**Приложение и его функциональность:**

1. Разработка функционала для регистрации продажи билетов в кинотеатре с возможностью выбора мест.

2. Внедрение механизма возврата билетов до начала сеанса для обеспечения удобства посетителей.

3. Введение простой системы регистрации пользователей с обязательным шифрованием паролей и авторизацией для сотрудников.

4. Внедрение возможности редактирования информации о фильмах в прокате и графиках сеансов

5. Реализация отображения доступных и занятых мест для выбранного сеанса.

6. Создание функционала для отметки занятых мест в зале посетителями на конкретном сеансе.

**Подробная информация о каждом слое:**

1. Basis: здесь расположены основные сущности и интерфейсы для работы, независимые от конкретной реализации. В репозитории models находятся сущности, представляющие данные в базе (film, session, ticket), а также классы для работы с этими данными: TicketRepository, FilmRepository, SessionRepository.

2. Application: Содержит класс CinemaService, реализующий интерфейсы CinemaPartService, TicketService. На этом уровне данные из репозиториев обрабатываются и объединяются в цельное для передачи на уровень представления. Здесь также находятся обработчик ответов от уровня данных (ResponseHandler), классы, отвечающие за бизнес-логику, и различные промежуточные сущности (FilmInf, SessionInf, SessionWithFilm).

3. Infrastructure: включает в себя внешние зависимости, в данном случае - базу данных, использующую формат JSON. Здесь находятся реализации репозиториев (FilmRepositoryImpl, TicketRepositoryImpl, SessionRepositoryImpl), имеющие зону ответственности за определенные модели данных (ticket, session, film). Есть класс DataContext, представляющий базу данных, а также сущности для самой базы данных (SessionJSON отличается от обычного сеанса тем, что у него временное поле в типе String). Присутствует класс Seed, наполняющий базу данных данными, и Mapper данных, выполняющий преобразование объектов из базы данных в объекты models (basis). Здесь также находятся функции для обработки и объединения объектов по их идентификаторам.

4. Main: Содержит утилиты для соблюдения принципа DRY и DI . App и Configuration регистрируют зависимости, а main представляет собой точку входа в программу, т.е. уровень представления.

5. Extra: infrastructure.auth: Включает сервис для регистрации пользователей, содержащий собственные сущности и утилиты.

Вся система подразумевает под собой выполнение принципов SOLID!

**Почему выбран именно JSON:**

1. Поддержка структурированных данных:

JSON обеспечивает поддержку структурированных данных с различными уровнями вложенности Это позволяет удобно организовывать информацию и представлять связанные данные в виде объектов. Таким образом, при расширении кинотеатра по местам JSON остается удобным выбором.

2. Читаемость и простота:

JSON обладает удобочитаемым форматом, что делает его легким в восприятии как для разработчиков, так и для других систем. Форматирование данных в JSON обеспечивает простоту восприятия информации и удобство для ручного редактирования, если необходимо.

3. Легкость использования:

JSON хорошо поддерживается в различных языках программирования, включая Kotlin. Существуют библиотеки работы с JSON, которые упрощают чтение и запись данных в/из файлов JSON.