Подробное техническое описание архитектуры программы Royal Stats (Hero-only)

Обзор Архитектуры

Программа Royal Stats построена на принципах многослойной архитектуры, разделенной на следующие слои:

1. **Presentation Layer (UI)**: Уровень взаимодействия с пользователем.
2. **Application Logic Layer (Services)**: Уровень бизнес-логики и координации.
3. **Domain Layer (Models)**: Уровень представления данных предметной области.
4. **Data Access Layer (Repositories)**: Уровень абстракции доступа к данным.
5. **Infrastructure Layer**: Уровень низкоуровневых компонентов (управление БД, парсинг).

Взаимодействие между слоями строго определено: вышележащие слои могут использовать нижележащие, но не наоборот. UI вызывает методы Сервисов, Сервисы используют Репозитории и Парсеры, Репозитории используют Менеджер БД.

Структура Базы Данных (SQLite)

База данных хранится в отдельном файле (.db) для каждого игрока/набора статистики. Схема БД включает следующие таблицы:

- **sessions**:
  - id (INTEGER PRIMARY KEY AUTOINCREMENT): Внутренний ID записи.
  - session_id (TEXT UNIQUE NOT NULL): Уникальный идентификатор сессии импорта (UUID).
  - session_name (TEXT NOT NULL): Имя сессии, заданное пользователем.
  - created_at (TIMESTAMP DEFAULT CURRENT_TIMESTAMP): Дата и время создания сессии.
  - tournaments_count (INTEGER DEFAULT 0): Количество турниров, импортированных в этой сессии.
  - knockouts_count (INTEGER DEFAULT 0): Общее количество KO Hero в турнирах этой сессии (агрегированное).
  - avg_finish_place (REAL DEFAULT 0): Среднее финишное место Hero по всем турнирам в этой сессии (агрегированное).
  - total_prize (REAL DEFAULT 0): Общая сумма выплат Hero по всем турнирам в этой сессии (агрегированное).
  - total_buy_in (REAL DEFAULT 0): Общая сумма бай-инов Hero по всем турнирам в этой сессии (агрегированное).
  - *Назначение:* Хранение метаданных и агрегированной статистики по каждой

операции импорта.

- **tournaments**:
  - id (INTEGER PRIMARY KEY AUTOINCREMENT): Внутренний ID записи.
  - tournament_id (TEXT UNIQUE NOT NULL): Уникальный идентификатор турнира (из HH/TS).
  - tournament_name (TEXT): Название турнира (из TS).
  - start_time (TEXT): Время начала турнира (из TS).
  - buyin (REAL): Полный бай-ин турнира (из TS).
  - payout (REAL): Общая выплата Hero в турнире (из TS).
  - finish_place (INTEGER): Финишное место Hero (из TS).
  - ko_count (INTEGER DEFAULT 0): Общее количество KO Hero в этом турнире (суммируется из hero_final_table_hands).
  - session_id (TEXT): session_id первой сессии, в которой был импортирован файл этого турнира (FOREIGN KEY к sessions.session_id с ON DELETE CASCADE).
  - reached_final_table (BOOLEAN DEFAULT 0): Флаг, указывающий, достиг ли Hero финального стола (9-max) в этом турнире.
  - final_table_initial_stack_chips (REAL): Стек Hero в фишках в первой раздаче финального стола.
  - final_table_initial_stack_bb (REAL): Стек Hero в BB в первой раздаче финального стола.
  - *Назначение:* Хранение агрегированной информации по каждому турниру Hero, объединяя данные из HH и TS.
- **hero_final_table_hands**:
  - id (INTEGER PRIMARY KEY AUTOINCREMENT): Внутренний ID записи.
  - tournament_id (TEXT NOT NULL): ID турнира (FOREIGN KEY к tournaments.tournament_id с ON DELETE CASCADE).
  - hand_id (TEXT NOT NULL): Уникальный ID раздачи из HH.
  - hand_number (INTEGER): Порядковый номер раздачи в турнире.
  - table_size (INTEGER): Количество игроков за столом в начале раздачи.
  - bb (REAL): Размер большого блайнда в раздаче.
  - hero_stack (REAL): Стек Hero в фишках в начале раздачи.
  - hero_ko_this_hand (INTEGER DEFAULT 0): Количество KO, сделанных Hero именно в этой раздаче.
  - session_id (TEXT): session_id сессии, в которой был импортирован HH файл с этой раздачей (FOREIGN KEY к sessions.session_id с ON DELETE CASCADE).
  - is_early_final (BOOLEAN DEFAULT 0): Флаг, указывающий, относится ли раздача к "ранней" стадии финального стола (9-6 игроков).
  - UNIQUE (tournament_id, hand_id): Комбинация ID турнира и ID раздачи уникальна.
  - *Назначение:* Хранение детальной информации по каждой раздаче финального стола, необходимой для точного подсчета KO и статов ранней стадии/стека на финалке.

- **overall_stats**:
  - id (INTEGER PRIMARY KEY CHECK (id = 1)): Гарантирует единственную строку.
  - total_tournaments (INTEGER DEFAULT 0): Общее количество турниров.
  - total_final_tables (INTEGER DEFAULT 0): Количество турниров, достигших финалки.
  - total_knockouts (INTEGER DEFAULT 0): Общее количество KO Hero.
  - avg_finish_place (REAL DEFAULT 0): Среднее место по всем турнирам.
  - avg_finish_place_ft (REAL DEFAULT 0): Среднее место только по финалке (1-9).
  - total_prize (REAL DEFAULT 0): Общая сумма выплат.
  - total_buy_in (REAL DEFAULT 0): Общая сумма бай-инов.
  - avg_ko_per_tournament (REAL DEFAULT 0): Среднее KO за турнир (по всем турнирам).
  - avg_ft_initial_stack_chips (REAL DEFAULT 0): Средний стек на старте финалки (фишки).
  - avg_ft_initial_stack_bb (REAL DEFAULT 0): Средний стек на старте финалки (BB).
  - big_ko_x1_5, big_ko_x2, big_ko_x10, big_ko_x100, big_ko_x1000, big_ko_x10000 (INTEGER DEFAULT 0): Количество "больших" KO.
  - early_ft_ko_count (INTEGER DEFAULT 0): Общее KO в ранней финалке (9-6 игроков).
  - early_ft_ko_per_tournament (REAL DEFAULT 0): Среднее KO в ранней финалке на турнир (достигший финалки).
  - last_updated (TIMESTAMP DEFAULT CURRENT_TIMESTAMP): Время последнего обновления статистики.
  - *Назначение:* Хранение всех агрегированных статистических показателей за всю историю Hero для быстрого доступа и отображения на дашборде.
- **places_distribution**:
  - place (INTEGER PRIMARY KEY): Финишное место на финальном столе (1-9).
  - count (INTEGER DEFAULT 0): Количество финишей на этом месте.
  - *Назначение:* Хранение данных для построения гистограммы распределения финишных мест на финальном столе.
- **stat_modules** и **module_settings**: Таблицы для управления плагинами статистики и их настройками (запланированы для расширяемости, но не полностью реализованы в предоставленном коде).

Последовательность Вызовов (Пример: Импорт Файлов)

Рассмотрим подробную последовательность вызовов при импорте файлов:

1. **UI (MainWindow)**:
   - Пользователь нажимает кнопку "Импорт файлов/папки".
   - Вызывается слот MainWindow.import_files() или MainWindow.import_directory().
   - Открывается QFileDialog для выбора файлов/папок.

- Открывается QInputDialog для ввода имени сессии.
- Создается QProgressDialog для отображения прогресса.
- Создается экземпляр ImportThread (наследник QThread), которому передается список путей, имя сессии и ссылка на application_service.
- Подключаются сигналы ImportThread.finished к MainWindow._import_finished и ImportThread.progress_update к MainWindow._update_progress.
- Вызывается import_thread.start().
- Вызывается progress_dialog.exec() для запуска модального диалога прогресса.

2. **Infrastructure (ImportThread)**:
   - В отдельном потоке выполняется метод ImportThread.run().
   - Внутри run, вызывается self.app_service.import_files(self.paths, self.session_name, progress_callback=self._report_progress).
   - Метод _report_progress в ImportThread просто вызывает self.progress_update.emit(current, total, text), отправляя сигнал в основной поток UI.
3. **Application Logic (ApplicationService)**:
   - Вызывается ApplicationService.import_files(paths, session_name, progress_callback).
   - Собирается полный список файлов для обработки, подсчитывается их количество.
   - Если есть progress_callback, он вызывается для установки максимального значения прогресс-бара.
   - Вызывается self.session_repo.create_session(session_name) для создания новой записи в таблице sessions. Получается session_id новой сессии.
   - Файлы разделяются на HH и TS и обрабатываются в определенном порядке (HH сначала).
   - Для каждого файла в списке:
     - Вызывается progress_callback для обновления текущего файла.
     - Читается содержимое файла.
     - Определяется тип файла.
     - Если **HH файл**:
       - Вызывается self.hh_parser.parse(content, filename).
       - Парсер HH возвращает словарь с данными турнира (ID, дата старта, факт достижения финалки, стек на старте финалки) и список словарей с данными по каждой раздаче финального стола (final_table_hands_data).
       - Получаются текущие данные турнира из БД по ID (self.tournament_repo.get_tournament_by_id).
       - Данные из парсера HH объединяются с существующими данными турнира (например, флаг reached_final_table становится TRUE, если он TRUE в парсере или уже был TRUE в БД; стек на старте финалки сохраняется только если его еще не было).

        - Создается объект Tournament из объединенных данных.
        - Вызывается self.tournament_repo.add_or_update_tournament(tournament_object) для сохранения/обновления записи в таблице tournaments.
        - Для каждой раздачи в final_table_hands_data:
            - Добавляется session_id текущей сессии.
            - Создается объект FinalTableHand.
            - Вызывается self.ft_hand_repo.add_hand(ft_hand_object) для сохранения раздачи в таблице hero_final_table_hands (с игнорированием дубликатов).
    - Если **TS файл**:
        - Вызывается self.ts_parser.parse(content, filename).
        - Парсер TS возвращает словарь с данными турнира (ID, название, дата старта, бай-ин, выплата, место).
        - Получаются текущие данные турнира из БД по ID (self.tournament_repo.get_tournament_by_id).
        - Данные из парсера TS объединяются с существующими данными турнира (данные из TS имеют приоритет для места, выплаты, бай-ина, названия, даты старта).
        - Создается объект Tournament из объединенных данных.
        - Вызывается self.tournament_repo.add_or_update_tournament(tournament_object).
  - После обработки всех файлов, вызывается self._update_all_statistics(session_id_just_imported).
4. **Application Logic (ApplicationService) - Обновление Статистики**:
  - Вызывается ApplicationService._update_all_statistics(session_id_just_imported).
  - **Обновление Overall Stats**:
    - Вызывается self._calculate_overall_stats().
    - В _calculate_overall_stats():
      - Запрашиваются все турниры (self.tournament_repo.get_all_tournaments()) и все руки финалки (self.ft_hand_repo.get_all_hands()).
      - Выполняются все расчеты агрегированных показателей (общее количество турниров, финалок, KO, средние места, средние стеки на финалке, статы ранней финалки) на основе полученных данных.
      - Вызываются стат-плагины (например, BigKOStat().compute(...)) для расчета специфических стат, передавая им необходимые данные и объект OverallStats (для доступа к уже посчитанным базовым агрегатам).
      - Создается объект OverallStats с рассчитанными значениями.
    - Вызывается

self.overall_stats_repo.update_overall_stats(overall_stats_object) для сохранения в БД.
    - **Обновление Place Distribution**:
        - Вызывается self.place_dist_repo.reset_distribution().
        - Запрашиваются все турниры (self.tournament_repo.get_all_tournaments()).
        - Для каждого турнира, достигшего финалки с известным местом (1-9): вызывается self.place_dist_repo.increment_place_count(place).
    - **Обновление Session Stats**:
        - Запрашиваются все сессии (self.session_repo.get_all_sessions()).
        - Для каждой сессии:
            - Запрашиваются турниры (self.tournament_repo.get_all_tournaments(session_id=...)) и руки финалки (self.ft_hand_repo.get_hands_by_session(session_id=...)) для этой сессии.
            - Рассчитываются агрегированные статы для этой сессии (количество турниров, KO, среднее место, общие выплаты/бай-ины).
            - Обновляется запись сессии в БД (self.session_repo.update_session_stats(session_object)).
5. **Infrastructure (ImportThread)**:
    - Метод run() завершается. Сигнал finished отправляется в основной поток.
6. **UI (MainWindow)**:
    - Слот MainWindow._import_finished() принимается.
    - Вызывается MainWindow.refresh_all_data().
7. **UI (Views)**:
    - MainWindow.refresh_all_data() вызывает reload() у StatsGrid, TournamentView, SessionView.
    - Каждый View в своем reload():
        - Запрашивает необходимые данные у application_service (например, StatsGrid запрашивает get_overall_stats и get_place_distribution; TournamentView запрашивает get_all_tournaments; SessionView запрашивает get_all_sessions).
        - Обновляет свое отображение на основе полученных данных.

Описание Компонентов (Подробно)

- **ApplicationService**: Синглтон. Является координатором. Не содержит логики прямого доступа к БД или парсинга файлов. Его ответственность - *управлять* этим процессом. Он решает, когда парсить, какие данные объединять, когда пересчитывать статистику и какие данные предоставить UI.
- **Репозитории (*.py в db/repositories)**: Каждый репозиторий - это шлюз к одной или группе связанных таблиц. Они предоставляют методы для CRUD (Create, Read,

Update, Delete) операций и простых агрегаций (SUM, AVG, COUNT) *на уровне БД*. Они не содержат бизнес-логики (например, логики подсчета Big KO или среднего стека на финалке). Они используют database_manager для выполнения SQL запросов.

- **DatabaseManager**: Синглтон. Низкоуровневая абстракция работы с файлом БД. Управляет жизненным циклом соединений (потокобезопасность), переключает активный файл БД, создает таблицы при необходимости. Репозитории получают соединения через него.
- **Модели (*.py в models)**: Простые структуры данных (dataclasses), представляющие строки таблиц или агрегированные данные. Не содержат сложной логики, в основном методы для преобразования в/из словаря.
- **Парсеры (*.py в parsers)**: Отвечают за чтение и интерпретацию сырых текстовых файлов HH/TS. Их задача - извлечь структурированные данные (в виде словарей или объектов моделей) из текста. Они не работают напрямую с БД. Они могут использовать базовые настройки из config.
- **Стат-плагины (*.py в stats)**: Не являются частью основного потока обработки данных. Это отдельные модули, инкапсулирующие логику расчета *конкретных* статистик. Они вызываются ApplicationService (или, возможно, UI для сессионных стат) и получают на вход данные (списки моделей, OverallStats) для выполнения своих расчетов. Они не работают напрямую с БД или парсерами.
- **UI Компоненты (*.py в ui)**: Отвечают исключительно за отображение данных и обработку действий пользователя. Они не содержат бизнес-логики. Они взаимодействуют только с ApplicationService для получения данных и инициирования операций (импорт, управление БД).

Взаимодействие Слоев

- **UI -> Application Logic**: UI инициирует действия (импорт, обновление) и запрашивает данные для отображения.
- **Application Logic -> Data Access**: Сервис сохраняет/обновляет данные через Репозитории, запрашивает данные для расчетов.
- **Application Logic -> Infrastructure (Parsers)**: Сервис вызывает парсеры для извлечения данных из файлов.
- **Application Logic -> Domain**: Сервис работает с объектами моделей, полученными от парсеров или Репозиториев.
- **Application Logic -> Stats**: Сервис вызывает стат-плагины для расчета специфических показателей.
- **Data Access -> Infrastructure (DatabaseManager)**: Репозитории получают соединения и выполняют запросы через Менеджер БД.
- **Parsers -> Config**: Парсеры могут использовать настройки из конфига (например, имя Hero, параметры финалки).

Эта подробная структура и описание потоков вызовов должны дать полное представление о том, как различные части программы взаимодействуют друг с другом

для достижения поставленных целей.