МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра компьютерных технологий и систем

ВИДЕВИЧ Александр Викторович

АВТОМАТИЗАЦИЯ ФИНАНСОВЫХ РАСЧЕТОВ, ПРЕДПОЛАГАЮЩИХ НАЛИЧИЕ ПОТОКОВ ПЛАТЕЖЕЙ

Дипломная работа

Научный руководитель:
кандидат физико-
математических наук, доцент
Е.Г. Красногир

Допущена к защите
«____»_____20___г.
Заведующий кафедрой компьютерных технологий и систем
доктор педагогических наук, кандидат физико-
математических наук, профессор В.В. Казачёнок

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Структура и объём дипломной работы

72 страницы, 22 рисунка, 2 таблицы, 7 приложений, 15 источников

Ключевые слова: ПОТОКИ ПЛАТЕЖЕЙ, АВТОМАТИЗАЦИЯ РАСЧЁТОВ, ФИНАНСОВАЯ ГРАМОТНОСТЬ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ШАБЛОН ПРОЕКТИРОВАНИЯ MVC.

Текст реферата

Объект исследования – потоки платежей, их разновидности и возможности использования.

Цель исследования – разработка приложения, позволяющего автоматизировать расчёты, связанные с потоками платежей, основываясь на параметрах, предоставленных пользователем, а также производить сравнения похожих потоков платежей для более глубокого понимания их разницы.

Методы исследования – изучение литературы, посвящённой потокам платежей, анализ существующих решений для автоматического расчёта потоков платежей, изучение существующих технологий и практик, применяемых при создании мобильных приложений, проектирование и создание приложения.

Полученные результаты – приложение, позволяющее проводить расчёты в рамках различных потоков платежей, предоставляющее возможность ознакомиться с используемыми формулами, их применением и возможными изменениями параметров и потоков платежей для получения оптимальных результатов.

Область возможного практического применения – использование приложения в качестве образовательной платформы для повышения финансовой грамотности пользователей, а также как инструмента для проведения расчётов в сфере потоков платежей.

РЭФЕРАТ

Структура і аб’ём дыпломнай працы

72 старонкі, 22 малюнка, 2 табліцы, 7 дадаткаў, 15 крыніц

Ключавыя словы: ПАТОКІ ПЛАЦЯЖОЎ, АЎТАМАТЫЗАЦЫЯ РАЗЛІКАЎ, ФІНАНСАВАЯ ГРАМАТНАСЦЬ, МАБІЛЬНЫ ДАДАТАК, ШАБЛОН ПРАЕКТАВАННЯ MVC.

Змест працы

Аб’ект даследавання – патокі плацяжоў, іх разнавіднасці і магчымасці выкарыстання.

Мэта даследавання – распрацоўка дадатка, які дазваляе аўтаматызаваць вылічэнні, звязаныя з патокамі плацяжоў, на аснове параметраў, прадастаўленых карыстальнікам, а таксама параўноўваць падобныя патокі плацяжоў для глыбейшага разумення іх адрозненняў.

Метады даследавання – вывучэнне літаратуры, прысвечанай патокам плацяжоў, аналіз існуючых рашэнняў для аўтаматычнага разліку патокаў плацяжоў, вывучэнне сучасных тэхналогій і практык, што выкарыстоўваюцца пры стварэнні мабільных прылад, праектаванне і распрацоўка дадатка.

Атрыманыя вынікі – мабільны дадатак, які дазваляе ажыццяўляць разлікі ў рамках розных патокаў плацяжоў, прадастаўляе магчымасць азнаёміцца з ужытымі формуламі, іх выкарыстаннем і магчымасцю змянення параметраў і патокаў плацяжоў для атрымання аптымальных вынікаў.

Вобласць магчымага практычнага прымянення – выкарыстанне дадатку ў якасці адукацыйнай платформы для павышэння фінансавай граматнасці карыстальнікаў, а таксама як інструмента для разлікаў у сферы патокаў плацяжоў.

SUMMARY

Structure and scope of the diploma work
72 pages, 22 figures, 2 tables, 7 appendixes, 15 references
Keywords: PAYMENT FLOWS, AUTOMATION OF CALCULATIONS, FINANCIAL LITERACY, MOBILE APPLICATION, MVC DESIGN PATTERN.
Summary text

The object of the research – payment flows, their types, and possibilities for their use.

The aim of the research – development of the application that automates calculations related to payment flows based on user-provided parameters and enables the comparison of similar payment flows for a deeper understanding of their differences.

Research methods – study of literature on payment flows, analysis of existing solutions for automatic payment flow calculations, examination of current technologies and practices in mobile application development, and the design and creation of the application.

The results of the work – the application that performs calculations within various payment flows, provides access to the formulas used, explains their application, and allows for modifications of parameters and payment flows to achieve optimal results.

Recommendations on the usage – educational platform to enhance users financial literacy and a tool for performing payment flow calculations in financial operations.

ВВЕДЕНИЕ

В условиях цифровой экономики и постоянного взаимодействия с финансовыми институтами потоки платежей играют ключевую роль в жизни каждого человека. Выплаты по кредитам, начисление процентов по депозитам, инвестиционные вклады и другие финансовые операции формируют личный бюджет и влияют на уровень финансовой стабильности. Управление этими потоками позволяет не только планировать текущие расходы, но и ставить долгосрочные финансовые цели. Однако сложные расчёты, необходимые для оценки обязательств, часто затруднительны без специальных инструментов.

Актуальность темы обусловлена постоянным ростом потребностей на услуги, предполагающие наличие потоков платежей и необходимостью создания удобных и доступных инструментов, позволяющих пользователям самостоятельно выполнять расчёты, связанные с их финансовыми обязательствами. Без автоматизированных расчётных средств оценка, например, полной стоимости кредита или расчёт доходности по депозиту может быть затруднена, особенно с учётом различных процентных схем, комиссий и сроков выплат. В условиях быстрого распространения цифровых финансовых продуктов и повышения финансовой осведомлённости населения, обеспечение лёгкого доступа к точным расчётам становится важным аспектом для поддержки финансовой самостоятельности и грамотности.

Целью дипломной работы является разработка приложения, позволяющего автоматизировать расчёты, связанные с потоками платежей, основываясь на параметрах, предоставленных пользователем, а также производить сравнения похожих потоков платежей для более глубокого понимания их разницы.

Для достижения цели были поставлены и решены следующие задачи:

1. Определить основные виды потоков платежей.
2. Определить существующие методы, применяемые для расчёта потоков платежей.
3. Определить функциональные требования к приложению.
4. Реализовать приложение, учитывая удобство пользовательского интерфейса и соответствие функционала поставленным требованиям.
5. Оценить потенциальные преимущества использования разработанного приложения для пользователей относительно аналогов, а также его роль в повышении финансовой грамотности.

ГЛАВА 1
ПОТОКИ ПЛАТЕЖЕЙ

1.1 Понятие потока платежей, классификация потоков платежей, основные параметры потоков платежей

Современные операции в области финансов и банковского дела часто включают в себя регулярные серии транзакций вместо единичных платежей. Например, студенты могут погашать кредит за обучение ежемесячно в течение года, а компании могут платить за услуги облачного хранения данных ежегодно. Такие последовательности, или ряды платежей называют потоком платежей, а отдельный элемент этого ряда – членом потока [8].

Выделяют следующую классификацию потоков платежей [7]:

1. По регулярности платежа:
   - 1.1. Регулярные. Регулярные платежи совершаются через равные интервалы. Регулярный поток, все члены которого положительны, называют финансовой рентой (или просто рентой). Параметры ренты: член ренты (размер отдельного платежа), период ренты (временной интервал между двумя последовательными платежами), срок ренты (время от начала первого периода ренты до конца последнего), процентная ставка.
   - 1.2. Нерегулярные.
2. По количеству выплат членов ренты в течение года:
   - 2.1. Годовые (выплата раз в году).
   - 2.2. P-срочные (выплата p раз в год).
   - 2.3. Непрерывные.
3. По числу начислений процентов в течение года:
   - 3.1. Ренты с ежегодным начислением.
   - 3.2. Ренты с начислением m раз в году или раз в k лет.
4. По величине членов потока:
   - 4.1. Постоянные.
   - 4.2. Переменные.
5. По вероятности выплат:
   - 5.1. Верные (подлежат безусловной выплате, число членов заранее известно).
   - 5.2. Условные (выплата ставится в зависимости от наступления некоторого случайного события, число членов заранее неизвестно).

6. По количеству членов ренты:

6.1. Ренты с конечным числом членов.

6.2. Бесконечные (или вечные) ренты.

7. По соотношению начала срока ренты и какого-либо момента времени, упреждающего начало ренты:

7.1. Немедленные.

7.2. Отложенные (или отсроченные).

8. По моменту выплат платежей в пределах периода ренты:

8.1. Обыкновенные (или постнумерандо). Платежи осуществляются конце периодов.

8.2. Пренумерандо. Платежи осуществляются в начале периодов.

К основным типам расчётов потоков платежей относятся [7]:

1. Наращенная сумма (S – для рент постнумерандо, $\ddot{S}$ –для рент пренумерандо) – наращенная сумма может представлять собой общую сумму накопленной задолженности к концу срока, итоговый объем инвестиций, накопленный денежный резерв и т. д. Параметр рассчитывается следующим образом:

$$S = \sum_t R_t(1+i)^{n-n_t}, \ddot{S} = S(1+i), \tag{1.1}$$

где R_t – ряд платежей,

n – общий срок выплат (в годах),

n_t время после некоторого начального момента времени,

i – сложная процентная ставка (процентная ставка – относительная величина дохода за фиксированный отрезок времени). Сложная процентная ставка – процентная ставка, при применении которой проценты начисляются на проценты).

2. Современная стоимость потока платежей (A – для рент постнумерандо, $\ddot{A}$ – для рент пренумерандо) – сумма всех членов потока, дисконтированных (приведённых) на начало срока ренты или некоторый упреждающий момент времени. Современная стоимость потока платежей характеризует приведённые к началу осуществления проекта инвестиционные затраты, суммарный капитализированный доход или чистую приведённую прибыль от реализации проекта и т. п. Параметр рассчитывается следующим образом:

$$A = \sum_t R_t v^{n_t}, \ddot{A} = A(1+i), \tag{1.2}$$

где R_t – ряд платежей,
ν – дисконтный множитель по ставке i,
n_t время после некоторого начального момента времени.

$$\nu = \frac{1}{1+i}. \tag{1.3}$$

Между величинами A и S существует функциональная зависимость [7]:

$$A\,(1+i)^n = S. \tag{1.4}$$

Формулы (1.1) и (1.2) представляют собой прямой метод расчёта наращенной суммы и современной стоимости потока платежей соответственно. Этим методом можно найти значения параметров S и A для любого потока платежей, однако для частных случаев рент существуют более компактные формулы [7]:

1. Годовая рента постнумерандо: в течение n лет в банк в конце каждого года вносится по R рублей. На взносы начисляются сложные проценты по ставке i% годовых, получим следующие члены ренты:

$$R(1+i)^{n-1}, R(1+i)^{n-2} \ldots R(1+i), R. \tag{1.5}$$

Ряд представляет собой геометрическую прогрессию с первым членом R и шагом $\frac{1}{1+i}$, поэтому для нахождения наращенной суммы можно воспользоваться формулой суммы геометрической прогрессии. Применив также формулу (1.1), получим.

$$S = R\frac{(1+i)^n - 1}{(1+i) - 1} = R\frac{(1+i)^n - 1}{i} = Rs_{n;i}. \tag{1.6}$$

Члены ряда дисконтированных величин

$$\mathrm{R}\nu, \mathrm{R}\nu^2 \ldots \mathrm{R}\nu^n \tag{1.7}$$

также представляют собой геометрическую прогрессию. Применив вместе с этим формулу (1.2), получим.

$$A = R\nu\frac{\nu^n - 1}{\nu - 1} = R\frac{1-\nu^n}{i} = R\frac{1-(1+i)^{-n}}{i} = Ra_{n;i}. \tag{1.8}$$

При увеличении срока ренты величина $a_{n;i}$ стремится к пределу [7]:

$$a_{n;i} = \frac{\lim\limits_{n\to\infty}(1-(1+i)^{-n})}{i} = \frac{1}{i}. \tag{1.9}$$

2. Годовая рента постнумерандо, начисление процентов m раз в году: аналогично случаю, когда проценты начисляются раз в году, получим члены ренты

$$R\left(1+\frac{j}{m}\right)^{(n-1)m}, R\left(1+\frac{j}{m}\right)^{(n-2)m}, \dots, R\left(1+\frac{j}{m}\right)^{m}, R \tag{1.10}$$

и ряд дисконтированных величин

$$\mathrm{R}v, \mathrm{R}v^2, \dots, \mathrm{R}v^n \tag{1.11}$$

с дисконтным множителем

$$v = \left(1+\tfrac{j}{m}\right)^{-m}. \tag{1.12}$$

Наращенная сумма и современная стоимость будут равны соответственно

$$S = R\frac{\left(1+\frac{j}{m}\right)^{nm}-1}{\left(1+\frac{j}{m}\right)^{m}-1} = Rs_{nm;\frac{j}{m}} \tag{1.13}$$

и

$$\mathrm{A} = R\frac{1-\left(1+\frac{j}{m}\right)^{-nm}}{\left(1+\frac{j}{m}\right)^{m}-1} = a_{nm;\frac{j}{m}}, \tag{1.14}$$

где j – номинальная ставка процентов

$$i = \frac{j}{m}. \tag{1.15}$$

3. Рента p-срочная ($m = 1$): пусть рента выплачивается p раз в году равными суммами, процент начисляется раз в конце года, тогда

$$S = R\frac{(1+i)^n - 1}{p\left[(1+i)^{\frac{1}{p}} - 1\right]} = Rs_{n;j}^{(p)}, \tag{1.16}$$

$$\mathrm{A} = R\frac{1-(1+i)^{-n}}{p\left[(1+i)^{\frac{1}{p}} - 1\right]} = Ra_{n;j}^{(p)}. \tag{1.17}$$

4. Рента p-срочная ($p = m$), число выплат в году равно числу начислений процентов: Используя формулы (1.6) и (1.8) с учётом того, что количество выплат равно mp, член ренты равен R/p и $(p = m)$, получим

$$S = R\frac{\left(1+\frac{j}{m}\right)^{mn} - 1}{j}, \tag{1.18}$$

$$\mathrm{A} = R\frac{1-\left(1+\frac{j}{m}\right)^{-mn}}{j}. \tag{1.19}$$

5. Рента p-срочная ($p \neq m$): в данном случае получим геометрическую прогрессию членов ренты с первым членом $\frac{R}{p}$ и шагом $\left(1+\frac{j}{m}\right)^{\frac{m}{p}}$. Наращенная сумма будет равна

$$S = R\frac{\left(1+\frac{j}{m}\right)^{mn} - 1}{p\left[\left(1+\frac{j}{m}\right)^{\frac{m}{p}} - 1\right]}. \tag{1.20}$$

Также получим и ряд дисконтированных величин

$$\mathrm{R}v, \mathrm{R}v^2 \ldots \mathrm{R}v^n \tag{1.21}$$

с дисконтным множителем

$$v = \left(1+\frac{j}{m}\right)^{-\frac{m}{p}}. \tag{1.22}$$

Современная стоимость потока платежей будет равна

$$\mathrm{A} = R\frac{1-\left(1+\frac{j}{m}\right)^{-mn}}{p\left[\left(1+\frac{j}{m}\right)^{\frac{m}{p}}-1\right]}. \tag{1.23}$$

Чем больше m, тем меньше промежуток между моментами начисления процентов [7], тогда при $m \to \infty$, используя формулу (1.4), получим

$$S = A * \lim_{m\to\infty}\left(1+\frac{j}{n}\right)^{mn} = Ae^{jn} = Ae^{\delta n}, \tag{1.24}$$

где δ – сила роста.

6. Непрерывное начисление процентов: рассмотрим ежегодные платежи постнумерандо. Используя выражение (1.24), получим ряд

$$R, Re^{\delta}, Re^{2\delta}, \dots, Re^{(n-1)\delta} \tag{1.25}$$

и процентную ставку

$$i = e^{\delta} - 1. \tag{1.26}$$

Наращенная сумма и современная стоимость потока платежей равны соответственно

$$S = R\frac{e^{\delta n}-1}{e^{\delta}-1} = Rs_{n;\delta} \tag{1.27}$$

и

$$\mathrm{A} = R\frac{1-e^{-\delta n}}{e^{\delta}-1} = Ra_{n;\delta}. \tag{1.28}$$

В случае p-срочной ренты получим

$$S = R\frac{e^{\delta n}-1}{p(e^{\delta/p}-1)} = Rs^{(p)}_{n;\delta}, \tag{1.29}$$

$$\mathrm{A} = R\frac{1-e^{-\delta n}}{p(e^{\delta}-1)} = Ra^{(p)}_{n;\delta}. \tag{1.30}$$

7. Отложенная рента: сдвиг во времени никак не отражается на величине наращенной суммы, изменяется только современная стоимость ренты

$$ {}_tA = Av^t = Ra_{n;i}v^t, \tag{1.31} $$

где ${}_tA$ – современная стоимость отложенной на t лет ренты.

8. Вечная рента: используется, когда срок потока платежей очень большой и конкретно не оговаривается. Применив формулу (1.9), получим

$$ A_\infty = \frac{R}{i}. \tag{1.32} $$

9. Переменная процентная ставка: пусть, например, для постоянной ренты постнумерандо со сроком 10 лет предусматриваются два уровня процентной ставки, применяемые по пятилетиям, тогда

$$ S = Rs_{5;i_1}(1+i_2)^5 + Rs_{5;i_2}, \tag{1.33} $$

$$ A = Ra_{5;i_1} + Ra_{5;i_2}(1+i_1)^5. \tag{1.34} $$

В случае, когда срок разбит на k частей, получим

$$ \mathrm{S} = \mathrm{R}\sum_{\mathrm{m}=1}^{\mathrm{k}}\left(\mathrm{s}_{\mathrm{n_m};\mathrm{i_m}} \cdot \prod_{\mathrm{p}=\mathrm{m}+1}^{\mathrm{k}}\left(1+\mathrm{i_p}\right)^{\mathrm{n_p}}\right), \tag{1.35} $$

$$ A = R\sum_{m=1}^{k}\left(a_{n_m;i_m} \cdot \prod_{p=1}^{m-1}\left(1+i_p\right)^{-n_p}\right), \tag{1.36} $$

где n_m – длительность m-го периода в годах,
$\mathrm{i_m}$ – процентная ставка m-го периода.

10. Ренты с постоянным абсолютным изменением во времени: пусть члены ренты образуют последовательность

$$ R,\ R+a, R+2a, \ldots, R+(n-1)a, \tag{1.37} $$

получим

$$ S = \left(R + \frac{a}{i}\right)s_{n;i} - \frac{na}{i}, \tag{1.38} $$

$$\mathrm{A} = \left(\mathrm{R} + \frac{a}{i}\right) a_{n;i} - \frac{nav^n}{i}. \tag{1.39}$$

11. Переменная p -срочная рента с постоянным абсолютным приростом: пусть R – базовая величина разовой выплаты, a – годовой прирост выплат, тогда

$$S = \sum_{t=1}^{pn} \left[R + \frac{a(t-1)}{p}\right](1+i)^{n-\frac{t}{p}}, \tag{1.40}$$

$$\mathrm{A} = \sum_{t=1}^{pn} \left[R + \frac{at}{p}\right] v^{\frac{t}{p}}. \tag{1.41}$$

12. Рента p-срочная с постоянными относительными изменениями членов: в этом случае последовательность платежей представляет собой геометрическую прогрессию

$$R,\ Rq, \dots, Rq^{np-1}, \tag{1.42}$$

где q – темп роста за период,

$$S = R\frac{q^{np} - (1+i)^n}{q - (1+i)^{\frac{1}{p}}}, \tag{1.43}$$

$$\mathrm{A} = R\frac{q^{np}v^n - 1}{q - (1+i)^{\frac{1}{p}}}. \tag{1.44}$$

Рассмотрим разницу различных рент на примере поступлений денег в фонд. Пусть рента выплачивается в течение 10 лет: ежегодно в конце каждого года поступают средства в размере 10000 рублей по ставке 13% годовых, тогда, используя формулы (1.6) и (1.8), получим

$$S = 10000 \cdot \frac{(1+0.13)^{10}-1}{0.13} \approx 184 \text{ тыс. руб.}$$

$$A = 10000 \cdot \frac{1-(1+0.13)^{-10}}{0.13} \approx 54 \text{ тыс. руб.}$$

В случае такой же ренты, но отложенной на полгода, получим по формуле (1.31)

$$_tA = 10000 * \frac{1-(1+0.13)^{-10}}{0.13} * 1.13^{-0.5} \approx 51 \text{ тыс. руб.}$$

А при увеличении с каждым годом платежа на 100 рублей, применив формулы (1.38) и (1.39), получим

$$S = \left(10000 + \frac{100}{0.13}\right) * \frac{(1+0.13)^{10}-1}{0.13} - \frac{10*100}{0.13} \approx 190 \text{ тыс. руб.}$$

$$A = \left(10000 + \frac{100}{0.13}\right) * \frac{1-(1+0.13)^{-10}}{0.13} - \frac{10*100*\frac{1}{1+0.13*10}}{0.13} \approx 55 \text{ тыс. руб.}$$

Если оставить фиксированный размер платежа, а проценты будут начисляться каждый месяц, по формулам (1.13) и (1.14) получим

$$S = 10000 * \frac{\left(1+\frac{0.13}{12}\right)^{120}-1}{\left(1+\frac{0.13}{12}\right)^{12}-1} \approx 191 \text{ тыс. руб.}$$

$$A = 10000 * \frac{1-\left(1+\frac{0.13}{12}\right)^{-120}}{\left(1+\frac{0.13}{12}\right)^{12}-1} \approx 52.6 \text{ тыс. руб.}$$

А при непрерывном начислении процентов получим по формулам (1.27) и (1.28)

$$S = 10000 * \frac{e^{0.13*10}-1}{e^{0.13}-1} \approx 192 \text{ тыс. руб.}$$

$$A = 10000 * \frac{1-e^{-0.13*10}}{e^{0.13}-1} \approx 52.4 \text{ тыс. руб.}$$

Теперь рассмотрим случай, когда при сохранении исходных значений размера платежа, процентной ставки и срока ренты платежи будут проходить ежемесячно. Тогда при начислении процентов раз в году в конце года по формулам (1.16) и (1.17) получим

$$S = 10000 * \frac{(1+0.13)^{10}-1}{12*((1+0.13)^{\frac{1}{12}}-1)} \approx 195 \text{ тыс. руб.}$$

$$A = 10000 * \frac{1-(1+0.13)^{-10}}{12*((1+0.13)^{\frac{1}{12}}-1)} \approx 57.4 \text{ тыс. руб.}$$

Если же проценты будут поступать ежемесячно, по формулам (1.18) и (1.19) получим

$$S = 10000 * \frac{\left(1+\frac{0.13}{12}\right)^{120}-1}{0.13} \approx 203 \text{ тыс. руб.}$$

$$A = 10000 * \frac{1-\left(1+\frac{0.13}{12}\right)^{-120}}{0.13} \approx 55.8 \text{ тыс. руб.}$$

Пусть теперь взносы в фонд поступают поквартально, а проценты начисляются ежемесячно (остальные параметры оставим такими же, как и в изначальном примере), тогда по формулам (1.20) и (1.23) получим

$$S = 10000 * \frac{\left(1+\frac{0.13}{12}\right)^{120}-1}{4*\left(\left(1+\frac{0.13}{12}\right)^{\frac{12}{4}}-1\right)} \approx 201 \text{ тыс. руб.}$$

$$A = 10000 * \frac{1-\left(1+\frac{0.13}{12}\right)^{-120}}{4*\left(\left(1+\frac{0.13}{12}\right)^{\frac{12}{4}}-1\right)} \approx 55 \text{ тыс. руб.}$$

Если же первые 5 лет начислять 12%, а последующие 5 лет 14%, используя формулы (1.35) и (1.36), получим

$$S = 10000 \cdot s_{5;0.12} \cdot (1 + 0.14)^5 + 10000 \cdot s_{5;0.14} \approx 188 \text{ тыс. руб.}$$

$$A = 10000 \cdot a_{5;0.12} + 10000 \cdot a_{5;0.14} \cdot (1 + 0.12)^{-5} \approx 55{,}5 \text{ тыс. руб.}$$

Для рент постнумерандо также справедливы соотношения [7]:

$$S(1;1) < \underset{m>1}{S(1;m)} < S(1;\infty) < \underset{p>1}{S(p;1)} < \underset{p>m>1}{S(p;m)} < \underset{p=m>1}{S(p;m)} < \underset{m>p>1}{S(p;m)} < S(p;\infty), \tag{1.45}$$

$$A(1;\infty) < A(1;m) < A(1;1) < A(p;\infty) < \underset{m>p>1}{A(p;m)} < \underset{p=m>1}{A(p;m)} < \underset{p>m>1}{A(p;m)} < A(p;\infty), \tag{1.46}$$

где $S(p;m)$ – сравниваемые суммы ($S(1;1)$ – наращенная сумма годовой ренты с ежегодным начислением процентов, $S(1;m)$ – аналогичная характеристика для ренты с начислением m раз в году, $S(p;\infty)$ – наращенная сумма p-срочной ренты с непрерывным начислением процентов),

$A(p;m)$ – современные стоимости ($A(1;1)$ – для годовой ренты с ежегодным начислением процентов, $A(p;\infty)$ – для p-срочной ренты с непрерывным начислением процентов).

Из формул (1.6) – (1.44) также можно выразить значения R, n и i, так, например, если рента годовая, постнумерандо, с ежегодным начислением процентов, получим [7]

$$R = \frac{S}{s_{n;i}} = \frac{A}{a_{n;i}}. \tag{1.47}$$

Для нахождения срока ренты можно использовать формулы для нахождения S или A относительно n, так, например, для годовой ренты постнумерандо, выразив n из формул (1.6) и (1.8) получим

$$S = R\frac{(1+i)^n - 1}{i} \Rightarrow n = \frac{\ln\left(\frac{S}{R}i + 1\right)}{\ln(1+i)}, \tag{1.48}$$

$$A = R\frac{1-(1+i)^{-n}}{i} \Rightarrow n = \frac{\ln\left(1 - \frac{A}{R}i\right)^{-1}}{\ln(1+i)}. \tag{1.49}$$

В случае других видов рент размер платежа и срок ренты будут определяться аналогично.

Для оценки параметра процентной ставки (i) можно применить следующую интерполяционную формулу [7]:

$$i = i_t + \frac{a - a_t}{a_d - a_t}(i_d - i_t), \tag{1.50}$$

где a_d и a_t – значения коэффициентов наращения или приведения рент для верхнего и нижнего уровня ставок (i_d, i_t),
a – значение коэффициента наращения или приведения, для которого определяется размер ставки.

Альтернативным способом нахождения процентной ставки является метод Ньютона-Рафсона [5]. Данный метод применяется для решения уравнений вида

$$f(x) = 0 \tag{1.51}$$

посредством приближения к корню с помощью итераций

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k = 0, 1 \ldots \quad (1.52)$$

Для нахождения начального приближения x_0 можно выбрать значение на одном из концов отрезка, на котором задана функция f и выполняется $f(x_0)f''(x_k) > 0$ [5]. Для расчета процентной ставки достаточно положить [7]

$$\tilde{a}_{n;\delta} = \int_0^n e^{-\delta t}\, dt = \frac{1-e^{-\delta t}}{\delta}, \quad (1.53)$$

где $\tilde{a}_{n;\delta}$ – коэффициент приведения дискретных платежей постнумерандо к непрерывным,

$$A = R\tilde{a}_{n;\delta}, \quad (1.54)$$

$$f(x) = R\frac{1 - e^{-xt}}{x} = 0,$$

$$f(x) = 1 - e^{-xn} - \frac{A}{R}x = 0,$$

$$f'(x) = ne^{-xn} - \frac{A}{R},$$

$$x_{k+1} = x_k - \frac{1 - e^{-x_k n} - \frac{A}{R}x_k}{ne^{-x_k n} - \frac{A}{R}}, k = 0, 1 \ldots \quad (1.55)$$

Ещё одним подходящим решением является метод деления отрезка пополам [5]. Аналогично методу Ньютона-Рафсона требуется найти корень уравнения (1.51), но вместо вычисления производных происходит деление отрезка, на котором задана функция, на две половины (предположим, что функция задана на отрезке $[a, b]$, а его серединой является точка c). Тогда, если $f(\mathrm{c}) = 0$, то c – корень уравнения, иначе, если $f(\mathrm{c}) > 0$, то требуется рассмотреть отрезок $[\mathrm{c}, b]$, если же $f(\mathrm{c}) > 0$ – отрезок $[a, c]$.

В случае нахождения процентной ставки можно положить

$$f(x) = S(x) \quad (1.56)$$

или

$$f(x) = A(x), \quad (1.57)$$

где x – значение процентной ставки, и решать уравнение

$$f(x) = S_{\text{зад}}(A_{\text{зад}}), \tag{1.58}$$

где $S_{\text{зад}}(A_{\text{зад}})$ – заданное значение наращенной суммы (современной стоимости).

1.2 Необходимость автоматизации процесса расчёта потоков платежей

В повседневной жизни потоки платежей выступают все более важным звеном между производителями, потребителями, государством и финансовыми учреждениями (рисунки 1.1 и 1.2). Эти потоки принимают различные формы: налоги, зарплаты, пенсии, пособия, стипендии, оплаты подписок, выплаты долгов, дивиденды, выплаты по облигациям и депозитам, рассрочки, кредиты и т. д. Большинство из них представляют собой периодические выплаты примерно одинаковых сумм, однако бóльшую сложность для людей представляют собой потоки, требующие большей гибкости и внимательности в расчётах и управлении.

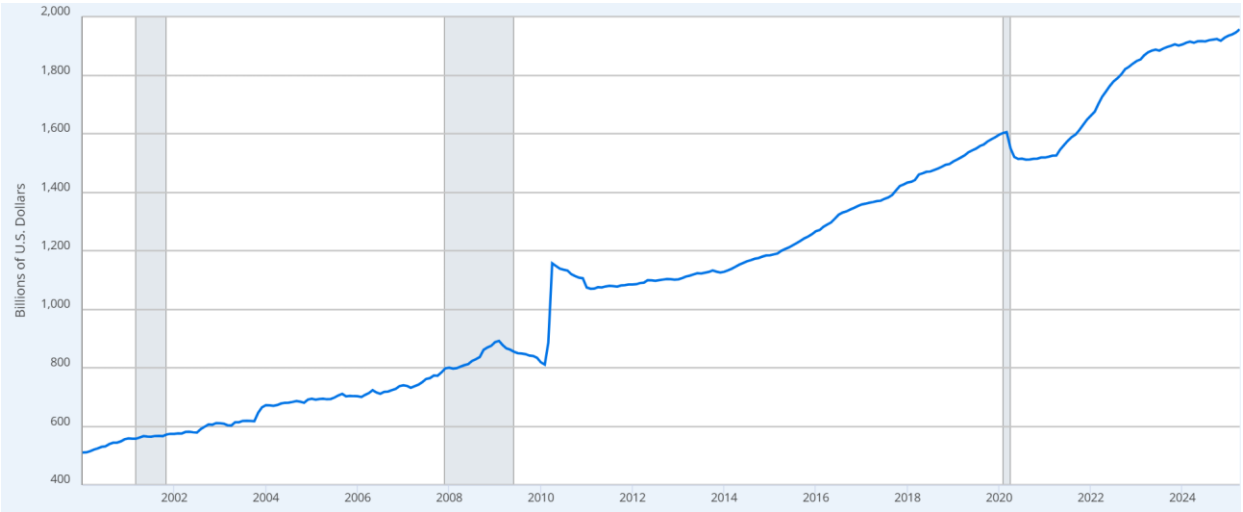


Рисунок 1.1 – График зависимости общего объёма потребительских кредитов от времени [9]

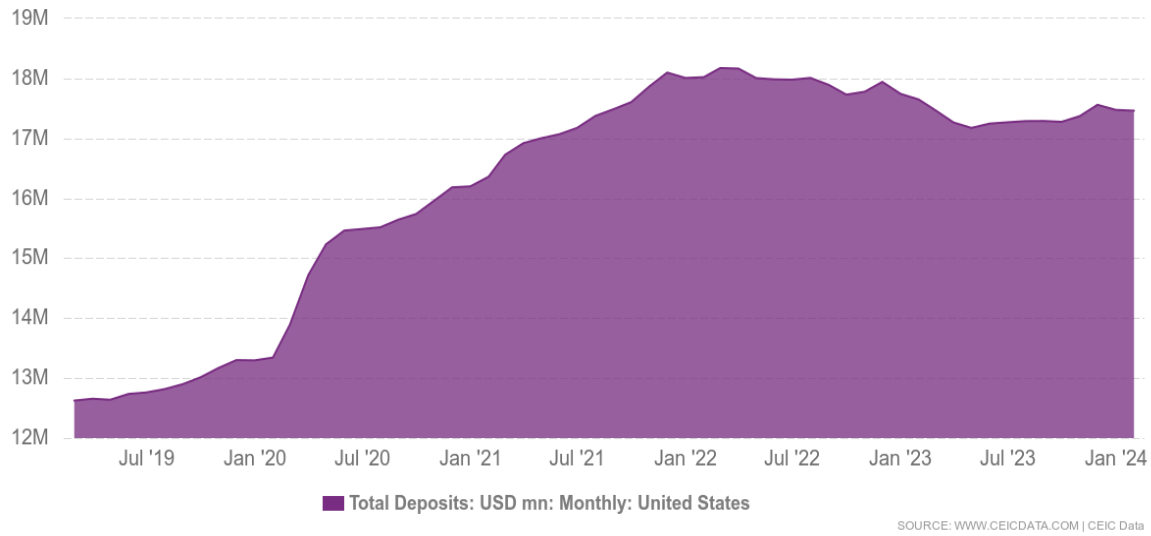


Рисунок 1.2 – График зависимости общего объёма депозитов от времени (данные за последние 5 лет) [15]

Поэтому все более актуальным становится вопрос о достаточной финансовой грамотности населения. Ведь человек, обладающий финансовой грамотностью, может более эффективно распоряжаться своими средствами, ставить финансовые цели и достигать их. Понимание, как работают потоки платежей, позволяет принимать обоснованные решения и увеличивать доходы. В то же время финансовая грамотность остаётся темой, которую часто откладывают «на потом», считая её чем-то сложным или избыточным. Поэтому нередко складываются ситуации, при которых финансово неграмотный человек оказывается в уязвимом положении, не осознавая всех последствий своих повседневных решений.

Стоит также отметить, что само население Республики Беларусь отмечает у себя нехватку финансовой грамотности. Анализ субъективных оценок уровня знаний населения по различным аспектам финансовой грамотности [6] показал, что значительная часть населения невысоко оценивает уровень владения знаниями по многим из них (рисунок 1.3). Из диаграмм можно заметить, что различные темы, связанные с потоками платежей, такие как инвестирование, кредитование, сбережение и др. недостаточно понятны населению, а значительная часть и вовсе не считает необходимым изучение подобных аспектов. Тем не менее, как ранее было показано на рисунках 1.1 и 1.2, люди все чаще сталкиваются с данными темами.

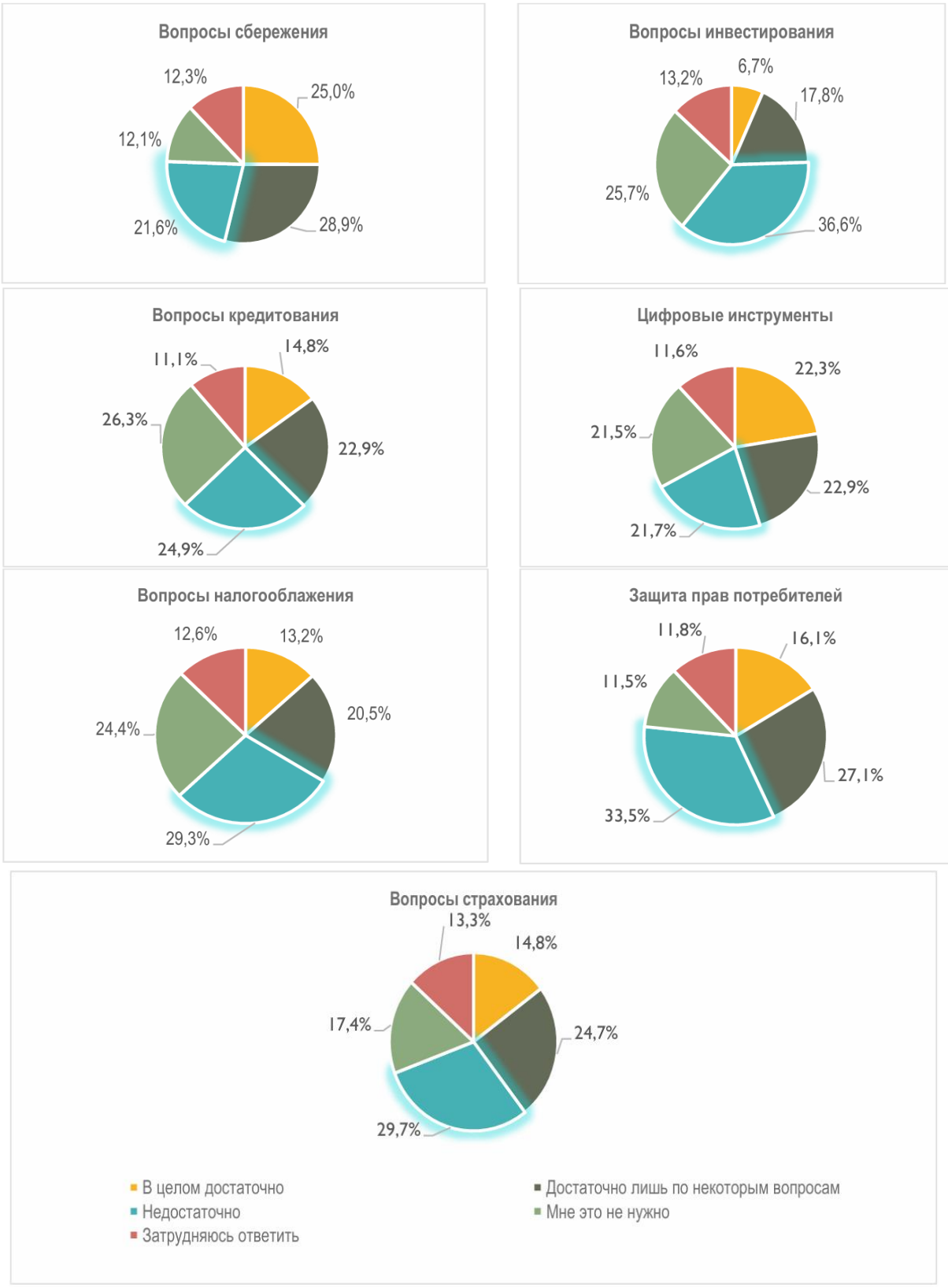


Рисунок 1.3 – Субъективная оценка знаний населения по различным аспектам финансовой грамотности [6]

Для упрощения работы с потоками платежей созданы различные решения, позволяющие производить необходимые расчёты без необходимости запоминания пользователем множества формул. Наиболее

распространёнными видами таких решений являются калькуляторы кредитов и депозитов (вкладов), например реализации калькуляторов от MYFIN [3], Беларусбанка [2], Сбербанка [4] или fingramota [1]. Однако большинство таких решений реализуют расчёты для одного или небольшого количества видов потоков платежей (причём для этого может создаваться несколько приложений) и, несмотря на то что такие реализации могут покрыть большинство запросов пользователей, значительное количество разновидностей потоков платежей остаётся нетронутым, из-за чего не удаётся полноценно понять их устройство. Также стоит отметить, что, так как реализация расчётов зачастую скрыта от пользователя, может быть затруднительно понять, как именно были получены данные результаты. Ещё одним недостатком подобных решений является отсутствие возможности выборов параметра для расчёта, так, например, кредитные калькуляторы зачастую позволяют рассчитать лишь ежемесячный платёж, а калькуляторы вкладов только сумму на момент возврата вклада. Сравнение наиболее популярных решений представлено в таблице 1. Стоит отметить, что выбранные решения используются для выбора подходящих услуг в банках, поэтому кроме функционала, относящегося к потокам платежей, в решениях присутствуют другие особенности, например, подбор кредитных тарифов, возможность выбора валют и. т.д. (такой функционал в работе не рассматривается).

Таблица 1 – Сравнение решений по автоматизации потоков платежей

	MYFIN	Беларусбанк	Сбербанк	fingramota
Возможность выбора параметра	нет	нет	нет	частично
Пояснение расчётов	нет	частично	нет	нет
Количество видов потоков платежей для расчёта	6	2	2	8

Учитывая ограниченную функциональность существующих решений, возникает необходимость в создании приложения, которое бы давало пользователям возможность более подробно ознакомиться с видами потоков платежей, процессом расчёта параметров и при этом могло использоваться в повседневной жизни для расчёта наиболее часто встречающихся в реальной жизни потоков платежей.

Также стоит отметить, что более 18%, принявших в участие в опросе в рамках анализа [6] считает необходимым разработку обучающих мобильных приложений (рисунок 1.4). Выгодность мобильных приложений также подтверждается наличием популярных приложений, таких как Duolingo, Quizlet, Mimo и других.

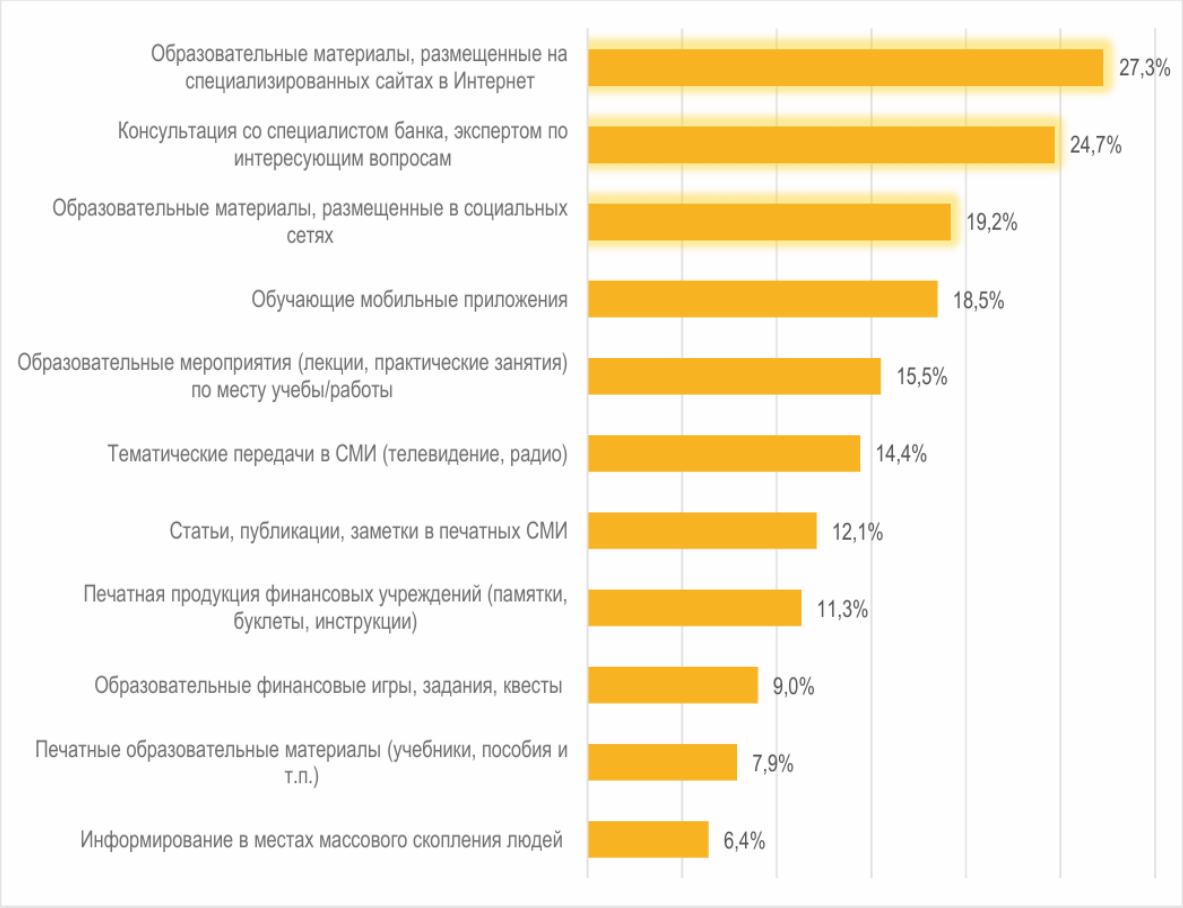


Рисунок 1.4 – Предпочтения населения в способе повышения уровня финансовой грамотности [6]

Таким образом, при существовании большого количества потоков платежей нельзя отрицать сложности в понимании их особенностей и отличий, а также необходимости уметь правильно выбирать потоки платежей и параметры для них в зависимости от реальной ситуации. Даже с учетом существующих решений по автоматизации расчетов присутствует необходимость в решениях с более детальной проработкой потоков платежей и возможностями демонстраций особенностей каждого из таких потоков.

ГЛАВА 2
ПРИЛОЖЕНИЕ ДЛЯ АВТОМАТИЧЕСКОГО РАСЧЁТА ПОТОКОВ ПЛАТЕЖЕЙ

2.1 Требования к приложению, стек технологий, описание работы приложения, план разработки

Перед реализацией приложения были сформулированы следующие требования:

1. Поддержка максимального количества видов потоков платежей, представленных в разделе 1.1, а также возможность расчёта максимального количества параметров.
2. Наличие понятных пользователю пояснений расчётов в виде формул.
3. Возможность демонстрации альтернативных параметров и потоков платежей для увеличения или уменьшения значения рассчитываемого параметра.
4. Кроссплатформенность приложения и простота портирования приложения.
5. Возможность работы приложения без подключения к интернету.
6. Наличие удобного и понятного пользователю интерфейса.
7. Эффективная скорость расчётов.
8. Возможность добавлять новые модули и функции без необходимости полного переписывания существующего кода.

Для реализации требований было принято решение создать мобильное приложение на языке Dart [10] с использованием фреймворка Flutter [11]. Выбор именно таких технологий имеет следующие преимущества:

1. Flutter позволяет создавать приложения, работающие на любом мобильном устройстве, а также позволяет расширить приложение для работы в браузерах и настольных устройствах.
2. Язык Dart, используемый в Flutter, компилируется в нативный код, что обеспечивает приемлемую скорость работы приложения.
3. Flutter предлагает обширную библиотеку виджетов, которые легко настраиваются, что позволяет быстро создавать удобный пользовательский интерфейс [11].
4. Фреймворк поддерживает модульную архитектуру и позволяет эффективно применять паттерн MVC при создании приложения.

5. Flutter имеет встроенную функцию hot reload, позволяющую производить изменения в программе, без необходимости её перезапуска, что ускоряет процесс разработки, а также упрощает обновления приложения.

6. Flutter имеет большое и активное сообщество разработчиков, а также обширную документацию [11], что упрощает решение возникающих проблем и поиск готовых решений.

Также было сформулировано примерное описание работы приложения: пользователь, попадая на главную страницу, с помощью меню навигации выбирает необходимый тип потоков платежей для расчётов. Далее пользователь выбирает необходимый параметр для расчёта и вводит параметры, требуемые приложением для произведения расчётов. При нажатии на кнопку «Рассчитать» пользователю предоставляется информация о результатах расчёта. При нажатии на кнопку «Пояснение расчётов» пользователю предоставляется информация об использованных при расчётах формулах. При нажатии на кнопку «Альтернативы» пользователю предоставляется меню для навигации, для выбора возможных способов уменьшения или увеличения рассчитываемого параметра, а при выборе какого-либо из способов пользователю демонстрируются пояснения к способу, которые сопровождаются примерами.

В соответствии с ожидаемым использованием приложения был создан следующий план создания приложения:

1. Создание архитектуры приложения и файловой структуры проекта.
2. Программная реализация необходимых для расчёта формул.
3. Создание пояснений для расчётов.
4. Создание страниц для потоков платежей и пояснений.
5. Создание меню навигации по потокам платежей.
6. Создание функционала для предоставления альтернатив.

2.2 Создание приложения

2.2.1 Создание архитектуры и файловой структуры приложения

Для реализации приложения было принято решение использовать паттерн MVC, который обеспечивает эффективное разделение приложения на модули: Model, View и Controller, что позволяет реализовывать части

приложения независимо, а также упрощает добавление нового функционала [12].

Концептуально работа приложения выглядит следующим образом:

1. Пользователь взаимодействует с некоторым интерфейсом (View).
2. Model представляет собой некоторую информацию о потоке платежей и при изменении параметров уведомляет View, что позволяет отобразить изменения пользователю.
3. Для обеспечения интерактивности View взаимодействует с Controller, отправляя запросы на получение или изменение данных, в частности – запросы на изменение параметров потока платежа либо получение результата расчётов.
4. Controller принимает запросы от View и обновляет Model, либо предоставляет о ней информацию.
5. Также View обращается к отдельному модулю – static data. Данный модуль предоставляет необходимые текста и функции для формирования пояснений и предоставлении альтернатив.

Визуальное представление архитектуры представлено на рисунке 2.1.

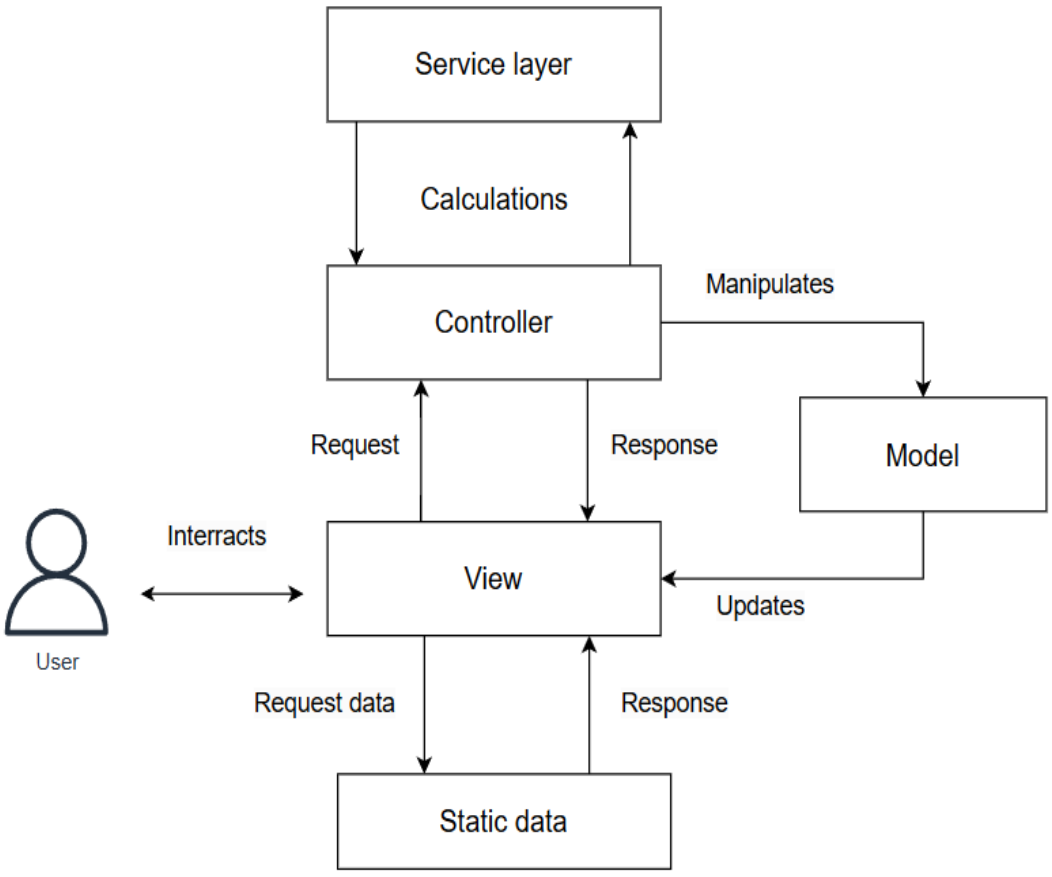


Рисунок 2.1 – Архитектура приложения

Ключевая логика приложения разбита по нескольким директориям проекта:

1. View: хранит файлы с реализацией клиентской части приложения, с которой будет взаимодействовать пользователь (экраны для расчёта потоков платежей, пояснений расчётов и т.д.).
2. Model: хранит файлы с объектным представлением потоков платежей (параметры потоков платежей, рассчитываемый в данный момент параметр и т.д.).
3. Controller: хранит файлы с реализацией интерактивного поведения приложения (расчёт параметров, сохранение введённых пользователем данных и т.д.).
4. Service: хранит формулы для расчёта параметров потоков платежей.
5. Static: хранит тексты с пояснениями расчётов и альтернативами при расчетах, а также функции для их удобного получения и изменения при необходимости.
6. Routes: хранит файлы, предназначенные для навигации по приложению.

При создании проекта фреймворком были автоматически созданы директории для хранения исполняемых файлов после сборки приложения (для каждой платформы файлы в отдельной директории).

Итоговое разбиение проекта представлено на рисунке 2.2.

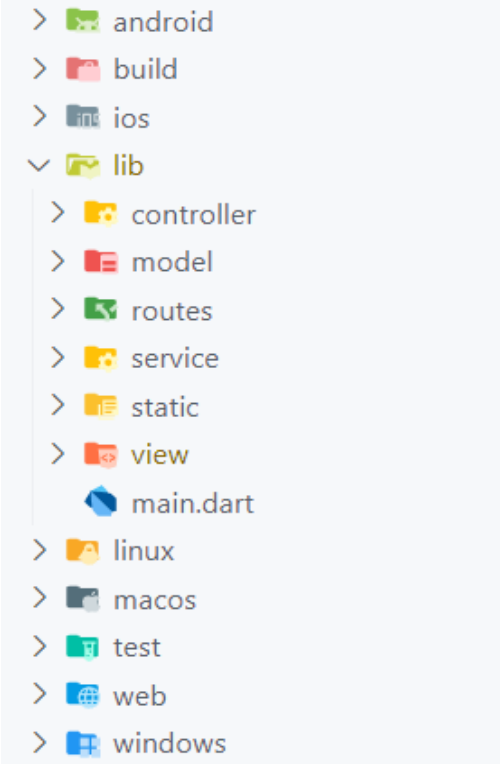


Рисунок 2.2 – Структура приложения

2.2.2 Среда для запуска приложения

Для запуска приложения было принято решение использовать виртуальное мобильное устройство. К основным преимуществам такого подхода можно отнести:

1. Эмуляция работы разных моделей телефонов без необходимости затрат на физические устройства.
2. Возможность выбора различных операционных систем, их версий и оболочек, что позволяет более детально тестировать приложение.
3. Встроенные возможности интеграции со средами разработки для более комфортного процесса разработки.

В качестве основной модели был выбран телефон Google Pixel 7a и операционная система Android 15.0 Vanilla Ice Cream. Интерфейс эмулятора представлен на рисунке 2.3.

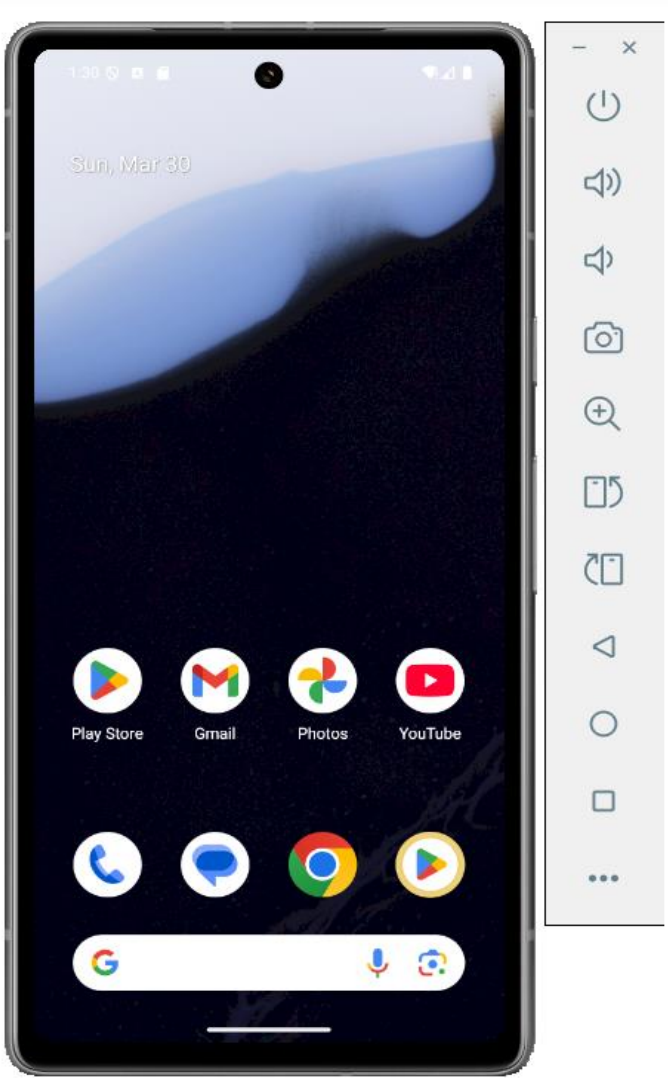


Рисунок 2.3 – Интерфейс эмулятора

Можно заметить, что интерфейс состоит из интерфейса самого устройства и вспомогательной боковой панели для более удобного взаимодействия с устройством. Фреймворк Flutter взаимодействует с эмулятором посредством подключения к устройству, установки файла с приложением в его систему и запуска самого приложения (рисунок 2.4).

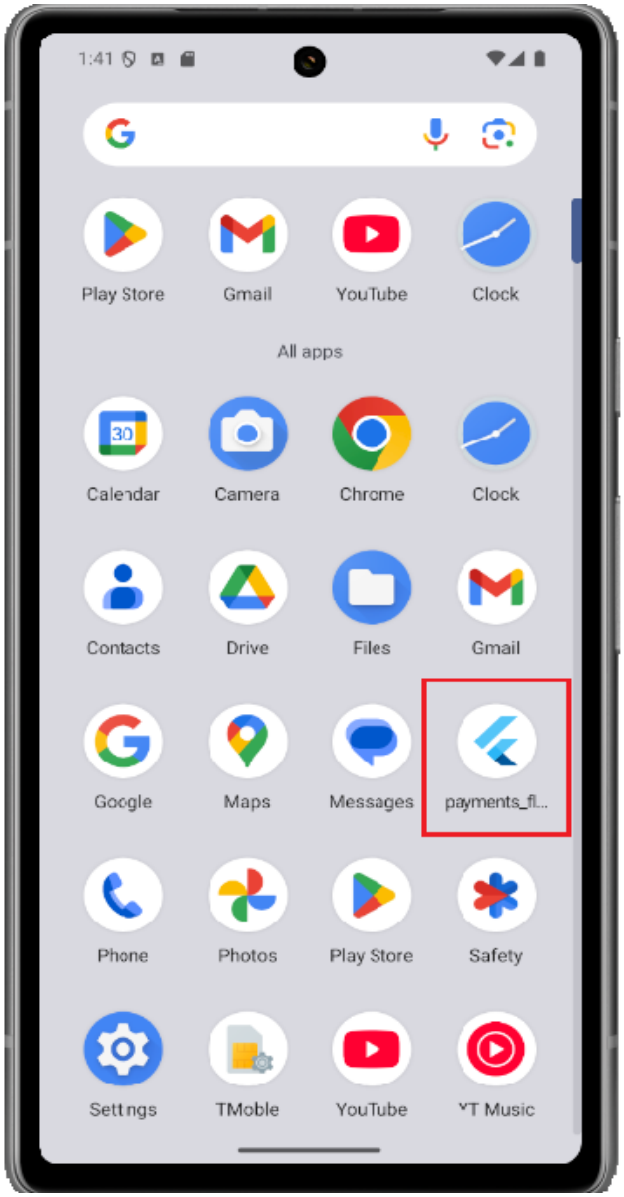


Рисунок 2.4 – Приложение на эмуляторе

2.2.3 Программная реализация необходимых для расчёта формул

Как было упомянуто в разделе 2.2.1, реализация формул расчёта параметров потоков платежей вынесена в файлы специальных сервисов. В

данном разделе на примере годовой ренты постнумерандо продемонстрирован процесс создания такого сервис файла.

Для начала были определены параметры и формулы для реализации:

1. Наращенная сумма (S): для расчёта были использованы формулы (1.4) и (1.6).
2. Современная стоимость (A): для расчёта были использованы формулы (1.4) и (1.8).
3. Ежегодный платёж (R): для расчёта были использованы формулы (1.47).
4. Годовая процентная ставка (i): для расчёта был использован метод деления отрезка пополам, представленный в главе 1.
5. Общий срок выплат в годах (n): для расчётов были использованы формулы (1.48) и (1.49).

Для реализации программного кода был создан файл year_postnumerando_rent_service.dart, содержащий класс YearPostnumerandoRentService и набор статических методов внутри:

1. double calculateSUsingR(double R, double i, int n) – реализация формулы (1.6).
2. double calculateSUsingA(double A, double i, int n) – реализация формулы (1.4).
3. double calculateAUsingR(double R, double i, int n) – реализация формулы (1.8).
4. double calculateAUsingS(double S, double i, int n) – реализация формулы (1.4).
5. double calculateAUsingR(double R, double i, int n) и double calculateRUsingS(double S, double i, int n) – реализации (1.47).
6. double calculateIUsingS(double S, double R, int n, double eps) и double calculateIUsingA(double A, double R, int n, double eps) – реализации метода деления отрезка пополам для нахождения процентной ставки;
7. double calculateNUsigS(double S, double R, double i) и double calculateNUsingA(double A, double R, double i) – реализации формул (1.48) и (1.49).

Реализация алгоритма расчёта годовой процентной ставки:

```
static double calculateIUsingS(double S, double R, int n,
    {double eps = 0.001}) { // eps – допустимая погрешность
  double iLower = 0.0001; // минимальное значение процентной ставки
  double iUpper = 1.0; // максимальное значение процентной ставки
  double iMid;
  while ((iUpper - iLower) > eps) {
   iMid = (iLower + iUpper) / 2;
```

```
    double testS = calculateSUsingR(R, iMid, n);
    if (testS < S) {
      iLower = iMid;
    } else {
      iUpper = iMid;
    }
  }
  return (iLower + iUpper) / 2;
}
static double calculateIUsingA(double A, double R, int n,
    {double eps = 0.001}) {
  double iLower = 0.0001;
  double iUpper = 1.0;
  double iMid;
  while ((iUpper - iLower) > eps) {
    iMid = (iLower + iUpper) / 2;
    double testA = calculateAUsingR(R, iMid, n);
    if (testA > A) {
      iLower = iMid;
    } else {
      iUpper = iMid;
    }
  }
  return (iLower + iUpper) / 2;
}
```

Реализация остальных формул представлена в Приложении А.

Для хранения результатов и последующего отображения данных была создана модель, представляющая собой класс TermRent. Класс хранит поля, представляющие собой параметры для годовой ренты постнумерандо и поле для определения рассчитываемого в данный момент параметра. Также стоит отметить, что параметры, которые являются общими для всех рент (в частности: наращенная сумма, современная стоимость потока, процентная ставка и размер члена ренты) были вынесены в отдельный класс BaseRent. Там же находится специальный объект fieldMap, позволяющий по названию параметра получить функцию для его обновления. Сам класс выглядит следующим образом:

```
abstract class BaseRent {
  double? S;
  double? A;
  double? R;
```

```
 double? i;
 Parameters selectedParameter;
late final Map<Parameters, void Function(double?)> fieldMap;
 BaseRent({
  this.S,
  this.A,
  this.R,
  this.i,
  this.selectedParameter = Parameters.S,
 }) {
  fieldMap = {
   Parameters.S: (value) => S = value,
   Parameters.A: (value) => A = value,
   Parameters.R: (value) => R = value,
   Parameters.i: (value) => i = value == null ? null : value / 100,
  };
 }
}
```

В свою очередь класс TermRent (как и все другие классы, описывающие ренты) наследует BaseRent:

```
class TermRent extends BaseRent {
 int? n;
 TermRent({
  super.S,
  super.A,
  super.R,
  super.i,
  super.selectedParameter,
  this.n,
 }) {
  fieldMap.addAll({
   Parameters.n:
      (value) => {
       if (value != null) {n = value.toInt()},
      },
  });
 }
}
```

Стоит отметить, что реализация моделей и сервисов для других видов потоков платежей с точки зрения проектирования аналогична представленной, изменяются только параметры и некоторые формулы.

2.2.4 Создание пояснений для расчётов

Для хранения пояснений к расчётам различных видов ренты было принято решение использовать вложенную структуру на основе объекта Map. Внешний Map использует в качестве ключа вид ренты (например, Rents.YEAR_POSTNUMERANDO – годовая рента постнумерандо), а в качестве значения – другой объект Map, который содержит пары «параметр – пояснение к расчётам для этого параметра». Такой подход обеспечивает удобный и быстрый доступ к необходимой информации, а также позволяет легко масштабировать систему при добавлении новых видов рент или параметров. Сам текст пояснения можно поделить на 3 части:

1. Теоретическое определение параметра – краткое, но ёмкое описание параметра. Например, для наращенной суммы ренты (S) указывается, что это общая сумма всех платежей, приведённых к концу срока ренты с учётом начисленных процентов.
2. Необходимые параметры для расчёта – входные данные, которые требуются для вычисления текущего параметра в рамках выбранного вида потока платежей. Например, для расчёта наращенной суммы годовой ренты постнумерандо ренты необходимо знать: размер регулярного платежа (R), процентную ставку (i) и срок ренты (n).
3. Формулы для расчёта параметра – математическое выражение, по которому вычисляется параметр.

Так как язык Dart не предоставляет встроенной возможности оформить формулу в читаемом для пользователя виде, был использован дополнительный модуль flutter_math_fork, предоставляющий возможности для использования языка разметки LaTeX [14]. Несмотря на то, что модуль не поддерживает все возможности оригинального LaTeX, функционала оказалось достаточно для корректного представления формул в приложении. Так, например для представления формулы (1.6) была создана следующая программная реализация:

```
Math.tex(
    r'''
      S = R \cdot \frac{(1 + i)^n - 1}{i}
    ''',
```

textStyle: const TextStyle(fontSize: 16),

);

На рисунке 2.5 представлен пример отображения формул на экране приложения.

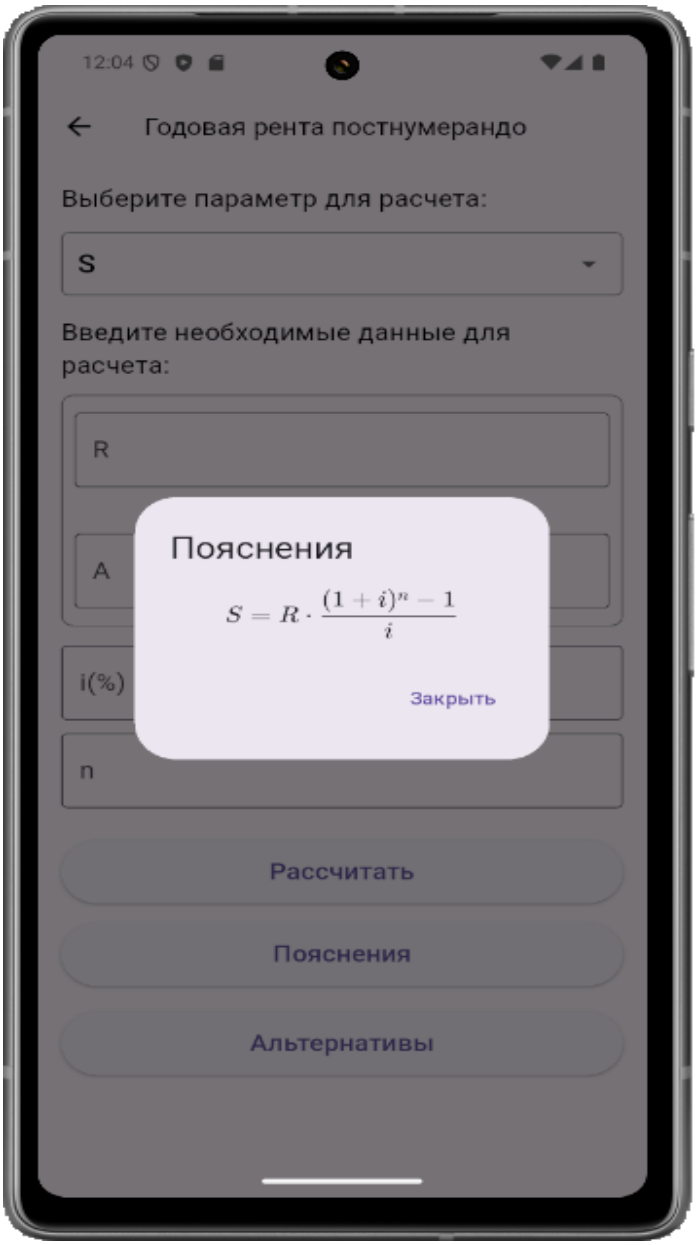


Рисунок 2.5 – Отображение формулы

Текст, отличный от формул, представляет собой объект класса Text, который позволяет добавить необходимое форматирование текста, а также позволяет легко отображать текст в будущем.

Также для будущего отображения, каждое пояснение как единое целое представляет собой объект класса Column, что позволяет расположить части текста в необходимом формате.

Так как некоторые части, такие как определения параметров, некоторые формулы и пояснения, являются одинаковыми для нескольких потоков платежей, для обеспечения переиспользуемости был создан специальный класс ExplanationsBuilder, реализующий шаблон проектирования Builder, который позволяет построить виджет для отображения из формул и текста пояснений.

Итоговый формат пояснений на примере годовой ренты постнумерандо и расчёта наращенной суммы:

```
Rents.YEAR_POSTNUMERANDO: {
   Parameters.S:
      ExplanationsBuilder.instance
         .addParagraph(yearRentPostnumerandoSExplanation)
         .addFormula(yearRentPostnumerandoSUsingRFormula)
         .addParagraph(sCalculationUsingAExplanation)
         .addFormula(sUsingAFormula)
         .build();
}
```

Таким образом, для получения необходимого пояснения и его отображения необходимо из объекта по значениям типа ренты и рассчитываемого параметра получить необходимый объект пояснения, содержащий всю необходимую информацию и готовый для отображения.

2.2.5 Создание страниц для потоков платежей и пояснений

Страница для расчёта отдельного вида потоков платежей состоит из нескольких частей:

1. Область выбора рассчитываемого параметра.
2. Область ввода необходимых данных для расчёта.
3. Кнопки получения результатов расчёта, пояснений и альтернатив.
4. Всплывающее окно с результатами расчёта.
5. Всплывающее окно с пояснениями к расчётам.
6. Навигационная панель для детального выбора альтернатив к параметрам и потокам платежей.

Для связи между отображением и моделью используется класс контроллера YearPostnumerandoRentController, реализация которого представлена в приложении Б.

На рисунках 2.6 и 2.7 представлены результаты создания страницы для годовой ренты постумерандо (реализация интерфейса представлена в приложении В).

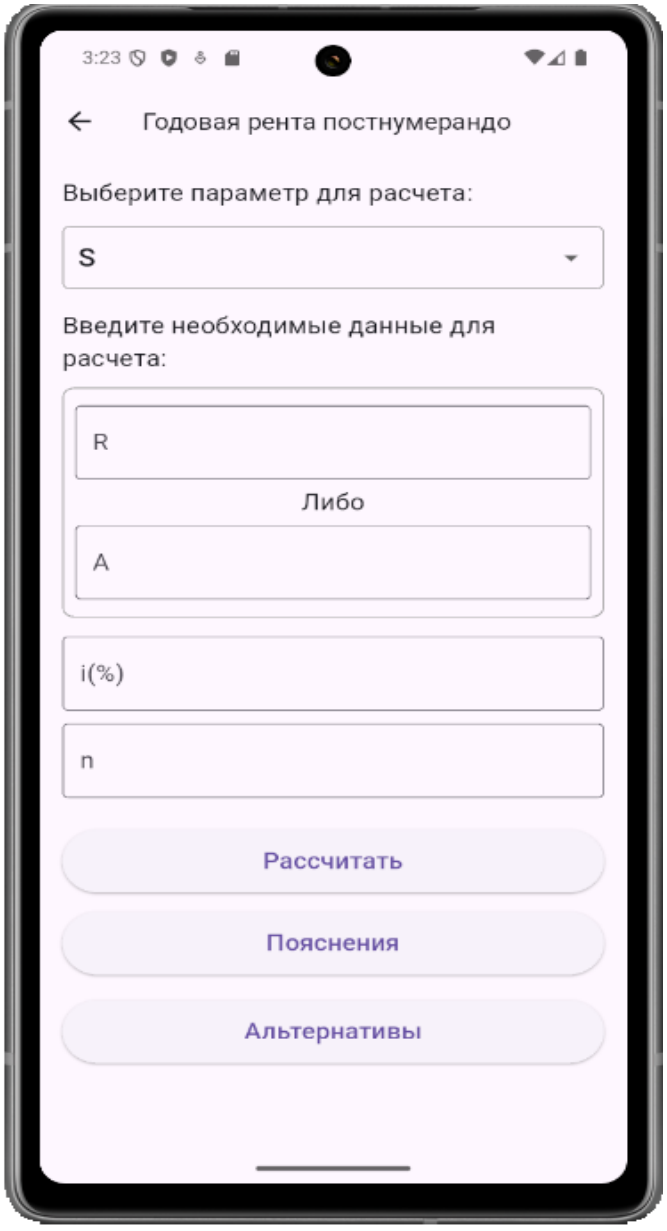


Рисунок 2.6 – Экран потока платежей

На рисунке 2.6 можно заметить, что поля ввода учитывают несколько возможных вариантов расчёта, что позволяет пользователю производить более гибкие вычисления, сохраняя при этом удобство интерфейса. Стоит отметить, что при вводе данных, позволяющий посчитать параметр несколькими способами, будет использована первая подходящая для расчёта формула (в случае с полями на рисунке 2.6, при заполнении всех полей, наращенная сумма будет посчитана с использованием формулы (1.6), при этом значение параметра *A* будет игнорироваться).

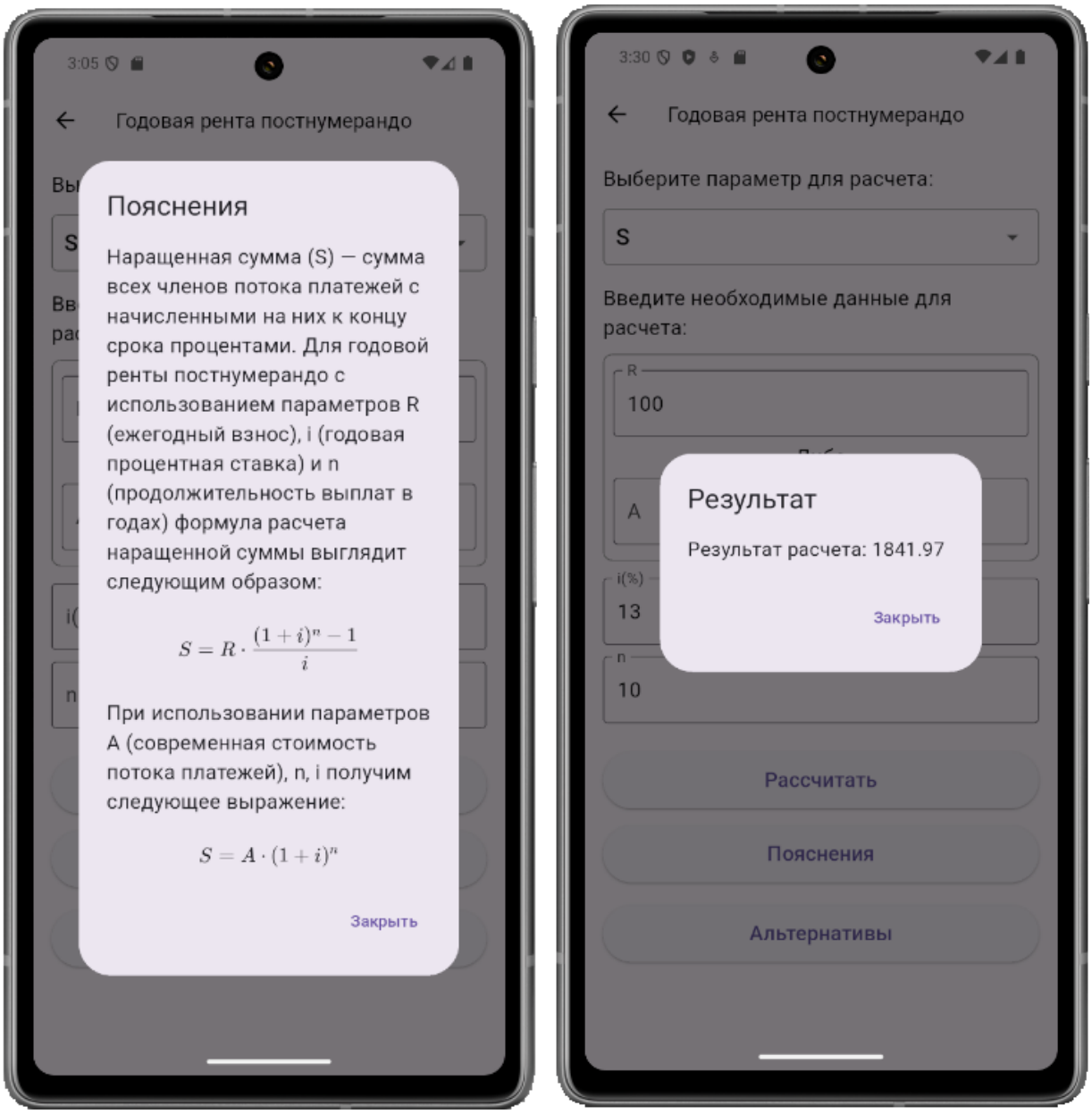


Рисунок 2.7 – Всплывающие окна пояснений и результатов расчёта

Также стоит отметить, что при вводе параметров происходит валидация, что позволяет предотвратить неверный формат данных. Так, например, в числовые поля нельзя вводить буквы, спецсимволы и т. д. При возникновении ошибки во время расчётов, например, если оставить значение какого-либо из обязательных параметров пустым, или ввести некорректное значение параметра (к примеру, при попытке рассчитать ренту при нулевой процентной ставке), при получении результатов расчёта отобразится сообщение об ошибке при вводе данных (рисунок 2.8).

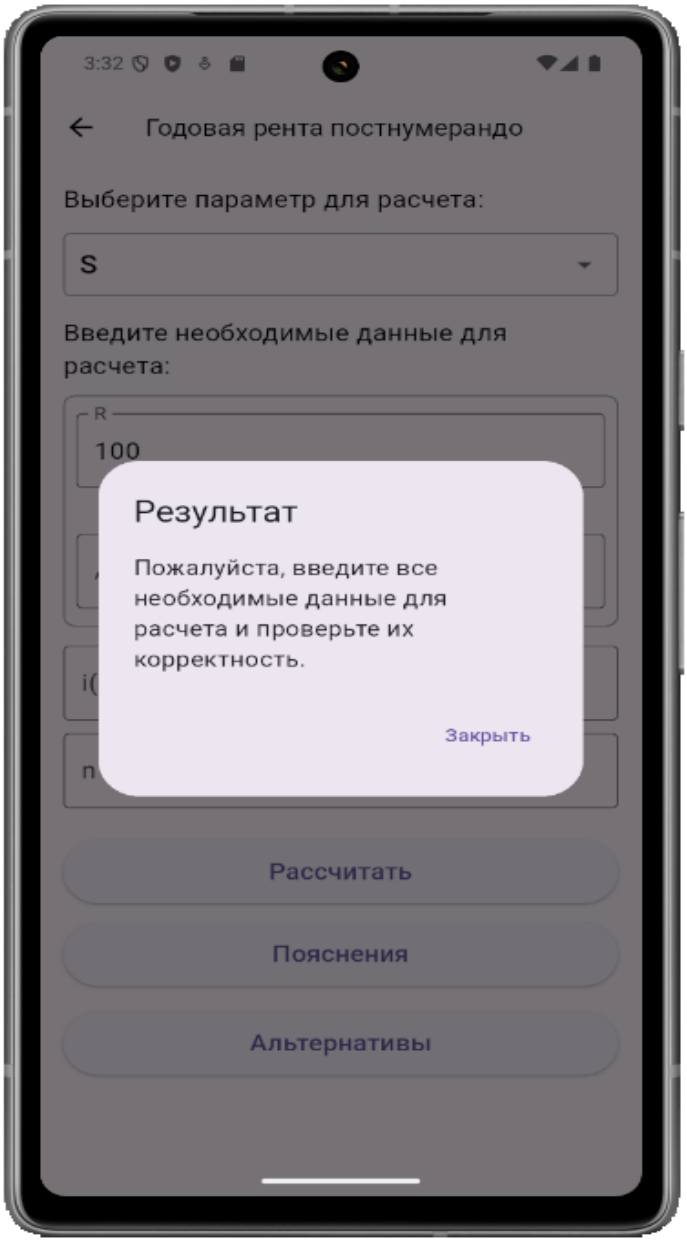


Рисунок 2.8 – Всплывающее окно при ошибке расчёта

При изменении рассчитываемого параметра поля также изменяются на необходимые для расчёта, а также сохраняются значения параметров с предыдущего расчёта, если они нужны для нахождения нового параметра, что позволяет не начинать процесс ввода с нуля (рисунок 2.9).

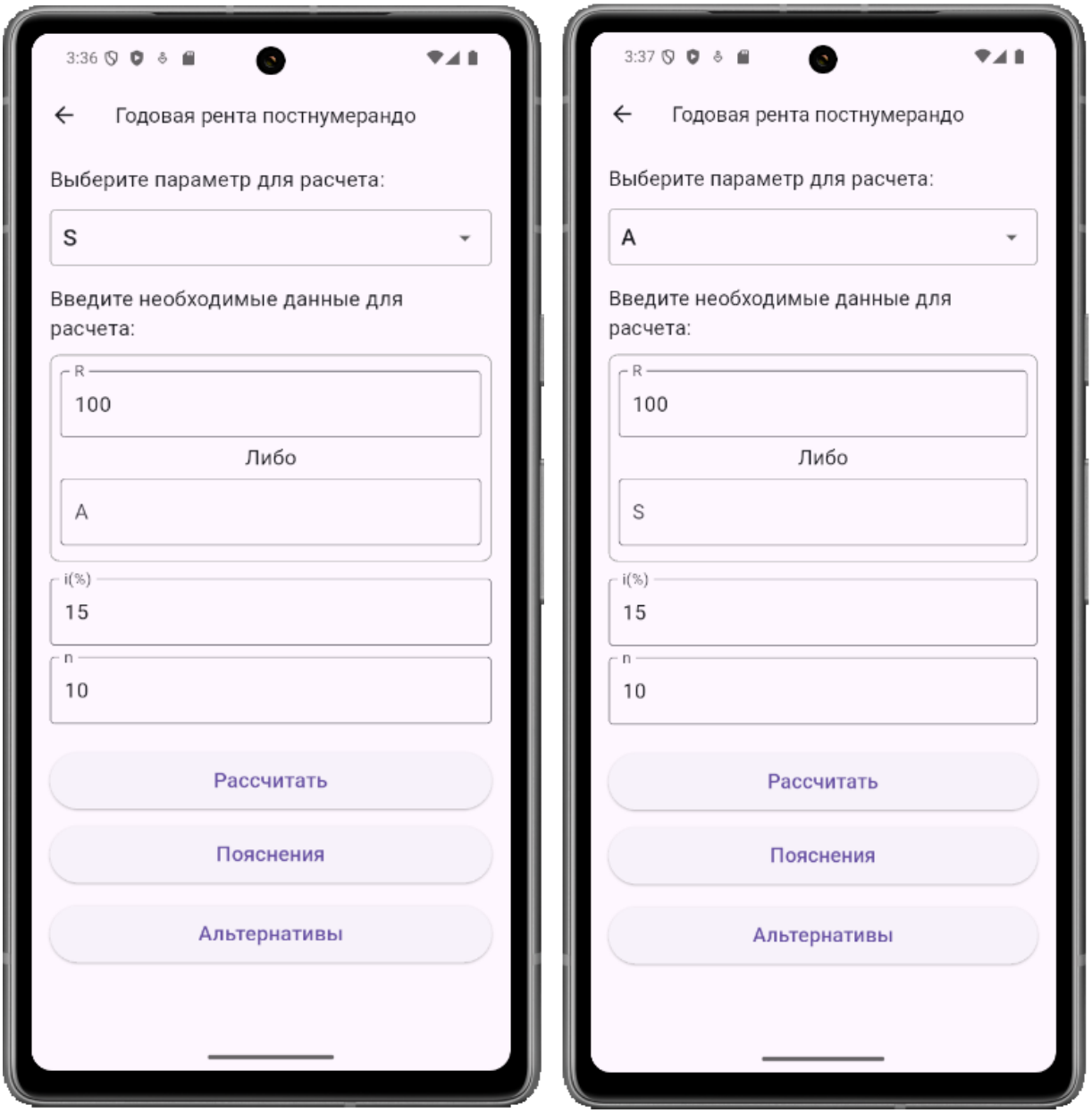


Рисунок 2.9 – Процесс смены рассчитываемого параметра

2.2.6 Создание меню навигации

Меню навигации представляет собой набор секций с названием ренты, на которую эта секция ведёт (рисунок 2.10). При нажатии на секцию пользователь может перейти на страницу с выбранным видом потоков платежей, после чего вернуться обратно к навигации.

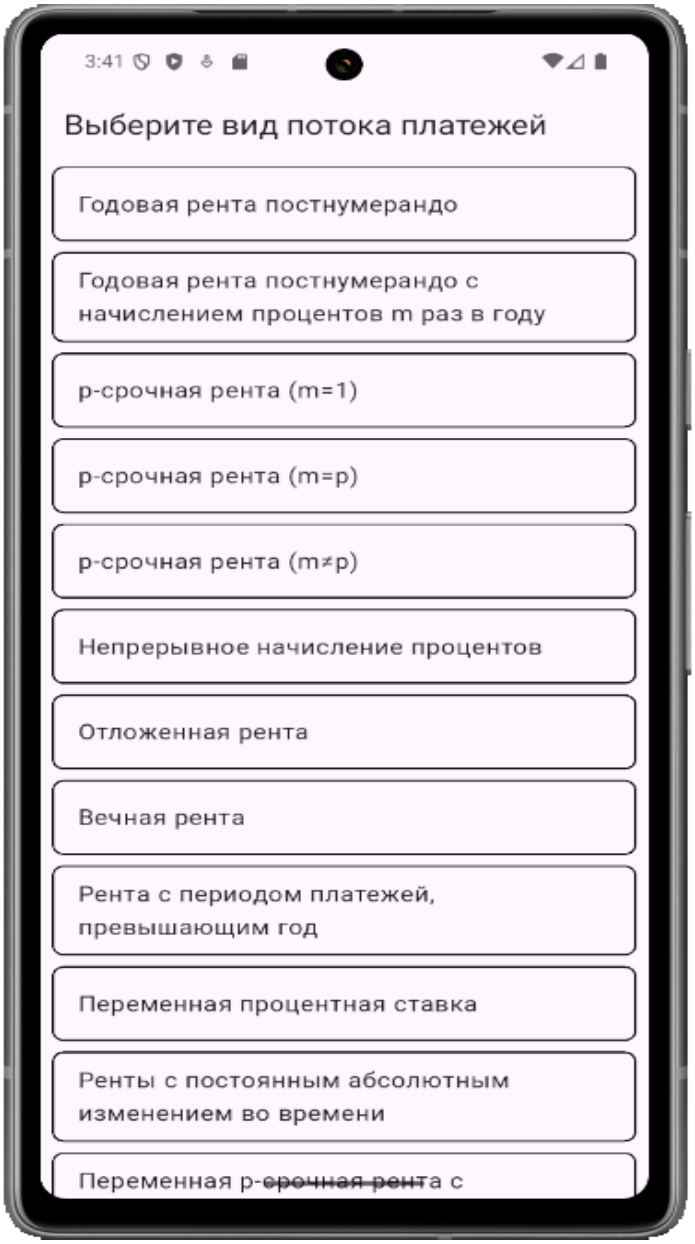


Рисунок 2.10 –Меню навигации

Данное меню навигации также позволяет добавлять неограниченное количество элементов, что позволяет добавить новые виды рент в будущем.

2.2.7 Создание функционала для предоставления альтернативных значений параметров и потоков платежей

Процесс выбора альтернатив разделен на две части:

1. Выбор уменьшения либо увеличения выбранного параметра.
2. Выбор одного из предложенных способов уменьшения (увеличения) параметра.

Разберём данный процесс на примере годовой ренты постнумерандо с начислением процентов m раз в году. Выбранным считается параметр, который выбран для расчёта в окне потока платежей. Так, например, на рисунке 2.11 выбранным параметром считается параметр S (наращенная сумма).

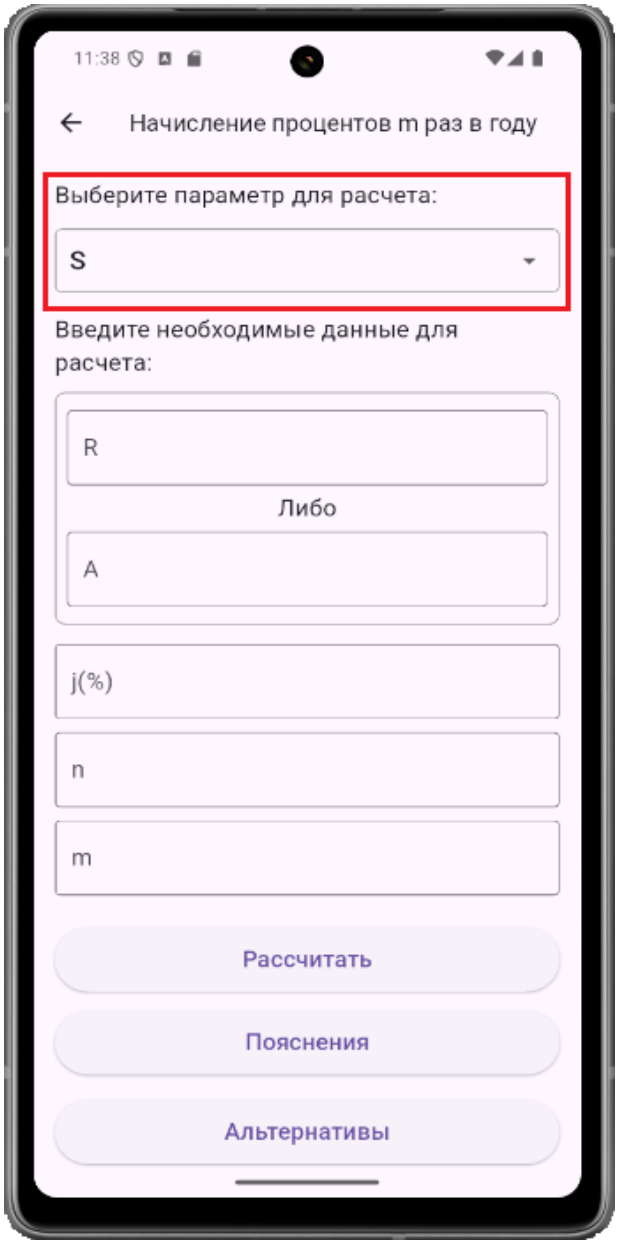


Рисунок 2.11 – Пример выбранного параметра

При нажатии на кнопку «Альтернативы» пользователю будет отображено новое окно, где он может выбрать: увеличить, либо уменьшить выбранный параметр (рисунок 2.12).

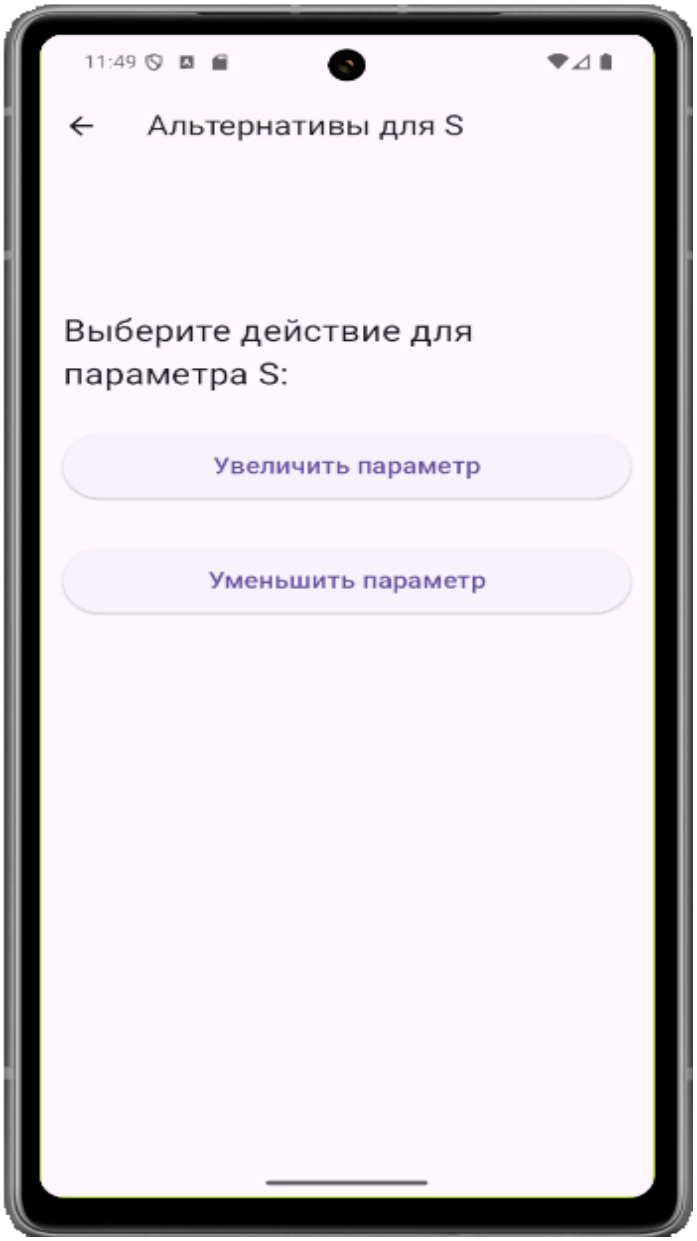


Рисунок 2.12 – Экран выбора изменения параметра

При выборе какой-либо из опций, пользователю демонстрируется окно с возможными способами увеличить (уменьшить) выбранный параметр. Подбор способов изменения параметров основан на формулах из главы 1. Так, например из формулы (1.13) можно заметить, что для увеличения наращенной суммы можно увеличить:

1. Член ренты (R);
2. Номинальную процентную ставку (j);
3. Срок ренты (n);
4. Количество начислений процентов в году (m).

В свою очередь альтернативные виды рент взяты по принципу наибольшего сходства параметров, принципа работы (при этом стоит отметить, что в приложении присутствует возможность добавления

альтернативных потоков платежей в случае необходимости), а также соотношений из формул (1.45) и (1.46). В частности, для годовой ренты постнумерандо с начислением процентов m раз в году для увеличения наращенной суммы можно использовать следующие ренты:

1. Годовую ренту постнумерандо с непрерывным начислением процентов;
2. P-срочную ренту с ежегодным начислением процентов в конце года;
3. P-срочную ренту с ежегодным начислением процентов m раз в году, $p > m > 1$;
4. P-срочную ренту с ежегодным начислением процентов m раз в году, $p = m > 1$.

В итоге, для данного примера, пользователю будет предоставлен список, изображённый на рисунке 2.13.

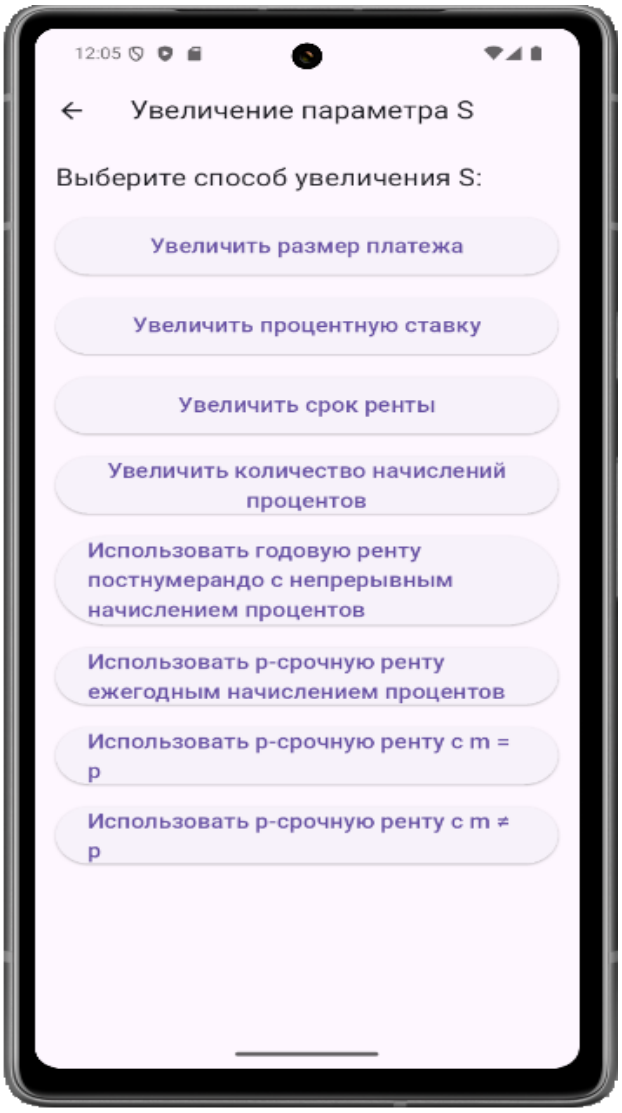


Рисунок 2.13 – Экран выбора способа увеличения параметра

При выборе какого-либо из способов увеличения (уменьшения) параметра пользователю отображается диалоговое окно, в котором расписаны пояснения к тому, почему изменение параметра влияет на увеличение (уменьшение) выбранного параметра. Также пояснения сопровождаются примером. В случае альтернативных потоков платежей, пояснения ограничиваются показательным примером. Примеры пояснений представлены на рисунке 2.14, а также в приложении Г.

Рисунок 2.14 – Примеры пояснений для альтернатив

Также стоит отметить, что, если в окне расчёта потока платежей были введены какие-либо значения, они будут использованы в пояснениях (рисунок 2.15).

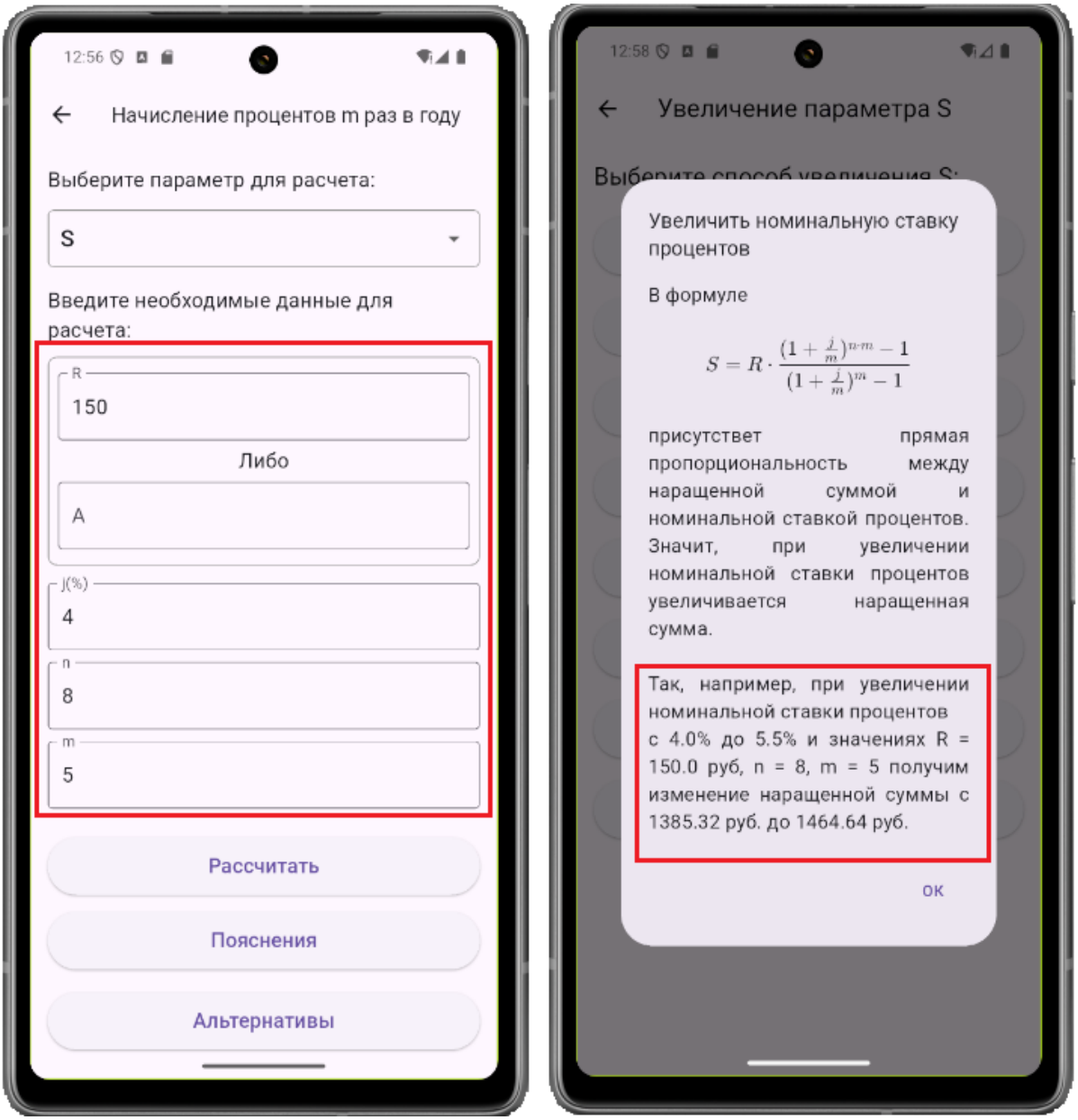


Рисунок 2.15 – Пример сохранения значений параметров

2.2.8 Особенности реализации некоторых потоков платежей

Несмотря на то, что в большинстве случаев разработка функционала, описывающего поток платежей, велась, используя одни и те же подходы и техники, в некоторых случаях приходилось учитывать специфику конкретного потока платежей. Так, например, для ренты с переменной процентной ставкой была необходима реализация дополнительного функционала, позволяющего пользователю добавлять и удалять периоды. Для обеспечения данного функционала на экран потока платежей была добавлена кнопка для добавления пары полей для ввода продолжительности периода и значения

процентной ставки в данный период времени. Для удаления какого-либо из периодов вне зависимости от порядка добавления для каждой пары полей была установлена специальная кнопка для удаления данной пары полей. Так как количество периодов может быть сколь угодно велико, на экране все поля для периодов располагаются на пролистываемой области, что позволяет эффективно просматривать любое количество полей. Результат реализации экрана для ренты с переменной процентной ставкой представлен на рисунке 2.16.

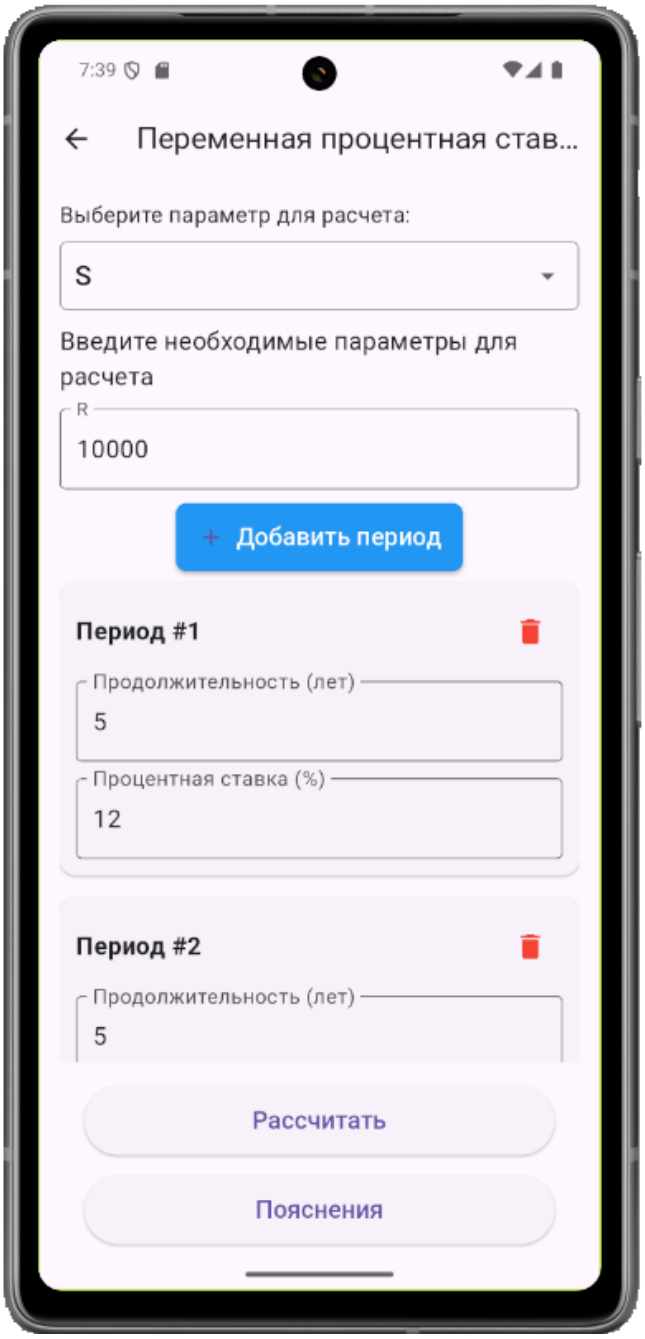


Рисунок 2.16 – Экран ренты с переменной процентной ставкой

Также на рисунке можно заметить, что для данного потока платежей не реализован функционал подбора альтернатив, что связано с отсутствием эффективного способа автоматического определения альтернативных потоков платежей, а также ограниченным набором рассчитываемых параметров (в качестве параметров для расчёта доступны: наращенная сумма, современная стоимость потока, размер члена ренты).

Ещё одним из уникальных случаев является реализация вечной ренты. Так как значения параметров продолжительности выплат и наращенной суммы для данного потока платежей всегда равны бесконечности, было принято решение отображать данную информацию сразу на экране расчёта, вместо экрана пояснений. Также для данных параметров отключены кнопки для пояснений, расчёта и подбора альтернатив. (рисунок 2.17).

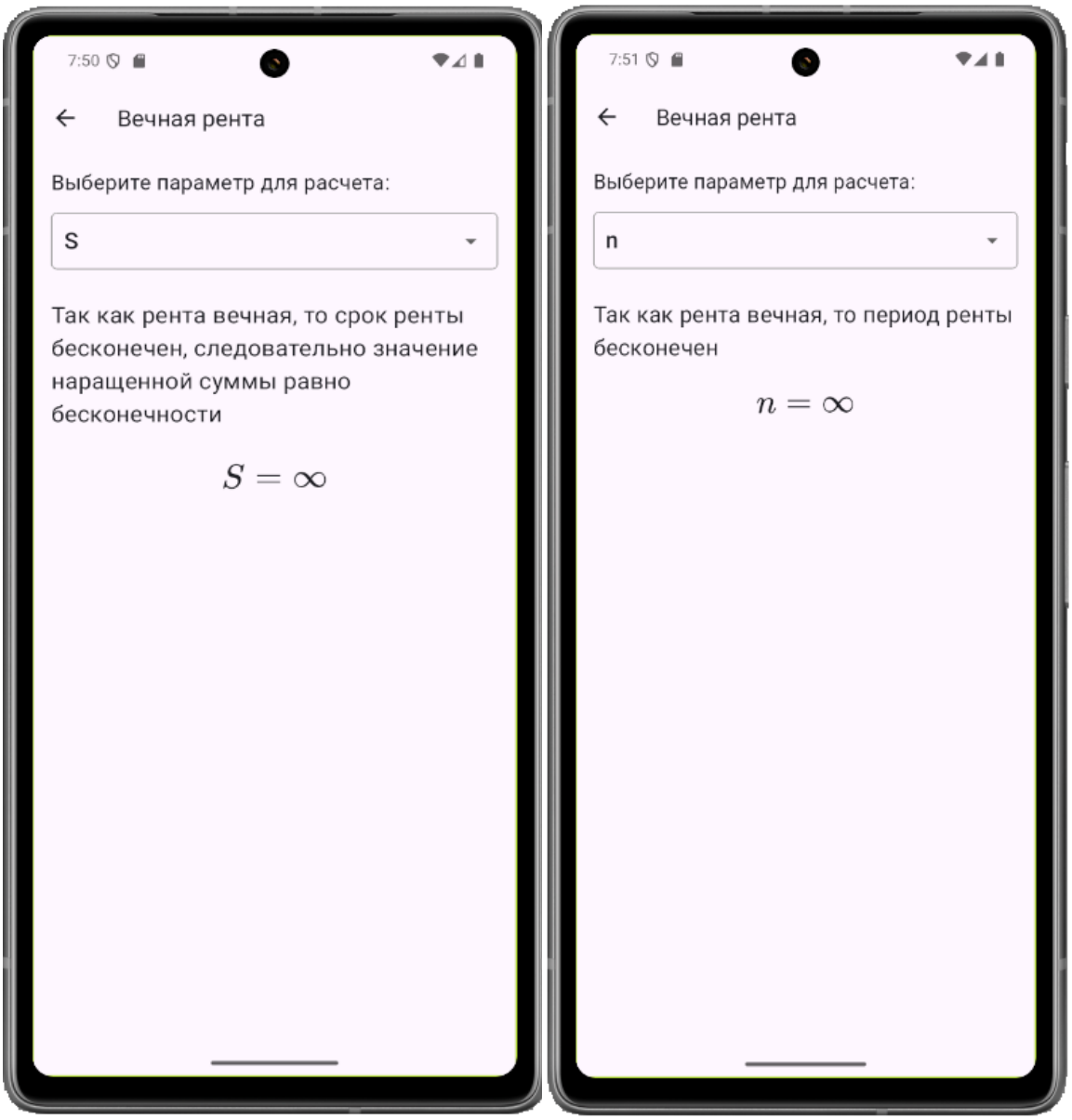


Рисунок 2.17 – Экраны расчёта наращенной суммы и продолжительности выплат для вечной ренты

При этом стоит отметить, что для остальных параметров внешний вид экрана остался стандартным (см. раздел 2.2.5).

Последней, но не менее важной особенностью является отсутствие возможности расчёта наращенной суммы, а также отсутствие возможности подбора альтернатив для отложенной ренты. Данный факт обоснован тем, что данная рента используется как дополнительная к какой-либо другой ренте, из-за чего не предоставляется возможным посчитать значение наращенной суммы и подобрать альтернативы. При этом стоит отметить, что сама рента используется как альтернатива для изменения параметра наращенной суммы (см. приложение Г).

2.2.9 Тестирование приложения

В условиях высокой конкуренции на рынке мобильных приложений и постоянно растущих пользовательских требований качество программного обеспечения становится критически важным фактором. В связи с этим тестирование программного обеспечения приобретает статус неотъемлемого этапа в процессе разработки.

Тестирование представляет собой комплекс мероприятий, направленных на проверку соответствия программного продукта установленным требованиям, выявление дефектов и оценку его качества. Этот процесс позволяет обнаружить ошибки на ранних стадиях разработки, повысить стабильность и производительность приложения, а также гарантировать положительный пользовательский опыт. Отсутствие должного уровня тестирования может привести к уязвимости, неудобству использования или полной неработоспособности приложения, что, в свою очередь, может повлечь за собой потерю пользователей и снижение репутации разработчиков.

Для тестирования приложения были созданы несколько видов тестов:

1. Unit тесты для слоя сервисов, которые позволяют проверить правильность реализации методов для расчёта параметров потоков платежей по формулам.
2. Тесты виджетов, которые без необходимости запуска приложения позволяют проверить, что на экране отображаются правильные UI элементы.
3. Интеграционные тесты для слоя контроллеров, которые проверяют правильность обновления данных.

Для написания тестов использовались дополнительные библиотеки для тестирования, в частности, библиотеки test и flutter_test, которые

предоставляют API для создания тестов и их запуска, а также библиотека mockito, которая позволяет создавать поддельные (mock) объекты, имитирующие поведение реальных зависимостей.

При создании тестов для слоя сервисов учитывалась возможность погрешностей из-за дробных значений, поэтому проверялось не точное совпадение с ожидаемым значением, а отличие от него не более чем на значение допустимой погрешности (выбирались значения в диапазоне от 10^{-2} до 10^{-3}).

Пример программной реализации unit теста для сервиса:

```
test('calculateIUsingS: вычисление i по S, R, n', () {
    double S = 184000;
    double R = 10000;
    int n = 10;
    double i = service.calculateIUsingS(S, R, n);
    expect(i, closeTo(0.13, 0.001));
  });
```

Полная реализация класса для тестирования сервиса на примере YearPostnumerandoRentService представлена в приложении Д.

Что касается создания тестов для UI, стоит отметить, что были протестированы именно мелкие компоненты и виджеты, из которых собирались страницы для потоков платежей. Такой подход позволил облегчить написание тестов (так как тест пишется только на отдельный компонент, а не на группу виджетов сразу) и исключить необходимость тестировать каждый класс View отдельно. При создании подобных тестов проверялась правильность рендеринга компонента, его косметические стили при появлении, соответствие отображаемого контента ожидаемому, а также корректное поведение при взаимодействии с ним.

Пример теста для UI компонента:

```
testWidgets('Отображает переданный label и стили', (
    WidgetTester tester,
  ) async {
    await tester.pumpWidget(
      MaterialApp(
        home: Scaffold(
          body: InputField(
            label: 'Test Label',
            controller: controller,
            onChanged: (value) => currentValue = value,
          ),
        ),
```

```
    ),
  );
  expect(find.text('Test Label'), findsOneWidget);
  final textField = tester.widget<TextField>(find.byType(TextField));
  expect(textField.style?.fontSize, 18);
  expect(textField.decoration?.labelStyle?.fontSize, 18);
  expect(textField.decoration?.border, isA<OutlineInputBorder>());
  expect(textField.keyboardType, TextInputType.number);
 });
```

Полная реализация класса для тестирования UI на примере InputField представлена в приложении Е.

При создании тестов для слоя контроллеров для изоляции зависимостей, которые не участвуют в тестировании, но могут значительно повлиять на его результаты, были использованы поддельные (mock) объекты таких классов. В данной группе тестов проверялась корректность обработки событий обновления параметра потока платежей, а также правильность выбора формулы, делегирования расчётов нужному методу сервиса и возвращению результата, который контроллер получил от сервиса.

Пример тестов для класса контроллера:

```
test('calculate: вычисление A через S, i, n', () {
  when(
    mockService.calculateAUsingS(12577.89, 0.05, 10),
  ).thenReturn(7721.73);
  model.S = 12577.89;
  model.i = 0.05;
  model.n = 10;
  controller.updateSelectedParameter(Parameters.A);
  expect(controller.calculate(), equals(7721.73));
  verify(mockService.calculateAUsingS(12577.89, 0.05, 10)).called(1);
 });
```

Полная реализация класса для тестирования контроллера на примере YearPostnumerandoRentController представлена в приложении Ж.

На рисунке 2.18 продемонстрирован пример успешного прохождения всех тестов для YearPostnumerandoRentController (аналогичный вид имеет любая группа тестов, описанных в данном разделе).

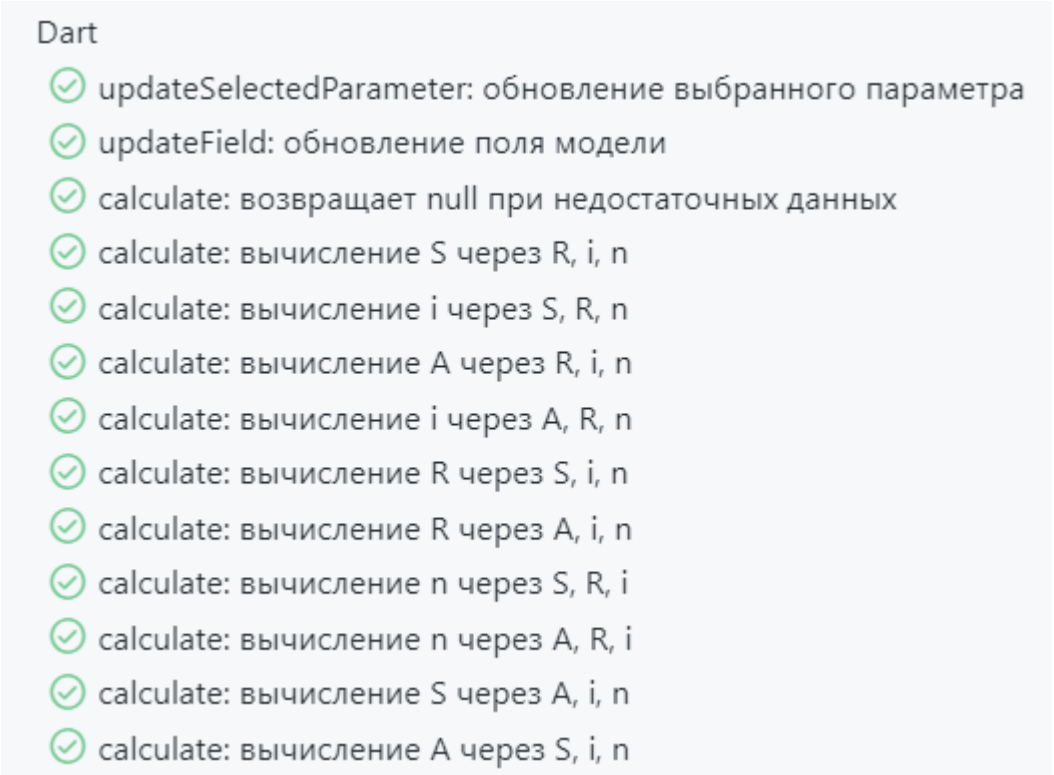


Рисунок 2.18 – Пример прохождения тестов

Стоит отметить, что также было проведено ручное тестирование приложения, чтобы проверить полный цикл его использования. На данном этапе проводилась проверка навигации, соответствие расположения элементов на экране и их контента ожидаемым, а также корректность работы в различных нестандартных вариантах использования приложения (например, ввод не валидных данных).

2.3 Результаты разработки приложения

Проанализировав процесс выбора технологий и создания приложения, можно убедиться, что оно полностью соответствует поставленным перед разработкой требованиям. При тестировании приложения ошибок в расчётах выявлено не было, из чего можно сделать выводы, что основной функционал приложения работает корректно в большинстве случаев.

Сравнение приложения с аналогами, приведёнными в разделе 1.2, представлено в таблице 2. Из таблицы можно заметить, что приложение поддерживает 12 потоков платежей (при этом стоит учесть, что не для всех потоков был реализован один и тот же функционал, как было продемонстрировано в разделе 2.2.8).

Таблица 2 – Сравнение решений по автоматизации потоков платежей

	MYFIN	Беларусбанк	Сбербанк	fingramota	Созданное приложение
Возможность выбора параметра	нет	нет	нет	частично	да
Пояснение расчётов	нет	частично	нет	нет	да
Количество видов потоков платежей для расчёта	6	2	2	8	12

Также стоит отметить, что полученное приложение не требует подключения к интернету, однако существуют и другие решения, не требующие интернета, поэтому нельзя однозначно выделить эту особенность как неоспоримое преимущество.

Разработанное решение может использоваться как в образовательных целях (например, для ознакомления с различными видами потоков платежей и процессом их расчёта), так и в повседневной жизни (например, для управления инвестициями).

Приложение разработано в первую очередь для мобильных платформ: Android и iOS. При этом используемые для его создания технологии позволяют адаптировать его для других платформ. Благодаря кроссплатформенному подходу, это же приложение может быть развёрнуто как веб-приложение, доступное через браузер, или как десктоп-приложение для Windows, macOS или Linux, что значительно расширяет его потенциальную аудиторию и сферы применения.

Ещё одним отличием является отсутствие потока платежей, аналогичного кредиту, однако архитектура приложения позволяет добавить его в будущем (впрочем, как и любой другой вид потоков платежей). Поэтому можно считать, что данное приложение является некоторым «ядром» с модульной архитектурой, которое позволяет в дальнейшем как добавлять новый функционал (например сравнение потоков платежей), так и редактировать уже существующий.

Таким образом, можно сделать вывод о том, что созданное приложение имеет существенные преимущества перед аналогами (большее количество возможных видов потоков платежей для расчёта, возможность динамического выбора параметров, а также наличия пояснений к расчётам) и при этом позволяет расширять функционал в будущем.

ЗАКЛЮЧЕНИЕ

Работа, посвящённая автоматизации финансовых расчётов, предполагающих наличие потоков платежей, состоит из двух частей: теоретической и практической. В теоретической части было рассмотрено понятие «поток платежей», определены основные параметры потоков платежей, дана классификация потоков платежей, а также определены формулы и методы для расчёта параметров в рамках отдельных потоков платежей, рассмотрены существующие решения для их автоматических расчётов. В практической части были определены основные требования к приложению для автоматического расчёта параметров потоков платежей, выбраны наиболее подходящие технологии для реализации, создана архитектура приложения, произведена разработка формул для расчёта, пояснений, экранов потоков платежей и альтернативных параметров, а также произведено тестирование для проверки соответствия поставленным требованиям, продемонстрирована работа приложения и некоторые его особенности. В конце практической части было произведено сравнение созданного приложения с аналогами и сделаны выводы касательно его преимуществ.

Созданное приложение, разработанное с использованием фреймворка Flutter, позволяет пользователям проводить расчёты различных параметров потоков платежей, получать пояснения того, как результат был рассчитан, а также находить возможные альтернативные наборы параметров и потоков платежей для получения наиболее оптимального результата. При этом имеется возможность использовать несколько формул для расчёта одного и того же параметра. Оно является кроссплатформенным, что позволяет ему работать на всех современных платформах, включая мобильные устройства, веб-браузеры и настольные операционные системы. Приложение поддерживает 12 потоков платежей, а его архитектура разработана с учетом принципов модульности и расширяемости, что позволяет в будущем без значительных изменений в кодовой базе добавлять реализации новых, более сложных видов потоков платежей.

Разработанное решение для автоматизации расчетов может поспособствовать повышению как финансовой грамотности населения, так и эффективности управления личными финансами. Созданное приложение является не только инструментом для упрощения финансовых расчетов, но и важным шагом в сторону цифровизации финансовой сферы, отвечающим современным требованиям и тенденциям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Единый портал финансовой грамотности: [Электронный ресурс]. – Режим доступа https://fingramota.by. Дата доступа 25.05.2025.
2. Калькулятор вкладов Беларусбанк [Электронный ресурс]. – Режим доступа https://belarusbank.by/ru/fizicheskim_licam/33357/vklady/kalkulyator_vkladov. – Дата доступа: 25.05.2025.
3. Кредитный калькулятор MYFIN [Электронный ресурс]. – Режим доступа https://myfin.by/kredity/kalkulyator. – Дата доступа: 25.05.2025.
4. Кредитный калькулятор СБЕР БАНК [Электронный ресурс]. – Режим доступа https://www.sberbank.com/ru/person/credits/money/kreditnyj_kalkulyator. – Дата доступа: 25.05.2025.
5. Морозов, А. А. Программирование задач численного анализа в система Mathematica / А. А. Морозов, В. Б. Таранчук. –Минск: БГПУ, 2005. – 145 с.
6. Оценка финансовой грамотности населения Республики Беларусь [Электронный ресурс]: отчет о выполнении научно-исследовательской работы по теме: «Оценка и анализ финансовой грамотности населения Республики Беларусь» / Национальная академия наук Беларуси. – Минск: Национальная академия наук Беларуси, 2022. – Режим доступа https://teacher.fingramota.by/ru/services/library/investigations/17. – Дата доступа: 25.05.2025.
7. Четыркин, Е. М. Финансовая математика / Е. М. Четыркин. – Москва: Академия народного хозяйства при правительстве Российской Федерации, 2000. – 398 с.
8. Якубова, У. Ш. Некоторые заметки о потоках платежей / У. Ш. Якубова, Н. Т. Парпиева, Н. Ш. Мирходжаева. – Ташкент: Бюллетень науки и практики, 2023. – с 321 – 328.
9. Consumer Loans, All Commercial Banks [Electronic resource]. – Mode of access: https://fred.stlouisfed.org/series/CONSUMER#. – Date of access: 25.05.2025.
10. Dart documentation [Electronic resource]. – Mode of access: https://dart.dev/guides. – Date of access: 25.05.2025.
11. Flutter documentation [Electronic resource]. – Mode of access: https://docs.flutter.dev. – Date of access: 25.05.2025.
12. Fowler, M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston: Addison Wesley, 2002. – 560 p.
13. Garret, S.J. An Introduction to the Mathematics of Finance. A Deterministic Approach / S.J. Garret. – Second Edition. – Elsevier, 2013. – 446 p.

14. LaTeX – A document preparation system [Electronic resource]. – Mode of access: https://www.latex-project.org. – Date of access: 25.05.2025.
15. US Total Deposits [Electronic resource]. – Mode of access: https://www.ceicdata.com/en/indicator/united-states/total-deposits. – Date of access: 25.05.2025.

ПРИЛОЖЕНИЕ А

РЕАЛИЗАЦИЯ РАСЧЁТОВ ПАРАМЕТРОВ ГОДОВОЙ РЕНТЫ ПОСТНУМЕРАНДО

```
class YearPostnumerandoRentService {
  static double calculateSUsingR(double R, double i, int n) {
    return R * ((pow(1 + i, n) - 1) / i);
  }

  static double calculateAUsingR(double R, double i, int n) {
    return R * ((1 - pow(1 + i, -n)) / i);
  }

  static double calculateRUsingS(double S, double i, int n) {
    return S * i / ((pow(1 + i, n) - 1));
  }

  static double calculateRUsingA(double A, double i, int n) {
    return A * i / ((1 - pow(1 + i, -n)));
  }

  static double calculateNUsigS(double S, double R, double i) {
    return log((S * i / R) + 1) / log(1 + i);
  }

  static double calculateNUsingA(double A, double R, double i) {
    return log(1 / (1 - (A * i / R))) / log(1 + i);
  }

  static double calculateIUsingS(double S, double R, int n,
      {double eps = 0.001}) { // eps – допустимая погрешность
    double iLower = 0.0001; // минимальное значение процентной ставки
    double iUpper = 1.0; // максимальное значение процентной ставки
    double iMid;

    while ((iUpper - iLower) > eps) {
      iMid = (iLower + iUpper) / 2;
```

```
    double testS = calculateSUsingR(R, iMid, n);

    if (testS < S) {
      iLower = iMid;
    } else {
      iUpper = iMid;
    }
  }
  return (iLower + iUpper) / 2;
}

static double calculateIUsingA(double A, double R, int n,
    {double eps = 0.001}) {
  double iLower = 0.0001;
  double iUpper = 1.0;
  double iMid;

  while ((iUpper - iLower) > eps) {
    iMid = (iLower + iUpper) / 2;
    double testA = calculateAUsingR(R, iMid, n);

    if (testA > A) {
      iLower = iMid;
    } else {
      iUpper = iMid;
    }
  }
  return (iLower + iUpper) / 2;
}

static double calculateSUsingA(double A, double i, int n) {
  return A * pow(1 + i, n);
}

static double calculateAUsingS(double S, double i, int n) {
  return S / pow(1 + i, n);
}
}
```

ПРИЛОЖЕНИЕ Б

РЕАЛИЗАЦИЯ КОНТРОЛЛЕРА ГОДОВОЙ РЕНТЫ ПОСТНУМЕРАНДО

```
class YearPostnumerandoRentController extends BaseController<TermRent> {
 final YearPostnumerandoRentService service;

 YearPostnumerandoRentController(super.model)
  : service = YearPostnumerandoRentService();

 @override
 Map<Parameters, double? Function()> createCalculationFunctions() {
  return {
   Parameters.S: () => _calculateS(),
   Parameters.A: () => _calculateA(),
   Parameters.R: () => _calculateR(),
   Parameters.i: () => _calculateI(),
   Parameters.n: () => _calculateN(),
  };
 }

 double? _calculateS() {
  if (model.R != null && model.i != null && model.n != null) {
   return service.calculateSUsingR(model.R!, model.i!, model.n!);
  }
  if (model.A != null && model.i != null && model.n != null) {
   return service.calculateSUsingA(model.A!, model.i!, model.n!);
  }
  return null;
 }

 double? _calculateA() {
  if (model.R != null && model.i != null && model.n != null) {
   return service.calculateAUsingR(model.R!, model.i!, model.n!);
  }
  if (model.S != null && model.i != null && model.n != null) {
   return service.calculateAUsingS(model.S!, model.i!, model.n!);
  }
  return null;
 }
```

```
  double? _calculateR() {
    if (model.S != null && model.i != null && model.n != null) {
      return service.calculateRUsingS(model.S!, model.i!, model.n!);
    }
    if (model.A != null && model.i != null && model.n != null) {
      return service.calculateRUsingA(model.A!, model.i!, model.n!);
    }
    return null;
  }

  double? _calculateI() {
    if (model.S != null && model.R != null && model.n != null) {
      return service.calculateIUsingS(model.S!, model.R!, model.n!) * 100;
    }
    if (model.A != null && model.R != null && model.n != null) {
      return service.calculateIUsingA(model.A!, model.R!, model.n!) * 100;
    }
    return null;
  }

  double? _calculateN() {
    if (model.S != null && model.R != null && model.i != null) {
      return service.calculateNUsigS(model.S!, model.R!, model.i!);
    }
    if (model.A != null && model.R != null && model.i != null) {
      return service.calculateNUsingA(model.A!, model.R!, model.i!);
    }
    return null;
  }
}
```

ПРИЛОЖЕНИЕ В

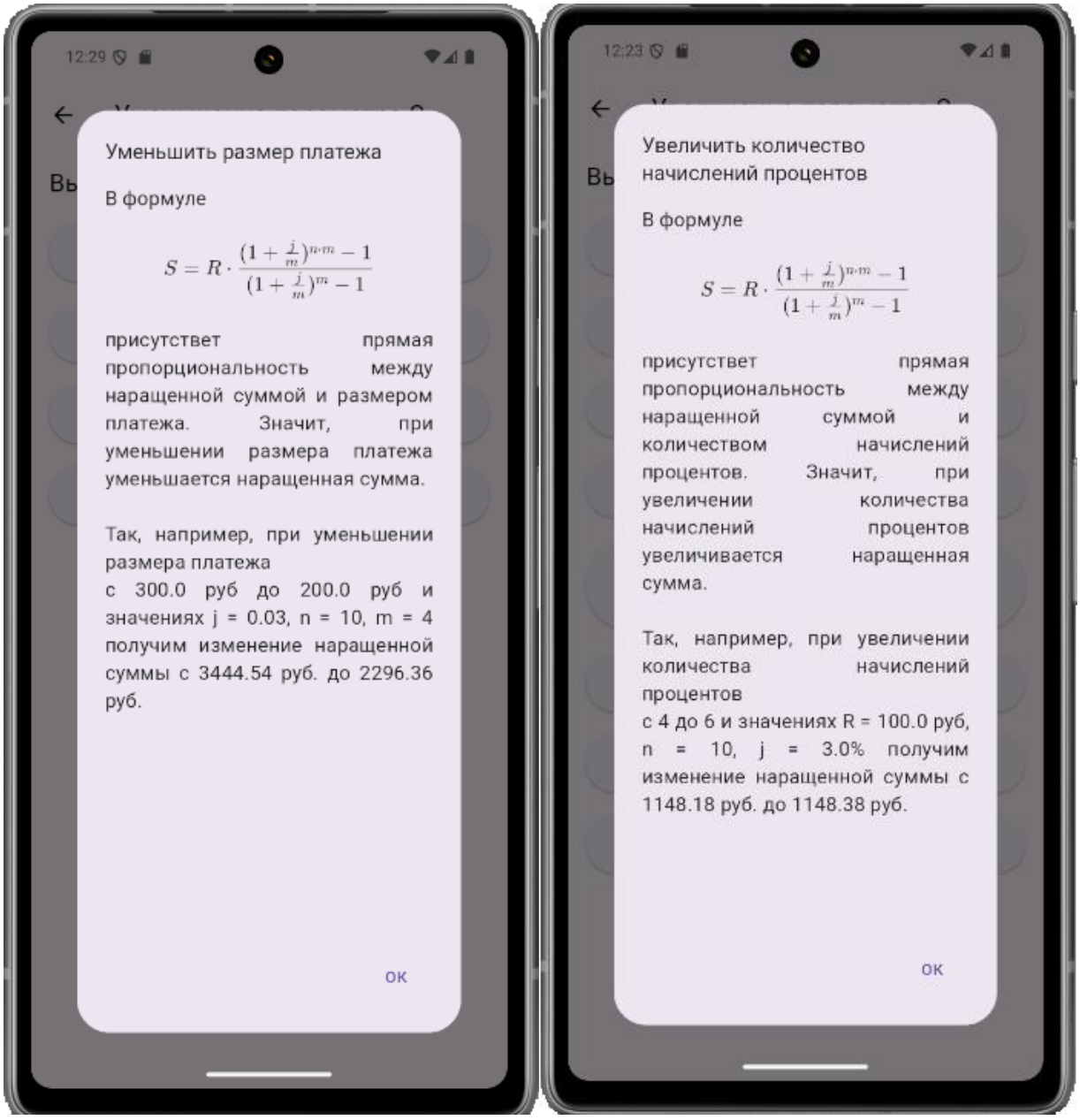
РЕАЛИЗАЦИЯ ИНТЕРФЕЙСА ГОДОВОЙ РЕНТЫ ПОСТНУМЕРАНДО

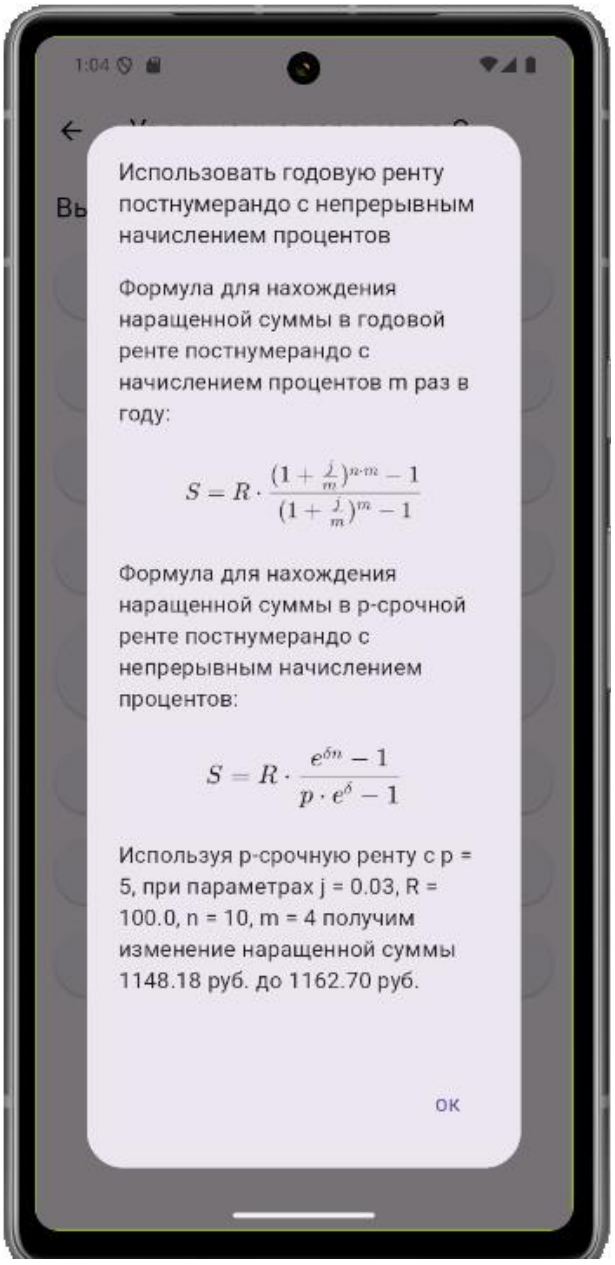
```
class YearPostnumerandoRentView extends StatefulWidget {
  final YearPostnumerandoRentController controller;

  const YearPostnumerandoRentView({super.key, required this.controller});

  @override
  State<StatefulWidget> createState() => _PostnumerandoRentViewState();
}

class _PostnumerandoRentViewState extends
State<YearPostnumerandoRentView> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Годовая рента постнумерандо',
          style: TextStyle(fontSize: 18),
        ),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            const Text(
              'Выберите параметр для расчета:',
              style: TextStyle(fontSize: 18),
            ),
            const SizedBox(height: 13),
            ParameterSelector(
              selectedParameter: widget.controller.selectedParameter,
              parameters: getTermParameters,
              onParameterChanged: (newValue) {
                setState(() {
                  widget.controller.updateSelectedParameter(
                    parameterFromString(newValue),
```

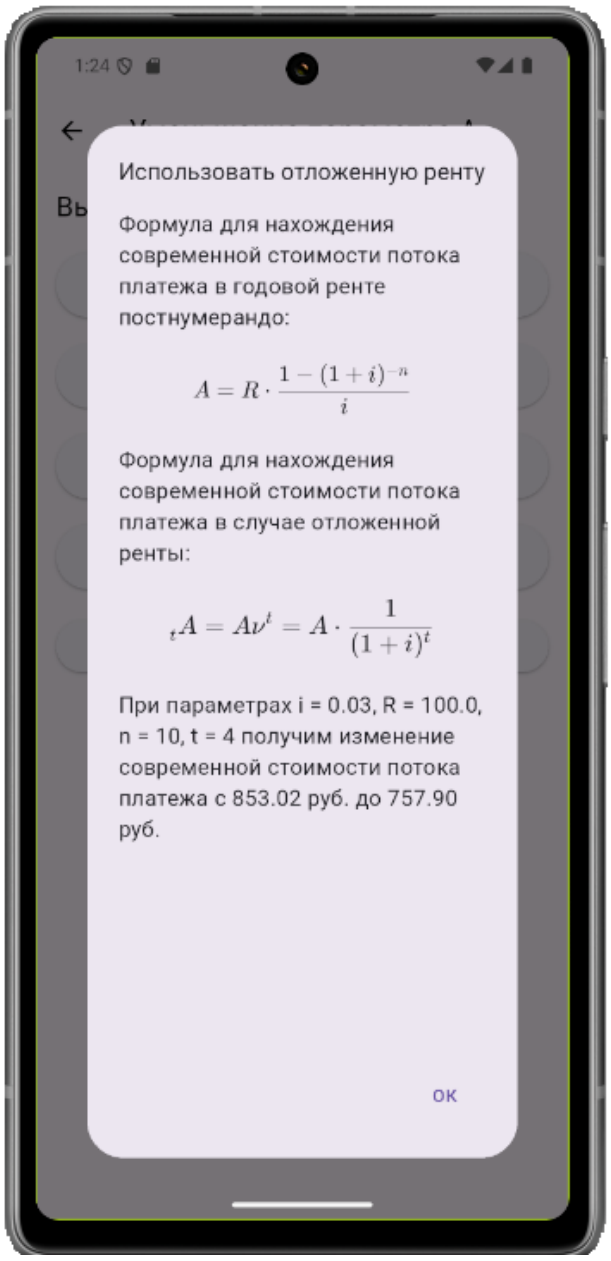
```
                );
              });
            },
          ),
          const SizedBox(height: 16),
          const Text(
            'Введите необходимые данные для расчета:',
            style: TextStyle(fontSize: 18),
          ),
          const SizedBox(height: 10),
          ParameterInputSection(
            selectedParameter: widget.controller.selectedParameter,
            inputFields: inputFields,
          ),
          const SizedBox(height: 20),
          ControlButtons(
            onCalculate: () {
              double? val = widget.controller.calculate();
              showDialog(
                context: context,
                builder: (BuildContext context) {
                  return ResultDialog(val: val);
                },
              );
            },
            onExplain: () {
              showDialog(
                context: context,
                builder: (BuildContext context) {
                  return ExplainDialog(
                    explanation:
                        explanations[Rents.YEAR_POSTNUMERANDO]![widget
                            .controller
                            .selectedParameter]!,
                  );
                },
              );
            },
          ),
        ],
      ),
    ),
  );
 }
}
```

ПРИЛОЖЕНИЕ Г

ПРИМЕРЫ ПОЯСНЕНИЙ К АЛЬТЕРНАТИВНЫМ ПАРАМЕТРАМ И ПОТОКАМ ПЛАТЕЖЕЙ




1:04
Использовать годовую ренту постнумерандо с непрерывным начислением процентов
Формула для нахождения наращенной суммы в годовой ренте постнумерандо с начислением процентов m раз в году:
$$S = R \cdot \frac{(1+\frac{j}{m})^{n \cdot m} - 1}{(1+\frac{j}{m})^{m} - 1}$$
Формула для нахождения наращенной суммы в p-срочной ренте постнумерандо с непрерывным начислением процентов:
$$S = R \cdot \frac{e^{\delta n} - 1}{p \cdot e^{\delta} - 1}$$
Используя p-срочную ренту с p = 5, при параметрах j = 0.03, R = 100.0, n = 10, m = 4 получим изменение наращенной суммы 1148.18 руб. до 1162.70 руб.
OK


1:24
Использовать отложенную ренту
Формула для нахождения современной стоимости потока платежа в годовой ренте постнумерандо:
$$A = R \cdot \frac{1 - (1+i)^{-n}}{i}$$
Формула для нахождения современной стоимости потока платежа в случае отложенной ренты:
$${}_tA = A\nu^t = A \cdot \frac{1}{(1+i)^t}$$
При параметрах i = 0.03, R = 100.0, n = 10, t = 4 получим изменение современной стоимости потока платежа с 853.02 руб. до 757.90 руб.
OK

ПРИЛОЖЕНИЕ Д

ПРИМЕР КЛАССА ДЛЯ ТЕСТИРОВАНИЯ ОТДЕЛЬНОГО СЕРВИСА

```
void main() {
  late YearPostnumerandoRentService service;
  setUp(() {
    service = YearPostnumerandoRentService();
  });
  group('Тесты для YearPostnumerandoRentService', () {
    test('calculateSUsingR: вычисление S по R, i, n', () {
      expect(service.calculateSUsingR(1000, 0.05, 10), closeTo(12577.89, 0.01));
    });
    test('calculateAUsingR: вычисление A по R, i, n', () {
      expect(service.calculateAUsingR(1000, 0.05, 10), closeTo(7721.73, 0.01));
    });
    test('calculateIUsingS: вычисление i по S, R, n', () {
      double S = 184000;
      double R = 10000;
      int n = 10;
      double i = service.calculateIUsingS(S, R, n);
      expect(i, closeTo(0.13, 0.001));
    });
    test('calculateIUsingA: вычисление i по A, R, n', () {
      double A = 54262.4;
      double R = 10000;
      int n = 10;
      double i = service.calculateIUsingA(A, R, n);
      expect(i, closeTo(0.13, 0.001));
    });
    test('calculateRUsingS: вычисление R по S, i, n', () {
      expect(
        service.calculateRUsingS(12577.89, 0.05, 10),
        closeTo(1000, 0.001),
      );
    });
    test('calculateRUsingA: вычисление R по A, i, n', () {
```

```
    expect(service.calculateRUsingA(7721.73, 0.05, 10), closeTo(1000, 0.001));
  });
  test('calculateNUsigS: вычисление n по S, R, i', () {
    expect(service.calculateNUsigS(12577.89, 1000, 0.05), closeTo(10, 0.01));
  });
  test('calculateNUsingA: вычисление n по A, R, i', () {
    expect(service.calculateNUsingA(7721.73, 1000, 0.05), closeTo(10, 0.01));
  });
  test('calculateSUsingA: вычисление S по A, i, n', () {
    expect(
      service.calculateSUsingA(7721.73, 0.05, 10),
      closeTo(12577.89, 0.01),
    );
  });
  test('calculateAUsingS: вычисление A по S, i, n', () {
    expect(
      service.calculateAUsingS(12577.89, 0.05, 10),
      closeTo(7721.73, 0.01),
    );
  });
 });
}
```

ПРИЛОЖЕНИЕ Е

ПРИМЕР КЛАССА ДЛЯ ТЕСТИРОВАНИЯ UI КОМПОНЕНТА

```
void main() {
  group('InputField Widget Tests', () {
    late TextEditingController controller;
    late String currentValue;
    setUp(() {
      controller = TextEditingController();
      currentValue = '';
    });
    tearDown(() {
      controller.dispose();
    });
    testWidgets('Отображает переданный label и стили', (
      WidgetTester tester,
    ) async {
      await tester.pumpWidget(
        MaterialApp(
          home: Scaffold(
            body: InputField(
              label: 'Test Label',
              controller: controller,
              onChanged: (value) => currentValue = value,
            ),
          ),
        ),
      );
      expect(find.text('Test Label'), findsOneWidget);
      final textField = tester.widget<TextField>(find.byType(TextField));
      expect(textField.style?.fontSize, 18);
      expect(textField.decoration?.labelStyle?.fontSize, 18);
      expect(textField.decoration?.border, isA<OutlineInputBorder>());
      expect(textField.keyboardType, TextInputType.number);
    });
    testWidgets('Обновляет значение через TextEditingController', (
```

```
  WidgetTester tester,
) async {
  await tester.pumpWidget(
    MaterialApp(
      home: Scaffold(
        body: InputField(
          label: 'Test',
          controller: controller,
          onChanged: (value) => currentValue = value,
        ),
      ),
    ),
  );
  await tester.enterText(find.byType(TextField), '123');
  expect(controller.text, '123');
  expect(currentValue, '123');
});
testWidgets('Вызывает onChanged при вводе текста', (
  WidgetTester tester,
) async {
  bool callbackCalled = false;
  await tester.pumpWidget(
    MaterialApp(
      home: Scaffold(
        body: InputField(
          label: 'Test',
          controller: controller,
          onChanged: (value) {
            callbackCalled = true;
            currentValue = value;
          },
        ),
      ),
    ),
  );
  await tester.enterText(find.byType(TextField), 'abc');
  expect(callbackCalled, isTrue);
  expect(currentValue, 'abc');
});
```

```
  testWidgets('Применяет marginTop и marginBottom', (
    WidgetTester tester,
  ) async {
    await tester.pumpWidget(
      MaterialApp(
        home: Scaffold(
          body: InputField(
            label: 'Test',
            controller: controller,
            onChanged: (value) {},
            marginTop: 10,
            marginBottom: 20,
          ),
        ),
      ),
    );
    final padding = tester.widget<Padding>(find.byType(Padding));
    expect(padding.padding, equals(EdgeInsets.only(top: 10, bottom: 20)));
  });
 });
}
```

ПРИЛОЖЕНИЕ Ж

ПРИМЕР КЛАССА ДЛЯ ТЕСТИРОВАНИЯ КОНТРОЛЛЕРА

```
@GenerateMocks([YearPostnumerandoRentService])
import 'year_postnumerando_rent_controller_test.mocks.dart';

void main() {
  late YearPostnumerandoRentController controller;
  late MockYearPostnumerandoRentService mockService;
  late TermRent model;
  setUp(() {
    mockService = MockYearPostnumerandoRentService();
    model = TermRent();
    controller = YearPostnumerandoRentController(model)..service = mockService;
  });
  group('Проверка YearPostnumerandoRentController', () {
    test('updateSelectedParameter: обновление выбранного параметра', () {
      controller.updateSelectedParameter(Parameters.S);
      expect(controller.selectedParameter, equals(Parameters.S));
      expect(model.selectedParameter, equals(Parameters.S));
    });
    test('updateField: обновление поля модели', () {
      controller.updateField('R', 1000.0);
      expect(model.R, equals(1000.0));
      controller.updateField('i', 0.0);
      expect(model.i, isNull);
    });
    test('calculate: вычисление S через R, i, n', () {
      when(mockService.calculateSUsingR(1000.0, 0.05, 10)).thenReturn(12577.89);
      model.R = 1000.0;
      model.i = 0.05;
      model.n = 10;
      controller.updateSelectedParameter(Parameters.S);
      expect(controller.calculate(), equals(12577.89));
      verify(mockService.calculateSUsingR(1000.0, 0.05, 10)).called(1);
    });
    test('calculate: вычисление i через S, R, n', () {
```

```
  when(
    mockService.calculateIUsingS(12577.89, 1000.0, 10, eps: 0.001),
  ).thenReturn(0.05);
  model.S = 12577.89;
  model.R = 1000.0;
  model.n = 10;
  controller.updateSelectedParameter(Parameters.i);
  expect(controller.calculate(), equals(5.0));
  verify(
    mockService.calculateIUsingS(12577.89, 1000.0, 10, eps: 0.001),
  ).called(1);
});
test('calculate: возвращает null при недостаточных данных', () {
  controller.updateSelectedParameter(Parameters.S);
  expect(controller.calculate(), isNull);
});
test('calculate: вычисление A через R, i, n', () {
  when(mockService.calculateAUsingR(1000.0, 0.05, 10)).thenReturn(7721.73);
  model.R = 1000.0;
  model.i = 0.05;
  model.n = 10;
  controller.updateSelectedParameter(Parameters.A);
  expect(controller.calculate(), equals(7721.73));
  verify(mockService.calculateAUsingR(1000.0, 0.05, 10)).called(1);
});
test('calculate: вычисление i через A, R, n', () {
  when(
    mockService.calculateIUsingA(7721.73, 1000.0, 10, eps: 0.001),
  ).thenReturn(0.05);
  model.A = 7721.73;
  model.R = 1000.0;
  model.n = 10;
  controller.updateSelectedParameter(Parameters.i);
  expect(controller.calculate(), equals(5.0));
  verify(
    mockService.calculateIUsingA(7721.73, 1000.0, 10, eps: 0.001),
  ).called(1);
});
test('calculate: вычисление R через S, i, n', () {
  when(mockService.calculateRUsingS(12577.89, 0.05, 10)).thenReturn(1000.0);
```

```
  model.S = 12577.89;
  model.i = 0.05;
  model.n = 10;
  controller.updateSelectedParameter(Parameters.R);
  expect(controller.calculate(), equals(1000.0));
  verify(mockService.calculateRUsingS(12577.89, 0.05, 10)).called(1);
});
test('calculate: вычисление R через A, i, n', () {
  when(mockService.calculateRUsingA(7721.73, 0.05, 10)).thenReturn(1000.0);
  model.A = 7721.73;
  model.i = 0.05;
  model.n = 10;
  controller.updateSelectedParameter(Parameters.R);
  expect(controller.calculate(), equals(1000.0));
  verify(mockService.calculateRUsingA(7721.73, 0.05, 10)).called(1);
});
test('calculate: вычисление n через S, R, i', () {
  when(
    mockService.calculateNUsigS(12577.89, 1000.0, 0.05),
  ).thenReturn(10.0);
  model.S = 12577.89;
  model.R = 1000.0;
  model.i = 0.05;
  controller.updateSelectedParameter(Parameters.n);
  expect(controller.calculate(), equals(10.0));
  verify(mockService.calculateNUsigS(12577.89, 1000.0, 0.05)).called(1);
});
test('calculate: вычисление n через A, R, i', () {
  when(
    mockService.calculateNUsingA(7721.73, 1000.0, 0.05),
  ).thenReturn(10.0);
  model.A = 7721.73;
  model.R = 1000.0;
  model.i = 0.05;
  controller.updateSelectedParameter(Parameters.n);
  expect(controller.calculate(), equals(10.0));
  verify(mockService.calculateNUsingA(7721.73, 1000.0, 0.05)).called(1);
});
test('calculate: вычисление S через A, i, n', () {
  when(
```

```
      mockService.calculateSUsingA(7721.73, 0.05, 10),
    ).thenReturn(12577.89);
    model.A = 7721.73;
    model.i = 0.05;
    model.n = 10;
    controller.updateSelectedParameter(Parameters.S);
    expect(controller.calculate(), equals(12577.89));
    verify(mockService.calculateSUsingA(7721.73, 0.05, 10)).called(1);
  });
  test('calculate: вычисление A через S, i, n', () {
    when(
      mockService.calculateAUsingS(12577.89, 0.05, 10),
    ).thenReturn(7721.73);
    model.S = 12577.89;
    model.i = 0.05;
    model.n = 10;
    controller.updateSelectedParameter(Parameters.A);
    expect(controller.calculate(), equals(7721.73));
    verify(mockService.calculateAUsingS(12577.89, 0.05, 10)).called(1);
  });
 });
}
```