УДК 004

ИНФОРМАЦИОННАЯ WEB-СИСТЕМА «ЗДРАВООХРАНЕНИЕ В Г. ГОМЕЛЕ»

ЦВЕТКОВ АЛЕКСАНДР АЛЕКСАНДРОВИЧ
студент
ГГТУ имени П.О. Сухого

Научный руководитель: Дорощенко Игорь Васильевич
старший преподаватель
ГГТУ имени П.О. Сухого

Аннотация: в работе представлена информационная *web*-система «Здравоохранение в г. Гомеле». Особое внимание уделяется архитектуре и методам разработки системы. Разработка прототипа позволила исследовать современные методы и средства разработки, а также способы проектирования информационных систем, что позволяет оптимизировать разработку и масштабируемость разрабатываемых решений.
Ключевые слова: информационная система, чистая архитектура, микросервисная архитектура, кэширование, здравоохранение.

INFORMATION WEB-SYSTEM «HEALTHCARE IN THE CITY OF GOMEL»

Tsvetkov Alexandr Alexandrovich

Scientific adviser: Doroschenko Igor Vasil'evich

Abstract: the paper presents the web-based information system «Healthcare in the city of Gomel». Special attention is paid to the architecture and methods of system development. The development of the prototype made it possible to explore modern development methods and tools, as well as ways to design information systems, which makes it possible to optimize the development and scalability of the solutions being developed.
Key words: information system, clean architecture, micro-service architecture, caching, healthcare.

Внедрение современных оптимизированных средств и методов проектирования и разработки информационных web-систем позволяет улучшить масштабируемость, производительность системы, а также ускорить процесс разработки за счёт внедрения специализированных гибких методологий, таких как CI/CD и др. В данной работе представлен опыт внедрения архитектурных решений и методов разработки на примере информационной web-системы «Здравоохранение в г. Гомеле», а также продемонстрирован результат принятых решений на примере масштабируемости и производительности системы.

Перед разработкой информационной системы необходимо спроектировать её архитектуру. Для этого необходимо определить функциональные требования системы и составить диаграмму вариантов использования. Это позволит определить роли пользователей и их возможности. В разрабатываемом решении выделены следующие роли пользователей: пациент, администратор, редактор, врач, регистратор, администратор учреждения. Пациент имеет возможность просматривать информацию, назначения (выписываются врачами) и осуществлять запись на приём (расписание и записи контролируются регистраторами). Администратор управляет пользователями и медучреждениями, зарегистрированны-

ми в системе. Редактор имеет доступ к редактированию новостного контента. Администратор учреждения управляет сотрудниками и информацией учреждения. Исходя из вышеперечисленных функциональных требований можно принять архитектурное решение для разделения пользовательского интерфейса на 3 приложения: публичное, для сотрудников учреждений и панель администратора. Такое разделение необходимо как для производительности системы, так и для безопасности. Система будет более производительна, поскольку для определённых групп пользователей не будет загружаться недоступная им информация. При составлении диаграммы вариантов использования необходимо выделять такие модули и выносить их в отдельные приложения. Кроме того, приложения, такие как панель администратора, можно размещать на локальных доменах, что снизит риск несанкционированного доступа.

При проектировании серверной части необходимо сразу определить: будет ли использоваться микросервисная архитектура. В настоящее время деление на микросервисы стало популярным решением, поскольку при использовании микросервисов можно вносить изменения в отдельный микросервис и развертывать его независимо от остальной системы [1, с. 29]. Однако это не всегда необходимо. Внедрение микросервисной архитектуры делает информационные системы низкой или средней степени сложности более трудными в разработке. Это связано с тем, что деление на микросервисы требует принятия сложных решений от архитектора, а также знаний оркестрации и грамотного осуществления передачи сообщений между микросервисами при разработке. Нередко микросервисы приходится разворачивать на отдельных машинах, что влечёт дополнительные денежные затраты на закупку и эксплуатацию оборудования. Исходя из вышеперечисленного, микросервисная архитектура является не универсальным решением, а крайней мерой для высоконагруженных информационных систем, над которыми работают сотни разработчиков или коллаборации больших команд разработки. В разработанном прототипе было принято решение использовать монолитную архитектуру, поскольку это не повлияло на работоспособность системы, однако ускорило разработку и упростило масштабирование.

Независимо от выбора микросервисной или монолитной архитектуры необходимо определить структуру монолита или каждого микросервиса. Обычно используется MVC или многослойная архитектура с инверсией зависимостей (чистая архитектура). В разработанном прототипе использовалась чистая архитектура, поскольку для масштабируемости системы необходимо разделять серверную и клиентскую часть, чего не позволяет делать MVC. В MVC часть бизнес-логики помещается в клиентскую часть, что не позволяет разделить обязанности разработчиков и повышает сложность масштабирования. Идея программного обеспечения состоит в том, чтобы дать простую возможность изменять поведение компьютеров. Для достижения этой цели программное обеспечение должно быть легко масштабируемым [2, с. 36]. В рамках чистой архитектуры можно добавлять или удалять слои по мере необходимости. В разработанном модуле была спроектирована структура, изображённая на рисунке 1.

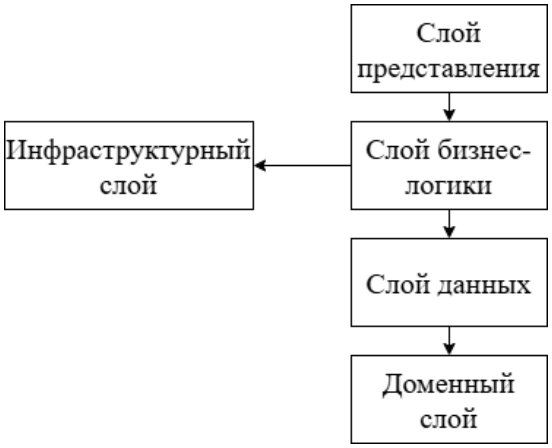

Рис. 1. Структура сервера на основе чистой архитектуры

Слой представления отвечает за интерфейс серверного приложения (API). Поскольку в разработанной системе используется база данных, был внедрён слой данных для настройки конфигурации и

выполнения CRUD-операций. Доменный слой представляет набор доменных моделей и абстрактных данных приложения. Поскольку в системе используются сторонние ресурсы, для работы с ними был внедрён инфраструктурный слой.

Среди сторонних сервисов можно выделить систему кэширования. В разработанной информационной системе для кэширования использовалась не оперативная память напрямую, а более усовершенствованный подход – через нереляционную СУБД, работающую в оперативной памяти, Redis. Данный подход позволяет, не теряя в производительности, осуществлять распределённое кэширование, если такое требуется, а также сохранять данные при перезапуске системы. Система кэширования значительно снижает нагрузку на базу данных. Таким образом, можно избежать высокой нагрузки на сервер, не внедряя сложные решения, как микросервисная архитектура. Таким образом, была получена работоспособная и производительная архитектура информационной системы (рис. 2).

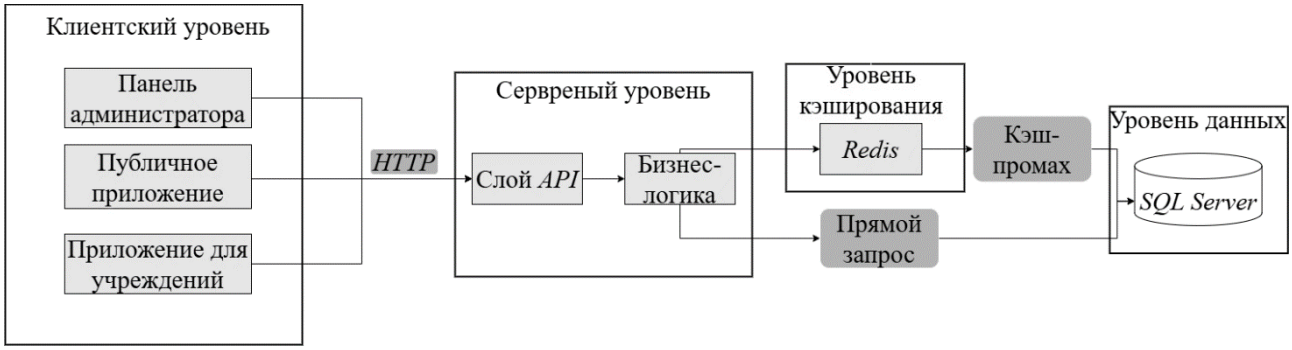

Рис. 2. Архитектура разработанной информационной системы

В качестве реляционной СУБД можно использовать любое удобное решение. В данном случае использовался SQL Server, поскольку использовалась технология ASP.NET Core, с которой используемая СУБД легко интегрируется.

При помощи данных подходов удалось создать продукт, нацеленный на цифровизацию городской системы здравоохранения, что позволило упростить доступ к актуальной информации по этой теме и медицинским услугам для граждан города. Дальнейшие исследования методов разработки информационных систем помогут создавать более сложные решения.

Список источников

1. Ньюмен С. Создание микросервисов. – СПб.: Питер, 2016. – 304 с.
2. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2021. – 352 с.