,

21-11-2025

Universidad Autónoma del Estado de México

Facultad de Ingeniería

Proyecto Tecnologías Computacionales I

Budgment Web Version

Profesor: Jose Antonio Hernández Servin

Equipo:

- Dávila Villavicencio Francisco Javier
- Guía Cruz Fabián Neftaly
- Rodríguez Nava José Bryan

Hoja de Evaluación

Puntos especificos considerados para evaluar el proyecto y acreditar la materia.

Tecnologías computacionales I, 2025b
Dr. J. Servín

1. (2 points) Portada

Nombres de la institución, organismo académico.	1
Título, Unidad de aprendizaje, Nombre del profesor	1
Nombres de los integrantes y fecha de entrega del reporte.	1
Criterios de evaluación	1
	2.00

2. (3 points) Objetivo

Es claro, cuantificable y bien estructurado	1
(comenzar con verbo en infinitivo, seguido del ¿cómo?,	1
y finalizando con el ¿para qué?).	1
	3.00

3. (15 points) Introducción

Claridad de los términos que tienen que ver con la experimentación y resultados,	1
se aborda la problemática planteada desde una postura o enfoque,	1
se hace uso de citas textuales con un solo estilo de referencias.	1
Máximo una sola cuartilla en extensión.	1
No hay faltas de ortografía y se usan adecuadamente las referencias.	1
	15.00

4. (35 points) Modelo

El o los modelos matemáticos están completos y sin errores.	1
La rutina de Polymath se puede ejecutar sin errores.	1
Explica los cambios realizados en el modelo para alcanzar el objetivo planteado.	1
	35.00

5. (25 points) Discusión

Presenta todos los resultados establecidos en el protocolo de la práctica de manera ordenada y con una breve descripción.	1
Interpreta y discute, a partir de principios disciplinares, datos importantes,	1
identificando tendencias, relaciones y diferencias útiles para el logro del objetivo.	1
	20.00

6. (2 points) Conclusión

Explican de manera precisa si, a través de los resultados,	1
se logró el objetivo planteado.	1
En caso de existir limitaciones en el estudio,	1
se propone soluciones y aportaciones para mejorar la práctica.	1
	2.00

7. (5 points) Referencias

Todas las fuentes que se utilizan son relevantes en la disciplina,	1
permiten la discusión de resultados a partir de un marco de referencia apropiado.	1
El documento se encuentra referenciado en estilo Vancouver.	1
Utilizar al menos 3 referencias: libro, artículo y sitio de internet.	1
	5.00

Budgment

Contenido

1. Objetivo

Diseñar y desarrollar una aplicación web (Budgment Web) que permita registrar, clasificar y visualizar transacciones financieras personales mediante una interfaz accesible y reportes interactivos, empleando Next.js en el frontend, Ktor (Kotlin) en el backend y SQLite para persistencia, con soporte de múltiples cuentas y monedas; de modo que el usuario pueda consultar su historial, detectar patrones de gasto y tomar decisiones informadas para optimizar su presupuesto.

2. Introducción

Budgment Web es una aplicación orientada a facilitar la administración de finanzas personales. Muchas personas no llevan un control estructurado de sus ingresos y gastos porque las herramientas existentes son complejas, requieren sincronizaciones con terceros o imponen modelos de suscripción. Estudios recientes muestran que los usuarios valoran mucho la facilidad de uso, reportes visuales claros, y funcionalidades de categorización automática para poder identificar y analizar sus hábitos financieros (Stefanov T et al., 2024).

Además, investigaciones sobre tecnologías financieras indican que aspectos como la seguridad, la transparencia y el soporte de diversas monedas o cuentas son factores determinantes para adoptar este tipo de aplicaciones (Panchal et al., 2024).

Budgment busca cubrir esa necesidad ofreciendo una interfaz sencilla y un flujo de trabajo claro: registrar transacciones, clasificarlas (automática o manualmente), y visualizar resúmenes y gráficas que permitan tomar decisiones informadas.

En este trabajo se plantea el problema de cómo transformar el registro financiero cotidiano en información útil y accionable. El enfoque adoptado es pragmático: una arquitectura de servicios desacoplados (frontend en Next.js y backend en Ktor) que garantiza portabilidad y seguridad; persistencia ligera mediante SQLite; y rutinas algorítmicas para clasificación de transacciones y agregación de datos para reportes.

2.1. Términos clave:

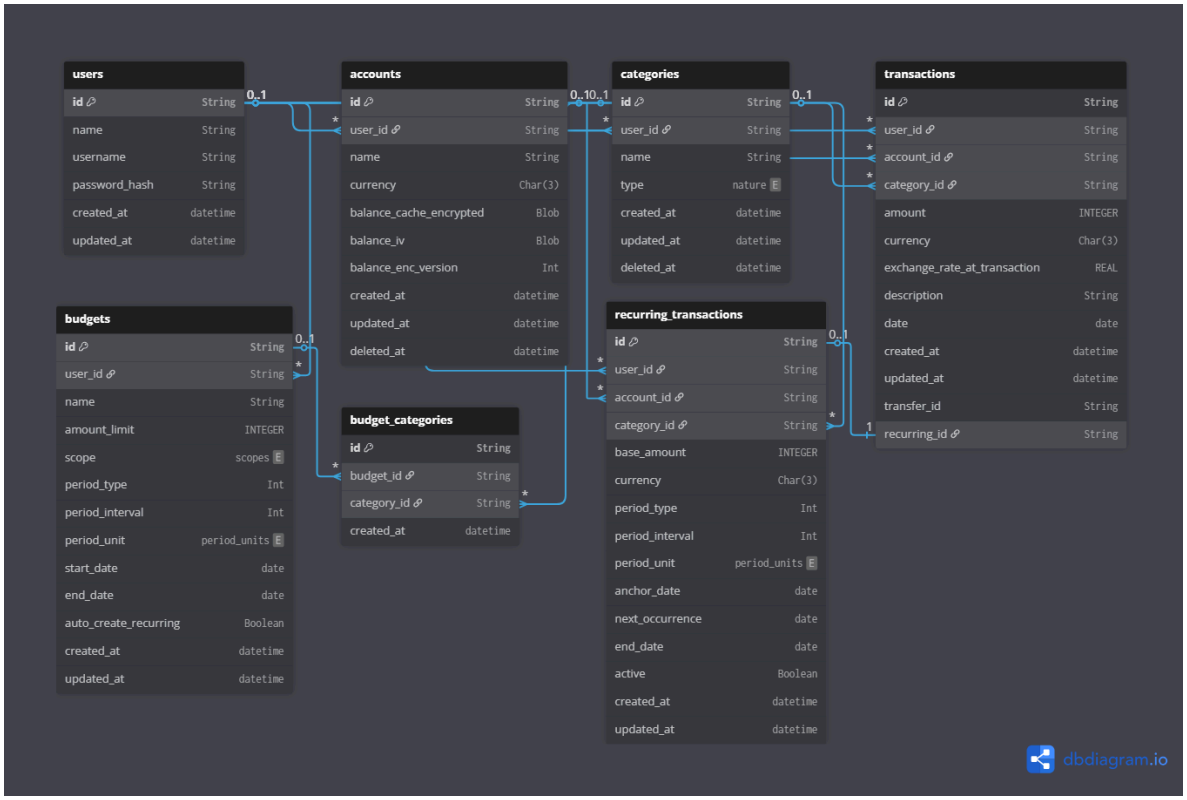
- **Transacción:** registro atómico con campos (monto, categoría, fecha, cuenta, moneda, descripción).
- **Cuenta:** contenedor de transacciones que puede tener moneda propia.

- **Categoría:** etiqueta aplicada a una transacción (p.ej. “Alimentos”, “Transporte”).
- Este documento describe los objetivos, la introducción conceptual, el modelo de datos y los algoritmos principales que sustentan Budgment Web. Las decisiones tecnológicas priorizan mantenibilidad, facilidad de despliegue (Docker Compose) y protección de la API, disponible únicamente a través del frontend.

3. Arquitectura y Modelos

La aplicación web **Budgment** está diseñada bajo una arquitectura de servicios separados, lo que permite mayor mantenibilidad, escalabilidad y seguridad en el flujo de datos.

3.1. Modelo de Datos



3.2. Arquitectura del proyecto

- **Elección de SQLite:** se priorizó una solución ligera y portable para usuarios individuales y despliegue sencillo; reduce complejidad operacional frente a un RDBMS cliente/servidor. (Cambio: del diseño inicial con DB remota a SQLite local por facilidad de despliegue.)
- **Arquitectura separada (Next.js / Ktor):** se optó por separación de responsabilidades para mejorar seguridad y mantenibilidad;

además facilita pruebas y escalado. (Cambio: modularizar para proteger API).

- **Soporte de múltiples monedas:** añadido para usuarios con cuentas en diferentes divisas; implica normalización y almacenamiento de tasas al insertar transacciones.
- **Interfaces y UX simplificada:** priorizar la simplicidad para bajar la fricción de registro, con el objetivo de aumentar la frecuencia de uso y facilitar la reducción de gastos.

4. Estructura Backend

El backend de Budgment es un servidor REST API desarrollado en Kotlin con el framework Ktor, encargado de manejar la lógica de la aplicación de finanzas personales. Su función principal es recibir solicitudes del frontend, autenticarlas de forma segura y responder con datos procesados en formato JSON. Utiliza SQLite como base de datos embebida mediante Exposed ORM, lo que facilita la manipulación y consulta de información como usuarios, cuentas, transacciones y categorías. Para la autenticación implementa JWT, permitiendo un sistema sin sesiones en el servidor, mientras que las contraseñas se protegen con hashing BCrypt. La arquitectura está modularizada en capas: Routes para definir endpoints HTTP, Services para ejecutar reglas y validaciones de negocio, Repositories para abstraer el acceso a datos con operaciones CRUD y permitir cambiar la base de datos si se requiere, y una capa de seguridad que configura y valida tokens. Incluye funcionalidades operativas como logging automático, identificadores únicos por request para debugging, manejo centralizado de errores, y serialización JSON automática con Content Negotiation y CORS habilitado para comunicarse con el cliente. Es una arquitectura ligera, segura y escalable, diseñada para gestionar múltiples usuarios, controlar saldos y registrar movimientos financieros de manera eficiente.

5. Estructura Frontend

El frontend de Budgment está desarrollado con Next.js 15, aprovechando el sistema de App Router, renderizado híbrido (SSR/CSR) y componentes de servidor para optimizar rendimiento. La interfaz se construye con React y componentes personalizados enfocados en usabilidad, simplicidad y consistencia visual. La comunicación con el backend se realiza mediante llamadas Fetch autenticadas con tokens JWT.

6. Mapeo de Endpoints

6.1. Notas generales

- Base URL por defecto: `http://localhost:8080`
- Autenticación: JWT (después de `/users/login`)
  - `accessToken` → usar como: `Authorization: Bearer <token>`
  - `refreshToken` → opcional para `/users/refresh`
  - `userId` → devuelto en el body del login
- Formato de montos:
  - Internamente centavos (minor units).
  - `12345` → `123.45`
- Fechas: `YYYY-MM-DD`
- `Content-Type: application/json`

6.2. Usuarios

6.2.1. POST /users/sign_in

Registrar usuario.

Body

```
{ "name": "Nombre Apellido", "username": "usuario", "password": "secreto" }
```

201 Response

```
{ "id": "uuid", "name": "...", "username": "...", "createdAt": "...",
"updatedAt": "..." }
```

6.2.2. POST /users/login

Login. Devuelve accessToken, refreshToken y userId.

Body

```
{ "username": "usuario", "password": "secreto" }
```

200 Response

```
{ "accessToken": "<jwt>", "refreshToken": "<jwt>", "userId": "3b1acdb5-..." }
```

6.2.3. GET /users/refresh (auth)

Regenera tokens usando refresh token en sesión.

200 Response

```
{ "accessToken": "<jwt>", "refreshToken": "<jwt>", "userId": "..." }
```

6.3. Cuentas (Accounts)

6.3.1. GET /users/{userId}/accounts (auth)

200 Response

```
[
  { "id": "...", "userId": "...", "name": "Cuenta A", "currency": "USD",
    "createdAt": "...", "updatedAt": null, "deletedAt": null }
]
```

6.3.2. POST /users/{userId}/accounts (auth)

Body

```
{ "name": "Cuenta A", "currency": "USD", "initialBalance": "150.50" }
```

201 Response

```
{ "id": "...", "userId": "...", "name": "Cuenta A", "currency": "USD" }
```

6.3.3. GET /users/{userId}/accounts/{accountId}/balance (auth)

200 Response

```
{
  "id": "<accountid>", "userId": "...", "name": "...",
  "currency": "USD", "balance": "150.50", "balanceMinorUnits": 15050
}
```

6.3.4. GET /users/{userId}/accounts/total_balance (auth)

```
{ "userId": "...", "total": "500.25", "totalMinorUnits": 50025 }
```

6.3.5. GET /users/{userId}/accounts/last_month/income (auth)

```
{ "userId": "...", "total": "1250.00", "totalMinorUnits": 125000 }
```

6.3.6. GET /users/{userId}/accounts/last_month/expense (auth)

```
{ "userId": "...", "total": "420.65", "totalMinorUnits": 42065 }
```

6.4. Transacciones

6.4.1. GET /users/{userId}/transactions (auth)

```
[
  { "id": "...", "userId": "...", "accountId": "...",
    "categoryId": null, "amount": -12345, "currency": "USD",
    "description": "Compra", "date": "2025-11-10" }
]
```

6.4.2. POST /users/{userId}/transactions (auth)

```
{
  "accountId": "<account-id>",
  "amount": -5000,
  "currency": "USD",
  "categoryId": "<category-id>",
  "description": "Pago supermercado",
  "date": "2025-11-28",
  "transferToAccountId": null
}
```

6.5. Categorías

6.5.1. POST /users/{userId}/categories (auth)

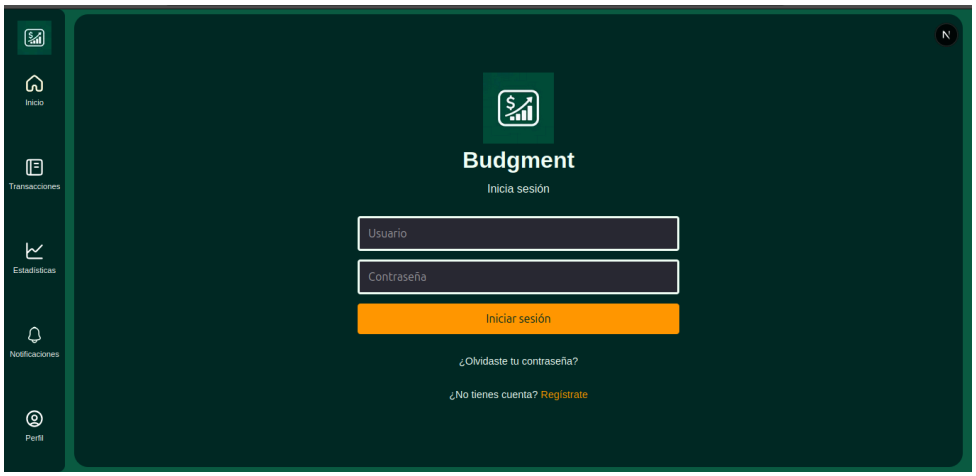
```
{ "name": "Comida", "type": "Gasto" }
```

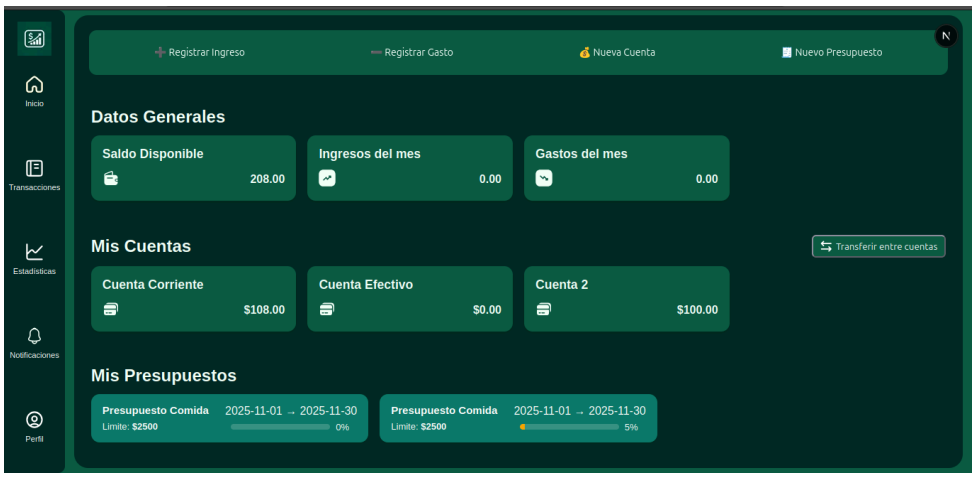
6.6. Presupuestos (Budgets)

6.6.1. POST /users/{userId}/budgets (auth)

```
{
  "name": "Presupuesto Comida",
  "amountLimit": 250000,
  "scope": "Category",
  "periodType": 2,
  "startDate": "2025-11-01",
  "endDate": "2025-11-30"
}
```

7. Pantallas





8. Como ejecutar

Budgment es una aplicación compuesta por dos partes:

- **Frontend:** Next.js + React
- **Backend:** Ktor + Kotlin

Este documento explica cómo ejecutar ambos proyectos localmente.

8.1. 🚀 Requisitos previos

8.1.1. Frontend

• Node.js 18+
• npm

8.1.2. Backend

• JDK 23 o superior
• Variable de entorno obligatoria: JWT_SECRET

8.2. 1. Ejecución del Frontend (Next.js)

8.2.1. 📂 Ubicación

budgment/frontend

8.2.2. ▶ Pasos

1. Entrar en la carpeta del frontend:

```
cd budgment/frontend
```

2. Instalar dependencias:

```
npm install
```

3. Ejecutar el servidor de desarrollo:

```
npm run dev
```

8.2.3. 🌐 URL de acceso

Frontend: «http://localhost:3000»

—

8.3. 2. Ejecución del Backend (Ktor)

8.3.1. 📂 Ubicación

budgment/backend

8.4. 🔐 Configurar JWT_SECRET

El backend requiere una variable de entorno llamada JWT_SECRET. Puedes generar un valor seguro así:

8.4.1. Linux / macOS

```
openssl rand -hex 32
```

O sin openssl:

```
uuidgen
```

8.4.2. Windows (PowerShell)

8.4.3. Windows (CMD)

```
powershell -command «guid
  ::NewGuid().ToString()«
  —
```

8.5. 🧩 Establecer la variable de entorno

8.5.1. Linux / macOS — Temporal

```
export JWT_SECRET=«tu_secreto_aqui»
```

8.5.2. Linux / macOS — Permanente

```
echo “export JWT_SECRET=«tu_secreto_aqui»” >>  /.bashrc source
   /.bashrc
```

8.5.3. Windows PowerShell — Permanente

```
setx JWT_SECRET «tu_secreto_aqui»
```

8.5.4. Windows CMD — Permanente

```
setx JWT_SECRET «tu_secreto_aqui»
```

8.5.5. Windows — Temporal (solo sesión actual)

PowerShell:

• env:JWT_SECRET=«tu_secreto_aqui»

CMD:

```
  set JWT_SECRET=tu_secreto_aqui
  —
```

8.6. ▶ Ejecutar el backend

1. Entrar a la carpeta:

```
cd budgment/backend
```

2. Compilar:

```
./gradlew build
  En Windows:
  gradlew build
```

3. Ejecutar:

```
./gradlew run
  En Windows:
  gradlew run
```

8.6.1. 🌐 URL de la API

Backend: «http://localhost:8080»

—

8.7. 🎉 Final

Cuando ambas partes estén ejecutándose:

Componente	URL
Frontend	«http://localhost:3000»
Backend API	«http://localhost:8080»

El frontend ya podrá comunicarse correctamente con la API.

9. Dockerización

Se tienen tres contenedores que cumplen roles distintos pero complementarios:

[Navegador] → (HTTP) → [nginx-proxy] → [frontend-test] → (API calls) → [backend-test]

9.1. nginx-proxy

Escucha en el puerto 80 público.

Es el reverse proxy que recibe todas las peticiones.

Redirige el tráfico al frontend según el dominio o configuración Docker.

9.2. frontend (Next.js)

Corre en puerto interno 3000.

No está expuesto al exterior.

nginx-proxy envía aquí las peticiones del usuario.

Para llamar al backend, usa la red Docker, ej:

http://backend:8080

9.3. backend (Ktor)

Corre en puerto interno 8080.

Solo accesible dentro de la red Docker.

Responde las solicitudes API del frontend.

10. Publicación

El codigo fuente se encuentra en GitHub en la siguiente URL https://github.com/NeftaliGC/Budgment-Web

Ademas tambien puede ver la aplicación en producción en: http://budgment.nintech.engineer

11. Referencias

Panchal, Mohadkar, & Anup D. (2024). Transforming Money Management: Analyzing The Impact Of Technology On Personal Finance. https://ssrn.com/abstract=5031477

Stefanov T, Stefanova M, Varbanova S, & Temelkov S. (2024). Personal finance management application. TEM J (Vol. 13). https://www.temjournal.com/content/133/TEMJournalAugust2024_2066_2075.pdf