1. Цель и контекст

Реализовать «ежедневные задачи» с добавлением через Telegram-бота и основным просмотром/управлением в mini-app. Логика дня: «Сегодня» показывает актуальные задачи; на следующий день автоматически остаются только невыполненные. Возможен просмотр «Вчера» и произвольной даты. Хранилище — Supabase (Postgres + RLS + Edge Functions + Scheduler).

---

2. Словарь

- **Task** — сущность задачи (текст, приоритет, сфера и т.п.).
- **Occurrence** — «вхождение» задачи на конкретную дату (статус на день, порядок в списке).
- **Rollover** — перенос невыполненных задач на следующий день путём создания нового Occurrence.
- **Today/Yesterday** — границы дня считаются в TZ пользователя (по умолчанию Asia/Nicosia).

---

3. Пользовательские сценарии (MVP)

1) **Добавить задачу в боте**: `/task add <текст> [#сфера] [@дата]` → задача появляется в списке «Сегодня» (или указанная дата). 2) **Список задач в mini-app**: экран с задачами на выбранную дату (по умолчанию — Сегодня). Доступна «Вчера» и выбор даты (date-picker). 3) **Статусы**: todo / in_progress / done. Быстрая смена статуса (чекбокс/свайп). Скрытие «done» по тумблеру. 4) **Порядок**: drag-and-drop сортировка в пределах дня (сохраняется `order_index`). 5) **Перенос**: на новый день остаются только невыполненные (rollover). Выполненные не появляются. История «Вчера» доступна read-only. 6) **Быстрый просмотр в боте**: `/tasks today` (топ-10), `/tasks yesterday`. Маркировка done по кнопке-инлайн.

Out of scope MVP: подзадачи, повторяющиеся задачи, сроки и напоминания, вложения, совместные списки.

---

4. Архитектура

- **Telegram Bot**: webhook → App Server (Node/TS). Команды парсятся, пишем в Supabase через сервис-ключ (сервер-side).
- **Mini-app**: Telegram WebApp (React/Next). Авторизация через `initData` → верифицируем на бэке → выдаём Supabase auth (JWT) с ограниченными правами.
- **Supabase**: Postgres (таблицы, представления, функции), RLS-политики; Edge Functions (Deno) для: `rollover_daily`, `task_add_rpc` (опционально), reorder.
- **Scheduler**: nightly job (UTC) запускает `rollover_daily` с учётом TZ пользователей.

---

5. Схема данных (SQL DDL)

```sql
-- USERS (если уже есть — дополнить tz)
create table if not exists app_user (
  id uuid primary key default gen_random_uuid(),
  tg_id bigint unique,
  tz text not null default 'Asia/Nicosia',
  created_at timestamptz not null default now()
);

-- TASKS: логическая задача
create type task_status as enum ('todo','in_progress','done','archived');

create table if not exists task (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references app_user(id) on delete cascade,
  title text not null,
  notes text,
  area text, -- #сфера (простая строка в MVP)
  priority smallint default 0,
  rollover boolean not null default true,
  source text default 'bot',
  archived_at timestamptz,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

create table if not exists task_occurrence (
  id uuid primary key default gen_random_uuid(),
  task_id uuid not null references task(id) on delete cascade,
  user_id uuid not null references app_user(id) on delete cascade,
  day date not null, -- дата в локальном дне пользователя (см. функции)
  status task_status not null default 'todo',
  order_index int not null default 1000,
  completed_at timestamptz,
  created_at timestamptz not null default now(),
  unique (task_id, day)
);

-- Триггеры
create or replace function set_updated_at() returns trigger as $$
begin new.updated_at = now(); return new; end; $$ language plpgsql;
create trigger task_updated_at before update on task for each row execute
function set_updated_at();

-- Индексы
```

```sql
create index if not exists idx_task_user on task(user_id);
create index if not exists idx_occ_user_day on task_occurrence(user_id, day);
create index if not exists idx_occ_status on task_occurrence(status);
```

Представление для списка

```sql
-- Возвращает occurrence с join на task (для дня)
create or replace view v_tasks_for_day as
select o.id as occurrence_id, o.day, o.status, o.order_index, o.completed_at,
       t.id as task_id, t.title, t.area, t.priority, t.rollover, t.source,
       o.user_id
from task_occurrence o
join task t on t.id = o.task_id;
```

---

6. Функции/Edge Functions

6.1 Построение «локального» дня

- Все timestamps храним в UTC.
- Локальный «сегодня» вычисляется по `app_user.tz`.
- Утилита `today_in_tz(user_id)` возвращает `date`.

Пример (SQL):

```sql
create or replace function today_local(u uuid) returns date as $$
declare tz text; d date; begin
  select app_user.tz into tz from app_user where id=u;
  select (now() at time zone tz)::date into d; return d;
end; $$ language plpgsql stable;
```

6.2 Добавление задачи (RPC)

Вставляет `task` и создаёт `task_occurrence` на указанную дату (или today_local).

```sql
create or replace function rpc_task_add(p_user uuid, p_title text, p_area text
default null, p_day date default null)
returns uuid as $$
declare t_id uuid; d date; begin
  d := coalesce(p_day, today_local(p_user));
  insert into task(user_id, title, area) values (p_user, p_title, p_area)
```

```
returning id into t_id;
  insert into task_occurrence(task_id, user_id, day) values (t_id, p_user, d);
  return t_id;
end; $$ language plpgsql;
```

6.3 Отметить статус occurrence

```
create or replace function rpc_occ_set_status(p_user uuid, p_occ uuid, p_status
task_status)
returns void as $$
begin
  update task_occurrence
     set status = p_status,
         completed_at = case when p_status='done' then now() else null end
   where id=p_occ and user_id=p_user;
end; $$ language plpgsql;
```

6.4 Ребейз порядка (drag-drop)

Клиент шлёт массив `{occurrence_id, order_index}`.

6.5 Rollover (Edge Function `rollover_daily`)

Псевдокод: - Для каждого пользователя `u`: - `y = today_local(u) - 1 day`, `t = today_local(u)`. - Найти occurrences за `y` со статусом != 'done' и `task.rollover=true`. - Для каждой — upsert occurrence на `t` если ещё нет: статус='todo', order_index наследовать. - Логировать кол-во перенесённых.

Запуск: ежедневный cron в 00:10 локального времени пользователя. Т.к. планировщик в UTC — запускаем каждый час и переносим для тех, у кого локально 00:10±15 мин.

---

7. RLS и безопасность

- Включить RLS на `task`, `task_occurrence`, `app_user`.
- Политики: пользователь видит/меняет только строки со своим `user_id`.

```
alter table task enable row level security;
alter table task_occurrence enable row level security;

create policy task_is_owner on task
for all using (auth.uid() = user_id) with check (auth.uid() = user_id);
```

```
create policy occ_is_owner on task_occurrence
for all using (auth.uid() = user_id) with check (auth.uid() = user_id);
```

- Auth: mini-app получает Supabase JWT после верификации Telegram `initData` на бэке. Бот использует сервис-роль (сервер) без RLS-ограничений для своих RPC.

---

8. Бот: команды и парсинг

- `/task add <текст> [#сфера] [@сегодня|@завтра|@YYYY-MM-DD]`
- `/tasks today` | `/tasks yesterday` | `/tasks <YYYY-MM-DD>`
- Инлайн-кнопки: ✅Done, ▶️In-progress, ↩️Undo.
- Ответ-карточка: порядковый номер, короткий id (первые 4 символа), статус.
- Анти-дубликаты: если в течение 10 сек приходит тот же текст от того же пользователя → idempotency key.

Грамматика парсинга (упрощённо):

```
TEXT := произвольная строка без суффиксов
AREA := '#' WORD
DATE := '@сегодня' | '@завтра' | '@' YYYY '-' MM '-' DD
```

---

9. Mini-app: UX

- Экран **Today**: дата-пикер (влево/вправо), список tasks (occurrences), чекбоксы, drag-drop, тумблер «показывать выполненные».
- Экран **Yesterday**: read-only, с суммой «сколько сделано».
- Плашка «перенесено N задач со вчера» в начале дня.
- Быстрые действия: «+» (добавить — открывает модал с текстом/сферой/датой).
- Локальные optimistic-updates; при ошибке — rollback и тост.

---

10. Нефункциональные требования

- **TZ-корректность**: границы дня по `app_user.tz`.
- **Производительность**: список дня ≤200 задач рендер ≤150 мс; операции статуса ≤300 мс.
- **Надёжность**: идемпотентность при повторных кликах; Edge Function с ретраями.
- **Телеметрия**: логируем RPC, длительность, ошибки, количество перенесённых.

---

11. Acceptance Criteria (AC)

1) `/task add` создаёт Task + Occurrence на нужную дату; отображается в mini-app «Сегодня». 2) Переключение статуса в mini-app и в боте синхронно отражается в обоих интерфейсах. 3) В 00:10 локального времени невыполненные «Вчера» появляются «Сегодня»; выполненные — нет. 4) «Вчера» доступен для просмотра; статусы там неизменяемы. 5) Перетаскивание сохраняет порядок и стабильно восстанавливается при перезагрузке. 6) Политики RLS не позволяют получить или изменить чужие задачи.

---

12. Тест-кейсы (Given-When-Then)

- **Добавление сегодня**: Given пользователь X, When `/task add Сделать матан`, Then задача видна в Today, статус todo.
- **Перенос**: Given 2 задачи, одна done, одна todo вчера; When наступает новый день, Then в Today 1 задача (todo), done исчезла.
- **TZ**: Given tz=Asia/Nicosia, When сейчас 23:30Z=02:30 местн., Then Today вычисляется корректно.
- **RLS**: Given другой пользователь Y, When пытается получить occurrences X, Then 0 строк.
- **Drag-drop**: Given 3 задачи, When меняем порядок, Then order_index сохранён.

---

13. Риски и решения

- **Сдвиг TZ/переезд**: сохранять `tz_change_log`, rollover ориентировать на актуальный tz.
- **Дубли при сбое сети**: idempotency key по (user, hash(text), 30 сек).
- **Конфликты порядка**: нормализовать `order_index` (шаг 100), периодически реиндексировать.

---

14. Следующие шаги

1) Миграции в Supabase (DDL + политики). 2) Edge Function: `rollover_daily` + cron. 3) RPC: `rpc_task_add`, `rpc_occ_set_status`, reorder. 4) Бот: парсер и ответы с инлайн-кнопками. 5) Mini-app: экран Today/Yesterday, drag-drop, optimistic updates. 6) QA по AC и тест-кейсам.