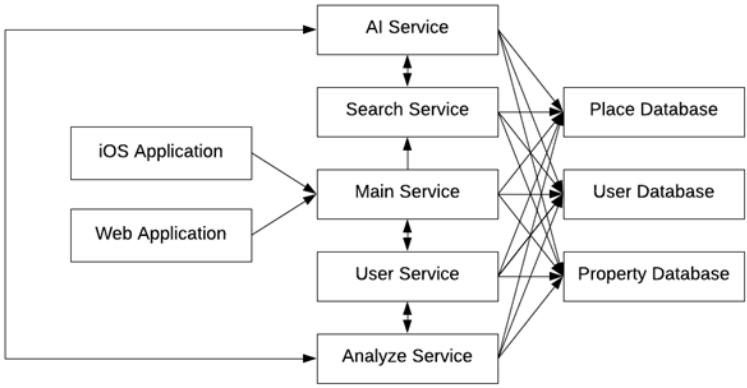

Рисунок 1 – Архитектура системы анализа и поиска объектов недвижимости, вакансий и сообществ

Наряду с возможностями анализа данных, актуальным также является и вопрос визуализации результата в доступном для пользователей виде. Так как интерфейсы компонентов в сервис-ориентированной архитектуре инкапсулируют детали реализации от остальных компонентов, таким образом обеспечи вая комбинирование и многократное использование компонентов для построения сложных распределенных программных комплексов, обеспечивая независимость от используемых платформ и инструментов разработки, способствую дальнейшему масштабированию. Это позволяет вести параллельную разработку предлагаемой информационной системы сокращая время разработки.

Визуализация релевантных рекомендаций результата и интерфейса клиентского приложения в процессе разработки использует наилучшие практики UI/UX дизайна, в том числе живые анимации.

UX-дизайн – это проектирование интерфейса на основе исследований пользовательского опыта и поведения. UX – это впечатления от работы с интерфейсом. Опыт пользователя зависит от различных компонентов: архитектуры сайта, графического

дизайна, понятного текста и отзывчивости интерфейса на конкретные действия пользователя.

UI-дизайн – процесс визуализации прототипа, который разработали на основании пользовательского опыта и исследования целевой аудитории. UI-дизайн включает в себя работу над графической частью интерфейса: анимацией, иллюстрациями, кнопками, меню, слайдерами, фотографиями и шрифтами. UI-дизайнер отвечает за то, как выглядит интерфейс продукта, и как пользователь взаимодействует с его элементами. Для этого необходимо грамотно организовать элементы интерфейса и выдержать единые стиль и логику их взаимодействия.

Исходя из высоких стандартов, заданных различными разработчиками программных продуктов, к клиентскому уровню информационной системы предъявляются определенные требования.

Требования к качественному UX/UI-дизайну:

Ясность. В интерфейсе не должно быть двусмысленности, а текст и структура направляют пользователя к цели.

Лаконичность. Интерфейс не должен быть перегружен подсказками, всплывающими окнами и анимацией. Это поможет сфокусировать внимание пользователя на конкретном элементе.

Узнаваемость. Элементы дизайна легко распознать, даже если пользователь видит приложение впервые. Интерфейс должен быть интуитивно понятным.

Отзывчивость. Хороший интерфейс реагирует на действия пользователя мгновенно. Он должен понимать, что происходит на экране прямо сейчас.

Постоянство. Элементы интерфейса – меню и слайдеры – должны вести себя одинаково на любой странице.

Эстетика. Интерфейс должен быть визуально привлекательным, чтобы пользователю было приятно работать, ничто его не раздражало и не отвлекало от решения задач.

Эффективность. Помимо внешней привлекательности хороший интерфейс экономит время пользователя и доставляет его в нужную точку с минимальными усилиями.

Снисходительность. Даже при самом продуманном интерфейсе ни один пользователь не застрахован от ошибки. Приложение должно обладать понятными обработчиками ошибок и оповещать пользователя сообщениями на случай, если что-то пошло не так. Это поможет сохранить деньги, время и лояльность клиентов в случае сбоя.

Предлагаемая информационная система будет иметь модульную клиент-серверную и сервис-ориентированную архитектуру, где каждый сервис выполняет определенную функцию и масштабируется без влияния на другие модули.

Предполагается, что целевой аудиторией могут быть, прежде всего, различные организации, застройщики и частные клиенты, заинтересованные в получении качественных клиентов и релевантных рекомендаций, что послужит сокращению цепочек для связи.

Модуль хранения данных представляется базой данных, в которую будут поступать объединенные данные из сервиса Main. В качестве хранения данных используется свободная реляционная база данных MySQL. MySQL – это одна из наиболее популярных и эффективных систем управления базами данных. Таблицы связаны определенными отношениями, делающими возможным объединять данные из нескольких таблиц в одном запросе для поиска информации. MySQL использует стандартизированный язык запросов SQL (Structured Query Language, язык структурированных запросов) для обращения к компьютерным базам данных. MySQL может в виде многопоточной библиотеки быть подключена к пользовательскому приложению, что даст в результате быстрый, легкий и компактный программный продукт [3].

Действия пользователей, в основном, будут связаны поиском и обменом сообщениями с другими пользователями. Данная информация и представляет наиболее ценный интерес. Можно получить следующие данные о записях пользователя: данные геолокации; поисковые запросы; друзья; интересы или сообщества, откуда пользователь взял информацию; тип информации. Все основные действия с данными происходят в слое анализа данных (рисунок 2).

Отметим также, что к основным модулям системы относят также и модуль предоставления результатов работы системы конечному пользователю, который реализуется в виде клиентского мобильного приложения на операционной системе iOS для работы с системой.

iOS – мобильная операционная система для смартфонов, электронных планшетов, носимых проигрывателей и некоторых других устройств, разрабатываемая и выпускаемая американской компанией Apple. App Store – магазин приложений, предлагает более 1,5 млн приложений для iPhone, число загрузок превысило 100 миллиардов, а пользовательская база составляет порядка 575 миллионов человек, что является отличным выбором при

необходимости дальнейшей монетизации информационного продукта (рисунок 3).

Рисунок 2 – Процесс анализа данных

При реализации интерфейса системы клиентского приложения будет использоваться декларативный подход SwiftUI. SwiftUI – это набор инструментов для создания пользовательского интерфейса, который позволяет декларативно разрабатывать приложения. SwiftUI позволяет говорить разработчику как должен выглядеть пользовательский интерфейс и как он должен работать, и система дальше организует все взаимодействие с пользователем.

Декларативный пользовательский интерфейс лучше всего понимается по сравнению с императивным, которым разработчики iOS занимались до iOS 13. В императивном пользовательском интерфейсе разработчики могли бы заставить функцию вызываться при нажатии кнопки, а внутри функции происходило бы считывание значения и изменение строки экране в зависимости от происходящего.

Императивный UI вызывает всевозможные проблемы, большинство из которых связано с состоянием (state), что является термином, означающим «значение, которое мы храним в нашем коде». Необходимо отслеживать, в каком состоянии находится приложение, и каждый раз убеждаться, что наш пользовательский интерфейс правильно отражает это состояние.

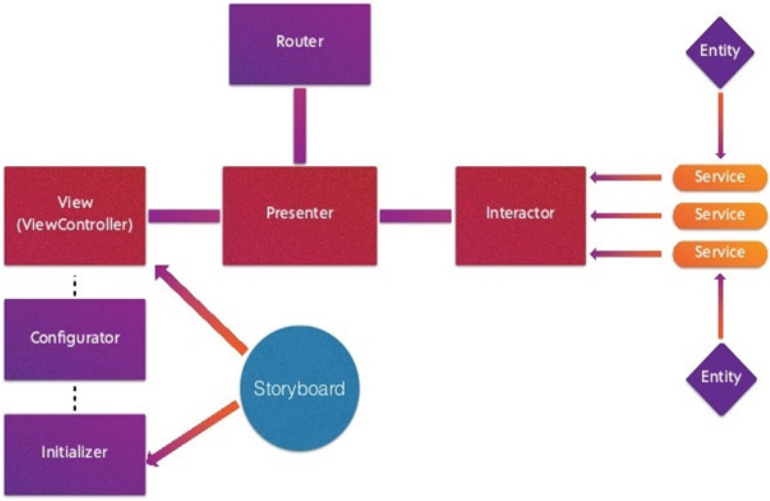

Рисунок 3 – Модель iOS приложения

Декларативный пользовательский интерфейс позволяет сообщить iOS обо всех возможных состояниях приложения сразу. Можем сказать, что если пользователь вошел в систему, то нужно показывать приветственное сообщение, но, если вышел из системы, показывать кнопку входа. Нет необходимости писать код для перемещения между этими двумя состояниями вручную.

Разработчик не указывает SwiftUI показывать или скрывать компоненты вручную, мы просто сообщаем ему все правила, которым мы хотим, чтобы он следовал, и оставляем его работать, чтобы он сам гарантировать соблюдение этих правил.

Так же SwiftUI позволяет быстро разрабатывать переиспользуемые элементы интерфейса, что ускоряет разработку и позволяет написать высоко переиспользуемый код. SwiftUI оснащен представлением интерфейса в реальном времени, то есть каждое изменения кода при создании дизайна элемента сразу же будет отображено для разработчика.

SwiftUI не останавливается на достигнутом: он также выступает в роли кроссплатформенного пользовательского интерфейса, который работает в iOS, macOS, tvOS и даже watchOS. Это означает, что при помощи SwiftUI можно разработать одно приложение, которое можно будет развернуть на разных платформах.

Архитектуры при разработке программного обеспечения используются, чтобы организовать код на разных уровнях, чтобы сделать его более чистым, легким для тестирования, обслуживания и написания меньшего количества шаблонного кода. VIPER – это шаблон «чистой» архитектуры, основанный на принципе единой ответственности. Которые делят наше приложение на модули в зависимости от вариантов использования. VIPER используется для разработки масштабируемых iOS приложений, поэтому это лучший выбор в этом отношении. Чистая архитектура делит логику приложения на отдельные уровни. Каждый уровень несет свою ответственность. VIPER сделает приложение модульным, где каждый модуль будет состоять из 5 уровней, определяющих архитектуру VIPER (View, Interactor, Presenter, Entity и Router). У каждого уровня есть своя роль или ответственность в модуле.

Уровень View. Уровень представления, который в основном представляет собой UIViewController и любой другой тип представления. Этот уровень содержит логику пользовательского интерфейса (отображение, обновление, анимацию) и отвечает за перехват действий пользователя и отправку их докладчику. Самое главное, что в нем нет бизнес-логики.

Уровень Interactor. Interactor «диспетчер сети»: он отвечает за получение данных из служб (сеть, база данных, датчики) по запросу докладчика. Interactor отвечает за управление данными из уровня модели.

Уровень Presenter. Presenter – это единственный уровень, который взаимодействует с представлением. По сути, это уровень, отвечающий за принятие решений на основе действий пользователя, отправленных View.

Уровень Entity. Entity – это просто модели, которые использует Interactor. Лучше всего разместить их вне модулей VIPER, поскольку эти объекты обычно используются разными модулями во всей системе.

Уровень Router. Этот уровень отвечает за обработку логики навигации для UIViewController.

Сервисы в VIPER это слои, которые отвечают за валидацию, обработку, и работу с данными. Уровень Interactor инкапсулирует все необходимые сервисы приложения, который в свою очередь использует сервисы для решения задач от пользователя.

Таким образом, VIPER помогает быть разработчикам более точными, что касается разделения проблем, разделяя большое

количество кода одного класса на несколько меньших классов. За счет поддержания единственной ответственности в каждом классе это упростит разработку классов, используя разработку через тестирование, позволяет быстро реагировать на изменяющиеся требования и создавать высокопроизводительное программное обеспечение.

Заключение. С учетом вышесказанного, несомненно, данная система является актуальной разработкой. Использование анализа и обработки данных в совокупности с популярной платформой для публикации приложений поможет создать полезную информационную систему эффективно решающую поставленные проблемы. Ожидается, что данная система будет востребованной широким кругом лиц, заинтересованных в изучении и анализе скрытых предпочтений пользователей, поиске вакансий или объектов недвижимости, а также людям, которые хотят найти себе единомышленников по разным вопросам.

СПИСОК ЛИТЕРАТУРЫ

1. Erl, T. Service-Oriented Architecture: Concepts, Technology, and Design / T. Erl. – Prentice Hall, 2016. – 143 p.

2. Рудикова, Л. В. Проектирование баз данных / Учебное пособие для студентов высш. учеб. заведений по специальностям «Программное обеспечение информационных технологий», «Экономическая кибернетика», «Прикладная математика (научно-педагогическая деятельность)», «Информационные системы и технологии (в экономике)» / Л. В. Рудикова. – Минск : ИВЦ Минфина, 2009. – 352 с.

3. Барсегян А. Методы и модели анализа данных: OLAP и DataMining / А. А. Баргесян, М. С. Куприянов, В. В. Степаненко, И. И. Холод – СПб: БХВ-Петербург, 2009. – 336 с.: ил.