Обоснование среды

Язык программирования:

Я выбрал язык программирования JavaScript. Выбор пал именно на него, так как уже имеется разработка программ и различных информационных систем с использованием данного языка, он является очень понятным и не нагруженным.

Главным плюсом выбора я считаю простую возможность связи серверной части и пользовательского интерфейса из-за возможности писать эти части ИС в одной среде. Таким образом сокращается трудозатраты на проектирования структуры всей информационной системы целом, ускоряется время разработки и простая обработка ошибок.

Сравнение с конкурентами:

Главным конкурентом при таком подходе является C#, у меня имеется опыт разработки на данном языке программирования. Данный язык программирования является хорошим выбором для разработки на платформе Windows. Я разрабатываю для этой платформы в основном, но также будет поддержка любой другой платформы, таких как Linux, Android и IOS. Это Основной недостаток данного выбора системы, но если его не учитывать, то C# + WPF является очень хорошим выбором для простой ИС без подключения к централизированному серверу. Такие требования в поддержке других платформ объясняется активным внедрения отечественных операционных операционных систем таких как Astra Linux.

При выборе языка я также рассматривал Java + фреймворки Java. Но после рассмотрения плюсов и минусов данного выбора пришел к выводу, что такой подход является слишком затратным и избыточным. Так как часто проектирование информационных систем происходит на крупных фреймворках таких как Spring и похожие. Для моей задачи потенциал данных фреймворков слишком большой и бесполезен

По итогу я выбрал JavaScript, потому что был опыт проектирования, и он прост в реализации.

База данных:

В качестве системы управления базами данных был выбран SQLite. Данный выбор обусловлен спецификой проекта — это локальная, однопользовательская информационная система, не требующая удаленного доступа или высоких нагрузок. SQLite идеально соответствует этим требованиям, так как представляет собой встраиваемую БД, не требующую отдельного серверного процесса. Данные хранятся в единственном файле на диске, что делает развертывание и распространение системы тривиальным: достаточно скопировать исполняемый файл и файл базы данных. Это полностью соответствует философии Tauri, направленной на создание легковесных приложений.

Сравнение с конкурентами:

PostgreSQL: Эти классические реляционные СУБД являются мощными и функциональными инструментами. Их основной недостаток в контексте данной задачи — избыточность и сложность развертывания. Они требуют установки и настройки отдельного серверного процесса (даже в локальном режиме), что усложняет архитектуру приложения и процесс его установки на машину конечного пользователя. Для моей системы, где не требуется сетевое взаимодействие, параллельная работа нескольких пользователей с сложными транзакциями, их использование было бы неоправданным усложнением.

NoSQL (например, MongoDB): хотя данные успеваемости не являются строго реляционными, реляционная модель отлично подходит для структурированного хранения информации о студентах, группах, дисциплинах и оценках. NoSQL-подход мог бы привести к дублированию данных и менее строгой схеме, что увеличило бы риски нарушения целостности данных. Вариант с хранением в JSON-файлах рассматривался как более простой, но был отвергнут из-за низкой эффективности при частых операциях поиска, обновления и агрегации данных, а также отсутствия встроенных механизмов гарантированной целостности (ACID), которые предоставляет SQLite.

Фреймворк для разработки интерфейса и back-end:

Для реализации пользовательского интерфейса планируется использование библиотеки React, а серверная логика (API для работы с БД) будет написана на Node.js (с использованием фреймворка Express.js или его аналогов). Такой выбор обеспечивает полную кодовую базу на JavaScript, что значительно ускоряет разработку. React, благодаря своей компонентной модели, позволяет создавать переиспользуемые, хорошо инкапсулированные блоки интерфейса (например, компоненты таблицы, модального окна для добавления оценки, выпадающего списка групп), что критически важно для поддержания кода и его масштабирования в будущем. Tauri же выступает в роли моста, который обеспечивает связь между этим веб-приложением и нативной частью (работа с файловой системой, окнами, и конечно, с SQLite через Rust-крэйты).

Сравнение с конкурентами:

PyQt (Python): Данные фреймворки позволяют создавать классические десктопные приложения с нативными виджетами операционной системы. Их главный недостаток — устаревший (в случае Tkinter) или сложный в освоении (PyQt) подход к созданию UI, а также низкая скорость итераций при разработке сложных интерфейсов. Современные пользователи ожидают от приложений reactivity и богатого визуального опыта, который на порядок проще и быстрее реализовать на React с использованием CSS.

WPF (C#): Эти технологии являются прямыми аналогами предлагаемого стека (JS + Tauri) в своих экосистемах. Они мощные и позволяют создавать сложные десктопные приложения. Однако, как и в случае с языком, их кроссплатформенность (особенно JavaFX) часто сопряжена с нюансами и не всегда обеспечивает идеально идентичный опыт на разных ОС. Кроме того, разработка современного UI на них требует больше усилий и менее гибка, чем верстка на HTML/CSS. Мой выбор на основе веб-технологий предоставляет доступ к огромной экосистеме библиотек UI (например, MUI, Chakra UI) и позволяет при необходимости легко портировать интерфейс в веб, если требования изменятся.

Java Spring Boot: Данный фреймворк является индустриальным стандартом для создания сложных, высоконагруженных enterprise-приложений. Он предоставляет полный спектр инструментов (Spring MVC, Spring Data, Spring Security) для построения мощного back-end. Однако именно эта мощь и является его главным недостатком в контексте данной задачи. Использование Spring Boot привело бы к значительной избыточности: приложению не требуются встроенный Tomcat/Jetty, сложные механизмы безопасности, микросервисная архитектура или ORM для работы с распределенной БД. Это существенно увеличило бы время разработки, сложность сборки и итоговый размер дистрибутива, что совершенно неоправданно для локального однопользовательского приложения. Кроме того, необходимость компиляции в bytecode и запуска в JVM усложняет процесс распространения и запуска для конечного пользователя по сравнению с единым исполняемым файлом Tauri.

Среда разработки:

В качестве основной среды разработки (IDE) будет использоваться Visual Studio Code. Это редактор с беспрецедентной поддержкой JavaScript/TypeScript, React и Node.js благодаря обширной экосистеме расширений (например, ES7+ React/Redux/React-Native snippets, SQLite). Он обладает мощным встроенным отладчиком, интегрированным терминалом и инструментами для работы с Git, что делает его идеальным выбором для полного цикла разработки в выбранном стеке.

Сравнение с конкурентами:

JetBrains WebStorm— это мощная IDE, заточенная специально под веб-разработку. Ее главный недостаток — платная лицензия для коммерческого использования, в то время как VS Code полностью бесплатен и обладает сравнимой, а зачастую и превосходящей функциональностью за счет сообщества.

Vim — это высокопроизводительный текстовый редактор, обладающий культовым статусом среди разработчиков благодаря своей скорости, режимному редактированию и полному управлению с клавиатуры. Однако его ключевой недостаток для данного проекта — это чрезвычайно высокая кривая обучения и необходимость длительной и кропотливой настройки (через плагины и конфигурационные файлы) для достижения уровня функциональности, сопоставимого с VS Code «из коробки». Интеграция с современными JavaScript-инструментами (дебаггером, линтерами, инструментами для React) в Vim требует глубоких знаний и времени на настройку, что неприемлемо для учебного проекта с ограниченными сроками, где фокус должен оставаться на логике приложения, а не на конфигурировании среды разработки.