DeployTime

Documento de Arquitectura y Guía para Desarrolladores

Versión: 1.0
Autor funcional: Tincho / ComunIA - DIXER
Propósito: Dejar documentadas las decisiones de diseño para que cualquier desarrollador pueda continuar el proyecto sin depender del contexto oral previo.

1. Objetivo del sistema

DeployTime es una aplicación de escritorio para Windows que permite registrar, por colaborador, el tiempo invertido en cada proyecto y tarea. El objetivo principal es medir el esfuerzo real por proyecto/tarea para mejorar la estimación de presupuestos y analizar la productividad.

El sistema debe funcionar aun sin conexión a internet (modo offline) y sincronizarse con un servidor remoto cuando haya conectividad. En el servidor se centralizan los datos, se exponen reportes y se prepara un frontend web para configuración y análisis.

2. Arquitectura general

Componentes principales:

- Aplicación Desktop (Electron + React) que corre en Windows 11.
- Base de datos local (MySQL en esta fase) para soportar modo offline.
- Servidor remoto con Laravel + MySQL, expuesto por HTTPS.
- API REST entre app Desktop y servidor remoto, usando JSON sobre HTTPS.
- Módulo de sincronización que mantiene coherencia entre base local y remota.

2.1. Flujo alto nivel

1) El usuario colabora desde la app de escritorio, seleccionando organización, proyecto y tarea. 2) La app registra los time entries en la base local, permitiendo pausa y stop del cronómetro. 3) Cuando hay conexión, la app llama al endpoint /api/sync del servidor Laravel. 4) El servidor integra los cambios locales, aplica reglas de conflicto y devuelve cambios remotos. 5) La app aplica los cambios en la base local y actualiza su estado de sincronización.

3. Multi-tenant y organizaciones

El sistema está diseñado para ser multi-empresa (multi-tenant lógico) a través del concepto de organización. Cada organización representa una empresa o cliente. Usuarios, proyectos, tareas y time entries están siempre asociados a una organización.

3.1. Regla principal de aislamiento

La aplicación Desktop debe trabajar siempre con UNA sola organización activa por instalación. En el servidor, todos los queries de proyectos, tareas y time entries deben estar filtrados por organization_id. Un usuario puede pertenecer a varias organizaciones, pero en una instalación concreta de DeployTime se selecciona una organización activa.

4. Modelo de datos (local y remoto)

El objetivo es que la base de datos local y la remota compartan el mismo modelo lógico, para simplificar la sincronización. La DB local almacena un subconjunto filtrado por organización, mientras que la remota almacza todos los tenants.

4.1. Tabla organizations

Representa una empresa/cliente.

Campos: - id (UUID, PK) - name (string, NOT NULL) - created_at (datetime) - updated_at (datetime)

4.2. Tabla users

Representa a los usuarios del sistema (colaboradores, administradores).

Campos: - id (UUID, PK) - name (string, NOT NULL) - email (string, NOT NULL, único) - password (string, NOT NULL en remoto, hash bcrypt/argon2; opcional/no usado en local) - active (boolean, default true) - created_at (datetime) - updated_at (datetime)

4.3. Tabla pivot organization_user

Relaciona usuarios con organizaciones y define su rol dentro de cada organización.

Campos: - organization_id (UUID, FK → organizations.id) - user_id (UUID, FK → users.id) - role (string, ej. 'admin', 'member') - created_at (datetime) PK compuesta por (organization_id, user_id).

4.4. Tabla projects

Representa los proyectos sobre los que se registran tareas y tiempos.

Campos: - id (UUID, PK) - organization_id (UUID, FK → organizations.id) - name (string, NOT NULL) - description (text, NULL) - archived (boolean, default false) - created_at (datetime) - updated_at (datetime)

4.5. Tabla tasks

Representa las tareas dentro de un proyecto.

Campos: - id (UUID, PK) - project_id (UUID, FK → projects.id) - name (string, NOT NULL) - description (text, NULL) - estimated_hours (decimal o int en minutos, NULL) - archived (boolean, default false) - created_at (datetime) - updated_at (datetime)

4.6. Tabla time_entries

Registra sesiones de trabajo concretas sobre una tarea de un proyecto.

Campos: - id (UUID, PK) - user_id (UUID, FK → users.id) - project_id (UUID, FK → projects.id) - task_id (UUID, FK → tasks.id) - start_time (datetime, NOT NULL) - end_time (datetime, NULL si está abierto) - duration_seconds (int, NOT NULL, >= 0) - created_at (datetime) - updated_at (datetime)

4.7. Tabla clients (solo remoto)

Registra las estaciones de trabajo conocidas por el servidor para diagnóstico de sincronización.

Campos: - id (UUID o autoincrement, PK) - client_id (UUID, único, requerido) - name (string, NULL, descripción opcional de la PC) - last_sync_at (datetime, NULL) - created_at (datetime) - updated_at (datetime)

4.8. Tabla client_state (solo local)

Tabla de una sola fila por instalación, que modela el estado de sincronización del cliente.

Campos: - id (UUID, PK) - client_id (UUID, requerido, identifica esta instalación) - organization_id (UUID, requerido) - user_id (UUID, requerido, usuario con el que se asoció esta instalación) - last_sync_at (datetime, NULL en primera sync) - created_at (datetime) - updated_at (datetime) Solo debe existir una fila en esta tabla.

5. API REST

La API expuesta por Laravel se basa en JSON sobre HTTPS. La autenticación se realiza mediante token Bearer obtenido por login. Todas las rutas protegidas deben requerir Authorization: Bearer . A continuación se definen los endpoints clave.

5.1. POST /api/login

Objetivo: autenticar al usuario y devolver token + organizaciones vinculadas.

Request (JSON):

```
{ "email": "string", "password": "string" }
```

Response 200 (JSON):

```
{ "token": "string", "user": { "id": "uuid", "name": "string", "email":
"string" }, "organizations": [ { "id": "uuid", "name": "string" } ] }
```

Errores típicos: 400 (datos inválidos), 401 (credenciales incorrectas), 403 (usuario inactivo).

5.2. GET /api/health

Objetivo: healthcheck para probar URL del backend desde la app Desktop.

Response 200 (JSON):

```
{ "status": "ok", "version": "x.y.z", "time": "ISO-8601" }
```

5.3. POST /api/sync

Objetivo: sincronización bidireccional entre la base local y la remota, filtrada por organización. La app envía su client_id, organization_id, last_sync_at y los time_entries modificados. El servidor devuelve usuarios, proyectos y tareas que cambiaron desde last_sync_at, junto con un nuevo next_sync_at.

Request (JSON, esquema):

```
{ "client_id": "uuid", "organization_id": "uuid", "last_sync_at": "ISO-8601 o
null", "time_entries": [ { "id": "uuid", "user_id": "uuid", "project_id":
"uuid", "task_id": "uuid", "start_time": "ISO-8601", "end_time": "ISO-8601 o
null", "duration_seconds": 1234, "created_at": "ISO-8601", "updated_at":
"ISO-8601" } ] }
```

Response 200 (JSON, esquema):

```
{ "next_sync_at": "ISO-8601", "users": [ { ... } ], "projects": [ { ... } ],
"tasks": [ { ... } ], "time_entry_results": [ { "id": "uuid", "status":
"ok|error", "error_code": "string?", "error_message": "string?" } ] }
```

Reglas de conflicto: - Usuarios, proyectos, tareas: REMOTO manda (los cambios remotos pisan lo local). - Time entries: LOCAL manda (se prioriza la versión local en caso de conflicto, o se registra conflicto explícito).

6. Estrategia de sincronización

6.1. Estado del cliente (client_state)

En la DB local existe una única fila en client_state que contiene: - client_id: UUID que identifica la instalación. - organization_id: organización activa en esta instalación. - user_id: usuario con el que se asoció inicialmentet esta instalación. - last_sync_at: última sincronización exitosa. Este estado se usa en cada llamada a /api/sync.

6.2. Primera sincronización

Flujo recomendado: 1) Usuario abre la app por primera vez. 2) App muestra wizard para configurar URL del servidor (guardado en archivo de config). 3) App llama a /api/health para validar conectividad. 4) Usuario hace login vía /api/login. 5) Servidor devuelve token + organizaciones asociadas. 6) Si hay varias organizaciones, la app solicita al usuario elegir una. 7) App genera client_id, crea fila en client_state y almacena organization_id y user_id. 8) App llama a /api/sync con last_sync_at = null para bajar usuarios, proyectos y tareas. 9) Luego de aplicar la respuesta, last_sync_at se actualiza y la app queda lista para uso offline.

6.3. Sincronizaciones posteriores

En uso normal: - La app registra time entries siempre en la DB local. - Marca internamente qué entries están pendientes de sincronización. - Cada vez que se cierra un time entry, o cada cierto intervalo, se llama a /api/sync. - En cada sync se envían solo los time entries pendientes y se recibe el delta de usuarios/proyectos/tareas cambiados en el servidor desde last_sync_at. - Si la sync es exitosa, se actualiza last_sync_at en client_state. - Si falla (sin conexión o error 5xx), los registros siguen marcados como pendientes y se reintentará más adelante.

7. Seguridad

7.1. Transporte

Todas las comunicaciones entre la app Desktop y Laravel deben realizarse por HTTPS. El servidor debe tener un certificado TLS válido y forzar redirección HTTP → HTTPS. Esto garantiza cifrado en tránsito de credenciales, tokens y datos de negocio.

7.2. Autenticación y autorización

La autenticación se basa en email + password vía /api/login. Las passwords se guardan siempre hasheadas en el servidor. La app nunca almacena la password, solo el token de acceso. Todas las peticiones a /api/sync requieren Authorization: Bearer . La autorización se aplica verificando que el usuario autenticado pertenece a organization_id en cada request de sync. El backend nunca debe devolver ni aceptar datos de organizaciones a las que el usuario no pertenece.

7.3. Base de datos remota y local

La base de datos remota solo debe ser accesible desde Laravel (misma máquina o red privada). No se debe exponer el puerto de MySQL a internet. La base local se protege principalmente mediante el sistema operativo (usuario de Windows, cifrado de disco). No se guardan passwords en la base local. El token puede almacenarse sin cifrado adicional en esta etapa, con la opción futura de cifrarlo si el contexto de riesgo lo requiere.

8. Checklist para implementación

- Definir físicamente las bases de datos local y remota siguiendo el modelo descrito.
- Implementar en Laravel las migraciones para organizations, users, organization_user, projects, tasks, time_entries, clients.
- Configurar autenticación en Laravel (Fortify/Sanctum o similar) y /api/login.
- Implementar /api/health para healthcheck simple.
- Implementar /api/sync respetando el contrato y las reglas de conflicto.
- Configurar HTTPS en el servidor (certificado y redirección HTTP → HTTPS).
- En la app Desktop, implementar el wizard de servidor (URL + /api/health).
- Implementar flujo de login, selección de organización y creación de client_state.
- Implementar lógica de creación de proyectos y tareas desde la app (alta y baja sin edición).
- Implementar cronómetro con start/pause/stop que genere time_entries consistentes.
- Implementar módulo de sincronización en la app: construcción de payload, manejo de respuesta, actualización de last_sync_at.
- Agregar logs básicos en servidor y cliente para poder diagnosticar problemas de sync.