Aurora
Informe diario de actividades

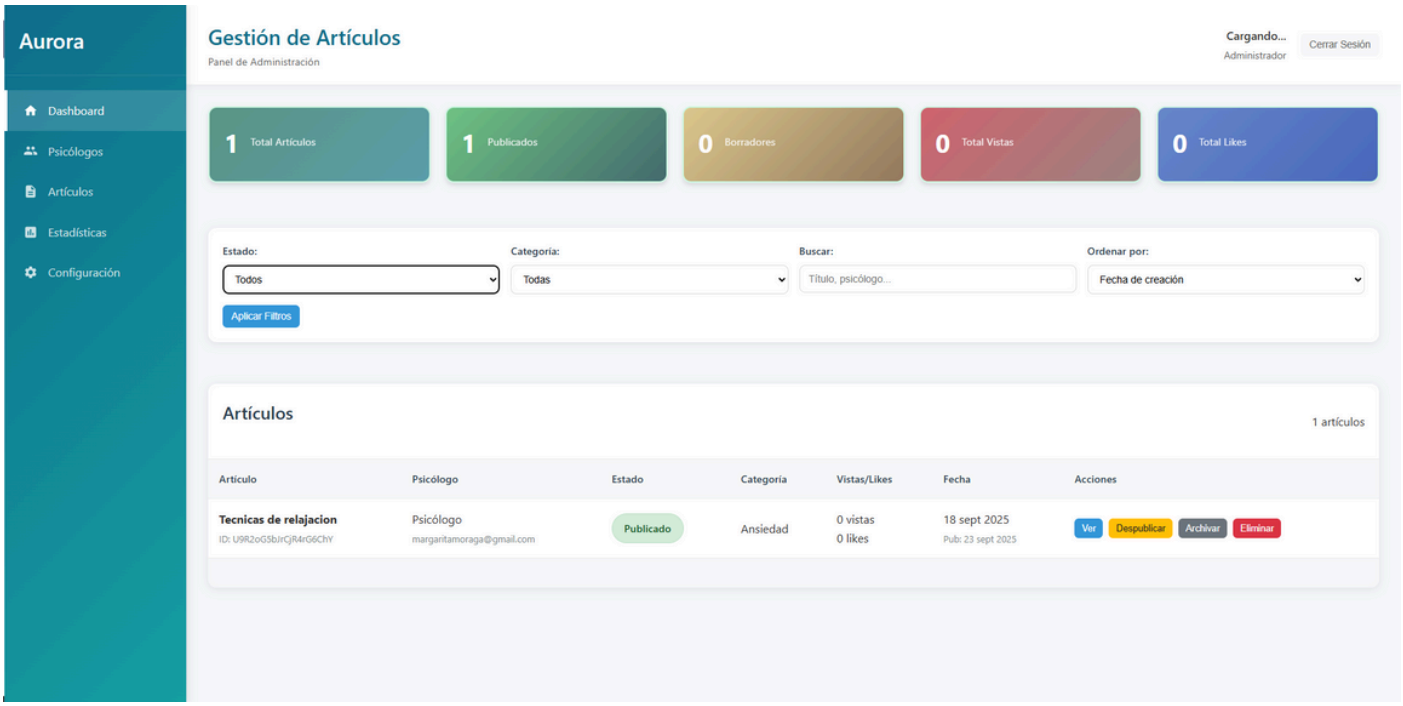
Fecha del informe	**23 septiembre 2025**
Fase y semana	**Fase 8- Semana 12**
Participantes	**Alan y Margarita**

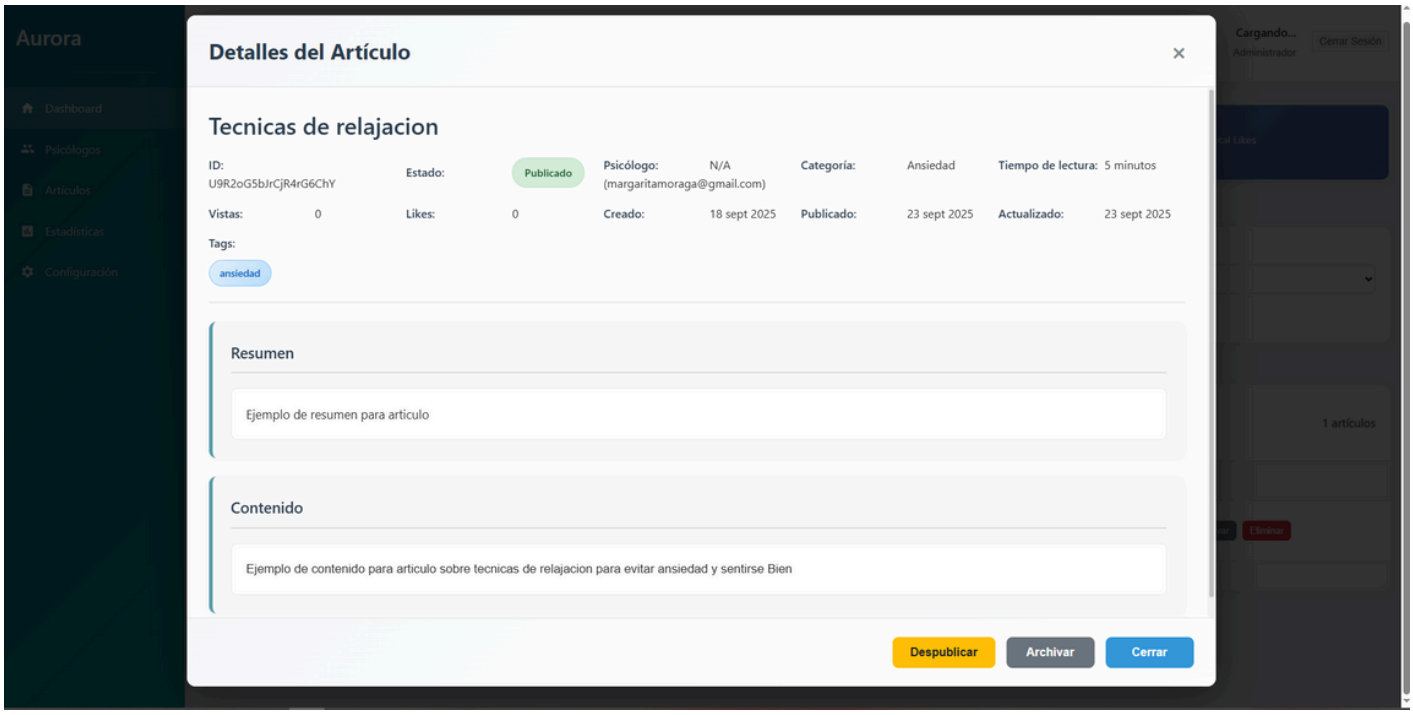
Actividades Realizadas

Tarea	% completado	Notas
• Implementación de funcionalidad de publicación de artículos en panel administrador. • Des publicación de articulo a través del panel administrador. • Archivamiento de artículos en el panel administrador • Des archivamiento de artículos en panel administrador. • Cambios de estado de artículos en la base de datos. • End point con cambio de estado de borrador a publicado al articulo y fecha de publicación. • Guardado de estado publicado en la base de datos. • Configuración de end point para obtener los artículos publicados. • Implementación de diseño para ejercicios diarios. • Implementación de funcionalidad de botón de comenzar ejercicio con temporizador y notas al finalizar. • Inicio de documento para plan de pruebas.	70%	Conectar backend para ejercicios y artículos .

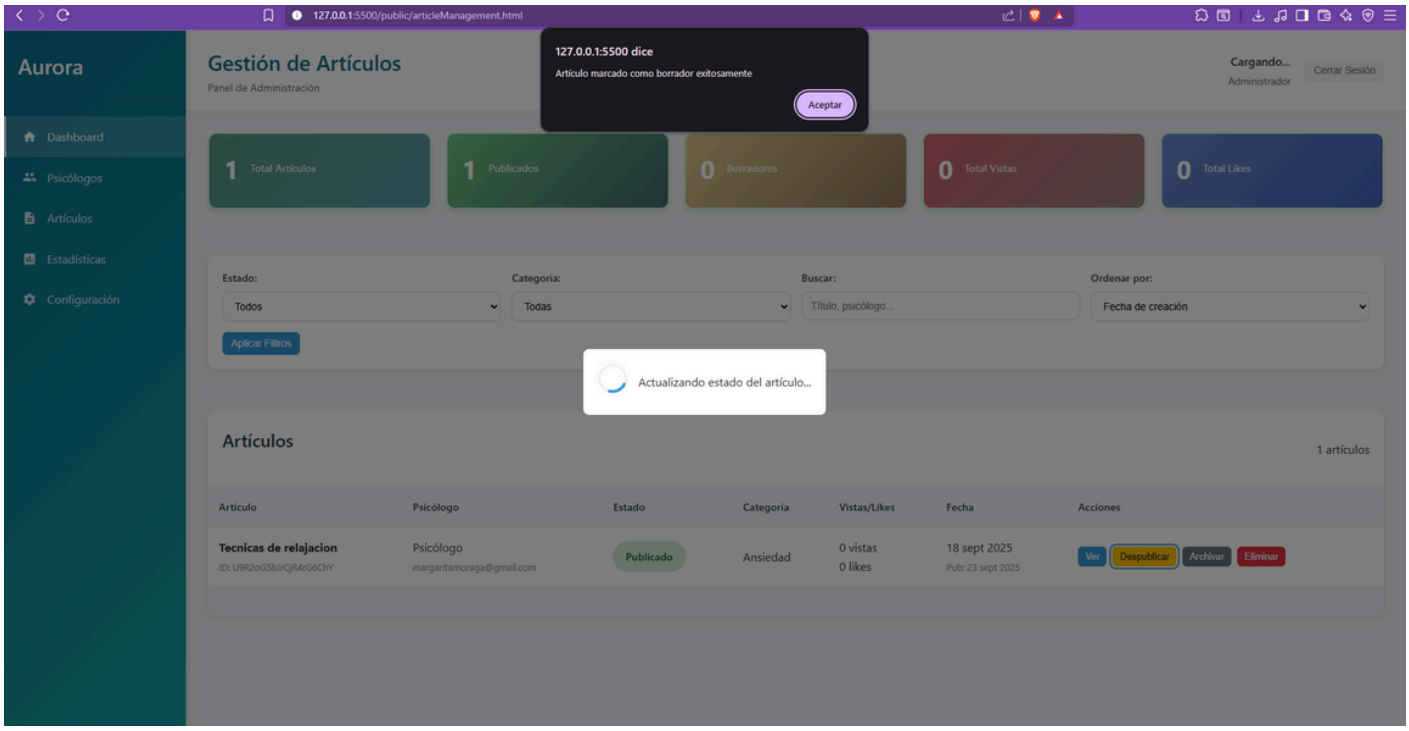
Evidencia de actividades

- Implementación de funcionalidad de publicación de artículos en panel administrador.

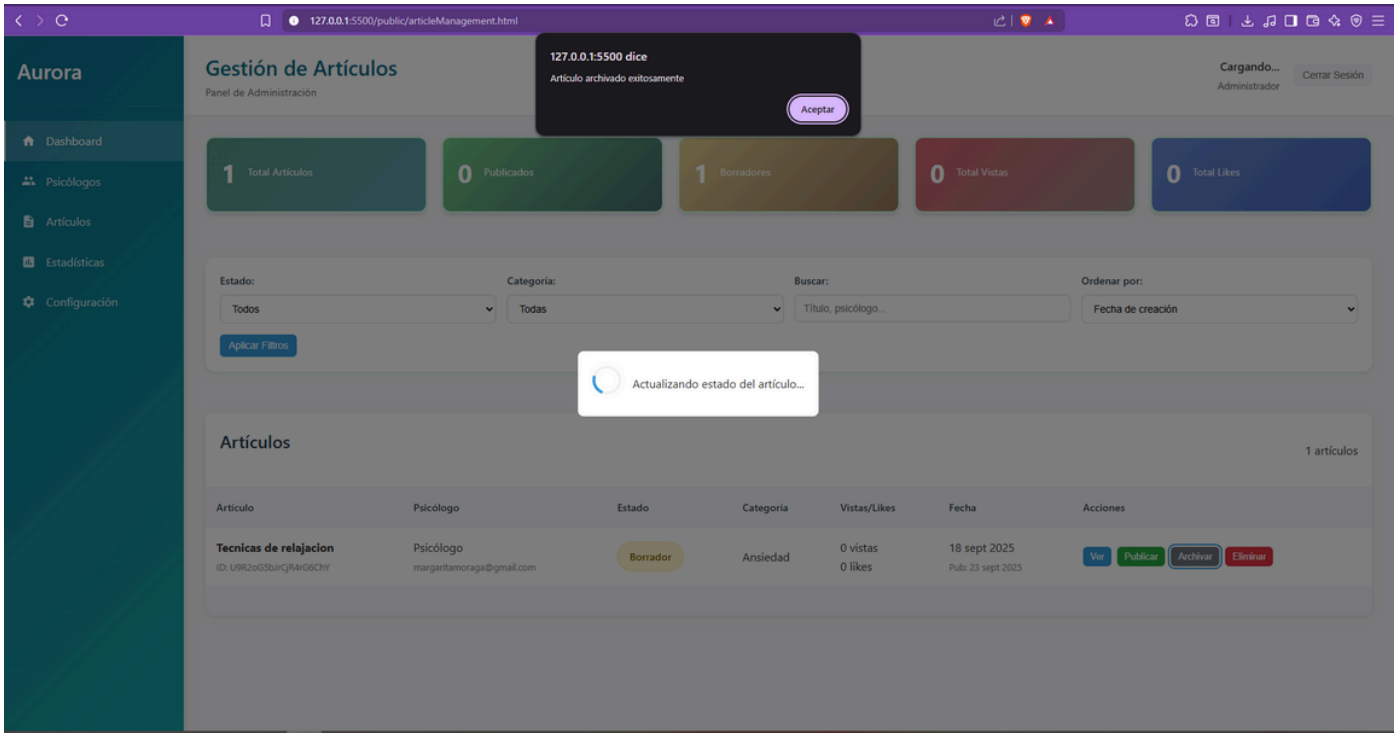




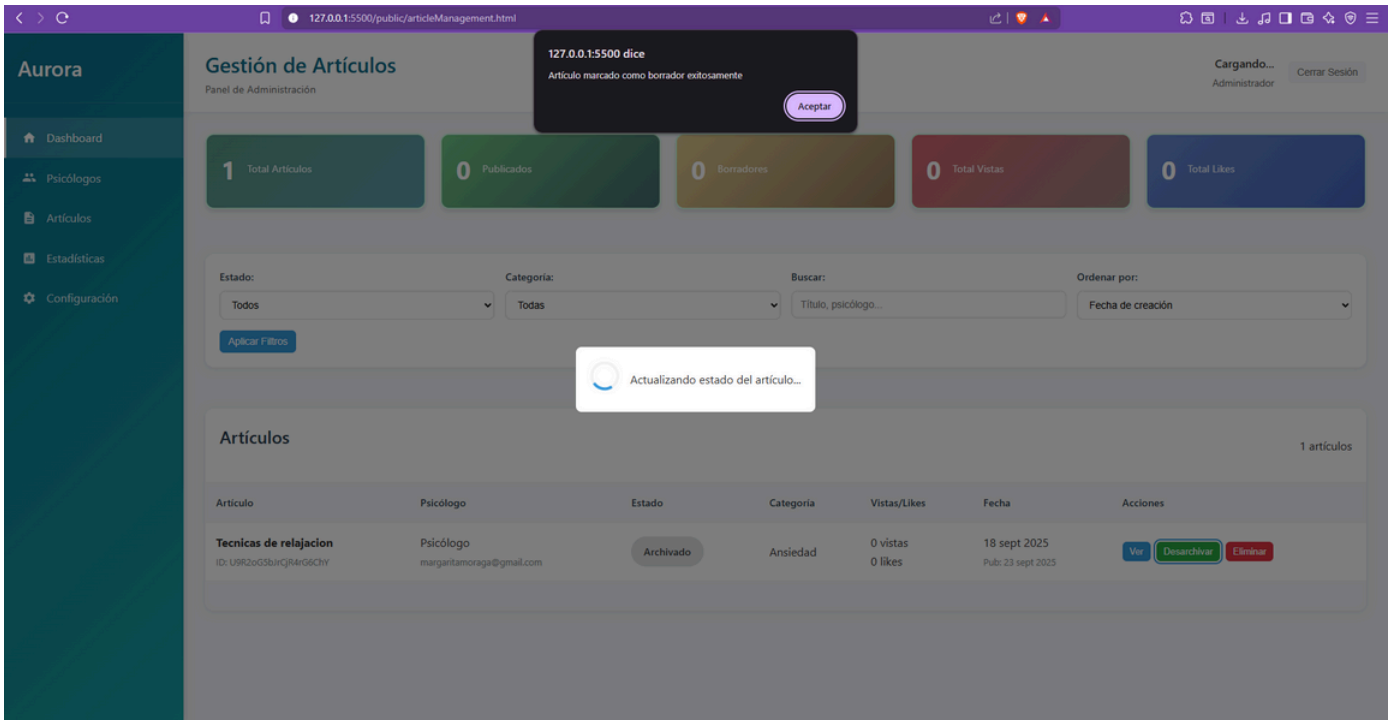
- Des publicación de articulo a través del panel administrador.



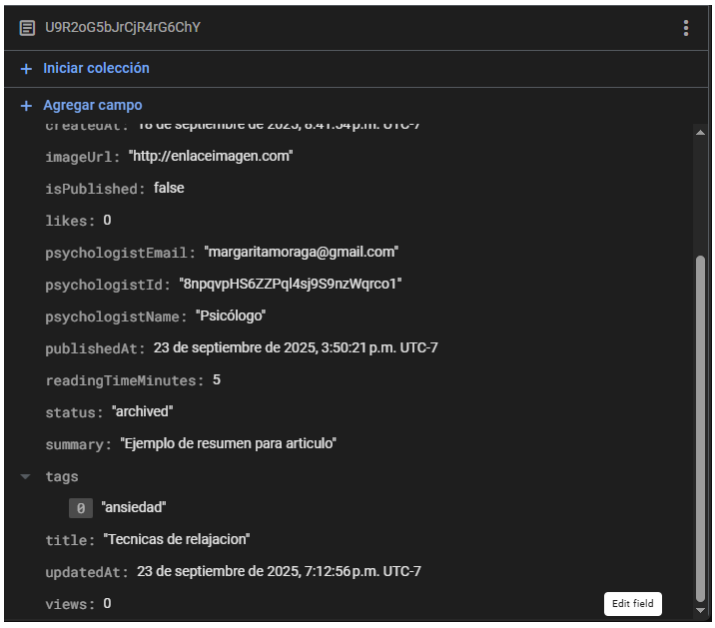
- Archivamiento de artículos en el panel administrador.

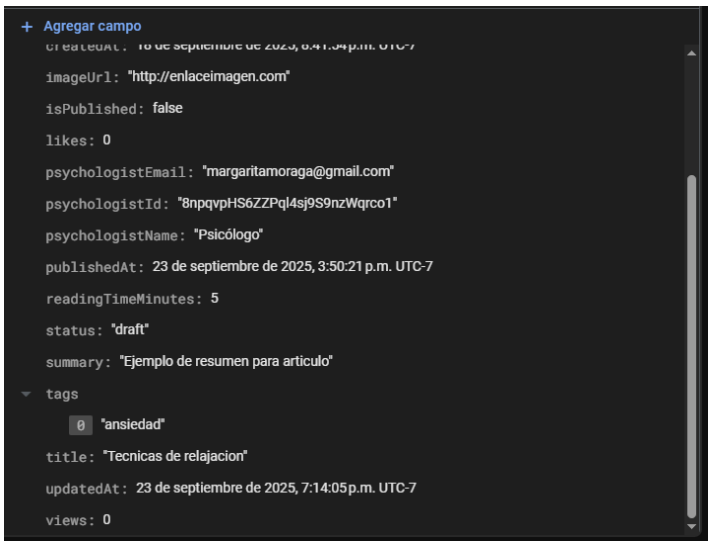


- Des archivamiento de artículos en panel administrador.



- Cambios de estado de artículos en la base de datos.

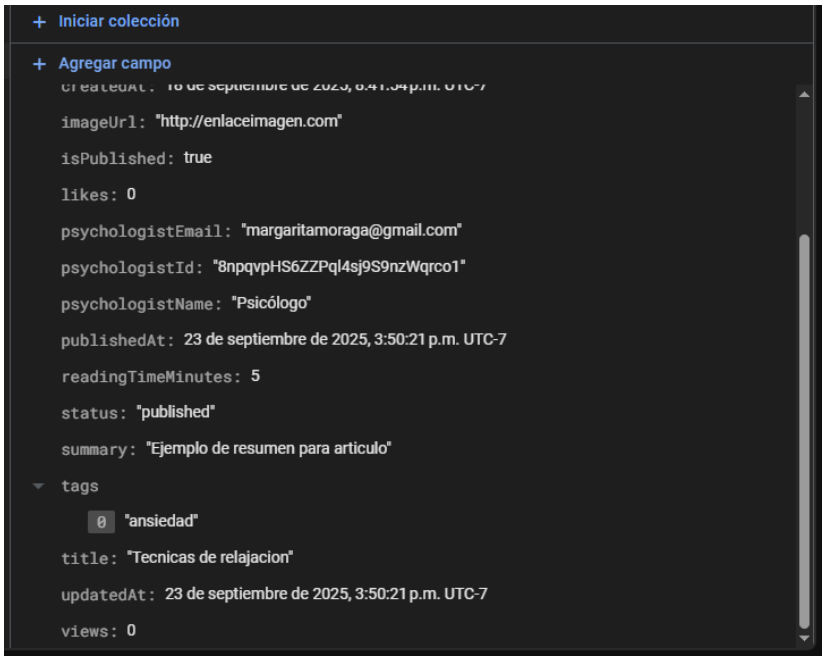




- End point con cambio de estado de borrador a publicado al articulo y fecha de publicación.

```
// Actualizar estado de un articulo (publicar/despublicar/archivar)
adminArticleRouter.put("/articles/:articleId/status", async (req, res) => {
  try {
    // Verificar que es admin
    await verifyIsAdmin(req.userId);
  } catch (adminError) {
    return res.status(403).json({ error: adminError.message });
  }

  const { articleId } = req.params;
  const { status, adminNote } = req.body;

  if (!['published', 'draft', 'archived', 'deleted'].includes(status)) {
    return res.status(400).json({
      error: "Estado inválido. Debe ser: published, draft, archived, o deleted"
    });
  }

  try {
    const articleDoc = await db.collection('articles').doc(articleId).get();

    if (!articleDoc.exists) {
      return res.status(404).json({ error: "Articulo no encontrado" });
    }

    const updateData = {
      status: status,
      updatedAt: FieldValue.serverTimestamp(),
      adminNote: adminNote || null
    };

    // Actualizar campos especificos según el estado
    if (status === 'published') {
      updateData.isPublished = true;
      if (!articleDoc.data().publishedAt) {
        updateData.publishedAt = FieldValue.serverTimestamp();
      }
    } else {
      updateData.isPublished = false;
    }

    if (status === 'deleted') {
      updateData.deletedAt = FieldValue.serverTimestamp();
    }

    await db.collection('articles').doc(articleId).update(updateData);

    res.json({
      success: true,
      message: `Articulo ${status === 'published' ? 'publicado' :
                status === 'draft' ? 'marcado como borrador' :
                status === 'archived' ? 'archivado' : 'eliminado'} exitosamente`,
      articleId: articleId,
      newStatus: status
    });

  } catch (error) {
    console.error("Error actualizando estado del articulo:", error);
    res.status(500).json({
      error: "Error al actualizar estado del articulo",
      details: process.env.NODE_ENV === 'development' ? error.message : undefined
    });
  }
});
```

- Guardado de estado publicado en la base de datos.



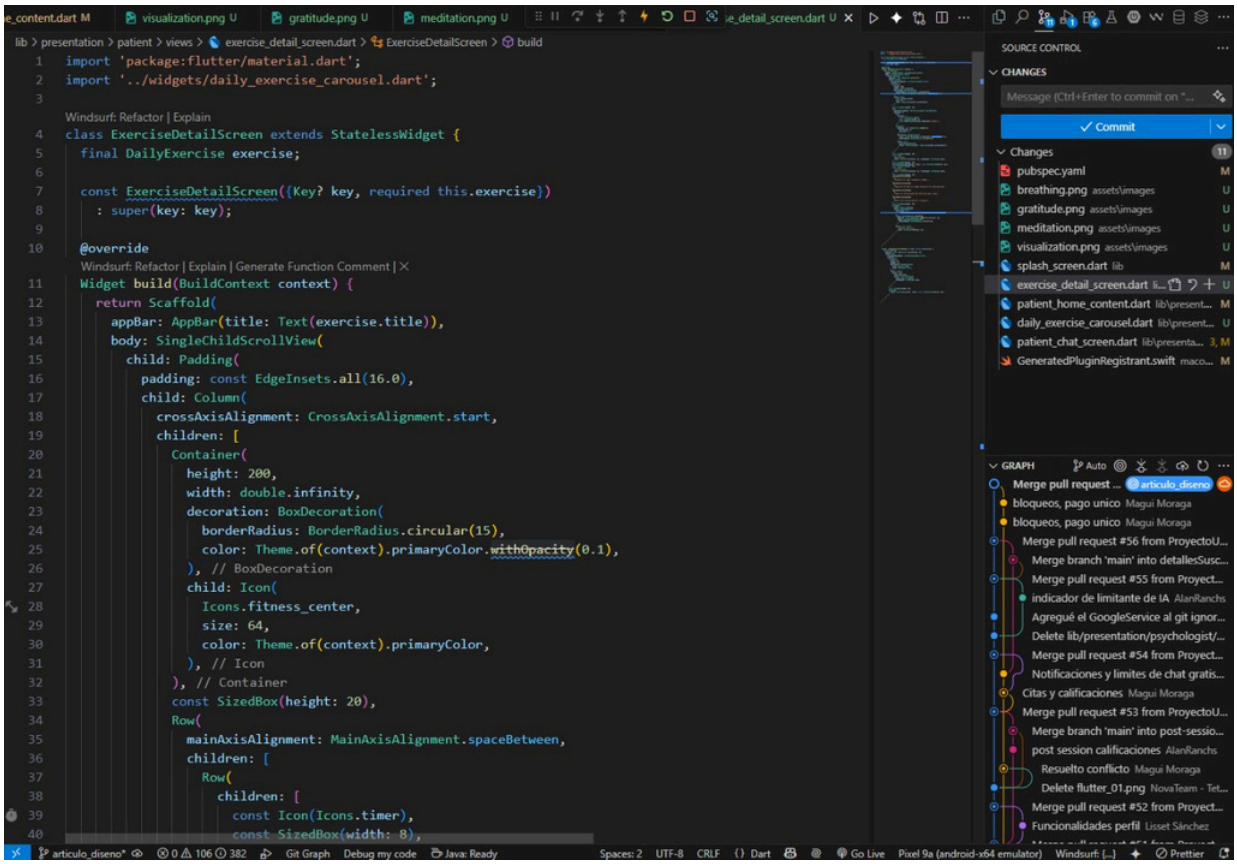
- Configuración de end point para obtener los artículos publicados.

```
// Endpoint para obtener todos los artículos públicos y publicados
articleRouter.get("/published", async (req, res) => {
  try {
    const { limit = 10, page = 1 } = req.query;
    const limitNum = Math.min(parseInt(limit), 50); // Máximo 50 artículos por página
    const pageNum = parseInt(page);
    const offset = (pageNum - 1) * limitNum;

    let query = db.collection('articles')
      .where('isPublished', '==', true)
      .where('status', '==', 'published')
      .orderBy('publishedAt', 'desc');

    const totalSnapshot = await query.get();
    const totalArticles = totalSnapshot.size;

    const articlesSnapshot = await query
      .offset(offset)
      .limit(limitNum)
      .get();

    const articles = articlesSnapshot.docs.map(doc => {
      const data = doc.data();
      return {
        id: doc.id,
        ...data,
        createdAt: data.createdAt?.toDate(),
        updatedAt: data.updatedAt?.toDate(),
        publishedAt: data.publishedAt?.toDate()
      };
    });

    res.json({
      articles: articles,
      totalArticles: totalArticles,
      page: pageNum,
      limit: limitNum
    });

  } catch (error) {
    console.error("Error al obtener artículos publicados:", error);
    res.status(500).json({
      error: "Error al obtener artículos publicados",
      details: process.env.NODE_ENV === 'development' ? error.message : undefined
    });
  }
});
```

- Implementación de diseño para ejercicios diarios.

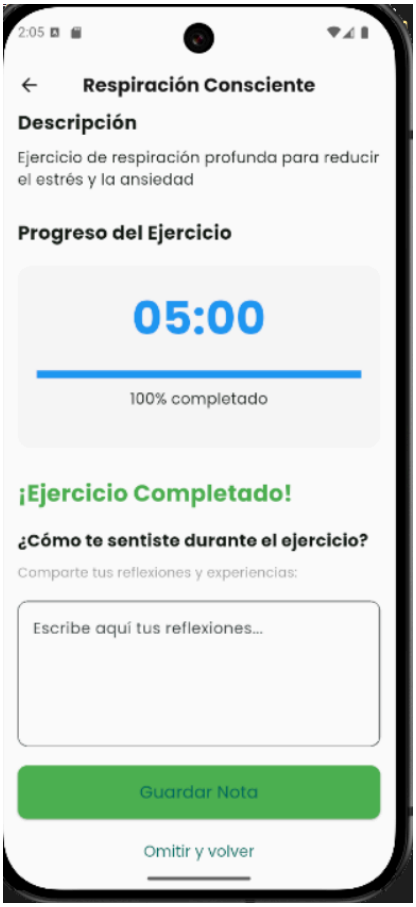

1:54
Buen día, Alan 👋
Frase del día
"El autocuidado es la forma de recuperar tu poder."
— Lalah Delia
Ejercicios diarios
Respiración Conscien...
Fácil
Ejercicio de respiración profunda para reducir el estrés y la ansiedad
5 min
Comenzar
¿Cómo te sientes?
¿Cómo te sientes hoy?
Terrible
Mal
Regular
Bien
Genial
Tips de Psicología Semanales
Practica la atención plena
Desconexión digital
Dic
Home
Chats
Psicólogos
Citas
Perfil


1:54
Ejercicio de Gratitud
3 minutos
Fácil
Descripción
Practica el reconocimiento de aspectos positivos en tu vida
Instrucciones
1 Encuentra un lugar tranquilo y cómodo.
2 Asegúrate de tener el tiempo necesario sin interrupciones.
3 Sigue las instrucciones del ejercicio paso a paso.
4 Mantén una actitud abierta y receptiva.
Comenzar Ejercicio

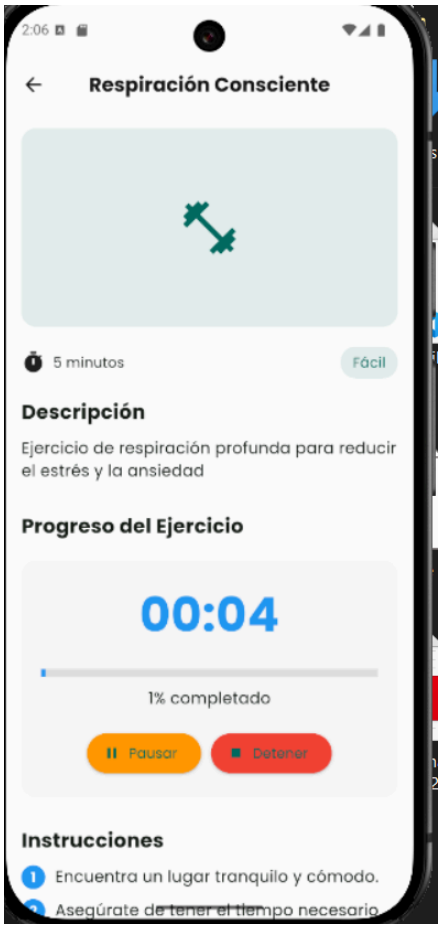


```dart
import 'package:flutter/material.dart';
import '../widgets/daily_exercise_carousel.dart';

Windsurf: Refactor | Explain
class ExerciseDetailScreen extends StatelessWidget {
  final DailyExercise exercise;

  const ExerciseDetailScreen({Key? key, required this.exercise})
    : super(key: key);

  @override
  Windsurf: Refactor | Explain | Generate Function Comment |
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text(exercise.title)),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Container(
                height: 200,
                width: double.infinity,
                decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(15),
                  color: Theme.of(context).primaryColor.withOpacity(0.1),
                ), // BoxDecoration
                child: Icon(
                  Icons.fitness_center,
                  size: 64,
                  color: Theme.of(context).primaryColor,
                ), // Icon
              ), // Container
              const SizedBox(height: 20),
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  Row(
                    children: [
                      const Icon(Icons.timer),
                      const SizedBox(width: 8),
```

- Implementación de funcionalidad de botón de comenzar ejercicio con temporizador y notas al finalizar.



- Opciones de detener y pausar ejercicio.



- Inicio de documento para plan de pruebas.

Plan de pruebas

Objetivo

Asegurar la calidad funcional, de rendimiento, seguridad y usabilidad de la aplicación **Aurora**, verificando que cumpla con los requisitos del negocio y técnicos establecidos, minimizando defectos en producción.

Alcance

- **Incluye:**
  - Módulos principales de Aurora (autenticación, gestión de pacientes, chat IA/psicólogo, reportes, notificaciones).
  - API backend, integración con Firebase, lógica de IA.
  - Interfaces de usuario (web y móvil).

Estrategia de pruebas

Se aplicará una combinación de pruebas manuales y automáticas:

1. **Pruebas funcionales (caja negra):**
   - Verificar flujos de usuario (inicio de sesión, registro, gestión de pacientes, generación de reportes).
   - Validación de reglas de negocio.
2. **Pruebas no funcionales:**
   - **Rendimiento:** carga concurrente en sesiones de chat y consultas a la base de datos.
   - **Seguridad:** validación de permisos, inyecciones, manejo seguro de credenciales.
   - **Usabilidad:** pruebas exploratorias con usuarios reales.
3. **Pruebas automatizadas:**
   - Unitarias (lógica de negocio en backend).
   - Integración (APIs, base de datos, servicios externos).
   - End-to-end (simulación de flujo completo).

Tipos de pruebas a ejecutar

Se aplicaran varios tipos de pruebas:

Tipo de prueba	Objetivo
Unitarias	Validar métodos individuales en backend y lógica de IA
Integración	Verificar que los módulos (API, DB, chat IA) funcionen en conjunto
Funcionales / Caja negra	Validar casos de uso del usuario final
Regresión	Garantizar que nuevas versiones no rompan funcionalidades existentes
Carga y estrés	Evaluar estabilidad con usuarios concurrentes
Seguridad	Comprobar autenticación, autorización y protección de datos sensibles
Usabilidad	Identificar problemas de experiencia de usuario