МИНПРОСВЕЩЕНИЯ РОССИИ



ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ИНФОРМАТИКИ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра информационных систем и технологий

Профиль «Разработка мобильных приложений»

Курс III, Группа 2201

Маркъ Корнилюк Корнилюковичъ

**Трекер долгов**

КУРСОВАЯ РАБОТА

по дисциплине «Разработка мобильных приложений»

Научный руководитель:

Звягин К.Н.

Санкт-Петербург

2025

**СОДЕРЖАНИЕ**

Оглавление

[ЗАДАНИЕ 2](#_Toc196513867)

[Реферат 7](#_Toc196513868)

[ГЛАВА 1. Теоретические основы и постановка задачи 8](#_Toc196513869)

[1.1 Описание предметной области 8](#_Toc196513870)

[1.2 Постановка задачи проекта 9](#_Toc196513871)

[1.3 Используемые технологии 10](#_Toc196513872)

[1.4 Особенности разработки для ОС Аврора 10](#_Toc196513873)

[ГЛАВА 2. Архитектура и реализация приложения 11](#_Toc196513874)

[2.1 Архитектурные решения: паттерн Model-View-Delegate 11](#_Toc196513875)

[2.2.1 Логика управления данными 13](#_Toc196513876)

[2.2.2 Модель данных для представления в QML 13](#_Toc196513877)

[2.3 Реализация пользовательского интерфейса на QML 14](#_Toc196513878)

[2.3.1 Общая структура приложения 14](#_Toc196513879)

[2.3.2 Основные элементы интерфейса и их назначение 14](#_Toc196513880)

[2.3.3 Делегат для отображения элементов списка 15](#_Toc196513881)

[2.4 Интеграция C++ и QML 16](#_Toc196513882)

[2.5 Стилизация и локализация 17](#_Toc196513883)

[ГЛАВА 3. Результаты и выводы по итогам работы 18](#_Toc196513884)

[3.1 Достигнутые результаты 18](#_Toc196513885)

[3.2 Анализ хода работы и полученный опыт 18](#_Toc196513886)

[3.3 Возникшие трудности и пути их решения 19](#_Toc196513887)

[3.4 Перспективы развития проекта 20](#_Toc196513888)

[СПИСОК ЛИТЕРАТУРЫ 21](#_Toc196513889)

[Приложение 21](#_Toc196513890)

# ЗАДАНИЕ

Разработать мобильное приложение для операционной системы Аврора с использованием фреймворка Qt/QML и языка C++, реализующее ведение списка личных долгов, предоставление базовых функций по добавлению, редактированию, удалению и пометке долгов как оплаченных, и отображение списка долгов в пользовательском интерфейсе.

ВВЕДЕНИЕ

Управление личными финансами является важной частью повседневной жизни для многих людей. Нередко возникают ситуации, когда необходимо отслеживать займы или долги как в качестве кредитора, так и в качестве должника. Ведение такого учета в удобном формате, доступном "на ходу", может значительно упростить финансовое планирование и контроль.

Мобильные приложения для отслеживания долгов предоставляют удобный интерфейс для записи необходимой информации: от кого или кому должен пользователь, сумму, статус (оплачено/не оплачено). Наличие такого инструмента на мобильной платформе Aurora OS, которая активно развивается, представляется актуальной задачей.

Целью данной курсовой работы является разработка мобильного

приложения под управлением ОС Аврора, позволяющего пользователю вести

список своих личных долгов, добавлять новые записи, помечать существующие как оплаченные и удалять ненужные, а также отображать этот список в удобном пользовательском интерфейсе.

Для достижения поставленной цели в ходе работы решались следующие задачи:

* Изучение принципов разработки пользовательских интерфейсов для ОСАврора с использованием стандартных компонентов Silica. Было уделеновнимание элементам ApplicationWindow, Page, ListView, Button, Dialog иContextMenu, что соответствует лекционным материалам.
* Ознакомление с особенностями интеграции логики, реализованной наC++, с пользовательским интерфейсом на QML. Изучались механизмысвязывания C++ и QML объектов (setContextProperty, Q\_INVOKABLE,сигналы/слоты), о чем шла речь на лекциях.
* Проектирование архитектуры приложения с разделением логикиуправления данными и представления. Был применен паттерн Model-View-Delegate, позволяющий четко разграничить ответственность между частями приложения.
* Реализация на языке C++ функционала для хранения данных о долгах иуправления ими (добавление, изменение статуса, удаление).
* Разработка пользовательского интерфейса приложения на языке QML сиспользованием стандартных компонентов Silica для отображения спискадолгов и взаимодействия с пользователем.
* Реализация взаимодействия между C++ логикой и QML интерфейсом дляобработки действий пользователя и обновления отображаемых данных.
* Применение подходов к стилизации и локализации интерфейса,основанных на системных настройках ОС Аврора. Использовалиськомпонент Theme и функция qsTr(), рассмотренные в лекциях покосметике приложения.

Объектом исследования выступает процесс ведения личного учета долгов пользователем посредством разрабатываемого мобильного приложения.

Предметом исследования является разрабатываемое мобильное приложение и применяемые в его создании технологии и подходы.

В процессе работы использовались методы изучения технической

документации, проектирования программной архитектуры, реализации функционала на основе фреймворка Qt с применением языков C++ и QML, а

также тестирование разработанного приложения.

Практическая значимость работы заключается в создании простого и

удобного инструмента для личного финансового учета на мобильной

платформе Aurora OS. Кроме того, работа способствует углублению

понимания процессов разработки нативных мобильных приложений,

особенностей интеграции C++ и QML.

Курсовая работа состоит из введения, трех глав, заключения, списка

литературы и приложения с исходным кодом. В первой главе рассмотрены

теоретические аспекты и сформулирована задача. Во второй главе описаны

архитектурные решения и процесс реализации приложения. Третья глава

посвящена анализу полученных результатов и выводам.

# Реферат

Настоящая курсовая работа посвящена разработке мобильного

приложения "Трекер долгов" для операционной системы Аврора. Целью

работы являлось создание инструмента, способного помочь пользователю в

учете личных финансов, связанных с займами и долгами, предоставляя функции добавления, пометки оплаты и удаления записей.

В ходе работы были изучены принципы построения пользовательских

интерфейсов для мобильных приложений на базе фреймворка Qt с

использованием декларативного языка QML и стандартных компонентов

Silica. Рассмотрены механизмы интеграции C++ логики, отвечающей за

управление данными, с QML интерфейсом, позволяющие эффективно

связывать эти части приложения.

Разработанное приложение имеет двухзвенную архитектуру, где C++

часть отвечает за хранение и управление данными о долгах, а QML часть – за

отрисовку пользовательского интерфейса и взаимодействие с пользователем.

Для представления данных о долгах в списке использован паттерн Model-View Delegate, стандартный для разработки UI на Qt/QML. Приложение также учитывает системные настройки стилизации и поддерживает базовую

локализацию.

В результате работы было создано функционирующее мобильное

приложение для ОС Аврора, успешно демонстрирующее базовые функции

трекера долгов и взаимодействие между C++ и QML частями приложения.

Работа подтвердила возможность эффективного применения фреймворка Qt и связки C++/QML для решения задач разработки мобильных приложений с нативным внешним видом.

# ГЛАВА 1. Теоретические основы и постановка задачи

## 1.1 Описание предметной области

Управление личными финансами включает в себя не только контроль доходов и расходов, но и учет взаимных обязательств – долгов и займов. В

современном мире, где финансовые операции становятся всё более

динамичными, удобное и доступное средство для отслеживания этих

обязательств может стать полезным инструментом для поддержания

финансовой дисциплины и избежания недоразумений. Приложение "Трекер

долгов" предназначено для ведения структурированного списка таких

обязательств, позволяя пользователю фиксировать ключевую информацию

(кто, кому, сколько, статус) и оперативно обновлять ее.

## 1.2 Постановка задачи проекта

Задача проекта заключается в разработке мобильного приложения для

операционной системы Аврора, способного выполнять функции трекера долгов. В соответствии с заданием, приложение должно:

* Позволять добавлять новые записи о долгах (например, имя контрагента,сумма долга, тип - "мне должны" или "я должен").
* Предоставлять возможность помечать существующие долги как оплаченные.
* Реализовывать функцию удаления записей о долгах.
* Отображать список всех долгов в пользовательском интерфейсе.
* Предоставлять базовый пользовательский интерфейс для взаимодействия с приложением (кнопки добавления, контекстное меню для действий над элементами списка).
* Быть разработанным с использованием фреймворка Qt/QML и языка C++.

## 1.3 Используемые технологии

Для реализации поставленной задачи был выбран следующий стек технологий, соответствующий требованиям и возможностям платформы ОС Аврора:

* **Операционная система Аврора (Aurora OS):** Целевая мобильная платформа для развертывания приложения. Выбор обусловлен требованиями к курсовой работе.
* **Фреймворк Qt:** Мощный кроссплатформенный фреймворк, предоставляющий обширные возможности для разработки как пользовательских интерфейсов, так и логики приложений. Является стандартом разработки для Aurora OS.
* **Язык QML:** Декларативный язык, входящий в состав Qt, оптимально подходящий для быстрого создания динамичных и адаптивных пользовательских интерфейсов мобильных приложений.
* **Язык C++:** Используется для реализации логики управления данными,хранения списка долгов и выполнения операций над ними.
* **Компоненты Silica:** Набор UI-компонентов из состава Qt for MCUs / QtQuick Controls 2, стилизованных под внешний вид ОС Аврора

## 1.4 Особенности разработки для ОС Аврора

Разработка приложений для ОС Аврора на базе Qt имеет ряд специфических особенностей, которые были учтены в проекте:

* **Использование Silica:** Применение компонентов Silica (таких как ApplicationWindow, Page, Button, ListView, Dialog, ContextMenu и др.)является стандартом для создания UI, соответствующего гайд лайнам Авроры. Как было показано в лекциях по UI-компонентам Silica, эти элементы обеспечивают правильное отображение и взаимодействие на устройствах под управлением Аврора.
* **Интеграция C++ и QML:** Разделение логики (C++) и интерфейса(QML) является рекомендуемым подходом. Механизмы экспорта C++объектов в QML (setContextProperty) и вызова методов C++ из QML(Q\_INVOKABLE) активно использовались для связи между бэкендом и фронтендом приложения, как это было подробно рассмотрено в лекциях по интеграции C++ и QML.
* **Стилизация и локализация:** Для обеспечения единообразного внешнего вида и возможности перевода интерфейса использовались системные возможности Qt и ОС Аврора. Компонент Theme предоставлял доступ к стандартным цветам, размерам и шрифтам, а функция qsTr() маркировала строки для последующей локализации. Была учтена работа с вырезами и закругленными углами через SafeZoneRect.insets.

# ГЛАВА 2. Архитектура и реализация приложения

## 2.1 Архитектурные решения: паттерн Model-View-Delegate

Для организации взаимодействия между данными, логикой их обработки и пользовательским интерфейсом в приложении был применен паттерн Model-View-Delegate (MVD). Этот паттерн является стандартным для разработки UI с использованием Qt/QML и позволяет четко разделить ответственность между компонентами, повышая гибкость и поддерживаемость кода. Как было рассмотрено в лекциях по Моделям и View, MVD включает:

* Модель (Model): Хранит и предоставляет данные для отображения. В данном проекте роль модели данных о коммитах авторов выполняет C++ класс, наследующий QAbstractListModel.
* Представление (View): Отображает данные из модели. В QML эту роль выполняет компонент ListView из состава Silica.
* Делегат (Delegate): Определяет, как выглядит каждый отдельный элемент данных из модели в представлении. В QML делегат описывается как отдельный визуальный компонент (например, Rectangle или Text) и связывается с данными из модели с помощью ролей.

Выбор MVD позволил эффективно управлять списком авторов и их статистикой: C++ обрабатывает получение и агрегацию данных, формируя модель, а QML просто "привязывается" к этой модели через ListView и его делегат для визуализации каждого элемента списка.

2.2 Реализация логики на языке C++

Основная бизнес-логика приложения, связанная с получением, обработкой и анализом данных, реализована на языке C++ с использованием библиотек Qt. Это обеспечивает необходимую производительность для сетевых операций и вычислений.

### 2.2.1 Логика управления данными

Для хранения и управления записями о долгах был создан C++ класс DebtModel, наследующий от QAbstractListModel. Этот класс предоставляет интерфейс для доступа к данным из QML-представлений и реализует логику добавления, удаления и изменения записей о долгах.

Внутри DebtModel данные хранятся, например, в QList структур (DebtItem), каждая из которых содержит поля для имени контрагента, суммы, типа долга и статуса оплаты. Методы rowCount(), data(), roleNames() реализованы для предоставления QML информации о количестве записей и данных для каждой записи по определенным ролям (например, "name", "amount", "isPaid").

Функции добавления (addDebt()), пометки оплаты (markAsPaid()) и удаления (removeDebt()) реализованы как Q\_INVOKABLE методы в классе DebtModel. Эти методы изменяют внутреннюю структуру данных (QList), добавляя, изменяя или удаляя элементы, и после этого эмитируют соответствующие сигналы QAbstractListModel (такие как rowsInserted(), dataChanged(), rowsRemoved()) для уведомления QML интерфейса об изменениях.

### 2.2.2 Модель данных для представления в QML

Обработанные и отсортированные данные о коммитах авторов должны быть представлены в QML. Для этого создан C++ класс AuthorModel, наследующий QAbstractListModel. Этот класс хранит список пар "автор коммиты" и реализует методы, необходимые для предоставления данных QML-представлениям (ListView).

Методы rowCount() возвращает количество элементов в списке, а data() предоставляет данные для конкретного элемента (по индексу и роли). Роли определяются в C++ классе (например, AuthorNameRole, CommitCountRole) и сопоставляются с именами строк, которые будут использоваться в QML (например, "authorName", "commitCount") через метод roleNames(). При завершении анализа данных в CommitAnalyzer, он обновляет данные в AuthorModel, и модель уведомляет QML представление об изменениях, вызывая его обновление.

## 2.3 Реализация пользовательского интерфейса на QML

Визуальная часть приложения разработана на языке QML с использованием компонентов Silica для обеспечения нативного внешнего вида и удобства использования на ОС Аврора.

### 2.3.1 Общая структура приложения

Основным контейнером приложения является компонент ApplicationWindow, как это принято для приложений ОС Аврора (согласно лекции по ApplicationWindow). Все видимое содержимое приложения размещается на компоненте Page, который установлен как initialPage главного окна. Такой подход соответствует стандартной структуре одностраничных или многостраничных (с использованием PageStack) приложений на Аврора.

### 2.3.2 Основные элементы интерфейса и их назначение

На главной странице приложения (Page) размещены следующие ключевые элементы:

* PageHeader: Стандартный заголовок страницы из Silica, отображающий название приложения (например, "Трекер долгов").
* ListView: Основной компонент для отображения списка долгов. Он привязан к модели данных DebtModel, предоставляемой C++ частью.
* Button: Кнопка для добавления новой записи о долге (например, "+"). По нажатию на эту кнопку происходит открытие диалогового окна ввода.

### 2.3.3 Делегат для отображения элементов списка

Для ListView определен компонент-делегат (QML Component). Делегат описывает внешний вид одной строки в списке долгов. В данном проекте делегат состоит из простых визуальных элементов, таких как Rectangle для фона элемента списка и Text для отображения информации о долге (например, имени контрагента, суммы). Внешний вид может меняться в зависимости от статуса долга (например, зачеркивание текста для оплаченного долга).

Данные для каждого элемента списка (имя, сумма, статус оплаты) доступны в делегате через специальные свойства, соответствующие ролям, определенным в C++ модели (DebtModel) (например, "name", "amount", "isPaid"). QML автоматически создает экземпляр делегата для каждого элемента в модели и привязывает данные из модели к свойствам делегата, обеспечивая отображение всего списка. Для действий над элементами списка (пометка как оплачено, удаление) используется ContextMenu внутри делегата.

## 2.4 Интеграция C++ и QML

Ключевым аспектом архитектуры является бесшовная интеграция между C++ логикой и QML интерфейсом. Это достигается посредством механизмов, предоставленных фреймворком Qt.

В точке входа приложения на C++ (main функции) создается экземпляр класса DebtModel, реализующего модель данных. Этот объект становится доступным в контексте QML с помощью метода QQmlContext::setContextProperty(). Это позволяет QML "видеть" и обращаться к этой C++ модели по заданному имени.

Взаимодействие между UI и логикой осуществляется через систему сигналов и слотов Qt. В QML, сигнал onClicked кнопки добавления долга связывается с функцией, открывающей диалоговое окно для ввода данных. Введенные данные из диалогового окна передаются в C++ модель путем вызова метода addDebt(), помеченного как Q\_INVOKABLE в классе DebtModel.

Для действий над элементами списка (markAsPaid(), removeDebt()), вызываемых из контекстного меню, также используются Q\_INVOKABLE методы класса DebtModel, вызываемые из обработчиков сигналов onClicked элементов ContextMenu в QML делегате.

Связывание модели DebtModel со свойством model компонента ListView в QML обеспечивает автоматическое обновление списка при изменении данных в C++ модели благодаря сигналам QAbstractListModel. Привязки свойств в делегате (text: modelData.name, text: modelData.amount, color: modelData.color) гарантируют, что отображаемая информация для каждого элемента списка всегда актуальна.

## 2.5 Стилизация и локализация

Для придания приложению единообразного внешнего вида, соответствующего стандартам ОС Аврора, и для обеспечения возможности перевода пользовательского интерфейса, были использованы компоненты Theme и функция qsTr().

Компонент Theme из состава Silica предоставляет доступ к системной палитре цветов (Theme.primaryColor, Theme.highlightColor и др.), стандартным размерам элементов (Theme.itemSizeMedium, Theme.itemSizeLarge) и шрифтов (Theme.fontSizeSmall, Theme.fontSizeMedium). Использование этих свойств вместо "жестко закодированных" значений позволяет приложению адаптироваться к текущей системной теме и настройкам, как было показано в лекциях по стилизации.

Все текстовые строки, видимые пользователю в QML-интерфейсе (например, название приложения, текст кнопок, пункты меню), были обернуты функцией qsTr(). Это стандартный механизм Qt для маркировки строк, предназначенных для перевода. Использование qsTr() позволяет в дальнейшем применять инструменты локализации Qt (например, Qt Linguist, рассмотренный в лекциях) для создания файлов переводов на другие языки без изменения исходного кода приложения. При запуске приложение автоматически загрузит перевод, соответствующий текущим языковым настройкам системы, если он доступен. Также была учтена работа с SafeZoneRect для корректного отображения контента в пределах безопасных зон экрана, как демонстрировалось на лекциях.

# ГЛАВА 3. Результаты и выводы по итогам работы

## 3.1 Достигнутые результаты

В результате выполнения курсовой работы было разработано мобильное приложение "Трекер долгов" для операционной системы Аврора, успешно реализующее поставленную задачу. Приложение способно вести список долгов, позволяя пользователю добавлять новые записи, помечать существующие как оплаченные и удалять их. Список долгов отображается в пользовательском интерфейсе и автоматически обновляется при внесении изменений.

Разработанная архитектура с разделением логики на C++ и представления на QML, основанная на паттерне Model-View-Delegate, продемонстрировала свою эффективность для данного типа задачи. Интеграция C++ и QML посредством стандартных механизмов Qt (сигналы/слоты, Q\_INVOKABLE, setContextProperty, QAbstractListModel) была успешно реализована и обеспечивает необходимый уровень взаимодействия между компонентами.

Пользовательский интерфейс, разработанный с использованием компонентов Silica, имеет нативный внешний вид и базовую функциональность для управления списком долгов.

## 3.2 Анализ хода работы и полученный опыт

Выполнение проекта позволило получить ценный опыт в области разработки нативных мобильных приложений для ОС Аврора с использованием фреймворка Qt. Работа с предоставленными лекционными материалами оказалась крайне полезной и послужила основной базой для понимания ключевых концепций и специфических подходов платформы.

Применение паттерна Model-View-Delegate (MVD), подробно рассмотренного в лекциях по Моделям и View, значительно упростило организацию данных и их представление в UI. Понимание ролей Модели, Представления и Делегата позволило структурировать QML и C++ код логичным образом.

Изученные в лекциях по интеграции C++ и QML механизмы экспорта C++ объектов и взаимодействия через сигналы и слоты оказались центральными для реализации связи между логикой управления данными и пользовательским интерфейсом. Практическое применение setContextProperty и Q\_INVOKABLE укрепило понимание этих важных концепций. Реализация модели данных, наследующей QAbstractListModel, дала глубокое понимание принципов работы с QML представлениями.

Работа с компонентами Silica и их свойствами, а также применение Theme для стилизации и qsTr для локализации (как показано в лекциях по UI и косметике), дало представление о том, как создавать приложения, соответствующие стандартам и внешнему виду ОС Аврора.

В целом, ход работы соответствовал запланированным этапам.

## 3.3 Возникшие трудности и пути их решения

В процессе работы над проектом возникли некоторые трудности, характерные для разработки, связанной с сетевым взаимодействием и интеграцией различных частей приложения:

* **Взаимодействие C++ и QML:** Несмотря на наличие стандартных механизмов, отладка ошибок на границе C++/QML может быть неочевидной. Решение заключалось в систематическом использовании отладочных сообщений (console.log в QML, qDebug в C++) и инструментов отладки Qt Creator для выявления и устранения проблем с передачей данных и вызовами методов.
* **Реализация логики модели данных:** Правильная реализация методов QAbstractListModel (rowCount, data, roleNames) и сигналов (rowsInserted, dataChanged, rowsRemoved) требовала тщательного изучения документации для обеспечения корректного и эффективного обновления QML представления.
* **UI/UX на мобильной платформе:** Адаптация интерфейса к особенностям мобильного устройства, использование стандартных компонентов Silica и работа с SafeZoneRect для корректного отображения контента в пределах безопасных зон требовали итеративного подхода и тестирования на реальном устройстве или эмуляторе.

## 3.4 Перспективы развития проекта

Разработанное приложение может быть расширено и улучшено в нескольких направлениях:

* **Персистентное хранение данных:** Реализовать сохранение списка долгов между сеансами работы приложения (например, с использованием QSettings для небольшого объема данных или SQLite для более сложных случаев).
* **Сортировка и фильтрация:** Добавить функционал сортировки списка долгов (например, по имени, сумме, дате) и фильтрации (например,только не оплаченные долги). Для этого можно использовать QSortFilterProxyModel.
* **Более детальная статистика:** Добавить расчет и отображение своднойстатистики, например, общей суммы долгов, общей суммыоплаченных долгов.
* **Категории долгов:** Ввести разделение долгов по категориям (например,личные, деловые).
* **Напоминания:** Реализовать систему напоминаний о приближающихся датах оплаты (если добавлять поле с датой).

# СПИСОК ЛИТЕРАТУРЫ

1. Лекционные материалы по дисциплине «Разработка мобильных приложений», преподаватель Звягин К.Н., 2025 г.
2. Официальная документация Qt. [[ссылка]](https://developer.auroraos.ru/doc/5.1.4/platform/architecture/qt)

# Приложение

Иллюстрации работы приложения:

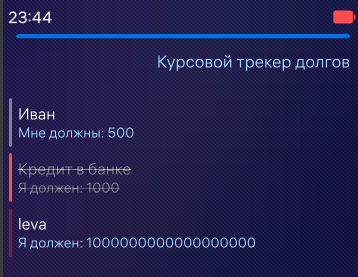


Рис. П.1.1. Интерфейс приложения после анализа репозитория "eeegoloauq/sber-split-app".

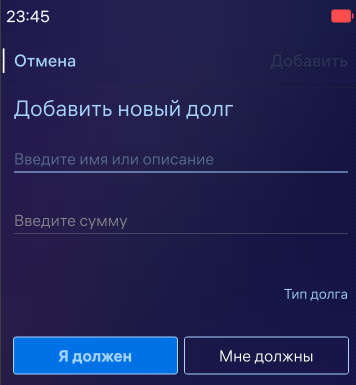


Рис. П.1.2. Результаты анализа более активного репозитория "hydralauncher/hydra".

