SpiritPlanner — Documentación técnica

1. Visión general

SpiritPlanner es una aplicación en Python con Flet y Firestore para gestionar Eras, Periodos, Incursiones y sus Sesiones. Firestore es la única fuente de verdad y la UI es declarativa. La organización del código sigue separaciones claras por pantalla (UI, estado derivado y efectos) y se apoya en componentes reutilizables para que la lectura y edición sean rápidas.

2. Principios de diseño aplicados

- SRP: cada fichero responde a una sola pregunta concreta.
- Separación de responsabilidades: UI, estado derivado y efectos viven en archivos distintos.
- KISS: funciones cortas y directas; sin abstracciones innecesarias.
- DRY moderado: reutilización solo cuando mejora claridad (componentes compartidos).
- Explícito sobre implícito: estados y decisiones visibles en código.
- Firestore como fuente de verdad: sin cachés locales ni duplicados persistentes.

3. Estructura de carpetas

```
app/
├─ main.py                      # Entrada Flet y enrutado
├─ screens/
│  ├─ shared_components.py      # Componentes comunes de UI
│  ├─ data_lookup.py            # Catálogos TSV en memoria
│  ├─ eras/                     # Pantalla de Eras (lista)
│  ├─ periods/                  # Pantalla de Periodos
│  ├─ incursions/               # Pantalla de Incursiones
│  └─ incursion_detail/         # Pantalla de detalle de Incursión
├─ services/
│  ├─ firestore_service.py      # Acceso a Firestore y reglas
│  └─ score_service.py          # Fórmula de puntuación
└─ utils/                       # Utilidades (logging, navegación, fe

pc/                             # Scripts CLI (generación de Era)
├─ generate_era.py
├─ firestore_service.py
└─ firestore_test.py
```

4. Flujo general de la app

1. `app/main.py` configura Flet, inicializa `FirestoreService` y enruta según URL.
2. Cada pantalla obtiene datos vía handlers que llaman a Firestore (fuente de verdad).
3. El estado derivado y permisos se calculan con funciones puras en `*_state.py`.
4. La UI se compone en `*_screen.py` usando componentes reutilizables (`*_components.py` y `shared_components.py`).
5. Los efectos de usuario (mensajes, navegación, escritura en Firestore) se manejan en `*_handlers.py`.

5. Referencia de módulos y funciones

app/main.py

- `main(page)`: configura tema y scroll, instancia `FirestoreService`, define `add_view`, engancha `handle_route_change` y `handle_view_pop`, y navega a `/eras`.
- `add_view(route, content)`: helper interno que envuelve un control en `ft.View` y lo añade a la pila de vistas.
- `handle_route_change(event)`: recompone la pila de vistas en función de la URL (`/eras`, `/periods`, `/incursions`, `/incursion_detail`) y actualiza la página.
- `handle_view_pop(event)`: gestiona el backstack manual; si no quedan vistas, navega a `/eras`.

app/utils/logger.py

- `_NoiseFilter.filter(record)`: filtra logs de librerías ruidosas salvo que sean `WARNING+`.
- `_configure_logging()`: crea `logs/spiritplanner_<timestamp>.log`, configura nivel DEBUG y aplica el filtro.
- `get_logger(name)`: asegura configuración y devuelve un logger con el nombre indicado.

app/utils/navigation.py

- `go(page, route)`: navegación asíncrona con `push_route`.
- `go_to(page, route)`: devuelve un handler asíncrono para usar en eventos de UI y navegar a la ruta dada.

app/utils/datetime_format.py

- `_parse_datetime(value)`: convierte `datetime` o ISO string en `datetime` consciente en UTC.
- `format_datetime_local(value)`: formatea un instante UTC a hora local con patrón `dd/mm/yy HH:MM` o `—` si falta.

app/services/score_service.py

- `calculate_score(...)`: aplica la fórmula del README (diferenciada para victoria/derrota) y devuelve el entero resultante.

app/services/firestore_service.py

- `ActiveIncursion`: dataclass con `era_id`, `period_id`, `incursion_id`.
- `FirestoreService.__init__()`: inicializa cliente Firestore (app Firebase si falta).
- `_init_firestore()`: helper estático para crear cliente.
- `_utc_now()`: instante actual en UTC.
- `list_eras()`: lee colección `eras`, añade `id` y devuelve lista.
- `list_periods(era_id)`: lee periodos de la Era, añade `id` y ordena por `index`.
- `list_incursions(era_id, period_id)`: lista incursiones del periodo ordenadas por `index`.
- `list_sessions(era_id, period_id, incursion_id)`: lista sesiones ordenadas por `started_at`.
- `get_active_incursion(era_id)`: busca `active_incursion` en la Era o recorre periodos/incursiones abiertas.
- `reveal_period(era_id, period_id)`: valida orden secuencial y marca `revealed_at`.
- `set_incursion_adversary(...)`: valida estado del periodo y actualiza `adversary_id` de una incursión.
- `assign_period_adversaries(...)`: valida reglas del README (4 incursiones, todas con adversario distinto) y fija `adversaries_assigned_at` en batch.
- `start_incursion(...)`: valida periodo revelado, adversarios asignados, unicidad de incursión activa y campos `adversary_level`/`difficulty`; marca `started_at`, `active_incursion` y crea sesión inicial.
- `update_incursion_adversary_level(...)`: actualiza nivel y dificultad (y adversario opcional) de la incursión.
- `resume_incursion(...)`: valida unicidad de incursión activa y, si no hay sesión abierta, crea una nueva.

- `pause_incursion(...)`: cierra la sesión abierta marcando `ended_at`.
- `finalize_incursion(...)`: calcula `score`, pausa sesión abierta, marca `ended_at` y limpia `active_incursion`; si todas las incursiones del periodo están finalizadas, fija `ended_at` del periodo.
- `_create_session(incursion_ref)`: crea sesión `{started_at, ended_at=None}`.
- `_period_complete(era_id, period_id)`: comprueba si todas las incursiones del periodo tienen `ended_at`.

app/screens/shared_components.py

- `header_text(text)`: título grande en negrita.
- `status_chip(label, color)`: chip compacto con color de fondo y texto blanco.
- `section_card(content, ...)`: contenedor con borde y padding para secciones.
- `action_button(...)`: devuelve `OutlinedButton` o `ElevatedButton` según `variant`.

Pantalla: Eras (`app/screens/eras`)

- `eras_state.get_era_status(era)`: devuelve etiqueta/color según `is_active`.
- `eras_state.get_incursion_status(active_count)`: etiqueta/color según incursiones activas.
- `eras_handlers.show_message(page, text)`: snack bar informativo.
- `eras_handlers.build_open_periods_handler(page, era_id)`: handler de navegación a periodos.
- `eras_handlers.build_open_active_handler(page, active_incursion)`: navegación a la incursión activa.
- `eras_handlers.handle_multiple_active_incursions(...)`: muestra aviso si hay más de una incursión activa.
- `eras_handlers.count_active_incursions(service, era_id)`: cuenta incursiones iniciadas y no finalizadas en la Era.
- `eras_handlers.get_active_incursion(service, era_id, active_count)`: obtiene la incursión activa si solo hay una.
- `eras_components.era_card(...)`: tarjeta de Era con chips de estado y acciones.
- `eras_screen.eras_view(page, service)`: compone la vista de Eras, carga la lista y decide botones por Era.

Pantalla: Periodos (`app/screens/periods`)

- `periods_state.can_reveal(periods, index)`: permite revelar si es el primer periodo o el anterior está finalizado.
- `periods_state.get_period_action(period, allow_reveal)`: devuelve la acción visible (`results`, `incursions`, `assign`, `reveal`, `None`).
- `periods_handlers.show_message(page, text)`: snack bar.
- `periods_handlers.close_dialog(page, dialog)`: cierra diálogo y refresca página.
- `periods_handlers.assign_period_adversaries(...)`: llama a `FirestoreService.assign_period_adversaries` y muestra error si falla.
- `periods_handlers.reveal_period(...)`: llama a `FirestoreService.reveal_period` y devuelve éxito/fracaso.
- `periods_components.period_card(...)`: tarjeta de periodo con acciones y vista previa de incursiones.
- `periods_components.incursions_preview(entries)`: contenedor con lista de incursiones.
- `periods_screen.periods_view(...)`: compone la pantalla de periodos, carga incursiones de cada periodo, abre diálogos de asignación y dispara revelado.

Pantalla: Incursiones (`app/screens/incursions`)

- `incursions_state.get_incursion_status(incursion)`: etiqueta/color según si está finalizada, activa o no iniciada.
- `incursions_state.get_spirit_info(incursion)`: texto con nombres de espíritus.
- `incursions_state.get_board_info(incursion)`: texto con nombres de tableros.
- `incursions_state.get_layout_info(incursion)`: nombre de la distribución.
- `incursions_state.get_adversary_info(incursion)`: nombre del adversario asignado.
- `incursions_handlers.build_open_incursion_handler(...)`: handler para abrir detalle de incursión.
- `incursions_handlers.list_incursions(...)`: proxy a `FirestoreService.list_incursions`.
- `incursions_components.incursion_card(...)`: tarjeta con resumen de incursión y botón Abrir.
- `incursions_screen.incursions_view(...)`: compone la lista de incursiones de un periodo y conecta acciones de apertura.

Pantalla: Detalle de incursión (`app/screens/incursion_detail`)

- `incursion_detail_state.resolve_session_state(incursion, open_session)`: determina estado de sesión (`NO_INICIADA`, `ACTIVA`, `PAUSADA`, `FINALIZADA`).
- `incursion_detail_state.can_edit_adversary_level(incursion, has_sessions)`: solo permite editar nivel si no hay sesiones ni `ended_at`.
- `incursion_detail_state.build_period_label(period)`: etiqueta “Periodo X” o “Periodo —”.
- `incursion_detail_state.get_result_label(result_value)`: “Victoria”/“Derrota”.
- `incursion_detail_state.get_score_formula(result_value)`: fórmula textual según resultado.
- `incursion_detail_state.total_minutes(sessions, now)`: suma minutos de todas las sesiones abiertas/cerradas.
- `incursion_detail_state.compute_score_preview(...)`: devuelve fórmula y puntuación calculada o `None`.
- `incursion_detail_handlers.show_message(page, text)`: snack bar.
- `incursion_detail_handlers.close_dialog(page, dialog)`: cierra diálogo modal.
- `incursion_detail_handlers.get_incursion(...)`: busca una incursión concreta dentro de la lista del periodo.
- `incursion_detail_handlers.get_period(...)`: obtiene el periodo actual desde Firestore.
- `incursion_detail_handlers.list_sessions(...)`: lista sesiones de una incursión.
- `incursion_detail_handlers.update_adversary_level(...)`: proxy a `FirestoreService.update_incursion_adversary_level`.
- `incursion_detail_handlers.finalize_incursion(...)`: proxy a `FirestoreService.finalize_incursion`.
- `incursion_detail_handlers.start_incursion(...)`: proxy a `FirestoreService.start_incursion`.
- `incursion_detail_handlers.pause_incursion(...)`: proxy a `FirestoreService.pause_incursion`.
- `incursion_detail_handlers.resume_incursion(...)`: proxy a `FirestoreService.resume_incursion`.
- `incursion_detail_components.dark_section(content)`: contenedor oscuro para cabecera.
- `incursion_detail_components.light_section(content)`: contenedor claro con borde.

- `incursion_detail_components.summary_tile(icon, label, on_click)`: baldosa resumida con icono y callback.
- `incursion_detail_screen.incursion_detail_view(...)`: pantalla de detalle; muestra setup, controla FAB o app bar inferior para sesiones, abre diálogos de sesiones y puntuación, permite finalizar incursión y actualizar dificultad.

app/screens/data_lookup.py

- `AdversaryLevel`, `AdversaryInfo`: dataclasses inmutables para catálogos.
- `_data_dir()`: resuelve la ruta `pc/data/input`.
- `_load_tsv_rows(filename, required_fields)`: lee TSV y devuelve filas válidas con campos obligatorios presentes.
- `_load_simple_map(filename, key_field, value_field)`: crea diccionario simple clave→valor desde TSV cacheado.
- `get_spirit_name(spirit_id)`: devuelve nombre de espíritu o `—`.
- `get_board_name(board_id)`: devuelve nombre de tablero o `—`.
- `get_layout_name(layout_id)`: devuelve nombre de distribución o `—`.
- `get_adversary_catalog()`: carga adversarios con niveles/dificultad, cacheado.
- `get_adversary_name(adversary_id)`: nombre del adversario o “Desconocido”.
- `get_adversary_levels(adversary_id)`: niveles de un adversario.
- `get_adversary_difficulty(adversary_id, level)`: dificultad asociada a un nivel concreto.

Scripts PC (`pc/`)

- `pc/firestore_service.py`:
  - `init_firestore()`: inicializa cliente Firebase si falta.
  - `era_exists(era_id)`: comprueba existencia de la Era.
  - `create_era(era_id)`: crea documento Era con `is_active` y `created_at`.
  - `create_period(era_id, period_id, index)`: inserta periodo con índice y `created_at`.
  - `create_incursion(era_id, period_id, incursion_id, data)`: inserta incursión con datos proporcionados.
- `pc/generate_era.py`:
  - Dataclasses `Spirit`, `Board`, `Layout` para tipar entradas.
  - `require_columns(fieldnames, required, path)`: valida que el TSV contiene columnas necesarias.
  - `load_spirits(path)`, `load_boards(path)`: leen TSV y devuelven listas tipadas.
  - `validate_adversaries(path)`: asegura columnas de adversarios.

    - `load_layouts(path)`: lee layouts con player_count y flag activo.
    - `generate_round_robin(spirits)`: genera emparejamientos rotatorios de espíritus.
    - `generate_board_rounds(boards)`: genera emparejamientos rotatorios de tableros.
    - `assign_boards(boards, match_count, rng)`: reparte tableros balanceados para un periodo.
    - `select_layouts(layouts)`: filtra layouts activos para 2 jugadores.
    - `assign_layouts(layouts, match_count, period_index)`: rota layouts por periodo.
    - `write_era_tsv(path, era_id, rounds, boards, layouts, rng)`: genera TSV de incursiones barajadas.
    - `write_era_firestore(...)`: crea Era/Periodos/Incursiones en Firestore según rondas.
    - `parse_args()`: define CLI y rutas por defecto de TSV.
    - `main()`: orquesta carga de datos, genera rondas y escribe en Firestore/TSV.
- `pc/firestore_test.py`:
    - Script mínimo para verificar conexión a Firestore y listar colecciones.

6. Cómo modificar de forma segura

- UI: edita `*_screen.py` o `*_components.py`. Mantén textos visibles en castellano y no añadas lógica de negocio aquí.
- Reglas y estado derivado: modifica funciones en `*_state.py` (puras, sin Flet ni Firestore).
- Handlers / efectos: usa `*_handlers.py` para llamadas a Firestore, mensajes y navegación. No construyas UI aquí.
- Catálogos: si cambian textos de espíritus/tableros/adversarios, actualiza los TSV en `pc/data/input/` y deja que `data_lookup.py` los consuma.
- Logging/Navegación: reutiliza `app/utils/logger.py` y `app/utils/navigation.py` para mantener consistencia.

7. Pendiente según README

- Implementar exportación a TSV (Android/CLI) para incursiones finalizadas con orden Era → Periodo → Incursión y timestamps en ISO UTC (no existe flujo ni script de exportación actualmente).
- Revisar conversión de zona horaria en cualquier nueva vista: debe aplicar hora local del dispositivo respetando DST (actualmente centralizado en

`format_datetime_local`, reusar en nuevas pantallas).

- Mantener el flujo “un único botón por periodo” y estados implícitos en UI: al añadir nuevas acciones, conservar la lógica de selección en `periods_state.get_period_action`.

8. Qué no hacer

- No introducir clases de dominio con estado ni patrones DDD/Clean/MVC.
- No duplicar datos de Firestore en cachés locales ni modificar nombres de campos.
- No mezclar lógica de negocio con UI ni añadir capas adicionales de abstracción.
- No alterar reglas de asignación, unicidad de incursión activa o cálculo de puntuación sin actualizar `FirestoreService` y `score_service`.