Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Объектно-ориентированное программирование

К защите допустить:

И.О. Заведующего кафедрой информатики

_____С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ВЕДЕНИЯ ЗАДАЧ

БГУИР КП 1-40 04 01 015 ПЗ

Студент М. А. Снежко

Руководитель Е. В. Тушинская

Нормоконтролер Е. В. Тушинская

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир стал таким благодаря технологиям. С самого раннего возраста родители учат чему-то своих детей, после этого обучение переходит в руки школьных учителей и не только. Из года в год количество обучающихся становится все больше и больше, и контролировать этот процесс становится все труднее. Однако технологии привнесли в жизнь общества новые возможности, в том числе различные программные продукты разных характеров. Одним из таких является система для ведения задач. Программное обеспечение для управления задачами позволяет пользователям легко и эффективно организовывать и контролировать свои дела, будь то учебные задания, рабочие проекты или личные цели. Такие системы упрощают процесс планирования, выполнения и отслеживания задач, что особенно актуально в условиях современного динамичного мира, где управление временем и ресурсами имеет решающее значение. Ключевые возможности приложения включают в себя:

– структурированное планирование и отслеживание личных задач и целей;

– гибкая система категоризации и фильтрации задач;

– визуализация прогресса и статистики выполнения;

– интеграция с календарями и системами напоминаний.

В данной курсовой работе был разработан программный продукт, обеспечивающий ведение и управление личными задачами. Пользователи могут использовать данное приложение для планирования своих дел, установки дедлайнов, отслеживания прогресса и повышения общей продуктивности. Программа позволяет легко создавать и редактировать задачи, устанавливать приоритеты, получать уведомления о предстоящих событиях и анализировать выполненные задачи с помощью отчетов и статистики.

Пояснительная записка оформлена в соответствии с СТП 01-2017 [1].

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

В данном пункте рассмотрим примеры популярных приложений для планирования задач. Приведем примеры 4 популярных систем для управления задачами, которые могут служить аналогами для реализации в данном курсовом проекте.

Первый пример – Trello. Данная платформа разработана для организации и управления задачами с использованием канбан-досок. В Trello пользователи могут создавать доски для различных проектов, добавлять на них карточки с задачами, присваивать метки, сроки выполнения и назначать ответственных. Система позволяет командам и индивидуальным пользователям визуализировать рабочий процесс, отслеживать выполнение задач и координировать действия в реальном времени. Среди особенностей Trello – интеграция с множеством сторонних сервисов, таких как Slack, Google Drive, и возможность автоматизации процессов с помощью Butler.

Второй пример – Asana. Asana является мощной системой для управления проектами и задачами, позволяющей организовать работу команд и индивидуальных пользователей. Приложение поддерживает создание задач, подзадач, назначение исполнителей, установку дедлайнов и отслеживание прогресса. В Asana также присутствуют функции для создания проектов, планирования с использованием календаря и просмотра задач в виде списка или доски. Приложение предлагает интеграцию с различными инструментами, такими как Slack, Google Calendar, и возможностью настройки автоматизации через Zapier.

Третий пример – Todoist. Это приложение ориентировано на управление личными задачами и проектами. Todoist позволяет пользователям создавать задачи, устанавливать сроки, назначать приоритеты и отслеживать прогресс. Приложение поддерживает создание проектов и подзадач, а также предоставляет систему тегов для организации и поиска задач. Todoist известен своим интуитивным интерфейсом и поддержкой кроссплатформенной синхронизации, что делает его удобным для использования на различных устройствах.

Четвертый пример – Microsoft To Do. Это приложение для управления личными и рабочими задачами, интегрированное с экосистемой Microsoft 365. Пользователи могут создавать списки задач, устанавливать напоминания, сроки выполнения и приоритеты. Microsoft To Do поддерживает синхронизацию задач с другими приложениями Microsoft, такими как Outlook и Teams, что позволяет эффективно планировать и управлять своими делами в одном месте. Приложение также предлагает возможность совместной работы

над задачами и списками с другими пользователями.

Таким образом, изучив аналоги программных продуктов для управления задачами, можно выделить следующие требования к курсовому проекту:

– создание и управление задачами: пользователи должны иметь возможность создавать задачи, устанавливать дедлайны и приоритеты, а также отслеживать прогресс их выполнения;

– интуитивный интерфейс: приложение должно быть простым в использовании;

– напоминания и уведомления: система должна напоминать пользователям о предстоящих задачах и дедлайнах;

– анализ и отчетность: приложение должно предоставлять инструменты для анализа выполненных задач и оценки продуктивности;

– масштабируемость и интеграция: возможность интеграции с другими популярными сервисами и адаптация под индивидуальные потребности пользователей.

В данной курсовой работе был разработан программный продукт, обеспечивающий планирование и управление личными задачами. Пользователи могут использовать данное приложение для организации своих дел, повышения продуктивности и эффективного управления временем. Приложение предоставляет инструменты для создания и редактирования задач, установки приоритетов и дедлайнов, получения уведомлений о предстоящих событиях и анализа выполненных задач с помощью отчетов и статистики.

1.2 Постановка задачи

В рамках данного курсового проекта была поставлена задача: разработать программное средство для планирования задач.

Для организации разработки продукта было составлено следующее техническое задание:

Суть проекта состоит в том, чтобы создать систему, которая будет включать функционал создания, редактирования, управления и отслеживания задач, как личных, так и рабочих. Задачи можно будет формировать с различными параметрами, такими как сроки выполнения, приоритеты, статус и возможность добавления подзадач. Также будет возможность устанавливать напоминания и уведомления для своевременного выполнения задач. В систему можно будет зайти только зарегистрировавшись.

Среди основных функций можно выделить:

– регистрация и авторизация: пользователи могут регистрироваться в системе, вводя логин и пароль, и авторизоваться для получения доступа к своим задачам;

– создание задач: возможность создания задач с указанием названия, описания, сроков выполнения, приоритетов и ответственных лиц;

– редактирование задач: возможность изменения параметров задачи, таких как сроки, приоритеты, статус и описание;

– удаление задач: возможность удаления задач из системы;

– управление подзадачами: возможность добавления, редактирования и удаления подзадач внутри основной задачи;

– установка напоминаний и уведомлений: система будет напоминать пользователям о предстоящих сроках выполнения задач;

– отслеживание статуса задач: возможность отслеживания текущего статуса задач (в процессе, завершена, просрочена и т.д.);

– просмотр списка задач: отображение всех задач в виде списка или доски с возможностью фильтрации и сортировки по различным параметрам.

– анализ: система будет предоставлять инструменты для анализа выполненных задач и оценки продуктивности пользователя;

В данной работе каждая функция содержит проверку корректности вводимых данных с последующей обработкой. Это необходимо для обеспечения стабильной и надежной работы программы, а также предотвращения ошибок при некорректном вводе пользователем. Перед выполнением основных операций функции проверяют входные параметры на соответствие заданным требованиям – допустимый диапазон, тип данных и т.д. В случае обнаружения ошибок выдается понятное сообщение, и выполнение функции прерывается. Таким образом, реализованная проверка корректности данных повышает надежность и безопасность работы программы, обеспечивая ее стабильное функционирование.

Разработав техническое задание и определив базовые необходимые функции для работы приложение можно перейти к проектированию программного продукта.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Разработка функциональности программного средства

Для создания данного приложения было решено использовать язык программирования C# и интегрированную среду разработки Microsoft Visual Studio 2022. Данная платформа предоставляет мощные инструменты для разработки, отладки и развертывания приложений. Благодаря своим возможностям, Visual Studio 2022 является идеальной средой для разработки кроссплатформенных приложений с использованием .NET MAUI, что позволяет разрабатывать приложения, которые могут работать на различных платформах, таких как Windows, macOS, iOS и Android.

NET MAUI (Multi-platform App UI) — это фреймворк, разработанный компанией Microsoft, который позволяет создавать современные кроссплатформенные приложения. Он предоставляет широкий набор инструментов для создания пользовательского интерфейса, работы с данными и интеграции с платформенными API. Среди достоинств .NET MAUI следует отметить высокую производительность приложений, возможность создания красивого и адаптивного пользовательского интерфейса, а также упрощение процесса разработки за счет использования единого кода для различных платформ.

C# продолжает развиваться и добавляет новые функциональные возможности с каждым релизом. Последние версии C# включают в себя улучшения в области обработки строк, синтаксиса, управления памятью и производительности. Что касается MAUI, это относительно новая кроссплатформенная разработка от Microsoft, выпущенная в 2022 году. MAUI позволяет разрабатывать приложения для различных платформ, включая Windows, macOS, iOS и Android, используя единую кодовую базу на C#. MAUI предлагает встроенную поддержку WinUI, XAML и .NET 6, что значительно упрощает разработку кроссплатформенных приложений. За последний год Maui получила множество улучшений, таких как лучшая интеграция с Azure, улучшенная производительность и новые визуальные элементы.

В целом, как C#, так и MAUI активно развиваются и предоставляют разработчикам все больше возможностей для создания современных, кроссплатформенных приложений.

2.2 Разработка функциональности программного средства

Проанализировав требования к проектируемой программной системе для управления задачами, были выделены возможности пользователя,

представленные в виде диаграммы прецедентов на рисунке 2.2.

Рисунок 2.2 – Диаграмма сценариев использования

2.3 Архитектура программного средства

Для реализации программного средства была выбрана многоуровневая (слоистая) архитектура. Она представляет из себя три уровня (или слоя): уровень представления, уровень приложения и уровень данных.

Самый верхний и первый уровень – уровень представления. В данном слое реализуются необходимые инструменты для отображения и получения контента от пользователя, то есть доступ к этому слою можно получить через

любое клиентское устройство. В разрабатываемом приложении данный слой будет представлен классами и ViewModel, которые занимаются транспортировкой информации между слоем бизнес-логики приложения и пользователем.

Следующий уровень – уровень приложения, или уровень бизнес-логики. Данный слой не знает о том, откуда пришли данные и куда пойдут после обработки. На этом уровне обеспечивается лишь общая логика работы приложения и преобразование данных в нем. В разрабатываемом приложении этот слой будет выражен классами, которые обрабатывают данные, которые приходят из уровня данных и отдают уже обработанную информацию далее на слой выше, а именно на слой представления.

Заключительный уровень – уровень данных. Данный слой занимается сохранением или же хранением данных, которые пришли из слоя бизнес-логики или же полученные из сторонних источников, например файлов или баз данных. В данном программном продукте планируется подключение базы данных, следовательно, в этом приложении данный слой будет выражен классом, который будет обеспечивать информацией слой бизнес-логики, если там это потребуется, а также сохранение данных пришедших оттуда.

Такая архитектура позволяет легко заменять или обновлять отдельные компоненты без необходимости переписывания всего приложения. Например, если возникнет необходимость в использовании иного способа хранения данных, можно будет заменить только уровень данных, не затрагивая при этом уровень представления или бизнес-логики. Благодаря четкому разделению ответственности, многоуровневая архитектура также повышает тестируемость и надежность системы в целом. Тестирование каждого уровня по отдельности становится более эффективным, что помогает выявлять и устранять ошибки на ранних этапах разработки.

Исходя из того, что данные могут быть переданы только между соседними уровнями, можно составить общую структуру взаимодействия отдельных компонент приложения в виде диаграммы классов, представленной на рисунке 2.3.

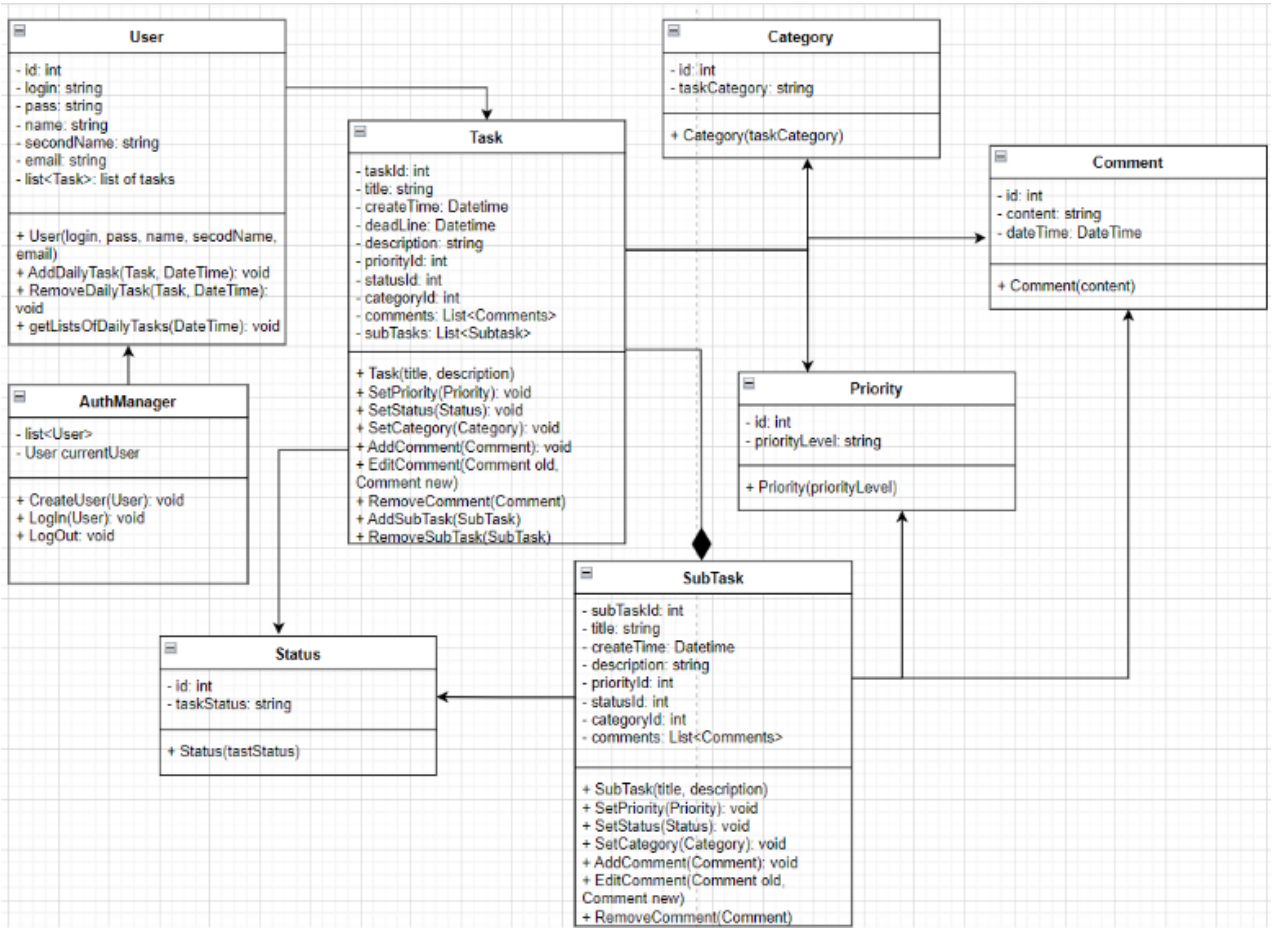

Рисунок 2.3 – Диаграмма классов приложения

Таким образом, многоуровневая архитектура обеспечивает четкое разделение ответственности между различными компонентами приложения, что повышает его гибкость, масштабируемость и модульность. Каждый уровень может быть реализован и протестирован независимо, что упрощает процесс разработки и поддержки программного средства. Помимо этого, многоуровневая архитектура обеспечивает более эффективное использование вычислительных ресурсов. Каждый уровень может быть реализован на отдельном сервере или кластере, что позволяет масштабировать приложение в соответствии с потребностями пользователей.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка уровня представления

Как было описано ранее, данный слой отвечает за отображение данных и взаимодействие с пользователем. В нашем проекте, использующем .NET MAUI и C#, уровень представления реализован в виде страниц и соответствующих им ViewModel. Компоненты уровня представления находятся в пространстве имен Manager.ui. На этом уровне представлены страницы, отвечающие за отображение интерфейса и получение данных от пользователя, а также ViewModel, обеспечивающие привязку данных и логику взаимодействия. Примеры реализации функций уровня представления:

– «TaskListPage.xaml» и «TaskListViewModel.cs»: эти файлы отвечают за отображение списка задач и взаимодействие с ним. TaskListViewModel содержит логику для загрузки, добавления и удаления задач, а TaskListPage – XAML разметку для отображения этого списка;

– «TaskDetailPage.xaml» и «TaskDetailViewModel.cs»: эти файлы обеспечивают отображение подробной информации о задаче и возможность ее редактирования. TaskDetailViewModel содержит логику для загрузки данных о задаче, сохранения изменений и управления состоянием задачи, а TaskDetailPage – XAML разметку для отображения и редактирования задачи.

Примеры функций, реализованных в ViewModel:

– «LoadTasks()»: загружает список задач из уровня бизнес-логики и передает их на отображение в TaskListPage;

– «AddTask()»: добавляет новую задачу, используя данные, введенные пользователем, и сохраняет ее через уровень бизнес-логики.

– «DeleteTask()»: удаляет выбранную задачу из списка и уровня данных;

– «SaveTask()»: сохраняет изменения, внесенные пользователем в детали задачи, и обновляет данные в уровне бизнес-логики.

Примеры функций, реализованных в страницах:

– «DisplayTaskList()»: отображает список задач, используя данные из TaskListViewModel;

– «ShowTaskDetails()»: переходит к отображению подробной информации о выбранной задаче на странице TaskDetailPage.

3.2 Разработка уровня приложения

На уровне приложения были разработаны различные сущности, а также менеджеры и интерфейсы, обеспечивающие их взаимодействие.

Category – класс, представляющий категорию задач. Необходим для классификации задач, помогая организовать их по разным категориям для

более удобного управления и поиска.

Comment – класс, представляющий комментарий к задаче. Предназначен для добавления комментариев к задачам пользователями, что позволяет отслеживать обсуждения и примечания по конкретным задачам.

Priority – класс, представляющий приоритет задачи. Нужен для определения важности задач, помогая пользователям фокусироваться на более приоритетных задачах.

Status – класс, представляющий статус задачи. Используется для отслеживания текущего состояния задачи, например, ожидается, выполняется, завершена, что помогает в управлении жизненным циклом задач.

Subtask – класс, представляющий подзадачу. Связан с классом Task как дочерний элемент. Используется для детализирования и разбиения основной задачи на более мелкие части, что облегчает управление комплексными задачами.

Task – основной класс, представляющий задачу. Связан с классами Category, Priority, Status, Subtask и Comment. Основная задача класса – хранение информации о задачах, таких как название, описание, дедлайн и другие параметры.

User – класс, представляющий пользователя системы. Содержит информацию о пользователях, таких как имя, электронная почта, роль и т.д., что необходимо для управления пользователями и их действиями в системе.

AuthManager – класс, ответственный за управление аутентификацией и авторизацией пользователей. Принадлежит уровню приложения. Использует сущность User для проверки учетных данных и управления сеансами пользователей. Обеспечивает безопасность системы, проверяя и подтверждая учетные данные пользователей.

IRepository – интерфейс, определяющий основные методы для работы с репозиторием данных. Принадлежит уровню данных и бизнес-логики. Реализуется различными классами репозиториев для работы с разными типами данных. Определяет контракт для операций CRUD (Create, Read, Update, Delete), что обеспечивает единообразие и гибкость работы с данными.

IUnitOfWork – интерфейс, определяющий методы для координации работы репозиториев. Принадлежит уровню данных и бизнес-логики. Используется для обеспечения атомарности операций, связанных с несколькими репозиториями. Управляет транзакциями и изменениями данных, гарантируя согласованность и целостность данных при выполнении сложных операций.

3.3 Разработка уровня данных

Данный слой обеспечивает взаимодействие с базой данных и передачу информации из нее на слой бизнес-логики и наоборот. В качестве базы данных была выбрана MongoDB, в которой хранятся данные сущностей приложения.

Основные классы, реализующие уровень данных:

– «AppDbContext»: класс контекста базы данных, отвечающий за конфигурацию и взаимодействие с MongoDB;

– «DataRepositories»: общие репозитории для работы с различными сущностями, реализующие интерфейсы IRepository, который определяет базовые операции для работы с данными (создание, чтение, обновление, удаление). Каждая сущность имеет собственный репозиторий, который инкапсулирует логику работы с ней;

– «UnitOfWork»: класс, реализующий транзакционную логику для координации работы с несколькими репозиториями.

Эта архитектура позволяет легко заменять или обновлять отдельные компоненты без необходимости переписывания всего приложения. Например, если возникнет необходимость в использовании иного способа хранения данных, можно будет заменить только уровень данных, не затрагивая при этом уровень представления или бизнес-логики. Благодаря четкому разделению ответственности, многоуровневая архитектура также повышает тестируемость и надежность системы в целом. Тестирование каждого уровня по отдельности становится более эффективным, что помогает выявлять и устранять ошибки на ранних этапах разработки.

4 ПРОВЕРКА РАБОТОСПОСОБНОСТИ ТЕСТОВ

Осуществлялось функциональное тестирование. Результаты проведенного тестирования приведены в таблице 4.1.

Таблица 4.1 – Тестирование программного средства

№	Тестируемая функция	Ожидаемый результат	Полученный результат
1	Регистрация пользователя	При выборе в начальном меню пункта регистрации и вводе корректных результатов в базе данных должен появиться созданный объект с информацией о данном пользователе	Соответствует ожидаемому
2	Фильтрация задач	При выборе функции фильтрации в меню должны отображаться все задачи, соответствующие выбранным критериям	Соответствует ожидаемому
3	Просмотр статистики пользователя	При выборе функции просмотра статистики должно быть отображение статистики текущего пользователя, его данные и информация о его списке задач	Соответствует ожидаемому
4	Создание новой задачи	При выборе функции создания новой задачи происходит добавление новой задачи с указанными параметрами в репозиторий	Соответствует ожидаемому
5	Добавление комментария к задаче	При выборе функции добавления комментария к задаче должно появиться окно ввода текста комментария, после чего комментарий добавляется к выбранной задаче	Соответствует ожидаемому
6	Изменение задачи	При выборе функции изменения задачи должно появиться окно редактирования параметров задачи, после чего обновленные данные сохраняются в репозитории	Соответствует ожидаемому

Продолжение таблицы 4.1

№	Тестируемая функция	Ожидаемый результат	Полученный результат
7	Загрузка задач из базы данных	При выборе данной функции и вводе корректных критериев из базы данных загружаются задачи, соответствующие указанным условиям	Соответствует ожидаемому
8	Удаление задачи	При выборе данной функции выбранная задача удаляется из базы данных без возможности восстановления	Соответствует ожидаемому
9	Авторизация пользователя	При выборе в начальном меню пункта авторизации и вводе корректных результатов из базы данных должна поступить информация о данном пользователе, происходит проверка хэшированных паролей и предоставляется доступ к дальнейшему использованию приложения или наоборот, если пароли не совпадают или данного пользователя среди зарегистрированных нет	Соответствует ожидаемому
10	Удаление комментария	При выборе функции удаления комментария из задачи и подтверждении действия комментарий удаляется из задачи	Соответствует ожидаемому

Таким образом, на основе выполненных тестов можно сделать вывод о том, что программное средство работает исправно и готово к дальнейшему использованию.

ЗАКЛЮЧЕНИЕ

В результате работы над курсовым проектом было разработано программное средство для тестирования, которое полностью удовлетворяет условиям поставленной задачи.

Реализация приложения включала в себя несколько ключевых этапов: анализ требований и проектирование, разработка и реализация, тестирование и отладка.

На первом этапе были определены основные функциональные требования к приложению, такие как возможность создания и редактирования задач, проведение тестирования с различными типами задач, и предоставление результатов тестирования. Было разработано проектное решение, включая архитектуру системы и пользовательский интерфейс.

На втором этапе использовались современные технологии и инструменты, такие как язык программирования C# на платформе .MAUI и MongoDB для хранения данных. Основное внимание уделялось обеспечению удобства использования и стабильности работы приложения.

Третий этап включал тестирование всех компонентов приложения, выявление и исправление ошибок. Также проводилось пользовательское тестирование для оценки удобства интерфейса и функциональности приложения.

Данное приложение для трекинга личных задач было разработано с применением принципов объектно-ориентированного программирования (ООП) и архитектурного паттерна SOLID. Благодаря этому, конечный продукт получился легко масштабируемым и открытым для дальнейшего совершенствования и расширения функциональности. С помощью этого программного средства пользователь может эффективно отслеживать, организовывать и контролировать свои личные задачи. Приложение также предоставляет возможности для самоанализа, что может быть полезно для личностного и профессионального развития. С помощью этого приложения вы можете эффективно отслеживать выполнение своих задач и управлять своим временем. Программа предоставляет возможность детально планировать задачи, устанавливать дедлайны и следить за их выполнением.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Доманов, А. Т. Стандарт предприятия / А. Т. Доманов, Н. И. Сорока. – Минск: БГУИР, 2017. – 167 с.

[2] Microsoft [Электронный ресурс], – Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/maui/. – Дата доступа: 30.05.2024.

[3] Jesse Liberty, Rodrigo Juarez NET MAUI for C# Developers: Build cross-platform mobile and desktop applications, 2023. – 400 с.

[4] Matt Goldman, .NET MAUI in Action, 2023. – 453 с.

[5] dev.to [Электронный ресурс], – Режим доступа: https://dev.to/smartmanapps/. – Дата доступа: 30.05.2024.

[6] habr [Электронный ресурс], – Режим доступа: https://habr.com/. – Дата доступа: 30.05.2024.

[7] Прайс Марк Дж., C# 10 и .NET 6. Современная кроссплатформенная разработка, 2023, - 848 с.

[8] Албахари Дж., Албахари Б., C# 9.0. Карманный справочник, 2021, с – 256

[9] Mark J. Price, Apps and Services with .NET 8: Build practical projects with Blazor, .NET MAUI, gRPC, GraphQL, and other enterprise technologies, Second Edition, 2023, - 798 с.

[10] Metanit [Электронный ресурс], – Режим доступа: https://metanit.com/sharp/maui/. – Дата доступа: 30.05.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код

```
<ContentPage
xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Manager.UI.Pages.MainPage"
             xmlns:entities="clr-
namespace:Manager.Domain.Entities;assembly=Manager.Domain"
             xmlns:models="clr-namespace:Manager.UI.ViewModels"
             Title=""
             x:DataType="models:MainPageViewModel"
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/too
lkit"
             NavigationPage.HasNavigationBar="False"
             Shell.NavBarIsVisible="False"
             xmlns:converter="clr-
namespace:Manager.UI.Converters">
    <ContentPage.Behaviors>
        <toolkit:EventToCommandBehavior EventName="Loaded"
        Command="{Binding TaskListCommand }"/>
    </ContentPage.Behaviors>
    <Grid>
        <ScrollView>
            <VerticalStackLayout Padding="30">
                <Label Text="Фильтр по приоритету"
FontAttributes="Bold"/>
                <Picker ItemsSource="{Binding Priorities}"
ItemDisplayBinding="{Binding PriorityLevel}"
SelectedItem="{Binding SelectedPriority}"/>
                <Label Text="Фильтр по статусу"
FontAttributes="Bold"/>
                <Picker ItemsSource="{Binding Statuses}"
ItemDisplayBinding="{Binding TaskStatus}" SelectedItem="{Binding
SelectedStatus}"/>
                <Label Text="Фильтр по категории"
FontAttributes="Bold"/>
                <Picker ItemsSource="{Binding Categories}"
ItemDisplayBinding="{Binding TaskCategory}"
SelectedItem="{Binding SelectedCategory}"/>
                <Label Text="Сортировка по"
FontAttributes="Bold"/>
                <Picker ItemsSource="{Binding SortCriteria}"
SelectedItem="{Binding SelectedSortCriterion}"/>
                <Label Text="Возвр/Убыв" FontAttributes="Bold"/>
                <Switch IsToggled="{Binding IsSortAscending}"/>
```

```
                <Button Text="Применить" Command="{Binding
ApplyFiltersCommand}" HorizontalOptions="FillAndExpand"
FontSize="18" Margin="0,20,0,0"/>
                <CollectionView ItemsSource="{Binding Tasks}">
                    <CollectionView.ItemTemplate>
                        <DataTemplate
x:DataType="entities:Task">
                            <Frame
                                BackgroundColor="{Binding
status, Converter={converter:TaskStatusToBorderColorConverter}}"
                                   CornerRadius="5" Margin="10"
Padding="10">
                                <Frame.GestureRecognizers>
                                    <TapGestureRecognizer
Command="{Binding Source={RelativeSource AncestorType={x:Type
models:MainPageViewModel}},
                                Path=ShowDetailsCommand}"
CommandParameter="{Binding}"/>
                                </Frame.GestureRecognizers>
                                <StackLayout>
                                    <Label Text="{Binding Title
}" FontSize="18" FontAttributes="Bold" />
                                    <Label Text="{Binding
Description }" FontSize="16" />
                                </StackLayout>
                            </Frame>
                        </DataTemplate>
                    </CollectionView.ItemTemplate>
                </CollectionView>
                <Button Text="Подробнее" Command="{Binding
GoToProfileCommand}" HorizontalOptions="FillAndExpand"
FontSize="18" Margin="10,20,0,0" />
                <Button Text="Добавить задачу" Command="{Binding
GoToAddTaskCommand}" HorizontalOptions="FillAndExpand"
FontSize="18" Margin="0,20,0,0" />
                <Button Text="Профиль" Command="{Binding
GoToUserPageCommand}" HorizontalOptions="FillAndExpand"
FontSize="18" Margin="0,20,0,0" />
            </VerticalStackLayout>
        </ScrollView>
    </Grid>
</ContentPage>

namespace Manager.UI.ViewModels
{
    public partial class MainPageViewModel : ObservableObject
    {
        private readonly AuthManager _authManager;
        public ObservableCollection<Domain.Entities.Task> Tasks
{ get; set; } = [];
        [ObservableProperty]
```

```
        private User user;
        [ObservableProperty]
        private string newTaskTitle;
        [ObservableProperty]
        private string newTaskDescription;
        [ObservableProperty]
        private ObservableCollection<string> sortCriteria;
        public MainPageViewModel(AuthManager authManager)
        {
            _authManager = authManager;
            User = _authManager.CurrentUser;
            Tasks = new
ObservableCollection<Domain.Entities.Task>(User.Tasks);
            SortCriteria = new ObservableCollection<string>
            {
                "Название",
                "Дата создания",
                "Дедлайн"
            };
        }
        [RelayCommand]
        public async Taskk GoToAddTask()
        {
            await Shell.Current.GoToAsync(nameof(AddTaskPage));
        }
        [RelayCommand]
        public async Taskk GoToProfile()
        {
            await Shell.Current.GoToAsync(nameof(ProfilePage));
        }
        [RelayCommand]
        async void AddTask() => await AddNewTask();
        public async Taskk AddNewTask()
        {
            if (User.Tasks == null)
            {
                User.Tasks = [];
            }
            var newTask = new Domain.Entities.Task
            {
                Title = NewTaskTitle,
                Description = NewTaskDescription,
            };
            User.Tasks.Add(newTask);
            Tasks.Add(newTask);
            await _authManager.UpdateUserAsync(User);
        }
        [RelayCommand]
        async Taskk TaskList() => await LoadTasksAsync();
        private async Taskk LoadTasksAsync()
        {
            if (User?.Tasks != null && User.Tasks.Count != 0)
```

```
            {
                var taskCopy = new List<Task>(User.Tasks);
                User.Tasks.Clear();
                foreach (var task in taskCopy)
                {
                    User.Tasks.Add(task);
                }
            }
            var priorityList = await
_authManager._unitOfWork.PriorityRepository.GetAllAsync();
            Priorities = new
ObservableCollection<Priority>(priorityList);
            var categoryList = await
_authManager._unitOfWork.CategoryRepository.GetAllAsync();
            Categories = new
ObservableCollection<Category>(categoryList);
            var statusList = await
_authManager._unitOfWork.StatusRepository.GetAllAsync();
            Statuses = new
ObservableCollection<Status>(statusList);
        }
        [ObservableProperty]
        private ObservableCollection<Priority> priorities;
        [ObservableProperty]
        private ObservableCollection<Category> categories;
        [ObservableProperty]
        private ObservableCollection<Status> statuses;
        [ObservableProperty]
        private Priority selectedPriority;
        [ObservableProperty]
        private Category selectedCategory;
        [ObservableProperty]
        private Status selectedStatus;
        [ObservableProperty]
        private bool isSortAscending = true;
        [ObservableProperty]
        private string selectedSortCriterion;
        [RelayCommand]
        public void ApplyFilters()
        {
            var filteredTasks = User.Tasks.AsEnumerable();
            if (SelectedPriority != null)
            {
                filteredTasks = filteredTasks.Where(t =>
t.priority.PriorityLevel == SelectedPriority.PriorityLevel);
            }
            if (SelectedStatus != null)
            {
                filteredTasks = filteredTasks.Where(t =>
t.status.TaskStatus == SelectedStatus.TaskStatus);
            }
            if (SelectedCategory != null)
```

```
            {
                filteredTasks = filteredTasks.Where(t =>
t.category.TaskCategory == SelectedCategory.TaskCategory);
            }
            switch (SelectedSortCriterion)
            {
                case "Title":
                    filteredTasks = IsSortAscending
                        ? filteredTasks.OrderBy(t => t.Title)
                        : filteredTasks.OrderByDescending(t =>
t.Title);
                    break;
                case "CreateTime":
                    filteredTasks = IsSortAscending
                        ? filteredTasks.OrderBy(t =>
t.CreateTime.Date).ThenBy(t => t.CreateTime.TimeOfDay)
                        : filteredTasks.OrderByDescending(t =>
t.CreateTime.Date).ThenByDescending(t =>
t.CreateTime.TimeOfDay);
                    break;
                case "Deadline":
                    filteredTasks = IsSortAscending
                        ? filteredTasks.OrderBy(t =>
t.Deadline.Date).ThenBy(t => t.Deadline.TimeOfDay)
                        : filteredTasks.OrderByDescending(t =>
t.Deadline.Date).ThenByDescending(t => t.Deadline.TimeOfDay);
                    break;
                default:
                    break;
            }
            Tasks.Clear();
            foreach (var task in filteredTasks)
            {
                Tasks.Add(task);
            }
            OnPropertyChanged(nameof(Tasks));
        }
        [RelayCommand]
        public async Taskk GoToUserPage()
        {
            await Shell.Current.GoToAsync(nameof(UserPage));
        }
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Схемы алгоритмов программного средства

Рисунок Б.1 – Блок-схема алгоритма добавления задачи

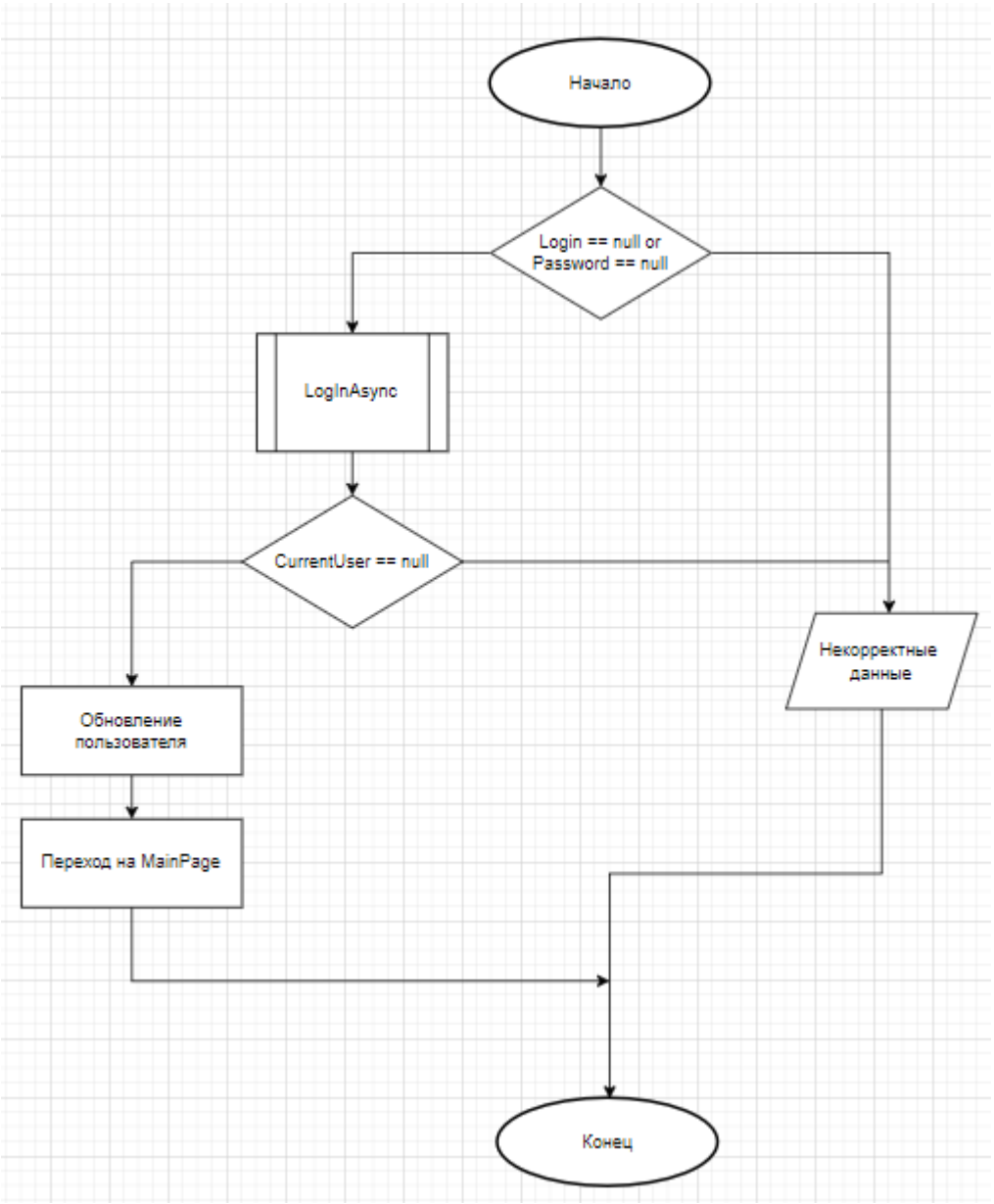

Рисунок Б.2 – Блок-схема алгоритма входа в аккаунт

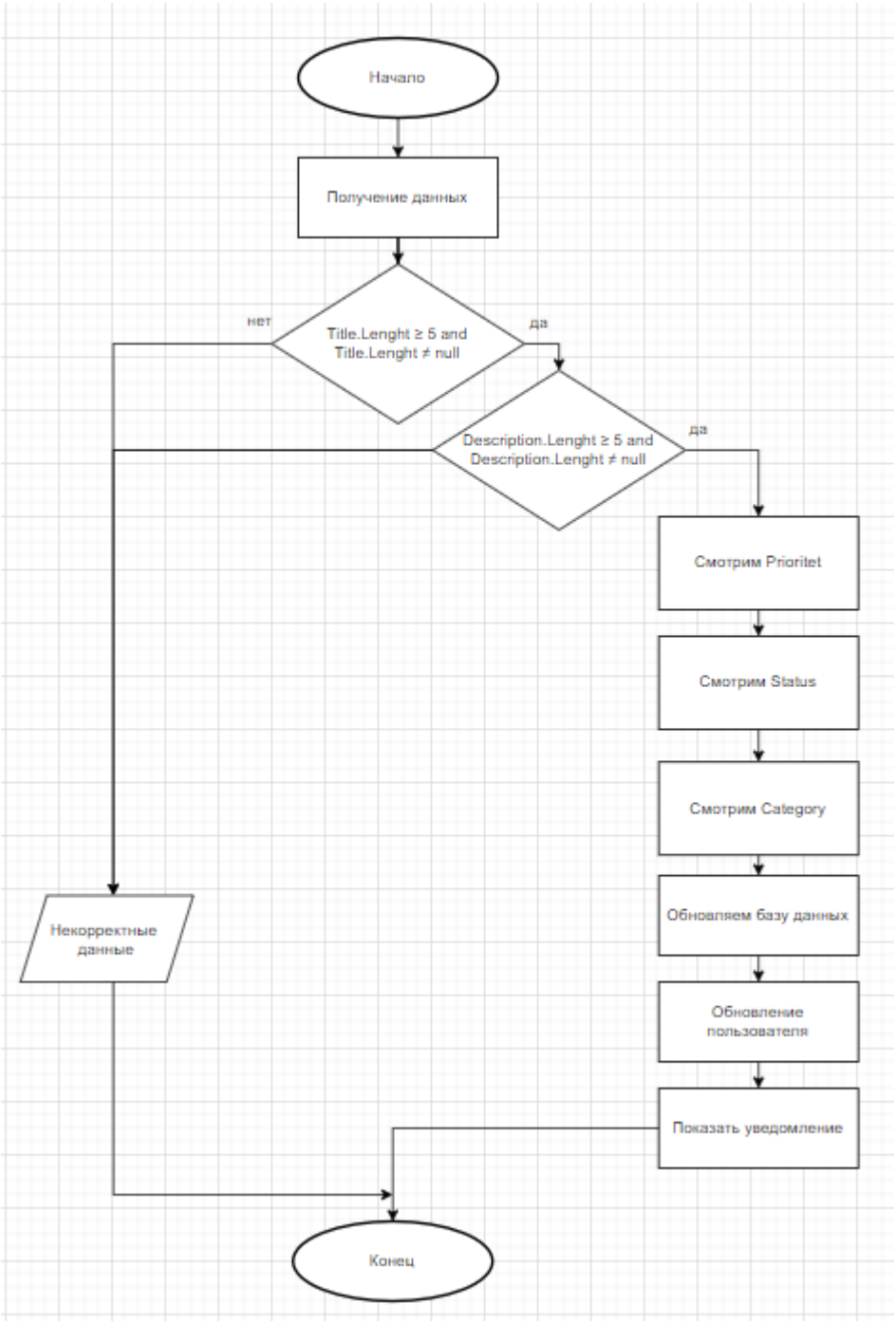

Рисунок Б.3 – Блок-схема алгоритма изменения задачи

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1-40 04 01	Пояснительная записка	28 с.
	Графические документы	
ГУИР 253505 014 СА	Алгоритм добавления задачи	Формат А4
	Алгоритм входа в аккаунт	
	Алгоритм изменения задачи	
	Схемы алгоритмов	

					БГУИР КП 1-40 04 01			
Изм.	Л.	№ докум.	Подп.	Дата	Программное средство для ведения задач		Лист	Листов
Разраб.		*Снежко М. А.*		05.06.24	Ведомость курсового проекта	Т	28	28
Пров.		*Тушинская Е.В.*		05.06.24		Кафедра Информатики гр. 253505		