Plan de Implementación de Reportes

1. Objetivo

Diseñar y exponer reportes administrables sobre reservas y documentos, con control de acceso por roles (usuarios vs. administradores) y exportación a CSV/PDF.

2. Tareas Prioritarias

2.1 Control de Acceso

- [ ] Agregar campo `role` al modelo de usuario (`admin` | `user`).
- [ ] Ajustar generación de JWT para incluir `role` .
- [ ] Crear dependencia FastAPI `require_admin` .
- [ ] Migrar la base de datos de auth para asignar el primer usuario administrador.
- [ ] Actualizar frontend (contexto de autenticación) para leer el rol y ocultar accesos no permitidos.

Snippets sugeridos

```python
# auth-service/db_auth.py
role: Mapped[str] = mapped_column(String(20), default="user")
```

```python
# auth-service/main.py
def create_access_token(data: dict, expires_delta: timedelta | None = None):
    ...
    to_encode["role"] = user.role
```

```python
# reservations-service/main.py
def require_admin(current_user: TokenData = Depends(get_current_user)):
    if current_user.role != "admin":
        raise HTTPException(status_code=403, detail="Solo Administradores")
```

2.2 Nuevo Módulo Backend de Reportes

- [ ] Crear servicio/módulo `reports-service` (FastAPI) o añadir módulo a `reservations-service` .
- [ ] Rutas iniciales:
  - `POST /api/reports/generate`
  - `GET /api/reports` (lista)
  - `GET /api/reports/{id}/download`

- [ ] Modelos SQLAlchemy:
  - `ReportFile(id, owner_id, scope, type, status, format, filters_json, storage_path, created_at, completed_at)`
  - `ReportSchedule(id, cron, type, params, active)`
- [ ] Task Celery `generate_report_task` .
- [ ] Guardar archivos en MinIO (`reports/{year}/{id}.csv`).
- [ ] Health check del servicio.

Snippets sugeridos

```
# reports-service/tasks.py
@celery_app.task(bind=True, autoretry_for=(Exception,), max_retries=3)
def generate_report_task(self, report_id: str):
    report = repo.get(report_id)
    data_iter = queries.fetch_reservations(report.filters)
    temp_file = writers.write_csv(data_iter)
    storage.save_object(report.storage_path, temp_file, content_type="text/csv")
    repo.mark_as_completed(report_id, temp_file.size)
```

2.3 Consultas de Datos

1. **Reservas por día/servicio**

```
SELECT date(start_time) AS day,
       service_type,
       COUNT(*) AS total,
       COUNT(*) FILTER (WHERE status = 'cancelled') AS cancellations
FROM reservations
WHERE start_time BETWEEN :start AND :end
GROUP BY day, service_type
ORDER BY day;
```

2. **Uso de Documentos**

```
SELECT document_type,
       COUNT(*) AS uploads,
       COUNT(*) FILTER (WHERE status = 'approved') AS approved,
       COUNT(*) FILTER (WHERE status = 'rejected') AS rejected
FROM documents
WHERE created_at BETWEEN :start AND :end
GROUP BY document_type;
```

2.4 Exportación

- [ ] Crear escritor CSV (`csv.writer`) y opción PDF (WeasyPrint o ReportLab).

- [ ] Generar metadatos (`size` , `checksum` , `expires_at`).
- [ ] Endpoint de descarga con URL firmada MinIO (`presigned_get_object`).

2.5 Integración Frontend

- [ ] Crear carpeta `src/services/reportsAPI.js` .
- [ ] Crear vista `src/components/admin/ReportsDashboard.jsx` .
- [ ] Contenido:
  - Formularios de filtro (rango fecha, estado, servicio).
  - Tabla de reportes generados con paginación.
  - Botones “Descargar CSV/PDF”, “Programar”.
  - Uso de `react-query` o SWR para polling de estados.

```
// src/services/reportsAPI.js
const api = axios.create({ baseURL: '/api/reports' });

export const createReport = (payload) => api.post('/generate', payload);
export const listReports = (params) => api.get('/', { params });
export const downloadReport = (id) => api.get(`/${id}/download`, { responseType: 'blob' });
```

2.6 Notificaciones

- [ ] Reutilizar notifications-service para notificar al administrador cuando el reporte esté listo.
- [ ] Payload sugerido:

```
{
  "notification_type": "report_ready",
  "recipient_email": "admin@municipio.cl",
  "data": {
    "report_name": "Reservas_Semana_45",
    "download_url": "https://...",
    "generated_at": "2025-11-07T03:40:00Z"
  }
}
```

2.7 Auditoría y Seguridad

- [ ] Registrar en `audit_logs` cada generación/descarga de reporte.
- [ ] Limitar la vigencia de los enlaces (24-48 horas).
- [ ] Validar filtros (rangos válidos, combos permitidos).
- [ ] Controlar tamaño máximo (ej. limitar a 50k filas por solicitud).

3. Roadmap sugerido

1. ⚙️ **(En progreso)** Añadir roles y dependencia `require_admin`.
2. 📦 Crear modelos/tablas `report_files`.
3. 🛠️ Endpoint `POST /generate` + task Celery para CSV de reservas.
4. 📤 Guardar en MinIO + endpoint `GET /download`.
5. 🖥️ UI admin básica para lanzar y ver reportes.
6. 🔁 Programador (opcional) para reportes recurrentes.
7. 🧪 Pruebas unitarias e integración (FastAPI + Celery).

4. Consideraciones

- **¿Nuevo servicio o módulo?** Recomendado separar como `reports-service` (Microservicio dedicado) para escalar y aislar lógica de BI.
- **Usuario vs Administrador:**
  - Usuario final: sólo acceso a “Mis reportes” (reservas personales).
  - Administrador: acceso completo + filtros especiales + programación.
- **Documentación:** Actualizar README y diagramas una vez completado.