Описание проекта

Содержание

1. Описание физической архитектуры системы

Физическая архитектура

Система представляет собой мобильное приложение, разработанное на языке программирования Dart версии 3.3.4 с использованием фреймворка Flutter версии 3.19.6. Приложение работает на операционной системе Android 6 и выше, а также на iOS старше 12 версии. Для хранения данных используется локальная база данных SQLite3, взаимодействие с которой осуществляется через пакет drift версии 2.17.0.

Для разработки был выбран фреймворк Flutter, так как он позволяет создавать кроссплатформенные мобильные приложения, а это позволяет сэкономить время и ресурсы на разработку и поддержку приложения. Фреймворк позволяет создавать красивые и быстрые приложения, которые могут работать на разных устройствах и операционных системах, таких как Android, iOS, Windows, macOS и Linux без изменения кода. Также Flutter имеет большое сообщество разработчиков, что позволяет быстро найти ответы на вопросы и решения проблем, а также использовать готовые решения и пакеты, что ускоряет разработку. Еще одним преимуществом Flutter является то, что он разрабатывается и поддерживается компанией Google, что гарантирует его актуальность и поддержку в будущем.

SQLite3 была выбрана в качестве базы данных, так как она является легковесной и простой в использовании, что позволяет быстро и легко создавать и изменять базу данных. К тому же данная

база данных позволяет хранить данные локально на устройстве пользователя, что обеспечивает быстрый доступ к данным и работу приложения без подключения к интернету. Также SQLite3 поддерживается пакетом drift, который позволяет работать с базой данных на языке Dart, что упрощает взаимодействие с базой данных и позволяет использовать привычный язык программирования.

В рамках текущей реализации было решено разработать полностью автономное приложение, не требующее подключения к интернету. Все данные хранятся локально на устройстве пользователя. Такой выбор обоснован тем, что приложение расчитано на использование одним пользователем и не предполагает совместной работы нескольких пользователей. В дальнейшем планируется коммерциализация приложения путем добавления возможности синхронизации данных с облачным хранилищем, что позволит пользователю иметь доступ к своим данным с разных устройств. Также это позволит реализовать возможность делиться данными с другими пользователями.

Структура проекта

Проект разделен на несколько основных частей:

- `lib` - основная директория, в которой находятся все файлы приложения
- `lib/src/` - директория, в которой находятся точка входа в приложение и файлы, отвечающие за инициализацию приложения
- `lib/src/common` - директория, в которой находятся общие для всего приложения файлы, а точнее файлы, содержащие общие константы, стили, виджеты, настройки маршрутизации и т.д.
- `lib/src/feature` - директория, в которой находятся файлы, отвечающие за отдельные части приложения
- `lib/src/feature/{feature_name}` - директория, в которой находятся файлы, отвечающие за возможности приложения, связанные с определенной областью жизни пользователя.

В нашем случае приложение разделено на 4 основные возможности:

- `lib/src/feature/tasks` - директория, в которой находятся файлы, отвечающие за отображение списка задач, т.ч. добавление и редактирование задач, отображение деталей задачи, фильтрация задач по статусу и приоритету, удаление задач, взятие задачи в работу, завершение задачи, а также самое основное - запуск алгоритма вычисления важности задач, позволяющего определить и предложить к решению наиболее важные на текущий момент задачи, что поможет пользователю оптимизировать свое время, повысить производительность и выполнить все задачи в срок.
- `lib/src/feature/calendar` - директория, в которой находятся файлы, отвечающие за работу с календарем, т.ч. отображение расписания и задач, добавление и редактирование занятий, выбор активной недели.
- `lib/src/feature/notes` - директория, в которой находятся файлы, отвечающие за работу с заметками, т.ч. добавление и редактирование заметок, удаление заметок, добавление в избранное. Заметки позволяют писать текст в markdown, что позволяет пользователю форматировать текст, добавлять ссылки, изображения, списки и т.д.
- `lib/src/feature/resources` - директория, в которой находятся файлы, отвечающие за работу с ресурсами, т.ч. добавление и редактирование ресурсов, удаление ресурсов, добавление в избранное. Ресурсы позволяют пользователю хранить ссылки на веб-страницы,

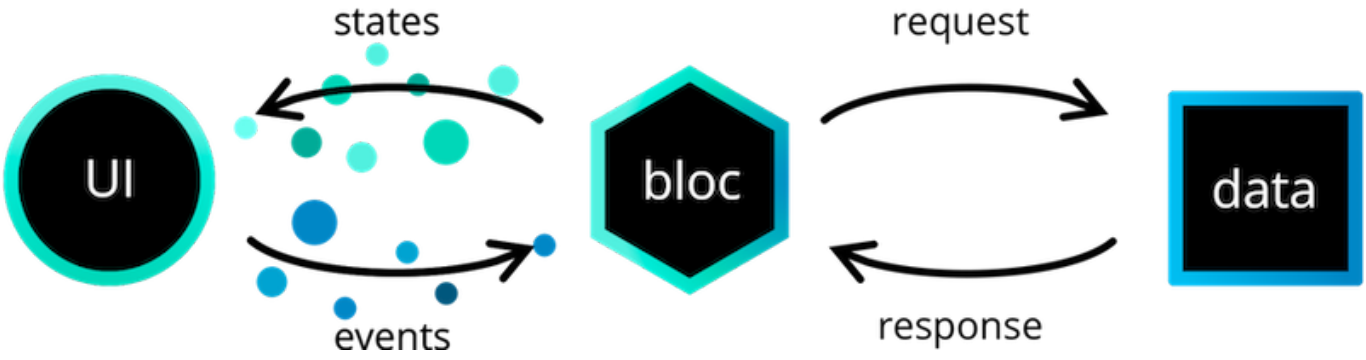
а в будущем планируется добавить возможность хранить файлы, что позволит пользователю хранить важные файлы на устройстве и иметь к ним доступ в любое время.

Каждая feature, как уже было сказано, отвечает за определенную область жизни пользователя. Каждая из них состоит из нескольких файлов, отвечающих за разные части функционала. Например, в feature tasks есть следующие файлы:

- `lib/src/feature/tasks/tasks_screen.dart` - файл, отвечающий за отображение списка задач, а также за добавление и редактирование задач. В этом файле находится виджет, который отображает список задач, а также кнопку добавления новой задачи. При нажатии на кнопку открывается новая страница, на которой пользователь может ввести название задачи, описание, приоритет, крайний срок и т.д. После ввода данных задача сохраняется в базе данных и отображается в списке задач.
- `lib/src/feature/tasks/task_put/task_put_screen.dart` - файл, отвечающий за редактирование задачи. В этом файле находится виджет, который отображает данные задачи, а также кнопку сохранения изменений. При нажатии на кнопку данные задачи обновляются в базе данных и отображаются в списке задач. На этой же странице можно удалить задачу.
- `lib/src/feature/tasks/entities/` - директория, в которой находятся файлы, отвечающие за сущности задач, т.ч. базовая сущность задачи и обычная, содеожащая дополнительные поля, такие как список идентификаторов подзадач и задач, от которых зависит данная задача.

Основной код логики каждой feature находится в файле `lib/src/feature/{feature_name}/bloc/{feature_name}_bloc.dart`, где находится класс, отвечающий за бизнес-логику данной feature. В случае с tasks это класс `TasksBloc`, который отвечает за работу с задачами, а именно за добавление, редактирование, удаление задач, а также за запуск алгоритма вычисления важности задач.

Bloc это паттерн, который позволяет разделить логику приложения на несколько частей, что упрощает разработку и поддержку приложения. В данном случае bloc отвечает за бизнес-логику, а виджеты за отображение данных и взаимодействие с пользователем. Таким образом, bloc отвечает за обработку событий, таких как добавление задачи, редактирование задачи, удаление задачи, запуск алгоритма вычисления важности задач и т.д. Также bloc отвечает за обработку запросов к базе данных, таких как получение списка задач, добавление задачи в базу данных, обновление задачи в базе данных и т.д.



В этом классе TaskBloc находятся методы, отвечающие за обработку событий, таких как добавление задачи, редактирование задачи, удаление задачи, запуск алгоритма вычисления важности задач. Также в этом классе находятся методы, отвечающие за запросы к базе данных, такие как получение списка задач, добавление задачи в базу данных, обновление задачи в базе данных и т.д. Таким

образом, данный класс отвечает за бизнес-логику приложения, а также за взаимодействие с базой данных.

На уровне с виджетами происходит отображение данных и взаимодействие с пользователем. Виджеты находятся в файлах `lib/src/feature/{feature_name}/widgets/`, где находятся виджеты, отвечающие за отображение данных и взаимодействие с пользователем. Например, в файле `lib/src/feature/tasks/widgets/tasks_screen.dart` находится виджет, отвечающий за отображение списка задач, а также за добавление новой задачи. В этом виджете находится список задач, а также кнопка добавления новой задачи. При нажатии на кнопку открывается новая страница, на которой пользователь может ввести название задачи, описание, приоритет, крайний срок и т.д. После ввода данных задача сохраняется в базе данных и отображается в списке задач.

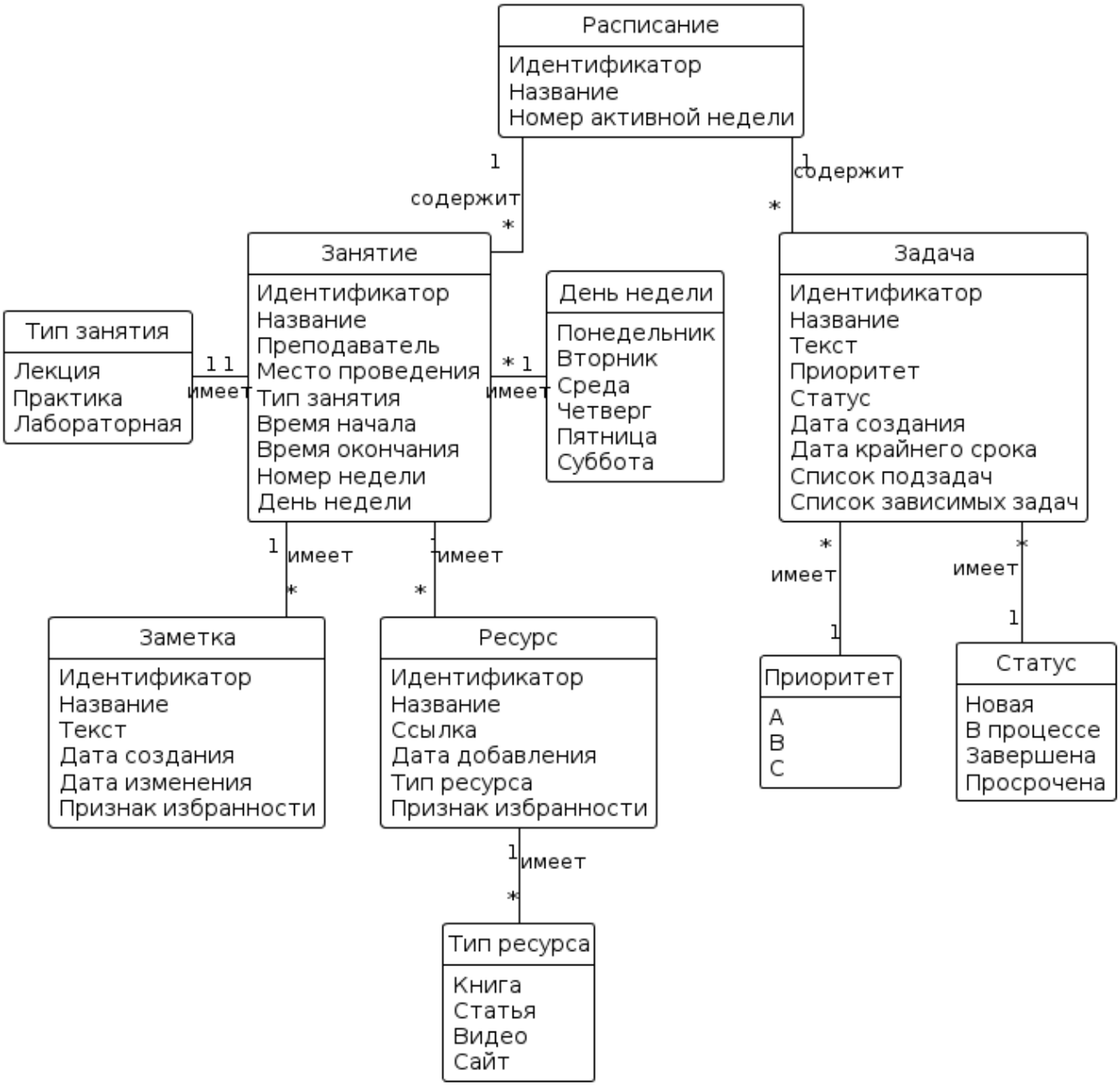
Также на уровне с директорией `lib` находится `packages` - директория, в которой находятся исходный код некоторых саморучно написанных библиотек, которые используются в проекте. В данной директории находятся следующие библиотеки:

- `packages/stook_database` - библиотека, отвечающая за работу с базой данных SQLite3. В данной библиотеке находятся классы, отвечающие за создание и обновление базы данных, а также за запросы к базе данных, такие как получение списка задач, добавление задачи в базу данных, обновление задачи в базе данных и т.д.
- `packages/stook_algorithm` - библиотека, отвечающая за алгоритм вычисления важности задач. В данной библиотеке находится класс, отвечающий за запуск алгоритма, который позволяет определить и предложить к решению наиболее важные на текущий момент задачи. Алгоритм основан на методе взвешенных сумм, который позволяет определить важность задачи на основе ее приоритета, статуса, крайнего срока и т.д. Исходный код библиотеки состоит из нескольких классов:
  - `AlgorithmItem`, представляющий собой элемент алгоритма, содержащий данные о задаче, такие как название, описание, приоритет, крайний срок, статус и т.д.
  - `AlgorithmResult`, представляющий собой результат алгоритма, содержащий список элементов алгоритма, отсортированных по важности, и дату и время запуска алгоритма.
  - `AlgorithmDataPreparer`, реализующий интерфейс `IAlgorithmDataPreparer`, отвечающий за подготовку данных для алгоритма.
  - `AlgorithmSolver`, реализующий интерфейс `IAlgorithmSolver`, отвечающий за запуск алгоритма.
  - `AlgorithmRunner`, реализующий интерфейс `IAlgorithmRunner`, инкапсулирующий логику алгоритма, а то есть подготовку данных, запуск алгоритма и обработку результата.

Каждый из классов внедряется в проект при помощи пакета `injector`, который позволяет внедрять зависимости в классы. Таким образом, при создании экземпляра класса `TasksBloc` внедряются зависимости `StookDatabase` и `AlgorithmRunner`, что позволяет использовать эти классы внутри `TasksBloc`. В случае необходимости замены реализации базы данных или алгоритма вычисления важности задач, достаточно заменить реализацию внутри `injector`, что позволяет легко изменять зависимости и упрощает тестирование.

2. Описание даталогической архитектуры системы

ER-диаграмма



ER-диаграмма представляет собой схему базы данных, которая показывает связи между сущностями и атрибутами. В данной схеме представлены следующие сущности:

- Задача - сущность, отвечающая за задачи. В данной сущности хранятся данные о задаче, такие как название, описание, приоритет, крайний срок, статус, список идентификаторов подзадач и задач, от которых зависит данная задача.
- Занятие - сущность, отвечающая за занятия. В данной сущности хранятся данные о занятии, такие как название, описание, дата и время начала и окончания, преподаватель, аудитория, тип занятия, номер недели и день недели.
- Заметка - сущность, отвечающая за заметки. В данной сущности хранятся данные о заметке, такие как название, текст, дата и время создания, дата и время последнего изменения, флаг избранное.
- Ресурс - сущность, отвечающая за ресурсы. В данной сущности хранятся данные о ресурсе, такие как название, описание, ссылка, дата и время создания, тип ресурса, флаг избранное.
- Расписание - сущность, отвечающая за расписание. В данной сущности хранятся данные о расписании, такие как название и номер активной недели.

Описание сущностей и атрибутов

Описание сущностей и атрибутов таким образом, как он представлен на ER-диаграмме и в базе данных. Разработанный пакет `stoop_database` позволяет работать с базой данных SQLite3 на языке Dart. В данном пакете определены следующие сущности:

Задача представляет собой сущность, отвечающую за задачи, которые пользователь хочет выполнить. В данной сущности хранятся следующие атрибуты:

- `id` - идентификатор задачи, уникальный идентификатор задачи.
- `title` - название задачи, строка, содержащая название задачи.
- `description` - описание задачи, строка, содержащая описание задачи.
- `priority` - приоритет задачи, строка, содержащая приоритет задачи.
- `createdDate` - дата и время создания задачи, дата и время, когда задача была создана.
- `deadlineDate` - крайний срок задачи, дата и время, когда задачу нужно выполнить.
- `status` - статус задачи, строка, содержащая статус задачи.

`Priority` и `Status` - это перечисления, которые содержат возможные значения приоритета и статуса задачи соответственно.

Занятие представляет собой сущность, отвечающую за занятия, которые пользователь хочет посетить. В данной сущности хранятся следующие атрибуты:

- `id` - идентификатор занятия, уникальный идентификатор занятия.
- `title` - название занятия, строка, содержащая название занятия.
- `description` - описание занятия, строка, содержащая описание занятия.
- `timeStart` - дата и время начала занятия, дата и время, когда занятие начинается.
- `timeEnd` - дата и время окончания занятия, дата и время, когда занятие заканчивается.
- `teacher` - преподаватель занятия, строка, содержащая имя преподавателя.
- `place` - аудитория занятия, строка, содержащая номер аудитории.
- `type` - тип занятия, строка, содержащая тип занятия.
- `weekNumber` - номер недели, на которой проходит занятие, целое число.
- `dayOfWeek` - день недели, на который проходит занятие, строка, содержащая день недели.

`Type` и `DayOfWeek` - это перечисления, которые содержат возможные значения типа занятия и дня недели соответственно.

Заметка представляет собой сущность, отвечающую за заметки, которые пользователь хочет сохранить. В данной сущности хранятся следующие атрибуты:

- `id` - идентификатор заметки, уникальный идентификатор заметки.
- `title` - название заметки, строка, содержащая название заметки.
- `text` - текст заметки, строка, содержащая текст заметки. Поддерживает markdown.
- `createdDate` - дата и время создания заметки, дата и время, когда заметка была создана.
- `lastModifiedDate` - дата и время последнего изменения заметки, дата и время, когда заметка была изменена.
- `isFavorite` - флаг избранное, булево значение, показывающее, является ли заметка избранной.

Ресурс представляет собой сущность, отвечающую за ресурсы, которые пользователь хочет сохранить. В данной сущности хранятся следующие атрибуты:

- `id` - идентификатор ресурса, уникальный идентификатор ресурса.
- `title` - название ресурса, строка, содержащая название ресурса.
- `description` - описание ресурса, строка, содержащая описание ресурса.
- `url` - ссылка на ресурс, строка, содержащая ссылку на ресурс.
- `createdDate` - дата и время создания ресурса, дата и время, когда ресурс был создан.
- `type` - тип ресурса, строка, содержащая тип ресурса.
- `isFavorite` - флаг избранное, булево значение, показывающее, является ли ресурс избранным.

`Type` - это перечисление, которое содержит возможные значения типа ресурса.

Описание связей между сущностями

Связь задачи с подзадачами реализована через таблицу `task_subtask`, которая содержит два поля: `task_id` и `subtask_id`, которые являются идентификаторами задачи и подзадачи соответственно. Таким образом, если задача зависит от другой задачи, то в таблице `task_subtask` будет создана запись с идентификаторами этих задач.

Связь задачи с задачами, от которых она зависит, реализована через таблицу `task_dependency`, которая содержит два поля: `task_id` и `dependency_id`, которые являются идентификаторами задачи и задачи, от которой зависит данная задача соответственно. Таким образом, если задача зависит от другой задачи, то в таблице `task_dependency` будет создана запись с идентификаторами этих задач.

3. Описание программы. Описание основных функций каждого раздела.

Расписание

Первой feature, с которой начнем, будет расписание. В данной feature пользователь может просматривать расписание занятий, добавлять и редактировать занятия, а также выбирать активную неделю. В данной feature определены следующие классы:

- `CalendarBloc` - класс, отвечающий за бизнес-логику расписания. В данном классе определены методы:
  - load - загрузка занятий, активной недели и задач, а также их преобразование в расписание
- `CalendarScreen` - виджет, отвечающий за отображение расписания.
- `CalendarDayCard` - виджет, отвечающий за отображение карточки дня в расписании.
- `CalendarLessonCard` - виджет, отвечающий за отображение карточки занятия в расписании.
- `CalendarModels` - файл, содержащий модели, используемые в расписании, такие как `CalendarLesson`, `CalendarDay`, `CalendarTask`.

Создание и изменение расписания происходит в отдельной под-feature `calendar_put`, в которой определены следующие классы:

- `CalendarPutBloc` - класс, отвечающий за бизнес-логику создания и изменения занятий. В данном классе определены методы:
    - `load` - загрузка занятия и формирование данных таким образом, чтобы их можно было редактировать в формате недельного расписания
    - `save` - сохранение изменений в расписании
- `CalendarPutBlocScope` - виджет, отвечающий за создание области видимости для `CalendarPutBloc`. Реализует в себе методы `load` и `save`, которые вызывают соответствующие методы `CalendarPutBloc`.
- `CalendarPutScreen` - виджет, отвечающий за отображение формы создания и изменения занятия.
- `CalendarModels` - файл, содержащий модели, используемые в создании и изменении занятий, такие как `CalendarWeek`, `CalendarDay`, `CalendarLesson`.
- Утилиту `CalendarProvider` - класс, отвечающий за работу с данными расписания, такие как
    - `addWeek` - добавление недели
    - `removeWeek` - удаление недели
    - `updateWeek` - обновление недели
    - `addLesson` - добавление занятия
    - `removeLesson` - удаление занятия
    - `updateLesson` - обновление занятия
    - `verifyCalendar` - проверка корректности расписания, такая как пересечение занятий, недопустимые даты, отсутствие занятий и т.д.

Создание и изменение задач происходит в отдельной под-feature `task_put`, благодаря работе с данными через `CalendarProvider`, который позволяет упростить работу с данными и уменьшить количество кода.

Задачи

Второй feature, с которой продолжим, будет задачи. В данной feature пользователь может просматривать задачи, добавлять и редактировать задачи, а также удалять задачи. В данной feature определены следующие классы:

- `TasksBloc` - класс, отвечающий за бизнес-логику задач. В данном классе определены методы:
    - `load` - загрузка задач
    - `openPutTask` - открытие формы создания и изменения задачи
    - `saveTask` - сохранение изменений в задаче
    - `deleteTask` - удаление задачи
    - `changeTaskStatus` - изменение статуса задачи
    - `runImportanceAlgorithm` - запуск алгоритма вычисления важности задач
- `TasksBlocScope` - виджет, отвечающий за создание области видимости для `TasksBloc`. Реализует в себе методы `load`, `openPutTask`, `saveTask`, `deleteTask`, `changeTaskStatus`, `runImportanceAlgorithm`, которые вызывают соответствующие методы `TasksBloc`.
- `ImportanceTasksStorage` - класс, отвечающий за хранение наиболее важных задач, полученных в результате алгоритма вычисления важности задач. Реализует методы:
    - `clearMostImportanceTasks` - очистка хранилища наиболее важных задач
    - `getMostImportanceTasks` - получение наиболее важных задач
    - `saveMostImportanceTasks` - сохранение наиболее важных задач

    - `getMostImportanceTasksTime` - получение времени получения наиболее важных задач
- Элементы пользовательского интерфейса, такие как `TaskCard`, `TaskStatusCard`, `TasksScreen`, отвечающие за отображение задач и их статусов и находящиеся в директории `widget`.

Создание и изменение задач происходит в отдельной под-feature `task_put`, благодаря работе с данными через `TasksBloc`, который позволяет упростить работу с данными и уменьшить количество кода.

При помощи метода `runImportanceAlgorithm` в `TasksBloc` запускается алгоритм вычисления важности задач, который позволяет определить и предложить к решению наиболее важные на текущий момент задачи. Алгоритм позволяет определить важность задачи на основе ее крайнего срока, приоритета и приоритета зависимости, вычисленного на основе приоритетов подзадач и задач, от которых зависит данная задача. Результат алгоритма сохраняется в `ImportanceTasksStorage`, который позволяет получить наиболее важные задачи и время их получения.

В рамках метода `runImportanceAlgorithm` происходит следующий алгоритм:

1. Получение списка задач из базы данных.
2. Вызов метода `AlgorithmRunner.run` с передачей списка задач в качестве аргумента и присваивания результата переменной `AlgorithmResult`.

Как писалось выше, `AlgorithmRunner` инкапсулирует в себе логику подготовки данных при помощи `AlgorithmDataPreparer`, в котором фильтруются задачи по статусам и для каждой из них высчитывается приоритет зависимости. После подготовки данных вызывается метод `AlgorithmSolver.solve`, который сортирует задачи по важности и возвращает результат в виде `AlgorithmResult`.

Заметки

Третьей feature, с которой продолжим, будет заметки. В данной feature пользователь может просматривать заметки, добавлять и редактировать заметки, а также удалять заметки. В данной feature определены следующие классы:

- `NotesBloc` - класс, отвечающий за бизнес-логику заметок. В данном классе определены методы:
    - load - загрузка заметок
    - openPutNote - открытие формы создания и изменения заметки
    - saveNote - сохранение изменений в заметке
    - deleteNote - удаление заметки
- `NotesBlocScope` - виджет, отвечающий за создание области видимости для `NotesBloc`. Реализует в себе методы `load`, `openPutNote`, `saveNote`, `deleteNote`, которые вызывают соответствующие методы `NotesBloc`.
- Модели `NoteEntity` - содержащие данные о заметке, такие как название, текст, дата и время создания, дата и время последнего изменения, флаг избранное.
- Элементы пользовательского интерфейса, такие как `NoteCard`, `NotesList` и `NotesScreen`, отвечающие за отображение заметок и находящиеся в директории `widget`.

Создание и изменение заметок происходит в отдельной под-feature `note_put`, благодаря работе с данными через `NotesBloc`, который позволяет упростить работу с данными и уменьшить количество кода.

Ресурсы

Четвертой feature, с которой продолжим, будет ресурсы. В данной feature пользователь может просматривать ресурсы, добавлять и редактировать ресурсы, а также удалять ресурсы. В данной feature определены следующие классы:

- `ResourcesBloc` - класс, отвечающий за бизнес-логику ресурсов. В данном классе определены методы:
  - `load` - загрузка ресурсов
  - `openPutResource` - открытие формы создания и изменения ресурса
  - `saveResource` - сохранение изменений в ресурсе
  - `deleteResource` - удаление ресурса
- `ResourcesBlocScope` - виджет, отвечающий за создание области видимости для `ResourcesBloc`. Реализует в себе методы `load`, `openPutResource`, `saveResource`, `deleteResource`, которые вызывают соответствующие методы `ResourcesBloc`.
- Модели `ResourceEntity` - содержащие данные о ресурсе, такие как название, описание, ссылка, дата и время создания, тип ресурса, флаг избранное.
- Элементы пользовательского интерфейса, такие как `ResourceCard`, `ResourcesList` и `ResourcesScreen`, отвечающие за отображение ресурсов и находящиеся в директории `widget`.

Создание и изменение ресурсов происходит в отдельной под-feature `resource_put`, благодаря работе с данными через `ResourcesBloc`, который позволяет упростить работу с данными и уменьшить количество кода.

Общие настройки приложения

В отдельную директорию, хранящуюся рядом с директорией `feature`, вынесены общие настройки приложения, такие как тема приложения, локализация, а также настройки маршрутизации. В данной директории определены следующие поддиректории:

- `theme` - содержит файлы, отвечающие за тему приложения, такие как
  - `dark_theme.dart` - тема приложения в темном режиме
  - `light_theme.dart` - тема приложения в светлом режиме
  - `theme_constants.dart` - константы темы приложения
  - `theme_colors.dart` - цвета темы приложения
  - `theme_extensions.dart` - расширения для темы приложения
- `router` - содержит файлы, отвечающие за маршрутизацию в приложении, такие как
  - `routes.dart` - маршруты приложения
  - `router_state_mixin.dart` - миксин, отвечающий за управление состоянием маршрутизации
- `infrastructure` - содержит файлы, отвечающие за инфраструктуру приложения, такие как
  - `di_configurator.dart` - конфигуратор зависимостей

    - `bloc_global_observer.dart` - глобальный наблюдатель за блоками, перегружающий методы `onEvent`, `onTransition`, `onError`, что позволяет конфигурировать блоки при их создании
- `extensions` - содержит файлы, отвечающие за расширения, такие как
    - `date_time_x.dart` - расширения для `DateTime`, добавляющий метод `isToday` для проверки, является ли дата сегодняшней
    - `time_of_day_x.dart` - расширения для `TimeOfDay`, добавляющий методы:
        - `isBefore` - проверка, является ли время до указанного времени
        - `isAfter` - проверка, является ли время после указанного времени
        - `addMinutes` - добавление минут к времени
        - `compareTo` - сравнение времени
        - `toDateTime` - преобразование времени в дату и время
    - `task_entity_x.dart` - расширения для `TaskEntity`, добавляющий метод:
        - `toTask` - преобразование `TaskEntity` в `Task` для работы с базой данных
        - `toTaskCompanion` - преобразование `TaskEntity` в `TaskCompanion` для работы с базой данных
    - `scaffold_x.dart` - расширения для `Scaffold`
- `constants` - содержит файлы, отвечающие за константы приложения.

Пакет stook_shared

Пакет `stook_shared` содержит общие классы и методы, которые используются в различных частях приложения. В данном пакете определены следующие классы:

- `TaskEntity` - класс, представляющий сущность задачи, содержащий данные о задаче, такие как
    - `id` - идентификатор задачи, уникальный идентификатор задачи
    - `title` - название задачи
    - `description` - описание задачи
    - `priority` - приоритет задачи
    - `createdDate` - дата и время создания задачи
    - `deadlineDate` - крайний срок задачи
    - `status` - статус задачи
    - `subtaskIds` - список идентификаторов подзадач
    - `dependOnTaskIds` - список идентификаторов задач, от которых зависит данная задача
- `TaskBaseEntity` - класс, представляющий базовую сущность задачи, содержащий данные о задаче, такие как
    - `id` - идентификатор задачи, уникальный идентификатор задачи
    - `title` - название задачи
    - `description` - описание задачи
    - `priority` - приоритет задачи
    - `createdDate` - дата и время создания задачи
    - `deadlineDate` - крайний срок задачи
    - `status` - статус задачи

Приоритет и статус задачи представлены в виде перечислений `TaskPriority` и `TaskStatus`, которые точно также определены в данном пакете.

Пакет stook_database

Как уже было сказано в разделе "Описание даталогической архитектуры системы", пакет `stook_database` отвечает за работу с базой данных SQLite3. Определение сущностей и атрибутов, а также связей между сущностями и атрибутами, который были описаны выше, реализованы в данном пакете.

Но данный пакет также содержить реализацию работы с данными в формате dao (Data Access Object). В данном пакете определены следующие dao:

- `TaskDao` - dao, отвечающий за работу с задачами. В данном dao определены методы:
  - `getAllTasks` - получение списка задач
  - `getTaskById` - получение задачи по идентификатору
  - `insertTask` - добавление задачи
  - `updateTask` - обновление задачи
  - `deleteTask` - удаление задачи
- `LessonDao` - dao, отвечающий за работу с занятиями. В данном dao определены методы:
  - `getAllLessons` - получение списка занятий
  - `getLessonById` - получение занятия по идентификатору
  - `insertLesson` - добавление занятия
  - `updateLesson` - обновление занятия
  - `deleteLesson` - удаление занятия
- `NoteDao` - dao, отвечающий за работу с заметками. В данном dao определены методы:
  - `getAllNotes` - получение списка заметок
  - `getNoteById` - получение заметки по идентификатору
  - `insertNote` - добавление заметки
  - `updateNote` - обновление заметки
  - `deleteNote` - удаление заметки
- `ResourceDao` - dao, отвечающий за работу с ресурсами. В данном dao определены методы:
  - `getAllResources` - получение списка ресурсов
  - `getResourceById` - получение ресурса по идентификатору
  - `insertResource` - добавление ресурса
  - `updateResource` - обновление ресурса
  - `deleteResource` - удаление ресурса
- `TaskDependOnRelationDao` - dao, отвечающий за работу с зависимостями задач. В данном dao определены методы:
  - `getDependOnTasks` - получение списка задач, от которых зависит задача
  - `getDependOnTaskById` - получение задачи, от которой зависит задача
  - `insertDependOnTask` - добавление задачи, от которой зависит задача
  - `deleteDependOnTask` - удаление задачи, от которой зависит задача
- `TaskSubtaskRelationDao` - dao, отвечающий за работу с подзадачами задач. В данном dao определены методы:
  - `getSubtasks` - получение списка подзадач
  - `getSubtaskById` - получение подзадачи
  - `insertSubtask` - добавление подзадачи
  - `deleteSubtask` - удаление подзадачи

Пакет stook_algorithm

Пакет `stook_algorithm` отвечает за алгоритм вычисления важности задач. В данном пакете определены следующие классы:

- `AlgorithmItem` - класс, представляющий собой элемент алгоритма, содержащий данные о задаче и вспомогательные данные, такие как
  - `id` - идентификатор задачи
  - `priority` - приоритет задачи, который из перечисления конвертируется в числовое значение по следующему правилу:
    - `TaskPriority.A` - 9
    - `TaskPriority.B` - 6
    - `TaskPriority.C` - 3
  - `dependsPriority` - приоритет задач, от которых зависит задача и которые зависят от нее, по умолчанию 0
  - `deadlineDate` - крайний срок задачи
- `AlgorithmResult` - класс, представляющий собой результат алгоритма, содержащий данные о наиболее важных задачах и времени получения этих задач, такие как
  - `taskIds` - список идентификаторов наиболее важных задач
  - `calculationTime` - время вычисления наиболее важных задач
- `AlgorithmDataPreparer` - класс, реализующий интерфейс `IAlgorithmDataPreparer`, отвечающий за подготовку данных для алгоритма. В рамках интерфейса необходимо реализовать следующие методы:

```
List<AlgorithmItem> getPreparedData(List<TaskEntity> items);
```

где `items` - список данных типа `TaskEntity`, которые необходимо подготовить для алгоритма.

- `AlgorithmSolver` - класс, реализующий интерфейс `IAlgorithmSolver`, отвечающий за запуск алгоритма. В рамках интерфейса необходимо реализовать следующие методы:

```
List<int> getMostImportantItems(
    List<AlgorithmItem> items, [
    int count = 3,
    DateTime? currentDate,
]);
```

где `items` - список элементов алгоритма, `count` - количество наиболее важных элементов, которые необходимо вернуть, `currentDate` - текущая дата и время, на основе которой необходимо вычислить важность элементов. Возвращает список идентификаторов наиболее важных элементов.

- `AlgorithmRunner` - класс, реализующий интерфейс `IAlgorithmRunner`, инкапсулирующий логику алгоритма, а то есть подготовку данных, запуск алгоритма и обработку результата. В рамках интерфейса необходимо реализовать следующие методы:

```
Future<AlgorithmResult> run(
    List<TaskEntity> items, [
```

```
    int count = 3,
    DateTime? currentDate,
]);
```

где `items` - список данных типа `TaskEntity`, которые необходимо подготовить для алгоритма, `count` - количество наиболее важных элементов, которые необходимо вернуть, `currentDate` - текущая дата и время, на основе которой необходимо вычислить важность элементов.

Данная структура и иерархия классов позволяет легко заменять реализацию алгоритма вычисления важности задач, что упрощает тестирование и разработку.

В текущей версии приложения реализован простой алгоритм вычисления важности задач, который основывается на приоритете и крайнем сроке задачи. Важность задачи вычисляется по формуле:

```
/// Получить важность задачи
double _calculateImportance(AlgorithmItem item, DateTime now) {
    return (item.priority * (item.dependsPriority + 1)) /
        (_findSecondsBetweenDates(now, item.deadlineDate) + 1);
}
```

где `item.priority` - приоритет задачи, `item.dependsPriority` - приоритет задач, от которых зависит задача и которые зависят от нее, `item.deadlineDate` - крайний срок задачи, `now` - текущая дата и время, а `_findSecondsBetweenDates` - функция, которая находит количество секунд между двумя датами, т.е.

```
/// Получить разницу между датами в секундах
int _findSecondsBetweenDates(DateTime first, DateTime second) {
    return second.difference(first).inSeconds;
}
```

Таким, образом, данный алгоритм вычисляет важность задачи на основе приоритета, крайнего срока и приоритета задач, от которых зависит задача и которые зависят от нее. Это позволяет вычислить наиболее важные задачи, которые необходимо выполнить в первую очередь, что дает возможность пользователю сосредоточиться на наиболее важных задачах.

Некоторые важные моменты

Правило 1. В целом для планирования подходят только задачи, имеющие указанный приоритет и крайний срок. Если у задачи не указан приоритет или крайний срок, то она не участвует в планировании.

Правило 2. Параметр `item.dependsPriority` стоит рассмотреть отдельно. Данный параметр вычисляется по следующему правилу:

1. Если у задачи (A) есть подзадачи, то к каждой подзадаче в приоритет зависимости добавляется приоритет задачи A.

2. Если у задачи (A) есть задачи, от которых она зависит, то к каждой задаче, от которой зависит задача A, в приоритет зависимости добавляется приоритет задачи A.

Для вычисления параметра `item.dependsPriority` используются все задачи, соответствующие правилу 1, что позволяет точнее определить важность задачи.

Код функции, вычисляющей параметр `item.dependsPriority`:

```
Map<int, int> _getTaskPriorityById(List<TaskEntity> tasks) {
    final taskPriorityById = <int, int>{
        for (final task in tasks) task.id:
task.priority!.toPriorityNumber,
    };

    final dependOnTasksById = <int, List<int>>{};

    for (final task in tasks) {
        for (final dependOnId in task.dependOnTasksIds) {
            dependOnTasksById.putIfAbsent(dependOnId, () => []);
            dependOnTasksById[dependOnId]!.add(task.id);
        }

        for (final subtaskId in task.subtasksIds) {
            dependOnTasksById.putIfAbsent(subtaskId, () => []);
            dependOnTasksById[subtaskId]!.add(task.id);
        }
    }

    for (final dependOnTasks in dependOnTasksById.entries) {
        final taskId = dependOnTasks.key;
        taskPriorityById[taskId] = dependOnTasks.value.fold<int>(
            0,
            (sum, id) => sum + (taskPriorityById[id] ?? 0),
        );
    }

    return taskPriorityById;
}
```

Эта функция `_getTaskPriorityById` принимает список объектов `TaskEntity` и возвращает `Map<int, int>`, где ключом является идентификатор задачи, а значением - её приоритет.

Вот объяснение шагов:

1. Создание taskPriorityById, который инициализируется пустым словарем. Затем используется синтаксис генератора коллекций для заполнения словаря значениями приоритета каждой задачи. Ключом является идентификатор задачи (task.id), а значением - приоритет задачи (task.priority!.toPriorityNumber).
2. Создание dependOnTasksById, который также инициализируется пустым словарем. Затем проходятся все задачи в списке tasks. Для каждой задачи добавляются зависимости в

dependOnTasksById. Если у задачи есть зависимости (dependOnTasksIds или subtasksIds), они добавляются в dependOnTasksById с ключом, равным идентификатору зависимости, и списком идентификаторов задач, от которых она зависит.

3. Затем происходит вычисление приоритета для каждой задачи. Для этого используется цикл for для итерации по парам ключ-значение в dependOnTasksById. Для каждой пары вычисляется сумма приоритетов всех задач из списка значений (списка идентификаторов задач, от которых зависит текущая задача). Затем эта сумма приоритетов устанавливается как новое значение приоритета для текущей задачи в словаре taskPriorityById.
4. В конце функция возвращает словарь taskPriorityById, содержащий идентификаторы задач и их приоритеты, учитывая зависимости.

Рассмотрим грубый пример, чтобы понять, как это работает. Предположим, у нас есть следующие задачи:

1. Задача `Защита проекта` зависит от выполнения задач `Оформление отчетности` и `Оформление презентации`. Приоритет задачи `Защита проекта` равен 6, а приоритет

зависимости равен 0.

2. Задача `Оформление отчетности` зависит от выполнения задач `Разработка приложения`, а от нее зависит задача `Защита проекта`. Приоритет задачи `Оформление отчетности` равен 3, а приоритет зависимости равен 6, потому что задача `Защита проекта` имеет приоритет 6, а приоритет зависимости равен 0.
3. Задача `Оформление презентации` зависит от выполнения задачи `Разработка приложения`, а от нее зависит задача `Защита проекта`. Приоритет задачи `Оформление презентации` равен 3, а приоритет зависимости равен 6, потому что задача `Защита проекта` имеет приоритет 6, а приоритет зависимости равен 0.
4. Задача `Разработка приложения` зависит от выполнения задачи `Проектирование архитектуры`, а от нее зависят задачи `Оформление отчетности` и `Оформление презентации`. Приоритет задачи `Разработка приложения` равен 9, а приоритет зависимости равен 18, потому что задачи `Оформление отчетности` и `Оформление презентации` имеют приоритет 3, а приоритет зависимости равен 6, что в сумме дает 18.
5. Задача `Проектирование архитектуры` зависит от выполнения задачи `Проектирование базы данных`, а от нее зависит задача `Разработка приложения`. Также задача `Проектирование архитектуры` имеет две подзадачи: `Выбрать стек технологий` и `Выбрать используемые фреймворки`, в свою очередь задача `Выбрать используемые фреймворки` зависит от выполнения задачи `Выбрать стек технологий`. Таким образом,
    - Приоритет задачи `Проектирование архитектуры` равен 9, а приоритет зависимости равен 27.
    - Подзадача `Выбрать используемые фреймворки` имеет приоритет 6, а приоритет зависимости равен 36, потому что задача `Проектирование архитектуры` имеет приоритет 9, а приоритет зависимости равен 27.
    - Подзадача `Выбрать стек технологий` имеет приоритет 6, а приоритет зависимости равен 78, потому что задача `Проектирование архитектуры` имеет приоритет 9, а приоритет зависимости равен 27, а подзадача `Выбрать используемые фреймворки` имеет приоритет 6, а приоритет зависимости равен 36.
6. Задача `Проектирование базы данных` не зависит от выполнения других задач, однако от нее зависит задача `Проектирование архитектуры`, также задача имеет три подзадачи: `Создать сущности`, `Создать таблицы` и `Создать связи`, которая в свою очередь зависит от задачи `Создать сущности`. Таким образом,
    - Приоритет задачи `Проектирование базы данных` равен 9, а приоритет зависимости равен 36, потому что задача `Проектирование архитектуры` имеет приоритет 9, а приоритет зависимости равен 27.
    - Подзадача `Создать таблицы` имеет приоритет 6, а приоритет зависимости равен 45, потому что задача `Проектирование базы данных` имеет приоритет 9, а приоритет зависимости равен 36.
    - Подзадача `Создать связи` имеет приоритет 3, а приоритет зависимости равен 45, потому что задача `Проектирование базы данных` имеет приоритет 9, а приоритет зависимости равен 36.
    - Подзадача `Создать сущности` имеет приоритет 6, а приоритет зависимости равен 93, потому что задача `Проектирование базы данных` имеет приоритет 9, а приоритет зависимости равен 36, а подзадача `Создать таблицы` имеет приоритет 3, а приоритет зависимости равен 45.

Таким образом, можно сделать вывод, что параметр `item.dependsPriority` учитывает приоритеты задач, от которых зависит задача и которые зависят от нее, что позволяет точнее определить важность задачи.

Пример рассмотрен грубо, потому что если мы посмотрим на схему, не учитывая приоритет зависимости, станет ясно, что выполнить на данный момент лучше всего задачи `Выбрать стек технологий`, `Создать сущности` и `Создать таблицы`, потому что они не имеют подзадать и не зависят от других задач. Однако, если учесть приоритет зависимости, то станет ясно, что выполнить на данный момент лучше всего все те же задачи, но в первую очередь задачу `Создать сущности`, потому что она имеет наибольший приоритет зависимости.

Правило 3. Однако задачи, соответствующие правилу 1, подтвергаются дополнительной фильрации перед планированием, чтобы они соответствовать следующим принципам:

1. Находятся в статусе "Создана" или "В процессе".
2. Не имеют подзадач, находящихся в статусе "Создана" или "В процессе".
3. Не зависят от задач, находящихся в статусе "Создана" или "В процессе".

Заключение

В данной работе было разработано приложение для управления задачами, заметками, ресурсами и расписанием. Приложение позволяет пользователю управлять своими задачами, заметками, ресурсами и расписанием, а также вычислять наиболее важные задачи, которые необходимо выполнить в первую очередь.