Федеральное государственное образовательное бюджетное учреждение

высшего образования

**«Финансовый университет при Правительстве**

**Российской Федерации»**

**(Финансовый университет)**

Колледж информатики и программирования

(наименование структурного подразделения)

Дипломный проект

Тема «Разработка приложения для мониторинга состояния людей с гипогликемией, гипергликемией»

(наименование)

Студент Шипилов Иван Андреевич

(фамилия, имя, отчество полностью)

Учебная группа 4ИСИП-020

Специальность 09.02.07 Информационные системы и программирование

(код и наименование специальности)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Руководитель  дипломного проекта |  |  |  | С. М. Володин |
|  |  | (подпись) |  | (инициалы, фамилия) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Консультант  дипломного проекта |  |  |  |  |
| (при наличии) |  | (подпись) |  | (инициалы, фамилия) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Председатель предметной  (цикловой) комиссии |  |  |  | Т. В. Соловьева |
|  |  | (подпись) |  | (инициалы, фамилия) |

Москва – 2024 г.

**Содержание**

[Введение 3](#_Toc169015187)

[Глава 1. Теоретическая часть 5](#_Toc169015188)

[1.1 Описание предметной области. 5](#_Toc169015189)

[1.2. Сравнение программ аналогов 5](#_Toc169015192)

[1.3. Постановка задачи 7](#_Toc169015194)

[1.4. Выбор программных и инструментальных средств реализации поставленной задачи 9](#_Toc169015195)

[Глава 2. Практическая часть 21](#_Toc169015196)

[2.1. Анализ требований и определение спецификаций ПО 21](#_Toc169015197)

[2.2. Проектирование программного обеспечения 23](#_Toc169015198)

[2.3. Разработка программного обеспечения 24](#_Toc169015199)

[2.4. Отладка и тестирование программы 40](#_Toc169015200)

[Заключение 42](#_Toc169015201)

[Список использованных источников и интернет ресурсов 43](#_Toc169015202)

# **Введение**

Сахарный диабет — это математика. Ты ни на секунду не можешь отключить мозг, если хочешь добиться хорошей компенсации и минимизировать риски развития осложнений. Нужно считать абсолютно все без перерывов на обед и выходные.

Многие ругают наш век, но в нем есть множество положительных моментов для людей с сахарным диабетом. Например, в наши дни существует масса мобильных приложений, которые значительно упрощают жизнь людей с сахарным диабетом 1 и 2 типов.

Сахарный диабет относится к числу наиболее распространённых хронических заболеваний, о чем могут свидетельствовать данные Российского статистического ежегодника за 2014 год – 3,43 млн. человек, страдающих СД. В общей структуре СД 90-95% составляет СД 2 типа, при этом фактическая распространённость превышает в 2-3 раза зарегистрированную. Именно данный тип диабета ассоциирован с «эпидемией диабета». Тенденции роста заболеваемости СД обуславливают ежегодное увеличение численности диспансерной группы у эндокринологов. Это приводит к высокой нагрузке врача на приеме, не позволяющей уделить достаточного внимания элементам обучения пациентов правилам самоконтроля заболевания.

Актуальность работы заключается в том, что данный сервис возник относительно недавно, но уже заработал огромную аудиторию. Приложения для контроля сахарного диабета активно развиваются. Создание приложений для диабета имеет большую актуальность по нескольким причинам:

* Удобство и доступность. Приложения позволяют легко и удобно отслеживать уровень глюкозы в крови, питание, физическую активность и другие показатели, что помогает контролировать заболевание.
* Персонализация и индивидуализация. Приложения могут быть настроены под конкретные потребности и особенности каждого пациента, что делает управление диабетом более эффективным.
* Мониторинг и анализ данных. Приложения собирают данные о пациенте, анализируют их и предоставляют информацию в удобном для восприятия виде, что помогает выявлять тенденции и оптимизировать лечение.
* Взаимодействие с врачом. Приложения могут упростить обмен информацией между пациентом и врачом, что помогает врачу более точно настраивать лечение в соответствии с индивидуальными потребностями пациента.

Таким образом, создание приложений для диабета является актуальным и важным направлением в сфере здравоохранения, которое помогает пациентам лучше контролировать своё заболевание и улучшить качество жизни

Целью проекта является разработка приложения для мониторинга состояния людей с гипогликемией, гипергликемией.

Практическая значимость работы заключается в том, что будет создано приложение для мониторинга состояния людей с гипогликемией, гипергликемией, которое сможет отвечать на вопросы пользователя, а так же помогать в контроле сахарного диабета.

# **Глава 1. Теоретическая часть**

## **1.1 Описание предметной области.**

## Сахарный диабет — группа эндокринных заболеваний, связанных с нарушением усвоения глюкозы вследствие абсолютной или относительной (нарушение взаимодействия с клетками-мишенями) недостаточности гормона инсулина. В результате развивается гипергликемия — стойкое увеличение содержания глюкозы в крови. Заболевание характеризуется хроническим течением, а также нарушением всех видов обмена веществ: углеводного, жирового, белкового, минерального и водно-солевого. Кроме человека данному заболеванию подвержены также некоторые животные, например кошки и собаки.

## Существует два типа диабета: инсулинозависимый и инсулиннезависимый. Сахарный диабет первого типа — это более тяжёлая форма гипергликемии, при которой по какой-то причине разрушаются клетки поджелудочной железы, вырабатывающие инсулин. Часть клеток продолжает работать, но количество гормона настолько мало, что не справляется с переработкой сахара даже при строгой без углеводной диеты. Сахарный диабет второго типа развивается у людей старше 40 лет как следствие нарушения углеводного обмена. При нём клетки поджелудочной железы не атрофируются, они вырабатывают инсулин в нужном количестве, но мышцы, печень и другие инсулинозависимые ткани теряют чувствительность к этому веществу. Глюкоза остаётся в крови и моче. Постепенно инсулина вырабатывается всё меньше, пока болезнь не перейдёт в первый тип.

## **1.2. Сравнение программ аналогов**

## Существует много приложений для ведения дневников для диабетиков и врачей, они бывают встроенными в глюкометры, компьютеры, смартфоны. В данных приложениях можно выстраивать графики глюкозы, следить за приёмом лекарств, пищи, а также считать кол-во углеводов для коррекции доз тех самых лекарств.

## 

1. Diabetes:M

Одно из наиболее функциональных приложений, позволяющих держать диабет под контролем. С его помощью можно отслеживать уровень инсулина, делая записи о приёме пищи и фиксируя данные с глюкометров и инсулиновых помп. На основе вашего дневника Diabetes:M формирует подробные отчёты, графики и статистику, которыми можно поделиться с лечащим врачом. В платной версии приложения также есть калькулятор расчёта значения короткого и пролонгированного болюсного инсулина.

Достоинства:

* Настройка собственного профиля;
* Тонкая настройка собственного профиля;
* Большой функционал приложения;
* Присутствуют графики и диаграммы;
* Возможность использования приложения без регистрации;

Недостатки:

* Неудобный интерфейс;

2. DiaMeter

Не менее удобное приложение для ведения записей об уровне сахара, количестве съеденных вами хлебных единиц, инъекциях короткого и продлённого инсулина, а также о самочувствии в целом. Есть наглядная статистика и возможность синхронизации всех данных в облаке. Дополнительно DiaMeter предлагает ряд интерактивных статей, в которых можно найти немало полезной информации о правильном питании при диабете, физических нагрузках, влиянии алкоголя и другом.

Достоинства:

* Регистрация своего профиля;
* Удобный интерфейс;
* Присутствуют графики и диаграммы;
* Синхронизация в облаке;

Недостатки:

* Невозможность использования приложения без регистрации;
* Реклама;

3. Диабет

Это приложение также позволяет вести учёт уровня глюкозы в крови и фиксировать употребление углеводосодержащих продуктов. Для каждого приёма пищи необходимо указывать не только что вы ели, но и сколько, чтобы расчёт дозы инсулина был максимально точным. Данные из приложения можно выгрузить в формате PDF или XLS (Excel), чтобы их удобнее было предоставлять лечащему врачу в печатном или электронном варианте. У iOS-версии предусмотрена интеграция с сервисом «Здоровье» от Apple.

Достоинства:

* Наличие каталога продуктов;
* Настройка напоминания об измерении через два часа;
* Простой интерфейс;

Недостатки:

* Отсутствие графиков;

Все приложения для контроля имеют свои преимущества и недостатки и каждое из них в равной степени может оказать положительное влияние на жизнь пользователя.

## **1.3. Постановка задачи**

Объектом исследования является разработка приложения для мониторинга состояния людей с гипогликемией, гипергликемией. Предметом исследования являются инструменты и технологии разработки приложения для мониторинга. Цель данной работы заключается в реализации программы мониторинга, которая будет выводить необходимую информацию пользователю.

Задачи, которые необходимо решить для достижения заданной цели, заключаются в следующем:

* выполнить пред проектное исследование предметной области на основе интернет-источников;
* выполнить анализ требований и разработать необходимые спецификации;
* спроектировать программное обеспечение в виде построения структурной и функциональной схем программы;
* разработать графический дизайн программы;
* реализовать функционал программы;
* провести тестирование и отладку программы;
* создать инсталлятор программы.

Для разработки данной дипломной работы были использованы следующие инструментальные средства:

* интегрированная среда разработки Visual Studio 2019;
* объектно-ориентированный язык программирования Python 3.11;
* система управления реляционными базами данных (РСУБД) SQLite;
* язык программирования структурированных запросов SQL;
* сервис для построения диаграмм Draw.io;
* текстовый процессор Microsoft Word.
* Инструмент для создания презентации- PowerPoint.

Функции программного продукта.

Приложение для мониторинга должно:

* Выводить необходимую информацию пользователю;
* Выводить графики глюкозы;

Ход работы приложения.

Пользователь вводит данные глюкозы или данные о еде. Приложение получает информацию, считает и выводит пользователю данные, график глюкозы.

Эксплуатационные требования для разрабатываемого приложения:

* операционная система Кроссплатформенность;
* процессор мощностью 2,6 ГГц;
* объем оперативной памяти не менее 4 Гб;
* 300 МБ свободной памяти.

## **1.4. Выбор программных и инструментальных средств реализации поставленной задачи**

Сколько существуют компьютеры, столько же и существуют средства разработки программного обеспечения. С течением времени они совершенствуются, постоянно прогрессируют и с каждым разом становятся все удобнее в использовании и проще в изучении.

Для начала подвергнем анализу наиболее популярные языки программирования: Java, C#, C++ и Visual Basic [Data Structures…, 1999].

Анализ языка программирования

Таблица 1. Парадигмы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможность |  |  | Язык |  |
| Java | C# | C++ | Visual  Basic |
| Императивная | + | + | + | + |
| Объектно-ориентированная | + | + | + | + |
| Рефлексивная | -/+ | +/- | +/- | +/- |
| Обобщенное программирование | + | + | + | + |
| Логическая | - | - | - | - |
| Распределительная | + | -/+ | +/- | - |

* Императивная. Императивный язык описывает не задачу, а способ ее решения.
* Объектно-ориентированная. Она основана на представлении всего в языке в виде объектов. Объекты, в свою очередь, могут содержать в себе как переменные, так и методы для работы с ними. Выполняется поддержка основополагающих принципов ООП: полиморфизм, наследование и инкапсуляция.
* Рефлексивная. Это возможность, предполагающая, что язык программирования может оперировать собственным кодов как данными.
* Обобщенное программирование. При написании алгоритмов есть возможность передавать им данные любых типов.
* Логическая. Программа является набором некоторых правил и фактов вывода в определенном логическом исчислении.
* Распределительная. Поддержка языком программирования возможности распараллеливания вычислений.

Таблица 2. Типизация

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Возможность |  |  |  | Язык |  |  |
| Java | C# |  | C++ |  | Visual  Basic |
| Статическая типизация | + |  | + | + |  | + |
| Явная типизация | + |  | + | + |  | + |
| Неявное приведение типов без потери данных | - |  | + | + |  | + |
| Неявное приведение типов с потерей данных | - |  | - | + |  | + |
| Неявное приведение типов в неоднозначных ситуациях | - |  | + | + |  | + |
| Алиасы типов | - |  | + | + |  | - |
| Информация о типах в runtime | + |  | + | -/+ |  | + |
| Информация о типах-параметрах в runtime | - |  | + | -/+ |  | + |

* Статическая типизация. Невозможность изменения типов в процессе выполнения после их объявления.
* Явная типизация. Параметры и типы переменных указываются явно.
* Неявное приведение типов без потери данных. Например, приведение числа с плавающей точкой к целому числу.
* Неявное приведение типов с потерей данных. Например, приведение целого числа к числу с плавающей точкой.
* Неявное приведение типов в неоднозначных ситуациях. Допустим, при попытке сложения числа 7 и строки «3» мы можем получить, как число 10, так и строку «73».
* Алиасы типов. Замена одного алиаса типа на другой, полностью эквивалентный ему, например, #typedef TYPE1 TYPE2.
* Информация о типах в runtime. Наличие возможности выяснить точный тип объекта в runtime.

Таблица 3. Компилятор-интерпретатор

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможность |  | Язык |  |  |
| Java | C# | C++ | Visual  Basic |
| Open-source | + | + | + | + |
| Возможность компиляции | + | + | + | + |
| Bootstrapping | + | + | + | ? |
| Многопоточная компиляция | + | + | + | + |
| Интерпретатор командной строки | - | - | +/- | + |
| Условная компиляция | -/+ | + | + | + |

* Open-source компилятор или интерпретатор. Эта возможность говорит о наличии полноценного open-source компилятора или интерпретатора.
* Возможность компиляции. Способность компилировать код в нативный или в byte-cod с возможностью JIT компиляции.
* Bootstrapping. Это значит, что компилятор написан на том же языке, что и компилируется им.
* Многопоточная компиляция. Способность компилятора распараллеливать процесс сборки на несколько потоков при условии наличии таковых.
* Интерпретатор командной строки. Возможность вводить инструкции языка в командную строку с последующим незамедлительным их выполнением.
* Условная компиляция. Наличие возможности включения или отключения компиляции участка программного кода в зависимости от выполнения условий, например, в C++ это делается с помощью #if …

#endif.

Таблица 4. Управление памятью

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможность |  | Язык |  |  |
| Java | C# | C++ | Visual  Basic |
| Создание объектов на стеке | - | + | + | - |
| Неуправляемые указатели | - | + | + | - |
| Ручное управление памятью | - | + | + | - |
| Сборка мусора | + | + | -/+ | + |

* Объекты на стеке. Способность создавать экземпляры любого типа данных не «в куче», а на стеке.
* Неуправляемые указатели. Характерен прямой доступ к памяти и наличие адресной арифметики.
* Ручное управление памятью. Возможность оперировать с данными «в куче» посредством, например, операторов new и delete в языке C++.
* Сборка мусора. Компилятор способен сам отслеживать неиспользуемые участи памяти в куче и освобождать их.

Таблица 5. Управление потоками вычислений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможность |  |  | Язык |  |
| Java | C# | C++ | Visual  Basic |
| Конструкция goto | - | + | + | + |
| Инструкция break без метки | + | + | + | + |
| Инструкция break с меткой | + | - | - | + |
| Поддержка try/catch | + | + | + | + |
| Ленивые вычисления | - | -/+ | + | - |

* Конструкция goto. Подразумевает возможность безусловного перехода к метке.
* Инструкция break без метки. Безусловный выход из ближайшего цикла.
* Инструкция break с меткой. Поддержка безусловного выхода из цикла, помеченного меткой.
* Поддержка try/catch. Наличие возможности обрабатывать исключения с помощью конструкции try/catch.
* Ленивые вычисления. Предполагает экономию времени на проведении вычислений, результаты которых в дальнейшем не понадобятся.

Таблица 6. Типы и структуры данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможность |  |  | Язык | |
| Java | C# | C++ | Visual  Basic |
| Кортежи | - | +/- | +/- | +/- |
| Многомерные массивы | +/- | + | + | + |
| Динамические массивы | +/- | +/- | + | + |
| Ассоциативные массивы | +/- | + | + | + |
| Контроль границ массивов | + | + | +/- | + |
| Цикл foreach | + | + | + | + |
| Целые числа произвольной длины | + | + | - | + |
| Целые числа с контролем границ | - | - | - | - |

* Многомерные массивы. Наличие в языке возможности создавать многомерные массивы, например, array[N][M].
* Динамические массивы. Наличие в языке возможности создавать массивы, которые способны изменять свой размер в процессе выполнения программного кода.
* Ассоциативные массивы. Это так называемые hash-таблицы.
* Цикл foreach. Наличие в языке конструкции, благодаря которой существует возможность перебора всех элементов коллекции.
* Наличие поддержки длинной целочисленной арифметики.
* Целые числа с контролем границ. Определение типа чисел с заданным диапазоном, например, int range [-77, 69], и при попытке присвоения переменной range значения, выходящего за границы, происходила бы ошибка.
* Кортежи. Способность компилятора вернуть из метода кортеж. Кортеж – это неименованный тип данных, содержащий безымянные поля произвольного типа.

Таблица 7. Объектно-ориентированные возможности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможность |  |  | Язык |  |
| Java | C# | C++ | Visual  Basic |
| Интерфейсы | + | + | + | + |
| Мультиметоды | - | -/+ | -/+ | - |
| Переименование членов при наследовании | - | - | -/+ | - |
| Множественное наследование | - | - | + | - |

* Интерфейсы. Наличие синтаксической и семантической конструкций в программном коде, обеспечивающая специфицирование услуг, предоставляемых классом.
* Множественное наследование. Способность при создании класса наследоваться от нескольких, что позволяет производному классу содержать в себе функционал базовых классов.

Таблица 8. Функциональные возможности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможность |  |  | Язык |  |
| Java | C# | C++ | Visual  Basic |
| First class function | - | + | + | ? |
| Анонимные функции | - | + | + | + |
| Лексические замыкания | + | + | + | + |
| Частичное применение | - | ? | +/- | ? |
| Каррирование | - | + | +/- | - |

* First class function или объекты первого класса. В контексте определенного языка это названия сущностей, способных передаваться в качестве параметра, быть полученными при выполнении функции или быть присвоенными переменной.
* Анонимные функции. Наличие особого вида функций, объявляемых в месте использования и не получающих уникального идентификатора для их вызова.
* Лексическое замыкание. Наличие в функции ссылки на переменную, объявленную вне тела функции и не переданную в нее в качестве параметра.

Таблица 9. Прочие свойства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Возможность |  | Я | зык |  |
| Java | C# | C++ | Visual  Basic |
| Макросы | - | - | + | + |
| Шаблоны | + | + | + | + |
| Поддержка Unicode в идентификаторах | + | + | + | + |
| Перегрузка функций | + | + | + | + |
| Динамические переменные | ? | ? | + | + |
| Значение параметров по умолчанию | - | + | + | + |
| Локальные функции | +/- | +/- | + | +/- |
| Наличие библиотек для работы с графикой | + | + | + | + |

* Макросы. Наличие макросистемы, обрабатывающей код программы до ее компиляции или выполнения.
* Шаблоны. Наличие возможности создания обобщенных классов для расширения функционала. Например, в C++ это template классы.
* Перегрузка функций. Возможность создания перегруженных функций, позволяющих принимать на вход различные наборы параметров.
* Динамические переменные. Способность языка создавать в нем переменные «в куче».
* Значение параметров по умолчанию. Способность вызывать конструкторы или функции без явной подстановки значений входных параметров, которые проинициализированы в описании функции.
* Наличие библиотек для работы с графикой. Возможность использовать функционал таких библиотек как: OpenGL, WebGL, OpenML, DirectX.

Исходя из приведенных сравнительных характеристик можно сказать, что по своей универсальности при написании программного обеспечения языку C++ нету равных. Считается, что при реализации программного продукта нужно отталкиваться от поставленной цели, и для этого в наибольшей степени подходят два языка: C++ и Java. Они оба обладают возможностью распараллеливания вычислений и созданием хорошего графического интерфейса, хотя в Java для этого требуется значительно больше оперативной памяти, что склоняет выбор не в его пользу. Также, в отличии от языка C++, Java не поддерживает прямой доступ к памяти компьютера и ленивые вычисления, что сильно увеличило бы объем кода. Помимо прочего C++ обладает более хорошей типизацией данных.

Язык программирования С++ задумывался как язык, который будет:

* лучше языка С
* поддерживать абстракцию данных;
* поддерживать объектно-ориентированное программирования

С++ - язык общего назначения. За исключением второстепенных деталей он содержит язык С как подмножество. Язык С расширяется введением более гибких и эффективных средств, предназначенных для построения новых типов. Программист структурирует свою задачу, определив новые типы, которые точно соответствуют понятиям предметной области задачи. Такой метод построения программы обычно называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определенных пользователем. С такими объектами можно работать надежно и просто даже в тех случаях, когда их тип нельзя установить на стадии трансляции. Программирование с использованием таких объектов обычно называют объектно-ориентированным. Если этот метод применяется правильно, то программы становятся короче и понятнее, а сопровождение их упрощается.

Ключевым понятием С++ является класс. Класс - это определяемый пользователем тип. Классы обеспечивают целостносностью данных, их инициализацию, неявное преобразование пользовательских типов, динамическое задание типов, контролируемое пользователем управление памятью и средства для перегрузки операций. В языке С++ концепции контроля типов и модульного построения программ реализованы более полно, чем в С. Кроме того, С++ содержит усовершенствования, прямо с классами не связанные: символические константы, функции-подстановки, стандартные значения параметров функций, перегрузка имен функций, операции управления свободной памятью и ссылочный тип. В С++ сохранены все возможности С эффективной работы с основными объектами, отражающими аппаратную "реальность" (разряды, байты, слова, адреса и т.д.). Это позволяет достаточно эффективно реализовывать пользовательские типы.

Объектно-ориентированное программирование — это метод программирования, способ написания "хороших" программ для множества задач. Если этот термин имеет какой-то смысл, то он должен подразумевать: такой язык программирования, который предоставляет хорошие возможности для объектно-ориентированного стиля программирования.

Язык С++ проектировался для поддержки абстракции данных и объектно-ориентированного программирования в добавление к традиционному стилю С., впрочем, это не значит, что язык требует какого-то одного стиля программирования от всех пользователей.

Развитие языка С++ происходило на базе языка С, и, за небольшим исключением, С был сохранен в качестве подмножества C++. Базовый язык С был спроектирован таким образом, что имеется очень тесная связь между типами, операциями, операторами и объектами, с которыми непосредственно работает машина, т.е. числами, символами и адресами. За исключением операций new, delete и throw, а также проверяемого блока, для выполнения операторов и выражений С++ не требуется скрытой динамической аппаратной или программной поддержки.

Первоначально язык С задумывался как конкурент ассемблера, способный вытеснить его из основных и наиболее требовательных к ресурсам задач системного программирования. В проекте С++ были приняты меры, чтобы успехи С в этой области не оказались под угрозой. Различие между двумя языками прежде все состоит в степени внимания, уделяемого типам и структурам. Язык С выразителен и в то же время снисходителен по отношению к типам. Язык С++ еще более выразителен, но такой выразительности можно достичь лишь тогда, когда типам уделяют большое внимание. Когда типы объектов известны, транслятор правильно распознает такие выражения, в которых иначе программисту пришлось бы записывать операции с утомительными подробностями. Кроме того, знание типов позволяет транслятору обнаруживать такие ошибки, которые в противном случае были бы выявлены только при тестировании. Отметим, что само по себе использование строгой типизации языка для контроля параметров функции, защиты данных от незаконного доступа, определения новых типов и операций не влечет дополнительных расходов памяти и увеличения времени выполнения программы.

Анализ среды реализации

Visual Studio 2017 (VS) представляет собой интегрированную среду разработки (Integrated Development Environment, IDE). IDE — это набор инструментов разработчика ПО, собранный в составе единого приложения и облегчающий труд программиста при написании приложений. Без IDE (в данном случае — без VS) для написания программы требуется текстовый редактор, с помощью которого программист вводит весь исходный код своей будущей программы. Затем, когда исходный код написан, необходимо запустить из командной строки компилятор, чтобы создать исполняемый файл приложения. Основная проблема, связанная с использованием текстового редактора и компилятора, запускаемого из командной строки, заключается в том, что вы выполняете большое количество ручной работы и теряете при этом много времени. К счастью, с помощью VS многие из этих рутинных и трудоемких задач, связанных с повседневной работой программиста, можно автоматизировать.

В состав VS входит целый набор типовых проектов, из которых каждый разработчик может подобрать именно то, что ему в данный момент требуется. Каждый раз, когда разработчик создает новый проект, VS автоматически моделирует "скелет" будущего приложения, причем этот код можно немедленно скомпилировать и запустить на исполнение. В составе каждого типового проекта имеются элементы, которые по желанию добавлять в ваш проект. Любой проект, в любом случае, содержит автоматически сгенерированный код, который представляет собой основу будущей программы. VS предлагает множество готовых к использованию элементов управления, включая и код, необходимый для их создания. Это экономит время разработчиков, избавляя их от необходимости каждый раз заново создавать типовой программный код для решения часто встречающихся задач. Многие из более сложных элементов управления содержат так называемые "программы мастера" (Wizards), которые помогают настроить поведение элементов управления, автоматически генерируя код в зависимости от выбранных вами опций.

Редактор VS оптимизирует работу программиста по кодированию. Существенная часть синтаксических элементов новой программы выделяется при помощи системы цветовых обозначений. В вашем распоряжении окажутся такие технологии, как Intellisense, известная как авто дополнение. В ходе того, как вы будете вводить новый код, на экране будут появляться всплывающие подсказки. Наконец, для ускорения выполнения многих задач вам будет предоставлено большое количество клавиатурных комбинаций (keyboard shortcuts). Существует и набор средств быстрой переработки или рефакторинга (refactoring), которые позволяют быстро усовершенствовать структуру кода, не отрываясь от процесса программирования. Например, функция переименования (Rename) позволяет изменить имя идентификатора там, где оно определено, и это повлечет за собой изменение имени данного идентификатора повсюду, где он встречается в коде программы. Кроме того, VS предоставляет и множество других, не менее удобных возможностей. Например, иерархия вызовов (callhierarchy) позволяет программисту проследить всю цепочку вызовов в коде приложения, с верхних уровней до самых нижних. Еще одна возможность, автоматически генерируемые фрагменты (snippets), позволяет вводить сокращения, которые разворачиваются в шаблоны кода (code template). Наконец, списки действий (action lists) предназначены для автоматической генерации нового кода.

Вам действительно необходимо научиться ориентироваться в среде VS, чтобы разобраться во всем множестве новых возможностей, призванных упростить нелегкую задачу быстрой разработки высококачественных приложений. Так, окно Toolbox просто "до отказа" забито различными элементами управления, окно Server Explorer предназначено для работы с сервисами и базами данных операционной системы, а окно Solution Explorer дает возможность работать с вашими проектами, тестировать утилиты, и предоставляет средства визуального проектирования. Кроме того, предоставляются и средства работы с компиляторами.

Многие из элементов, образующих среду VS, являются настраиваемыми. Это значит, что вы можете менять цвета отображения элементов кода, опции редактора, а также общее оформление. Набор опций настолько обширен, что и в самом деле необходимо потратить некоторое время, чтобы ознакомиться с ними и привыкнуть к тому, где и какую из них следует искать. Если стандартные настройки среды VS, которые по умолчанию предлагаются сразу же после установки, вас не устраивают, или не предлагают нужных вам опций, вы можете написать собственные макросы, чтобы автоматизировать последовательности часто выполняемых шагов. Для более сложных вариантов настройки VS предлагает специальный интерфейс прикладного программирования (Application Programming Interface, API), предназначенный для создания собственных дополнительных модулей (add-ins) и расширений (extensions). Некоторые сторонние разработчики уже предлагают приложения, способные интегрироваться в среду VS. В качестве примера можно назвать среду разработки на Delphi от компании Embarcadero1, которая встраивается в Visual Studio.

Исходя из всех данных был выбран язык Python на котором и написано наше приложение.

# **Глава 2. Практическая часть**

## **2.1. Анализ требований и определение спецификаций ПО**

Спецификации данного проекта определяются следующими диаграммами из списка:

* диаграмма вариантов использования;
* детализированная диаграмма потоков данных;
* диаграмма потоков данных.

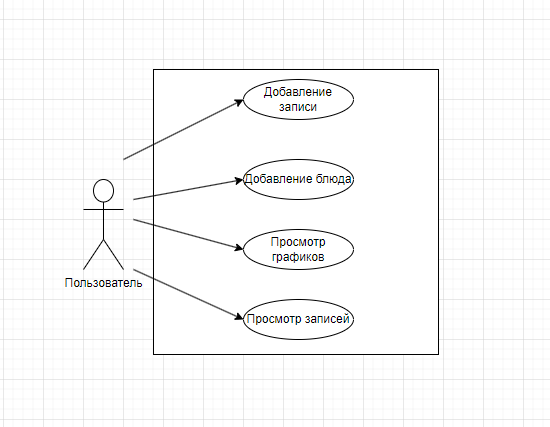


Рисунок 1. Диаграмма вариантов использования

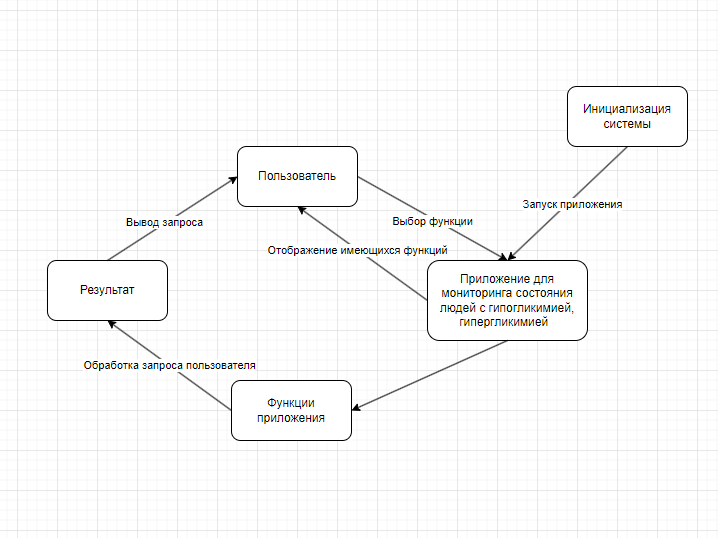


Рисунок 2. Детализированная диаграмма потоков данных

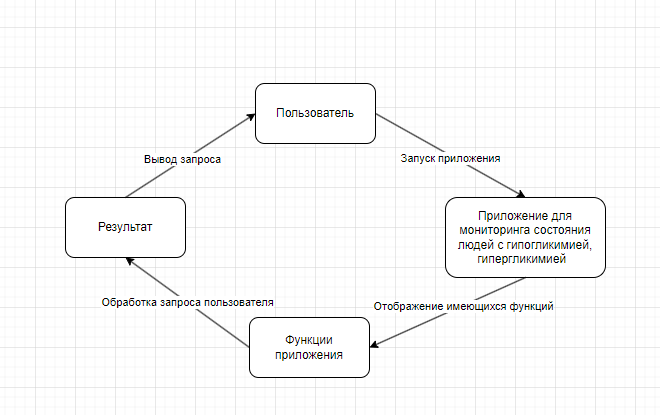


Рисунок 3. Диаграмма потоков данных

## **2.2. Проектирование программного обеспечения**

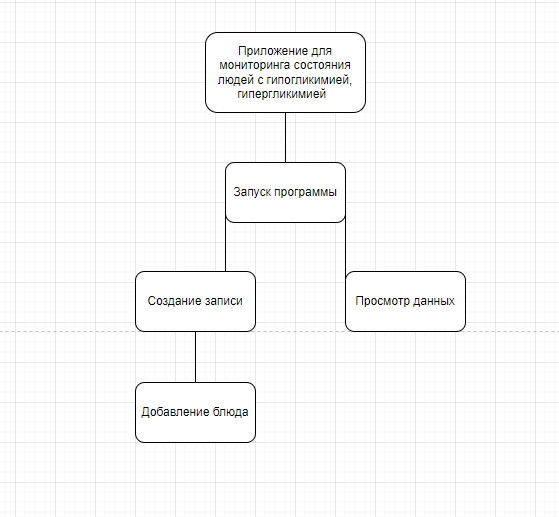


Рисунок 4. Структурная диаграмма

## **2.3. Разработка программного обеспечения**

Приложение для мониторинга состояния людей с гипогликемией, гипергликемией представляет из себя десктопное приложение, разработана на базе платформы Windows. Приложение обладает такими функциями как: добавление записи в дневник, добавление блюда. Так же приложение позволяет просматривать графики нагрузки и сахара в крови. Ниже указано описание этих функций в виде кода.

В данном коде прописано главное окно.

import sys

import logging

import asyncio

from pathlib import Path

import yaml

import platformdirs

import PySide6.QtAsyncio as QtAsyncio

from PySide6.QtWidgets import QApplication

from sqlalchemy import Column, MetaData, ForeignKey

import sqlalchemy.types as sqlt

import sqlalchemy.sql.functions as sqlf

import glycemia\_tracker.components as components

from glycemia\_tracker.common.db import init\_db\_engine, get\_or\_create\_table

from glycemia\_tracker.common.stateful\_main\_window import StatefulMainWindow

async def init\_env\_config(env):

    env['data\_dir'] = Path(platformdirs.user\_data\_dir(env['appname']))

    env['data\_dir'].mkdir(parents = True, exist\_ok = True)

    env['config\_dir'] = Path(platformdirs.user\_config\_dir(env['appname']))

    env['config\_dir'].mkdir(parents = True, exist\_ok = True)

    config\_path = env['config\_dir'] / 'config.yaml'

    if not config\_path.exists():

        config\_path.write\_text(yaml.dump(dict(

            # default config

        )))

    env['config'] = yaml.load(config\_path.read\_text(), yaml.Loader)

    return env

async def init\_env\_db(env):

    engine = init\_db\_engine(

        env['data\_dir'] / 'storage.sqlite',

        pragmas = ['foreign\_keys=ON'])

    metadata = MetaData()

    env['db'] = dict(

        engine = engine,

        metadata = metadata,

        tables = dict(

            users = await get\_or\_create\_table(

                engine, metadata, 'users',

                Column('id', sqlt.Integer(), primary\_key = True),

                Column('c\_dt', sqlt.DateTime(), server\_default = sqlf.current\_timestamp()),

                Column('username', sqlt.String(), nullable = False, unique = True),

                Column('password\_hash', sqlt.String(), nullable = False)),

            dishes = await get\_or\_create\_table(

                engine, metadata, 'dishes',

                Column('id', sqlt.Integer(), primary\_key = True),

                Column('c\_dt', sqlt.DateTime(), server\_default = sqlf.current\_timestamp()),

                Column('name', sqlt.String(), nullable = False, unique = True),

                Column('glycemic\_index', sqlt.Double(), nullable = True)),

            entries = await get\_or\_create\_table(

                engine, metadata, 'entries',

                Column('id', sqlt.Integer(), primary\_key = True),

                Column('user\_id', sqlt.Integer(), ForeignKey('users.id'), nullable = False),

                Column('dish\_id', sqlt.Integer(), ForeignKey('dishes.id'), nullable = True),

                Column('c\_dt', sqlt.DateTime(), server\_default = sqlf.current\_timestamp()),

                Column('dt', sqlt.DateTime()),

                Column('glycemic\_index', sqlt.Double(), nullable = True),

                Column('amount', sqlt.Double(), nullable = True))))

    return env

async def init\_env():

    env = dict(appname = 'Glycemia Tracker')

    for f in [init\_env\_config, init\_env\_db]:

        env |= await f(env)

    return env

class MainWindow(StatefulMainWindow):

    def \_\_init\_\_(self, \*args, \*\*kwargs):

        super(MainWindow, self).\_\_init\_\_(\*args, \*\*kwargs)

        self.setWindowTitle(self.\_env['appname'])

        self.\_init\_menu()

    def \_init\_menu(self):

        menu = self.menuBar().addMenu("&Menu")

        menu.addAction("&Exit", self.close)

def main():

    logging.basicConfig(

        format = '%(asctime)s %(levelname)s %(message)s',

        datefmt = '%Y.%m.%d %H:%M:%S',

        level = logging.INFO)

    QApplication(sys.argv)

    env = asyncio.run(init\_env())

    main\_window = MainWindow(env, [

        components.login,

        components.register,

        components.main\_menu,

        components.diary,

        components.statistics,

        components.add\_entry,

        components.add\_dish])

    main\_window.show()

    QtAsyncio.run()

if \_\_name\_\_ == '\_\_main\_\_':

    sys.exit(main())

В данном коде прописаны графики.

Код \_\_init\_\_.py

from . import controls  # noqa: F401

from . import handlers  # noqa: F401

from . import view  # noqa: F401

Код controls.py

from typing import Callable, Any

from PySide6.QtWidgets import (

    QWidget,

    QPushButton)

from pyqtgraph import PlotWidget, DateAxisItem

from glycemia\_tracker.common.qt import with\_props, noop

def mk\_plot():

    plt = with\_props(

        PlotWidget(axisItems = {'bottom': DateAxisItem(orientation = 'bottom')}),

        (PlotWidget.setBackground, 'w'))

    return plt

def init(handlers: dict[str, Callable[[Any], None]]) -> dict[str, QWidget]:

    controls = dict(

        pw\_intake = mk\_plot(),

        pw\_load = mk\_plot(),

        pb\_back = QPushButton('Назад'))

    for c in ['back']:

        controls[f'pb\_{c}'].clicked.connect((

            lambda c:

            lambda: handlers.get(f'on\_{c}', noop)())(c))

    return controls

Код handlers.py

from glycemia\_tracker.common.types import (

    StackPop)

async def on\_back(env, state):

    yield StackPop()

\_\_all\_\_ = [

    'on\_back']

Код view.py

from typing import Any

from PySide6.QtWidgets import (

    QWidget,

    QVBoxLayout,

    QHBoxLayout)

from pyqtgraph import mkPen

from glycemia\_tracker.common.qt import (

    mk\_layout)

def init(controls: dict[str, QWidget]) -> QWidget:

    return mk\_layout(

        QVBoxLayout,

        mk\_layout(

            QHBoxLayout,

            controls['pw\_intake'],

            controls['pw\_load']),

        controls['pb\_back'])

def plot\_intake(plt, state):

    xs, ys = list(map(list, zip(\*[

        (e['dt'], e['glycemic\_index'])

        for e in state['entries']])))

    ys\_clean = list(filter(lambda x: x is not None, ys))

    if len(ys\_clean) == 0:

        return

    ys\_mean = sum(ys\_clean) / len(ys\_clean)

    plt.clear()

    pen = mkPen(color = (0xFF, 0x40, 0x40), width = 5)

    plt.plot(

        list(map(lambda x: x.timestamp(), xs)),

        list(map(lambda x: x or ys\_mean, ys)),

        pen = pen,

        symbol= '+')

    plt.setTitle('Мои показатели', color = 'k', size = '20pt')

def plot\_load(plt, state):

    xs, ys = list(map(list, zip(\*[

        (e['dt'],

         None

         if ((e['glycemic\_index'] is None) or (e['amount'] is None))

         else (e['glycemic\_index'] \* e['amount']))

        for e in state['entries']])))

    ys\_clean = list(filter(lambda x: x is not None, ys))

    if len(ys\_clean) == 0:

        return

    ys\_mean = sum(ys\_clean) / len(ys\_clean)

    plt.clear()

    pen = mkPen(color = (0xFF, 0x40, 0x40), width = 5)

    plt.plot(

        list(map(lambda x: x.timestamp(), xs)),

        list(map(lambda x: x or ys\_mean, ys)),

        pen = pen,

        symbol= '+')

    plt.setTitle('Гликемическая нагрузка', color = 'k', size = '20pt')

def update(controls: dict[str, QWidget], state: dict[str, Any]) -> ():

    plot\_intake(controls['pw\_intake'], state)

    plot\_load(controls['pw\_load'], state)

В данном коде прописаны графики.

Код handlers.py

from sqlalchemy.exc import IntegrityError

from glycemia\_tracker.common.types import (

    StackPop, GlobalStateUpdate, StateUpdate, ViewUpdate)

from glycemia\_tracker.common.handlers import (with\_lock)

from glycemia\_tracker.components.common import (

    parse\_glycemic\_index,

    fetch\_dish\_items)

async def on\_back(env, state):

    yield StackPop()

@with\_lock

async def on\_add(env, state, name: str, glycemic\_index\_raw: str):

    yield StateUpdate(dict(

        name = name,

        glycemic\_index\_raw = glycemic\_index\_raw))

    yield ViewUpdate()

    if len(name) == 0:

        yield StateUpdate(dict(

            feedback = 'Название не должно быть пустым.'))

        return

    glycemic\_index = None

    match parse\_glycemic\_index(glycemic\_index\_raw):

        case (False, reason):

            yield StateUpdate(dict(feedback = reason))

            return

        case (True, glycemic\_index):

            pass

    async with env['db']['engine'].begin() as tx:

        tbl\_dishes = env['db']['tables']['dishes']

        try:

            stmt = (

                tbl\_dishes

                .insert()

                .values(

                    name = name,

                    glycemic\_index = glycemic\_index)

                .returning(tbl\_dishes.c.id))

            (dish\_id, ) = (await tx.execute(stmt)).first()

        except IntegrityError:

            yield StateUpdate(dict(

                feedback = 'Блюдо с таким названием уже существует.'))

            return

    yield GlobalStateUpdate(dict(

        dish\_items = await fetch\_dish\_items(env)))

    yield StateUpdate(dict(

        name = '',

        glycemic\_index\_raw = '',

        feedback = 'Блюдо успешно добавлено.'))

\_\_all\_\_ = [

    'on\_back',

    'on\_add']

Код controls.py

from typing import Callable, Any

from PySide6.QtWidgets import (

    QWidget,

    QLabel,

    QPushButton,

    QLineEdit)

from glycemia\_tracker.common.qt import noop

def init(handlers: dict[str, Callable[[Any], None]]) -> dict[str, QWidget]:

    controls = dict(

        le\_name = QLineEdit(),

        le\_glycemic\_index = QLineEdit(),

        lbl\_feedback = QLabel(),

        pb\_add = QPushButton('Добавить'),

        pb\_back = QPushButton('Назад'))

    controls['pb\_add'].clicked.connect(lambda: handlers.get('on\_add', noop)(

        name = controls['le\_name'].text(),

        glycemic\_index\_raw = controls['le\_glycemic\_index'].text()))

    for c in ['back']:

        controls[f'pb\_{c}'].clicked.connect((

            lambda c:

            lambda: handlers.get(f'on\_{c}', noop)())(c))

    return controls

Код view.py

from typing import Any

from PySide6.QtWidgets import (

    QWidget,

    QVBoxLayout,

    QHBoxLayout,

    QLabel)

from glycemia\_tracker.common.qt import (

    mk\_form, mk\_layout,

    with\_lock, sync\_lineedit)

def init(controls: dict[str, QWidget]) -> QWidget:

    return mk\_layout(

        QVBoxLayout,

        QLabel('Добавить новое блюдо:'),

        mk\_form(

            ('Название: ', controls['le\_name']),

            ('Гликемический индекс: ', controls['le\_glycemic\_index'])),

        controls['lbl\_feedback'],

        mk\_layout(

            QHBoxLayout,

            controls['pb\_back'],

            controls['pb\_add']))

@with\_lock('pb\_add')

@sync\_lineedit('le\_name', 'name')

@sync\_lineedit('le\_glycemic\_index', 'glycemic\_index\_raw')

def update(controls: dict[str, QWidget], state: dict[str, Any]) -> ():

    controls['lbl\_feedback'].setText(state.get('feedback', ''))

Код init.py

from . import controls  # noqa: F401

from . import handlers  # noqa: F401

from . import view  # noqa: F401

В данном коде прописано добавление записей в дневник.

Код controls.py

from typing import Callable, Any

import datetime

from PySide6.QtWidgets import (

    QWidget,

    QLabel,

    QComboBox,

    QDateTimeEdit,

    QPushButton,

    QLineEdit)

from PySide6.QtCore import (

    QDateTime,

    QDate,

    QTime)

from glycemia\_tracker.common.qt import with\_props, noop, from\_qdatetime

def init(handlers: dict[str, Callable[[Any], None]]) -> dict[str, QWidget]:

    now = datetime.datetime.now()

    controls = dict(

        dte\_dt = with\_props(

            QDateTimeEdit(),

            (QDateTimeEdit.setCalendarPopup, True),

            (QDateTimeEdit.setDisplayFormat, 'yyyy-MM-dd HH:mm'),

            (QDateTimeEdit.setDateTime, QDateTime(

                QDate(now.year, now.month, now.day),

                QTime(now.hour, now.minute)))),

        cb\_dish = with\_props(

            QComboBox(),

            (QComboBox.setCurrentIndex, -1)),

        le\_glycemic\_index = QLineEdit(),

        le\_amount = QLineEdit(),

        lbl\_feedback = QLabel(),

        pb\_add = QPushButton('Добавить'),

        pb\_add\_dish = QPushButton('Добавить блюдо'),

        pb\_back = QPushButton('Назад'))

    controls['pb\_add'].clicked.connect(lambda: handlers.get('on\_add', noop)(

        dt = from\_qdatetime(controls['dte\_dt'].dateTime()),

        dish\_id\_raw = controls['cb\_dish'].currentData(),

        glycemic\_index\_raw = controls['le\_glycemic\_index'].text(),

        amount\_raw = controls['le\_amount'].text()))

    for c in ['back', 'add\_dish']:

        controls[f'pb\_{c}'].clicked.connect((

            lambda c:

            lambda: handlers.get(f'on\_{c}', noop)())(c))

    return controls

Код handlers.py

from glycemia\_tracker.common.types import (

    StackPop, StateUpdate, StackPush, ViewUpdate)

from glycemia\_tracker.common.handlers import (with\_lock)

from glycemia\_tracker.components.common import parse\_glycemic\_index, parse\_amount

async def on\_back(env, state):

    yield StackPop()

async def on\_add\_dish(env, state):

    yield StackPush('add\_dish')

@with\_lock

async def on\_add(

        env, state,

        dt,

        dish\_id\_raw: str | None,

        glycemic\_index\_raw: str,

        amount\_raw: str):

    yield StateUpdate(dict(

        dish\_id\_raw = dish\_id\_raw,

        glycemic\_index\_raw = glycemic\_index\_raw,

        amount\_raw = amount\_raw))

    yield ViewUpdate()

    dish\_id = (None if dish\_id\_raw is None else int(dish\_id\_raw))

    glycemic\_index = None

    match parse\_glycemic\_index(glycemic\_index\_raw):

        case (False, reason):

            yield StateUpdate(dict(feedback = reason))

            return

        case (True, glycemic\_index):

            pass

    amount = None

    match parse\_amount(amount\_raw):

        case (False, reason):

            yield StateUpdate(dict(feedback = reason))

            return

        case (True, amount):

            pass

    async with env['db']['engine'].begin() as tx:

        tbl\_entries = env['db']['tables']['entries']

        stmt = (

            tbl\_entries

            .insert()

            .values(

                user\_id = state['user\_id'],

                dish\_id = dish\_id,

                dt = dt,

                glycemic\_index = glycemic\_index,

                amount = amount)

            .returning(tbl\_entries.c.id))

        (entry\_id, ) = (await tx.execute(stmt)).first()

    yield StateUpdate(dict(

        dish\_id\_raw = None,

        glycemic\_index\_raw = '',

        amount\_raw = '',

        feedback = 'Запись успешно добавлена.'))

\_\_all\_\_ = [

    'on\_back',

    'on\_add',

    'on\_add\_dish']

Код view.py

from typing import Any

from PySide6.QtWidgets import (

    QWidget,

    QVBoxLayout,

    QHBoxLayout,

    QLabel)

from glycemia\_tracker.common.qt import (

    mk\_form, mk\_layout,

    with\_lock, sync\_lineedit)

def init(controls: dict[str, QWidget]) -> QWidget:

    return mk\_layout(

        QVBoxLayout,

        QLabel('Добавить новую запись:'),

        mk\_form(

            ('Дата и время: ', controls['dte\_dt']),

            ('Блюдо: ', controls['cb\_dish']),

            ('Гликемический индекс: ', controls['le\_glycemic\_index']),

            ('Масса: ', controls['le\_amount'])),

        controls['lbl\_feedback'],

        controls['pb\_add'],

        mk\_layout(

            QHBoxLayout,

            controls['pb\_back'],

            controls['pb\_add\_dish']))

@with\_lock('pb\_add')

@sync\_lineedit('le\_glycemic\_index', 'glycemic\_index\_raw')

@sync\_lineedit('le\_amount', 'amount\_raw')

def update(controls: dict[str, QWidget], state: dict[str, Any]) -> ():

    controls['lbl\_feedback'].setText(state.get('feedback', ''))

    controls['cb\_dish'].clear()

    for dish\_id, label in state['dish\_items']:

        controls['cb\_dish'].addItem(label, str(dish\_id))

    if ('dish\_id\_raw' not in state) or (state['dish\_id\_raw'] is None):

        controls['cb\_dish'].setCurrentIndex(-1)

Код \_\_init\_\_.py

from . import controls  # noqa: F401

from . import handlers  # noqa: F401

from . import view  # noqa: F401

На ниже представленных изображениях показаны окна приложения.

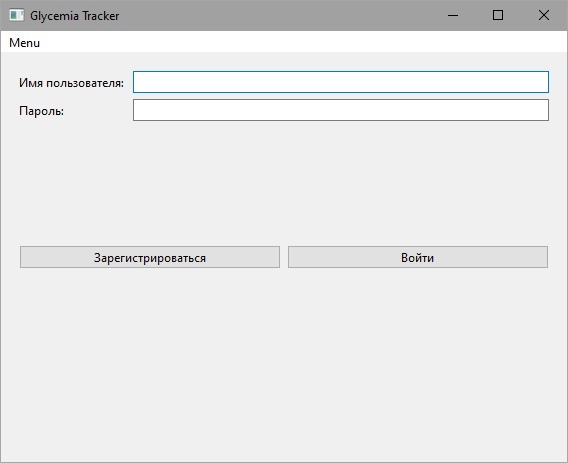


Рисунок 5. Начальная страница

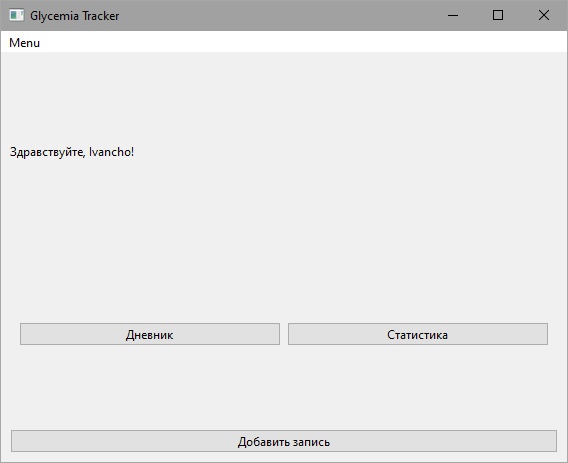


Рисунок 6. Главная страница

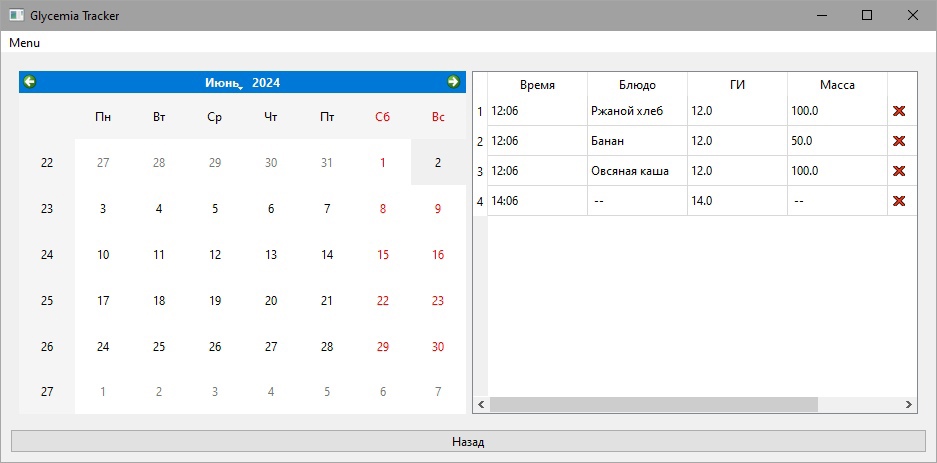


Рисунок 7. Дневник

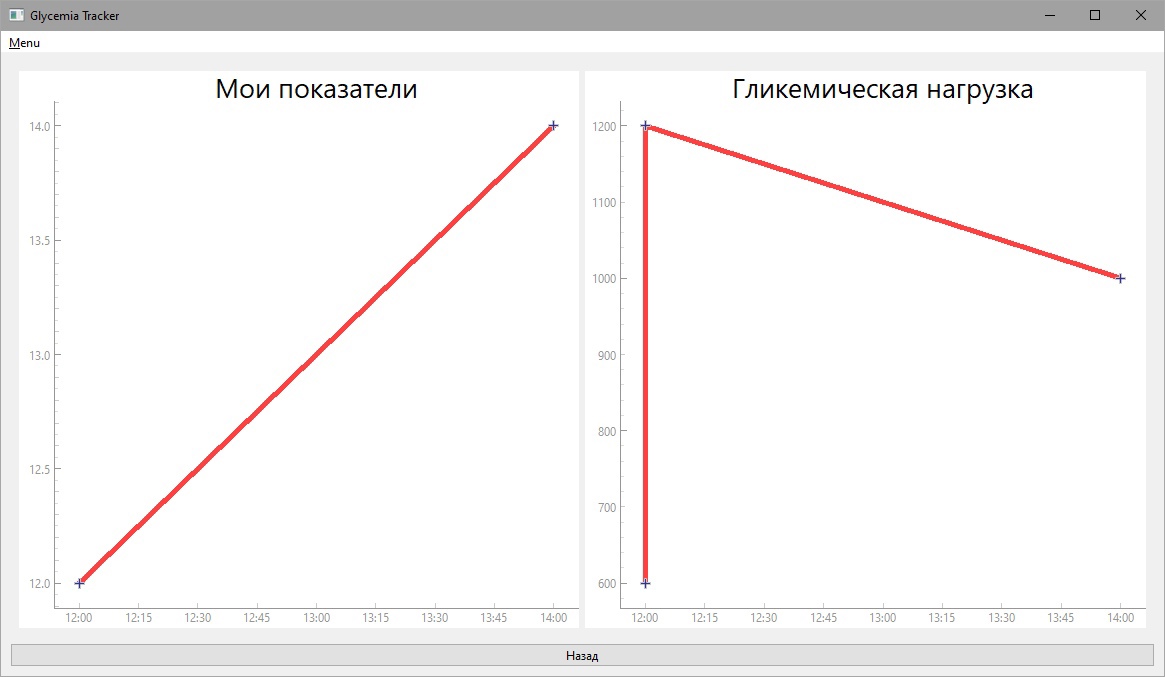


Рисунок 8. Статистика

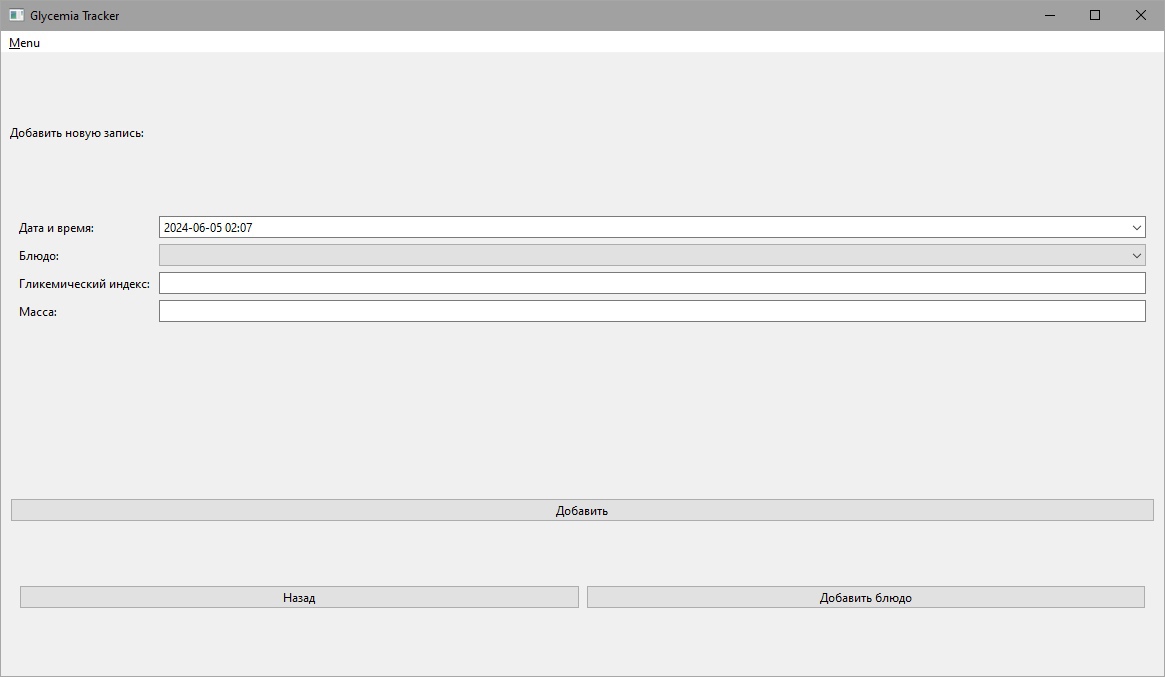


Рисунок 9. Добавление записи

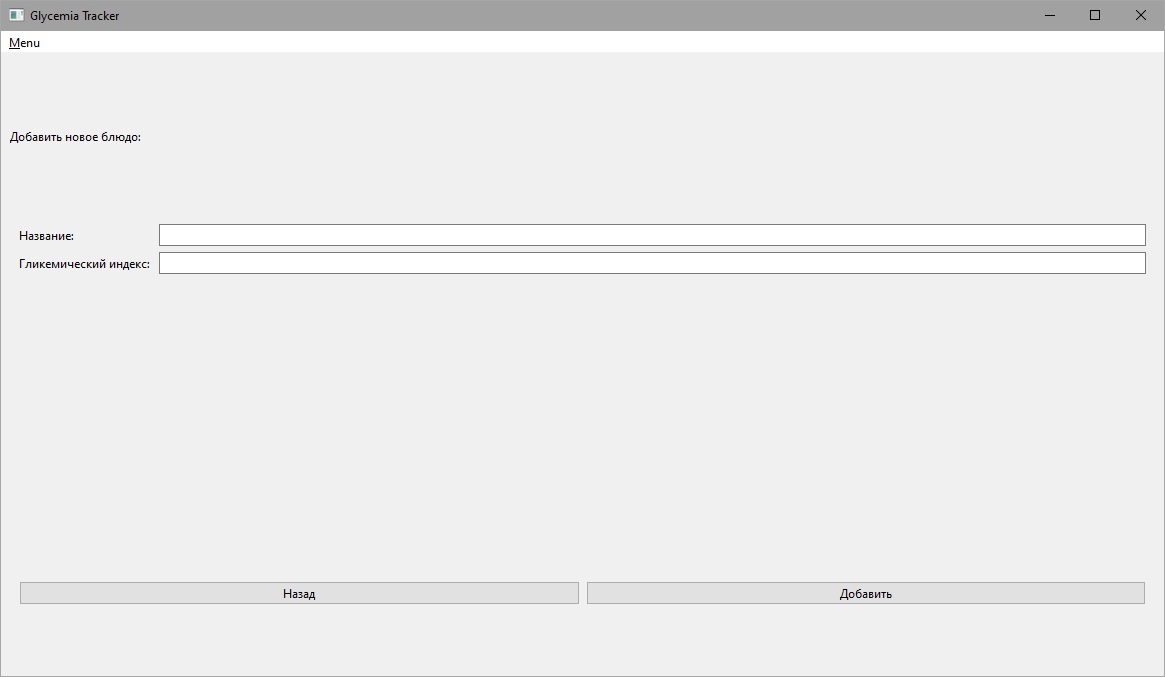


Рисунок 10. Добавление блюда

## **2.4. Отладка и тестирование программы**

Таблица 1. Тестовый пример #1

|  |  |
| --- | --- |
| Тестовый пример | 1.0 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Проверка входа пользователя |
| Краткое изложение теста | Ввод данных пользователя для входа. |
| Этапы теста | Ввод данных пользователя. |
|  | Нажатие на кнопку войти. |
| Тестовые данные | Приложение |
| Ожидаемый результат | Вход пользователя в систему. |
| Фактический результат | Вход пользователя в систему, переход на главную страницу. |
| Статус | Успешно |
| Предварительное условие | Наличие приложения на компьютере. |
| Постусловие |  |
| Примечания/комментарии |  |

Таблица 2. Тестовый пример #2

|  |  |
| --- | --- |
| Тестовый пример | 2.0 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Вход пользователя с неправильными данными. |
| Краткое изложение теста | Вход пользователя с неправильными данными. |
| Этапы теста | Ввод неправильных данных. |
|  | Нажатие на кнопку войти. |
| Тестовые данные | Приложение. |
| Ожидаемый результат | Вывод сообщения о неправильном пароле и имени пользователя. |
| Фактический результат | Вывод сообщения о неправильном пароле и имени пользователя. |
| Статус | Успешно |
| Предварительное условие | Наличие приложения на компьютере. |
| Постусловие |  |
| Примечания/комментарии |  |

Таблица 3. Тестовый пример #3

|  |  |
| --- | --- |
| Тестовый пример # | 3 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Добавление блюда с таким же названием. |
| Краткое изложение теста | Ввод названия блюда, ввод гликемического индекса. |
| Этапы теста | Ввод названия блюда. |
| Тестовые данные | Приложение. |
| Ожидаемый результат | Вывод сообщения «Блюда с таким названием уже существует» |
| Фактический результат | Рассказ анекдота, озвучен |
| Статус | Успешно |
| Предварительное условие | Наличие приложения на компьютере. |
| Постусловие |  |
| Примечания/комментарии |  |

Таблица 4. Тестовый пример #4

|  |  |
| --- | --- |
| Тестовый пример # | 4 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Появление графика нагрузки и моих показателей. |
| Краткое изложение теста | После добавления записи в дневник на странице статистика должны появиться графики. |
| Этапы теста | Добавление записи в дневник. |
| Тестовые данные | Приложение |
| Ожидаемый результат | Вывод графика на странице статистика. |
| Фактический результат | Вывод графика на странице статистика. |
| Статус | Успешно |
| Предварительное условие | Наличие приложения на компьютере. |
| Постусловие |  |
| Примечания/комментарии |  |

# Заключение

В данном дипломном проекте была рассмотрена задача по разработке приложения для мониторинга состояния людей с гипогликемией, гипергликемией, которая может использоваться на компьютерных устройствах.

Для разработки данной программы был использован высокоуровневый язык программирования Python в среде разработки Visual Studio Code. Разработанный программный продукт интуитивно понятный в использовании.

В результате был освоен высокоуровневый язык программирования Python и среда разработки Visual Studio Code благодаря которой было разработано приложение для мониторинга состояния людей с гипогликемией, гипергликемией.

Кроме того, разработанный продукт имеет большой потенциал для дальнейшего развития. Возможно направление для будущих улучшений таких функций как: подключение к глюкометру, добавление различных видов инсулина, благодаря чему упростится использование приложения для мониторинга состояния людей с гипогликемией, гипергликемией в повседневной жизни. Приложение позволяет отслеживать уровень глюкозы в крови, вести дневник питания и физической активности, а также получать рекомендации по лечению от врачей. Разработанное приложение является важным инструментом для контроля и управления диабетом, помогая пациентам улучшить качество своей жизни и снизить риск осложнений. Приложения для диабета представляют собой ценный инструмент для контроля уровня сахара в крови, наблюдения за питанием и упражнениями, а также для мониторинга общего состояния здоровья. Они помогают пациентам лучше ориентироваться в своем заболевании, более эффективно управлять своими симптомами и следить за своими показателями. Приложения могут значительно улучшить качество жизни людей с диабетом, обеспечивая им доступ к необходимой информации и поддержке в любое время и в любом месте. Однако важно помнить, что приложения не заменяют профессионального медицинского совета и консультации специалиста.

# Список использованных источников и интернет-ресурсов

*Законодательные и нормативные акты:*

1. ГОСТ Р 7.0.12-2011 Библиографическая запись. Сокращение слов и словосочетаний на русском языке. Общие требования и правила. – М.: Стандартинформ, 2012. – 61 с.

2. ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: Стандартинформ, 2010. – 92 с.

3. ГОСТ 7.32-2017 Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: Стандартинформ, 2017. – 47 с.

4. ГОСТ 7.82-2001 Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2001. – 39 с.

5. ГОСТ Р 7.0.100-2018 Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: Стандартинформ, 2018. – 122 с.

6. ГОСТ Р 7.0.5-2008 Библиографическая ссылка. Общие требования и правила составления. – М.: Стандартинформ, 2008. – 32 с.

7. Единая система программной документации. – М.: Стандартинформ, 2005. – 128 с.

*Учебная и научная литература:*

8. Иванова, Г.С. Технология программирования: учебник для студентов вузов обуч. по напр. «Информатика и вычислительная техника» / Г.С. Иванова. – 3-е изд., стер. – Москва: Кнорус, 2018. – 333 с.

9. Павловская, Т.А. C#. Программирование на языке высокого уровня: учебник для студентов вузов. – СПб: Питер, 2020. – 432 с.

10. Перлова, О.Н., Ляпина, О.П., Гусева, А.В. Проектирование и разработка информационных систем: учебник. – 2-е изд, стер. – М.: Издательский центр «Академия», 2018. – 256 с.

11. Федорова, Г.Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности: Учебное пособие / Г.Н. Федорова. – М.: КУРС: ИНФРА-М, 2019. – 336 с.

12. Фуфаев, Д.Э., Фуфаев, Э.В., Разработка и эксплуатация автоматизированных информационных систем: учебное пособие для студентов учреждений среднего профессионального образования по специальности "Информатика и вычислительная техника" / Д.Э. Фуфаев, Э.В. Фуфаев. – 6-е изд., стер. – М.: Издательский центр «Академия», 2018. – 302 с.

13. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.

14. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.

15. Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

16. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

17. Гэддис Т. Начинаем программировать на Python. – 4-е изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2019. – 768 с.

18. Свейгарт, Э. Автоматизация рутиных задач с помощью Python: практическое руководство для начинающих. Пер. с англ. — М.: Вильямc, 2016. – 592 с.

19. Рейтц К., Шлюссер Т. Автостопом по Python. – СПб.: Питер, 2017. – 336 с.

20. Дронов В.А. Django 2.1. Практика создания веб-сайтов на Python. – СПб.: БХВ-Петербург, 2019. – 672 с.

*Интернет-документы:*

21. Введение в язык C# и .NET Framework. – [Электронный ресурс]. – URL: https://msdn.microsoft.com/ru-ru/library/z1zx9t92.aspx (дата обращения: 25.09.2023).

22. Гагарина, Л. Г. Технология разработки программного обеспечения: учебное пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Сидорова-Виснадул; под ред. Л.Г. Гагариной. – Москва: ФОРУМ: ИНФРА-М, 2020. – 400 с. – (Среднее профессиональное образование). – ISBN 978-5-8199-0812-9. – Текст: электронный. – URL: https://znanium.com/catalog/product/1067012 (дата обращения: 30.09.2023).

23. Гниденко, И.Г. Технологии и методы программирования: учебное пособие для вузов / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. – Москва: Издательство Юрайт, 2020. – 235 с. – (Высшее образование). – ISBN 978-5- 534-02816-4. – Текст : электронный // ЭБС Юрайт [сайт]. – URL: https://urait.ru/bcode/450999 (дата обращения: 25.09.2023).

24. Гниденко, И.Г. Технология разработки программного обеспечения: учебное пособие для среднего профессионального образования / И.Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. – Москва: Издательство Юрайт, 39 2020. – 235 с. – (Профессиональное образование). – ISBN 978-5-534-05047-9. – Текст: электронный // ЭБС Юрайт [сайт]. – URL: https://urait.ru/bcode/453640 (дата обращения: 26.09.2023).

25. Гуриков, С.Р. Введение в программирование на языке Visual C#: учебное пособие / С.Р. Гуриков. — МОСКВА: ФОРУМ: ИНФРА-М, 2020. — 447 с. — (Высшее образование: Бакалавриат). – ISBN 978-5-00091-458-8. – Текст: электронный. – URL: https://znanium.com/catalog/product/1092167 (дата обращения: 29.09.2023).

26. Интернет-сервис для построения схем и диаграмм Draw.io. – [Электронный ресурс]. – URL: https://www.draw.io/ (дата обращения: 30.09.2023).

27. Интернет-сервис для построения UML-диаграмм. – [Электронный ресурс]. – URL: https://plantuml.com/ (дата обращения: 30.09.2023).

28. Казанский, А. А. Программирование на Visual C#: учебное пособие для вузов / А. А. Казанский. – 2-е изд., перераб. и доп. – Москва: Издательство Юрайт, 2020. – 192 с. – (Высшее образование). – ISBN 978-5-534-12338-8. – Текст: электронный // ЭБС Юрайт [сайт]. – URL: <https://urait.ru/bcode/451467> (дата обращения: 30.09.2023).

29. Полное руководство по языку программирования C# 6.0 и платформе .NET 4.6. – [Электронный ресурс]. – URL: <http://metanit.com/sharp/tutorial/> (дата обращения: 27.09.2023).

30. Руководство по программированию в Windows Forms. – [Электронный ресурс]. – URL: http://metanit.com/sharp/windowsforms/ (дата обращения: 27.09.2023).

31. Руководство по программированию в WPF. – [Электронный ресурс]. – URL: https://metanit.com/sharp/wpf/ (дата обращения: 29.09.2023).

32. Руководство по работе в среде Visual Studio. – [Электронный ресурс]. – URL: https://docs.microsoft.com/ru-ru/visualstudio/ (дата обращения: 25.09.2023).