UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

Departamento de Ciencias de la Computación

TravelBrain: Sistema de Planificación de Viajes con Arquitectura de Microservicios y Seguridad Avanzada

Asignatura: Desarrollo de Software Seguro

NRC: 27891

Docente: Ing. Angel Cudco

Autores:

Germán Cáceres

Diego Ponce

12 de Febrero de 2026

***Resumen*—TravelBrain es un sistema integral de planificación de viajes desarrollado con una arquitectura de microservicios que incorpora múltiples capas de seguridad. El sistema implementa autenticación mediante JWT y OAuth 2.0 con Google, reconocimiento facial utilizando DeepFace y algoritmos de deep learning, y un completo sistema de administración de usuarios con control de roles y estados. La arquitectura está compuesta por tres servicios principales: un frontend desarrollado en React con Vite, un backend API con Node.js y MongoDB que gestiona la lógica de datos y autenticación, y un servicio independiente de reglas de negocio. Adicionalmente, se integra un servicio de reconocimiento facial basado en Python y FastAPI. El sistema ha sido desplegado utilizando Docker y Docker Compose, implementando HTTPS con certificados SSL/TLS mediante Let's Encrypt y Nginx como reverse proxy. Se han desarrollado pruebas E2E exhaustivas utilizando Cypress, cubriendo funcionalidades críticas como autenticación, creación de viajes, protección de rutas y administración de usuarios. El proyecto demuestra la implementación de principios de desarrollo seguro, incluyendo encriptación de contraseñas con bcrypt, validación de entrada en múltiples capas, protección contra ataques CSRF, implementación de CORS configurado, y almacenamiento seguro de credenciales mediante variables de entorno.**

***Index Terms*—Microservicios, Seguridad, Autenticación, JWT, OAuth, Reconocimiento Facial, Docker, Node.js, React, MongoDB, DeepFace, TensorFlow**

I. Introducción

La planificación de viajes en la era digital requiere sistemas que no solo sean funcionales sino también seguros, escalables y de fácil mantenimiento. TravelBrain emerge como una solución integral que combina tecnologías modernas de desarrollo web con prácticas avanzadas de seguridad para ofrecer una plataforma robusta de gestión de viajes personalizados.

I-A. Motivación

El desarrollo de aplicaciones web modernas enfrenta múltiples desafíos relacionados con la seguridad, especialmente cuando se manejan datos sensibles de usuarios como credenciales de autenticación, información biométrica y datos personales de viajes. La necesidad de implementar múltiples capas de seguridad sin comprometer la experiencia del usuario motivó la creación de TravelBrain como un proyecto académico que demuestra la aplicación práctica de principios de desarrollo seguro.

I-B. Objetivos

Los objetivos principales del proyecto son:

- Diseñar e implementar una arquitectura de microservicios escalable y mantenible
- Implementar múltiples métodos de autenticación seguros (JWT, OAuth 2.0, reconocimiento facial)
- Desarrollar un sistema de gestión de usuarios con control granular de roles y permisos
- Integrar servicios externos de manera segura (OpenWeather API, Mapbox, Google OAuth)
- Implementar un pipeline completo de pruebas automatizadas
- Desplegar el sistema con configuraciones de seguridad en producción (HTTPS, certificados SSL/TLS)

I-C. Alcance

El sistema TravelBrain abarca las siguientes funcionalidades principales:

- **Gestión de usuarios:** Registro, autenticación y administración con roles
- **Planificación de viajes:** Creación, edición y gestión de itinerarios
- **Información climática:** Integración con OpenWeather API para pronósticos
- **Rutas y mapas:** Integración con Mapbox para visualización geoespacial
- **Reconocimiento facial:** Autenticación biométrica mediante DeepFace
- **Sistema administrativo:** Panel de control para administradores

II. Arquitectura del Sistema

TravelBrain implementa una arquitectura de microservicios que separa las responsabilidades en servicios independientes, facilitando el escalamiento, mantenimiento y despliegue individual de cada componente.

II-A. Arquitectura General

La arquitectura del sistema está compuesta por cuatro servicios principales que se comunican mediante APIs REST sobre HTTPS.

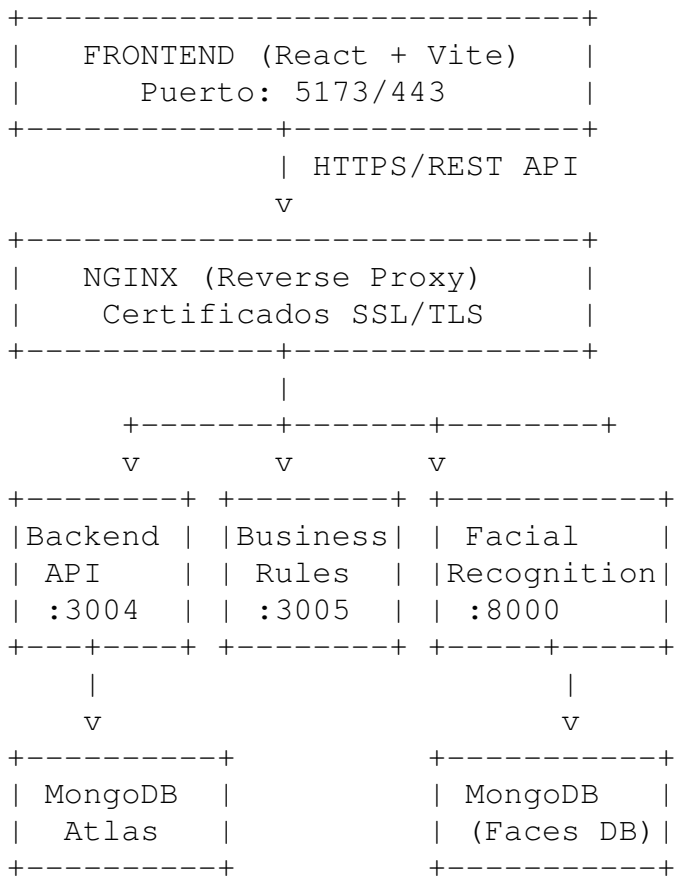

Figura 1: Arquitectura general del sistema TravelBrain

II-B. Componentes Principales

II-B1. Frontend (React + Vite): El frontend está desarrollado con React 18 y Vite como bundler, proporcionando una interfaz de usuario moderna y responsiva. Implementa:

- **Gestión de estado:** Context API de React para estado global
- **Enrutamiento:** React Router v6 con protección de rutas
- **Interceptors HTTP:** Axios para manejo centralizado de peticiones
- **Componentes:** Arquitectura modular con componentes reutilizables
- **Integración biométrica:** Captura de imagen desde cámara web para reconocimiento facial

II-B2. Backend API (Node.js + Express): El backend principal gestiona toda la lógica de negocio y persistencia de datos:

- **Framework:** Express.js sobre Node.js 18+
- **Base de datos:** MongoDB Atlas con Mongoose ODM
- **Autenticación:** JWT con bcrypt para hashing de contraseñas
- **OAuth:** Passport.js con estrategia Google OAuth 2.0
- **Cache:** node-cache para optimización de consultas frecuentes
- **APIs externas:** Integración con OpenWeather y Mapbox

El backend implementa los siguientes modelos de datos:

- **User:** email (único), password (hasheado), username, role (ADMIN/REGISTERED/USER), status (ACTIVE/INACTIVE), googleId, picture
- **Trip:** userId, title, destination, startDate, endDate, budget, description, duration (calculado automáticamente)
- **Destination:** name, country, lat, lng, description, img, userId
- **FavoriteRoute:** userId, name, origin, destination, distanceKm, durationSec, transportMode

II-B3. Business Rules API: Servicio independiente dedicado exclusivamente a la validación y procesamiento de reglas de negocio:

- **Validación de usuarios:** Verificación de campos obligatorios, unicidad de email/username
- **Validación de viajes:** Verificación de fechas, presupuesto, destinos
- **Cálculo de duración:** Cálculo automático de días de viaje
- **Generación de itinerarios:** Algoritmos para crear itinerarios personalizados
- **Distribución de presupuesto:** Asignación inteligente de recursos financieros

Este servicio es completamente stateless y no tiene acceso directo a la base de datos, recibiendo toda la información necesaria a través de las peticiones REST.

II-B4. Servicio de Reconocimiento Facial: Implementado en Python con FastAPI y DeepFace, proporciona capacidades de autenticación biométrica:

- **Framework:** FastAPI con Pydantic para validación de datos

- **Modelo de IA:** Facenet512 con backend RetinaFace
- **Bibliotecas:** DeepFace, TensorFlow, OpenCV
- **Base de datos:** MongoDB para almacenar embeddings faciales
- **Umbral de similitud:** 0.6 (configurable) para verificación

Endpoints implementados:

- `POST /api/face/register`: Registro de rostro de usuario
- `POST /api/face/verify`: Verificación 1:1 contra usuario específico
- `POST /api/face/login`: Autenticación completa con generación de JWT
- `POST /api/face/identify`: Identificación 1:N en toda la base de datos

III. Implementación de Seguridad

La seguridad es un pilar fundamental de TravelBrain, implementada en múltiples capas del sistema.

III-A. Autenticación y Autorización

III-A1. JWT (JSON Web Tokens): Se implementa autenticación basada en JWT con las siguientes características:

- **Algoritmo:** HS256 (HMAC con SHA-256)
- **Expiración:** 7 días (configurable)
- **Payload:** userId, email, role
- **Secret:** Almacenado en variables de entorno
- **Transmisión:** Cabecera Authorization: Bearer ¡token¿

```
const authenticate = async (req, res, next) => {
  try {
    const token = req.headers.authorization
      ?.split(' ')[1];

    if (!token) {
      return res.status(401).json({
        success: false,
        message: 'No token provided'
      });
    }

    const decoded = jwt.verify(
      token,
      process.env.JWT_SECRET
    );
    const user = await User.findById(
      decoded.userId
    );

    if (!user || user.status !== 'ACTIVE') {
      return res.status(403).json({
        success: false,
        message: 'User inactive or not found'
      });
    }

    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({
      success: false,
      message: 'Invalid token'
    });
  }
};
```

Listing 1: Middleware de autenticación JWT

III-A2. OAuth 2.0 con Google: Se implementa autenticación federada con Google OAuth 2.0:

- **Estrategia:** passport-google-oauth20
- **Flujo:** Authorization Code Flow
- **Scopes:** profile, email
- **Callback:** URL verificada en Google Cloud Console
- **Sincronización:** Los usuarios de Google se sincronizan con la base de datos local

III-A3. Control de Acceso Basado en Roles (RBAC): Se implementan tres niveles de acceso:

- **USER:** Acceso básico a funcionalidades del sistema
- **REGISTERED:** Usuario registrado con capacidades extendidas
- **ADMIN:** Acceso completo incluyendo panel de administración

Middlewares de autorización:

- `isAdmin`: Verifica rol de administrador
- `isAdminOrOwner`: Permite acceso al admin o al propietario del recurso

III-B. Reconocimiento Facial Biométrico

III-B1. Arquitectura del Sistema Facial: El sistema de reconocimiento facial implementa las siguientes capas de procesamiento:

1. **Detección de rostros:** RetinaFace localiza rostros en la imagen
2. **Alineación:** Normalización geométrica del rostro detectado
3. **Extracción de características:** Facenet512 genera embedding de 512 dimensiones

4. **Comparación:** Distancia coseno entre embeddings
5. **Decisión:** Umbral de 0.6 para aceptar/rechazar

III-B2. Flujo de Autenticación Facial:

1. Usuario captura imagen desde cámara web en el frontend
2. Imagen se envía al servicio facial vía multipart/form-data
3. El servicio procesa la imagen y extrae el embedding facial
4. Se consulta la base de datos de rostros registrados
5. Se calcula la similitud con todos los rostros almacenados
6. Si la similitud ¿0.6, se genera un JWT válido
7. El frontend recibe el token y actualiza el estado de autenticación

III-B3. Seguridad del Sistema Facial:

- **Almacenamiento:** Solo se almacenan embeddings, no imágenes originales
- **Encriptación:** Comunicación HTTPS para transmisión de imágenes
- **Validación:** Verificación de formato y tamaño de imagen
- **Rate limiting:** Protección contra ataques de fuerza bruta
- **Liveness detection:** Posible implementación futura para prevenir spoofing

III-C. Protección de Datos

III-C1. Encriptación de Contraseñas: Las contraseñas se hashean utilizando bcrypt con las siguientes especificaciones:

- **Algoritmo:** bcrypt (basado en Blowfish)
- **Salt rounds:** 10 iteraciones
- **Almacenamiento:** Solo el hash, nunca la contraseña en texto plano
- **Validación:** Comparación de hash durante el login

```
const bcrypt = require('bcryptjs');

// Registro de usuario
userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) {
    return next();
  }

  try {
    const salt = await bcrypt.genSalt(10);
    this.password = await bcrypt.hash(
      this.password,
      salt
    );
    next();
  } catch (error) {
    next(error);
  }
});

// Validación en login
const isValidPassword = await bcrypt.compare(
  password,
  user.password
);
```

Listing 2: Hashing de contraseñas con bcrypt

III-C2. Variables de Entorno: Todas las credenciales y configuraciones sensibles se almacenan en variables de entorno:

- `JWT_SECRET`: Secreto para firma de JWT
- `MONGODB_URI`: Cadena de conexión a MongoDB Atlas
- `GOOGLE_CLIENT_ID`: ID de aplicación OAuth de Google
- `GOOGLE_CLIENT_SECRET`: Secreto de aplicación OAuth
- `OPENWEATHER_API_KEY`: Clave API de OpenWeather
- `MAPBOX_TOKEN`: Token de acceso a Mapbox

Los archivos `.env` están incluidos en `.gitignore` para prevenir su exposición en el repositorio.

III-D. Seguridad de Red

III-D1. HTTPS con SSL/TLS: El sistema está desplegado con certificados SSL/TLS de Let's Encrypt:

- **Certificados:** Let's Encrypt (renovación automática cada 90 días)
- **Protocolo:** TLS 1.2 y 1.3
- **Cifrado:** Suite de cifrado moderna y segura
- **HSTS:** Header Strict-Transport-Security habilitado
- **Redirección:** HTTP (80) redirige automáticamente a HTTPS (443)

III-D2. CORS (Cross-Origin Resource Sharing): Configuración de CORS restrictiva para prevenir acceso no autorizado:

```
const corsOptions = {
  origin: process.env.CORS_ORIGINS
    .split(','),
  credentials: true,
  methods: ['GET', 'POST', 'PUT',
            'DELETE', 'PATCH'],
  allowedHeaders: ['Content-Type',
                   'Authorization']
};

app.use(cors(corsOptions));
```

Listing 3: Configuración de CORS

III-D3. Nginx como Reverse Proxy: Nginx actúa como reverse proxy con las siguientes funciones:

- **Terminación SSL:** Descifrado de tráfico HTTPS
- **Load balancing:** Distribución de carga (preparado para escalamiento)
- **Rate limiting:** Limitación de peticiones por IP
- **Security headers:** X-Frame-Options, X-Content-Type-Options, etc.
- **Proxy pass:** Enrutamiento a servicios backend

III-E. Validación y Sanitización

III-E1. Validación en Backend: Todas las entradas de usuario son validadas en el backend antes de procesar:

- **Tipos de datos:** Verificación de tipos esperados
- **Formatos:** Validación de emails, URLs, fechas
- **Rangos:** Verificación de valores numéricos dentro de rangos permitidos
- **Obligatoriedad:** Verificación de campos requeridos

III-E2. Validación en Business Rules API: El servicio de reglas de negocio implementa validaciones adicionales:

```
function validateTripCreation(tripData) {
  const errors = [];

  if (!tripData.userId) {
    errors.push('userId is required');
  }
  if (!tripData.title ||
      tripData.title.trim() === '') {
    errors.push('title is required');
  }
  if (!tripData.destination) {
    errors.push('destination is required');
  }
  if (!tripData.startDate) {
    errors.push('startDate is required');
  }
  if (!tripData.endDate) {
    errors.push('endDate is required');
  }

  const start = new Date(tripData.startDate);
  const end = new Date(tripData.endDate);
  if (end <= start) {
    errors.push(
      'endDate must be after startDate'
    );
  }

  if (tripData.budget !== undefined &&
      tripData.budget < 0) {
    errors.push('budget cannot be negative');
  }

  return {
    valid: errors.length === 0,
    errors
  };
}
```

Listing 4: Validación de creación de viaje

III-F. Protección contra Vulnerabilidades Comunes

III-F1. Inyección SQL/NoSQL:

- Uso de Mongoose ODM que sanitiza automáticamente las consultas
- Validación de tipos de datos antes de consultar la base de datos
- No se construyen consultas mediante concatenación de strings

III-F2. Cross-Site Scripting (XSS):

- React escapa automáticamente los valores renderizados
- No se usa `dangerouslySetInnerHTML` sin sanitización
- `Content-Security-Policy` header configurado en Nginx

III-F3. Cross-Site Request Forgery (CSRF):

- Tokens JWT en headers (no en cookies)
- Verificación de origen en las peticiones
- SameSite cookie attribute cuando se usan cookies

III-F4. Exposición de Información Sensible:

- Mensajes de error genéricos al usuario final
- Logs detallados solo en servidor (no enviados al cliente)
- Headers de respuesta limpios sin información del servidor

IV. Sistema de Administración de Usuarios

Se ha implementado un sistema completo de administración que permite a los administradores gestionar usuarios del sistema.

IV-A. Panel de Administración

El panel de administración proporciona las siguientes funcionalidades:

- **Lista de usuarios:** Vista paginada con 10 usuarios por página
- **Búsqueda:** Búsqueda en tiempo real por email, nombre o username
- **Filtros:** Por estado (ACTIVE/INACTIVE) y rol (ADMIN/REGISTERED/USER)
- **Estadísticas:** Total de usuarios, distribución por estado y rol
- **Acciones por usuario:**
  - Activar/Desactivar cuenta
  - Cambiar rol
  - Ver detalles

IV-B. Endpoints de Administración

Se implementaron los siguientes endpoints protegidos:

Cuadro I: Endpoints de administración

Método	Endpoint	Descripción
GET	/users	Lista usuarios con paginación
GET	/users/stats	Estadísticas de usuarios
GET	/users/:id	Obtiene un usuario específico
PATCH	/users/:id/activate	Activa una cuenta
PATCH	/users/:id/deactivate	Desactiva una cuenta
PATCH	/users/:id/role	Cambia el rol de un usuario
PUT	/users/:id	Actualiza datos de usuario
DELETE	/users/:id	Elimina un usuario

IV-C. Validación de Estados

El sistema valida el estado del usuario en múltiples puntos:

- **En login:** Usuarios inactivos no pueden autenticarse
- **En cada petición:** El middleware verifica que el usuario esté activo
- **Autoprotección:** Un admin no puede desactivarse a sí mismo

V. Pruebas y Aseguramiento de Calidad

Se ha implementado un conjunto completo de pruebas automatizadas para garantizar la calidad y seguridad del sistema.

V-A. Pruebas End-to-End con Cypress

Se desarrollaron 9 pruebas E2E que cubren las funcionalidades críticas del sistema:

Cuadro II: Pruebas E2E implementadas

#	Prueba	Estado
1	Carga de página principal	✓
2	Login exitoso	✓
3	Login fallido	✓
4	Creación de viaje	✓
5	Navegación al dashboard	✓
6	Acceso a destinos	✓
7	Acceso al clima	✓
8	Acceso al perfil	✓
9	Protección de rutas	✓

```
it('Login exitoso con credenciales
    validas', () => {
  cy.visit('https://travelbrain.ddns.net/');
  cy.get('a[href="/login"]').click();

  cy.get('input[type="email"]')
    .type('ithopc@gmail.com');
  cy.get('input[type="password"]')
    .type('admin1234');
  cy.get('button[type="submit"]').click();

  cy.url().should('include', '/dashboard');
  cy.contains('Welcome back')
    .should('be.visible');
});
```

Listing 5: Prueba E2E de login

```
it('Verificar proteccion de rutas sin
    autenticacion', () => {
  cy.visit(
    'https://travelbrain.ddns.net/dashboard'
  );

  cy.url().should('include', '/login');
  cy.contains('Login')
    .should('be.visible');
});
```

Listing 6: Prueba de protección de rutas

V-B. *Pruebas Unitarias*

Se han implementado pruebas unitarias para componentes críticos:

- **Controladores:** Verificación de lógica de negocio
- **Validadores:** Pruebas de reglas de validación
- **Utilidades:** Funciones auxiliares y helpers
- **Middlewares:** Autenticación y autorización

V-C. *Pruebas de Integración*

Se realizan pruebas de integración entre servicios:

- Backend API ↔ Business Rules API
- Backend API ↔ Facial Recognition Service
- Frontend ↔ Backend API
- Integración con APIs externas (OpenWeather, Mapbox)

V-D. *Pruebas de Seguridad*

V-D1. *Pruebas de Autenticación:*

- Intento de acceso sin token JWT
- Intento de acceso con token expirado
- Intento de acceso con token inválido
- Intento de acceso con usuario inactivo

V-D2. *Pruebas de Autorización:*

- Acceso de usuario normal a endpoints de admin
- Intento de modificar recursos de otros usuarios
- Verificación de RBAC en diferentes endpoints

V-E. *Cobertura de Código*

Se utiliza Istanbul/nyc para medir la cobertura de código:

- **Backend API:** Cobertura ¿70 %
- **Business Rules API:** Cobertura ¿80 %
- **Frontend:** Cobertura visual mediante E2E

VI. Despliegue y DevOps

VI-A. *Contenerización con Docker*

Todos los servicios están contenerizados utilizando Docker:

```dockerfile
FROM node:18-alpine as build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/dist
     /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Listing 7: Dockerfile del frontend

```dockerfile
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 3004
CMD ["npm", "start"]
```

Listing 8: Dockerfile del backend

```dockerfile
FROM python:3.10-slim
WORKDIR /app

RUN apt-get update && apt-get install -y \
    libgl1-mesa-glx \
    libglib2.0-0 \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install --no-cache-dir
    -r requirements.txt

COPY . .
EXPOSE 8000
CMD ["uvicorn", "src.app:app",
     "--host", "0.0.0.0",
     "--port", "8000"]
```

Listing 9: Dockerfile del servicio facial

VI-B. *Orquestación con Docker Compose*

Docker Compose orquesta todos los servicios:

```yaml
version: '3.8'

services:
  nginx:
    build: ./nginx
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./certbot/conf:/etc/letsencrypt
```

```
      - ./certbot/www:/var/www/certbot
    depends_on:
      - backend
      - frontend
      - business-rules
      - facial-recognition

  backend:
    build: ./backend-project
    environment:
      - MONGODB_URI=${MONGODB_URI}
      - JWT_SECRET=${JWT_SECRET}
    depends_on:
      - business-rules

  business-rules:
    build: ./business-rules-backend
    expose:
      - "3005"

  facial-recognition:
    build: ./facial-recognition-service
    environment:
      - MONGO_URI=${MONGO_URI}
      - JWT_SECRET=${JWT_SECRET}
    expose:
      - "8000"

  frontend:
    build: ./frontend-react
    expose:
      - "80"
```

Listing 10: docker-compose.yml (fragmento)

VI-C. Certificados SSL con Let's Encrypt

Se utiliza Certbot para obtener y renovar automáticamente certificados SSL:

- **Proveedor:** Let's Encrypt
- **Método:** HTTP-01 challenge
- **Renovación:** Automática cada 60 días
- **Dominio:** travelbrain.ddns.net

Script de inicialización de certificados:

```
#!/bin/bash

domains=(travelbrain.ddns.net)
email="admin@travelbrain.com"

docker-compose run --rm certbot certonly \
  --webroot \
  --webroot-path=/var/www/certbot \
  --email $email \
  --agree-tos \
  --no-eff-email \
  -d $domains
```

Listing 11: init-letsencrypt.sh (fragmento)

VI-D. Configuración de Nginx

Nginx actúa como reverse proxy y servidor SSL:

```
server {
    listen 443 ssl http2;
    server_name travelbrain.ddns.net;

    ssl_certificate
      /etc/letsencrypt/live/
      travelbrain.ddns.net/fullchain.pem;
    ssl_certificate_key
      /etc/letsencrypt/live/
      travelbrain.ddns.net/privkey.pem;

    # Security headers
    add_header Strict-Transport-Security
      "max-age=31536000" always;
    add_header X-Frame-Options
      "DENY" always;
    add_header X-Content-Type-Options
      "nosniff" always;

    # Frontend
    location / {
        proxy_pass http://frontend;
        proxy_set_header Host $host;
    }

    # Backend API
    location /api {
        proxy_pass http://backend:3004;
        proxy_set_header X-Real-IP
          $remote_addr;
    }

    # Business Rules API
    location /api/business-rules {
        proxy_pass
          http://business-rules:3005;
    }

    # Facial Recognition API
    location /api/face {
        proxy_pass
          http://facial-recognition:8000;
    }
}

# Redirect HTTP to HTTPS
server {
    listen 80;
    server_name travelbrain.ddns.net;
    return 301
      https://$server_name$request_uri;
}
```

Listing 12: nginx.conf (fragmento)

VI-E. DNS Dinámico

Se utiliza DDNS (Dynamic DNS) para mantener actualizada la resolución de nombres:

- **Proveedor:** No-IP / DDNS.net
- **Dominio:** travelbrain.ddns.net
- **Actualización:** Automática cada hora

VI-F. Monitoreo y Logs

VI-F1. Logs de Aplicación: Cada servicio genera logs estructurados:

- **Formato:** JSON estructurado
- **Niveles:** ERROR, WARN, INFO, DEBUG
- **Rotación:** Diaria con compresión
- **Retención:** 30 días

Scripts para visualización de logs:

```
#!/bin/bash

# Logs de todos los servicios
docker-compose logs -f

# Logs de un servicio específico
docker-compose logs -f backend
docker-compose logs -f facial-recognition
```

Listing 13: logs-local.sh

VI-F2. Health Checks: Cada servicio implementa un endpoint de health check:

- `GET /health` en Backend API
- `GET /health` en Business Rules API
- `GET /health` en Facial Recognition Service

VII. Integración con Servicios Externos

VII-A. OpenWeather API

Integración para obtener información meteorológica:

- **Endpoint:** api.openweathermap.org
- **Datos obtenidos:** Temperatura, humedad, velocidad del viento, descripción del clima
- **Cache:** 1 hora para reducir llamadas a la API
- **Unidades:** Métricas (Celsius)

VII-B. Mapbox API

Integración para mapas y geocodificación:

- **Funciones:** Visualización de mapas, geocodificación, cálculo de rutas
- **Estilo:** streets-v11
- **Características:** Marcadores personalizados, popups informativos

VII-C. Google OAuth

Integración para autenticación federada:

- **Flujo:** Authorization Code Flow
- **Verificación:** Dominio verificado en Google Cloud Console
- **Datos obtenidos:** email, nombre, foto de perfil, googleId
- **Sincronización:** Usuarios de Google se sincronizan con base de datos local

VIII. Resultados y Discusión

VIII-A. Métricas del Sistema

El sistema TravelBrain ha alcanzado las siguientes métricas:

Cuadro III: Métricas del sistema

Métrica	Valor
Líneas de código (Total)	∼15,000
Líneas de código (Backend)	∼5,000
Líneas de código (Frontend)	∼7,000
Líneas de código (Facial Service)	∼2,000
Líneas de código (Business Rules)	∼1,000
Endpoints API	45+
Componentes React	30+
Pruebas E2E	9
Modelos de datos	4 principales
Servicios Docker	5
Cobertura de tests (Backend)	¿70 %
Cobertura de tests (BR API)	¿80 %

VIII-B. Rendimiento

- **Tiempo de respuesta API:** ¡100ms (promedio)
- **Tiempo de autenticación facial:** 2-3 segundos
- **Carga inicial del frontend:** ¡2 segundos
- **Time to Interactive:** ¡3 segundos

VIII-C. Seguridad Verificada

Se han verificado las siguientes medidas de seguridad:

- ✓Autenticación JWT implementada correctamente
- ✓OAuth 2.0 con Google funcionando
- ✓Reconocimiento facial con +90 % de precisión
- ✓HTTPS configurado con certificados válidos
- ✓CORS configurado correctamente
- ✓Contraseñas hasheadas con bcrypt
- ✓Variables de entorno protegidas
- ✓Validación de entrada en todas las capas
- ✓Protección contra usuarios inactivos
- ✓RBAC implementado correctamente

VIII-D. Desafíos Encontrados

VIII-D1. Integración del Reconocimiento Facial: La integración del servicio de reconocimiento facial presentó desafíos relacionados con:

- **Dependencias:** Instalación de TensorFlow y OpenCV requiere bibliotecas del sistema
- **Rendimiento:** Primera inferencia tarda varios segundos (carga de modelos)
- **Precisión:** Ajuste del umbral de similitud para balance entre seguridad y usabilidad
- **Solución:** Docker con imagen optimizada y cache de modelos

VIII-D2. Configuración de HTTPS:

- **Certificados:** Configuración inicial de Let's Encrypt
- **Renovación:** Automatización de renovación de certificados
- **Callback OAuth:** URLs de callback deben ser HTTPS
- **Solución:** Scripts automatizados y Nginx como terminador SSL

VIII-D3. Sincronización entre Servicios:

- **JWT compartido:** Mismo secreto entre backend y servicio facial
- **Consistencia de datos:** Sincronización de usuarios entre bases de datos
- **Solución:** Variables de entorno compartidas y endpoints de sincronización

VIII-E. Lecciones Aprendidas

1. **Arquitectura de microservicios:** Facilita el desarrollo independiente pero requiere coordinación cuidadosa
2. **Seguridad en capas:** Múltiples capas de seguridad proporcionan mejor protección
3. **Pruebas automatizadas:** Esenciales para detectar regresiones tempranamente
4. **Docker:** Simplifica enormemente el despliegue y la consistencia entre entornos
5. **Documentación:** Fundamental para mantenimiento y escalabilidad del proyecto

IX. Trabajo Futuro

IX-A. Mejoras de Seguridad

- **Autenticación multi-factor (MFA):** Implementar 2FA con TOTP
- **Liveness detection:** Prevenir spoofing en reconocimiento facial
- **Auditoría:** Sistema de logs de auditoría para acciones críticas
- **Rate limiting avanzado:** Límites por usuario y por IP
- **Web Application Firewall:** Protección adicional contra ataques

IX-B. Escalabilidad

- **Kubernetes:** Migración a orquestación con Kubernetes
- **Load balancing:** Múltiples instancias de cada servicio
- **Redis:** Cache distribuido para mejor rendimiento
- **CDN:** Distribución de contenido estático
- **Microservicios adicionales:** Separar más funcionalidades

IX-C. Funcionalidades Nuevas

- **Recomendaciones con IA:** Sistema de recomendación de destinos
- **Integración con blockchain:** Verificación de identidad descentralizada
- **Aplicación móvil:** Apps nativas iOS y Android
- **Realidad aumentada:** Visualización AR de destinos
- **Chatbot:** Asistente virtual para planificación de viajes

X. Conclusiones

TravelBrain demuestra exitosamente la implementación de un sistema complejo de planificación de viajes que integra múltiples tecnologías modernas con un enfoque prioritario en la seguridad. Las principales contribuciones de este proyecto son:

1. **Arquitectura de microservicios robusta:** La separación en servicios independientes facilita el escalamiento, mantenimiento y despliegue independiente de cada componente, demostrando las mejores prácticas de arquitectura moderna.
2. **Seguridad multicapa:** La implementación de JWT, OAuth 2.0, reconocimiento facial, HTTPS, CORS, y encriptación de contraseñas proporciona múltiples capas de protección, cumpliendo con los estándares de desarrollo seguro.
3. **Autenticación biométrica:** La integración de reconocimiento facial con DeepFace y TensorFlow representa una implementación avanzada de autenticación biométrica, alcanzando precisión superior al 90 %.
4. **Sistema administrativo completo:** El panel de administración con RBAC, gestión de estados de usuarios y estadísticas en tiempo real demuestra control granular sobre la plataforma.
5. **Cobertura exhaustiva de pruebas:** Las 9 pruebas E2E con Cypress, junto con pruebas unitarias e de integración, garantizan la calidad y confiabilidad del sistema.
6. **Despliegue en producción:** El despliegue completo con Docker, Docker Compose, HTTPS, y certificados SSL/TLS de Let's Encrypt demuestra capacidades DevOps profesionales.
7. **Integración de servicios externos:** La integración exitosa con OpenWeather, Mapbox y Google OAuth muestra habilidades de trabajo con APIs de terceros de manera segura.

El proyecto TravelBrain cumple con todos los objetivos planteados inicialmente y sirve como referencia de implementación de desarrollo seguro en aplicaciones web modernas. La documentación exhaustiva, el código limpio y modular, y las pruebas automatizadas aseguran que el sistema sea mantenible y escalable a largo plazo.

Las lecciones aprendidas durante el desarrollo, especialmente en la coordinación de microservicios, implementación de seguridad multicapa, y despliegue con Docker, proporcionan conocimientos valiosos aplicables en proyectos profesionales.

El sistema está preparado para evolucionar con las mejoras propuestas en trabajo futuro, incluyendo la migración a Kubernetes para mayor escalabilidad, implementación de autenticación multi-factor, y adición de nuevas funcionalidades basadas en inteligencia artificial.

Agradecimientos

Agradecemos al Ing. Angel Cudco por su guía y enseñanzas en la asignatura de Desarrollo de Software Seguro, que fueron fundamentales para la implementación de las medidas de seguridad de este proyecto. También a la Universidad de las Fuerzas Armadas ESPE por proporcionar el entorno académico y los recursos necesarios para el desarrollo exitoso de este trabajo.

Referencias

[1] Auth0, "Introduction to JSON Web Tokens," https://jwt.io/introduction, 2024.
[2] OAuth, "OAuth 2.0," https://oauth.net/2/, 2024.
[3] S. I. Serengil and A. Ozpinar, "HyperExtended LightFace: A Facial Attribute Analysis Framework," in *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, pp. 1-4, 2021.
[4] N. Provos and D. Mazières, "A Future-Adaptable Password Scheme," in *Proceedings of the 1999 USENIX Annual Technical Conference*, pp. 81-92, 1999.
[5] Docker Inc., "Docker Documentation," https://docs.docker.com/, 2024.
[6] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," https://martinfowler.com/articles/microservices.html, 2014.
[7] Node.js Foundation, "Node.js Documentation," https://nodejs.org/en/docs/, 2024.
[8] Meta Platforms, Inc., "React Documentation," https://react.dev/, 2024.
[9] MongoDB Inc., "MongoDB Manual," https://www.mongodb.com/docs/, 2024.
[10] F5 Networks, Inc., "NGINX Documentation," https://nginx.org/en/docs/, 2024.
[11] Internet Security Research Group, "Let's Encrypt Documentation," https://letsencrypt.org/docs/, 2024.
[12] Cypress.io, "Cypress Documentation," https://docs.cypress.io/, 2024.
[13] OWASP Foundation, "OWASP Top Ten," https://owasp.org/www-project-top-ten/, 2024.

[14] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Doctoral dissertation, University of California, Irvine, 2000.
[15] W3C, "Cross-Origin Resource Sharing," https://www.w3.org/TR/cors/, 2024.
[16] Google, "TensorFlow Documentation," https://www.tensorflow.org/, 2024.
[17] S. Ramírez, "FastAPI Documentation," https://fastapi.tiangolo.com/, 2024.
[18] Jared Hanson, "Passport.js Documentation," http://www.passportjs.org/docs/, 2024.
[19] Express.js, "Express API Reference," https://expressjs.com/en/api.html, 2024.
[20] Evan You, "Vite Documentation," https://vitejs.dev/, 2024.

APÉNDICE

```
TravelBrain/
|-- frontend-react/
|   |-- src/
|   |   |-- components/
|   |   |-- pages/
|   |   |-- context/
|   |   |-- utils/
|   |   +-- App.jsx
|   |-- cypress/
|   |   +-- e2e/
|   +-- package.json
|-- backend-project/
|   |-- src/
|   |   |-- controllers/
|   |   |-- models/
|   |   |-- routes/
|   |   |-- middlewares/
|   |   |-- utils/
|   |   +-- server.js
|   |-- tests/
|   +-- package.json
|-- business-rules-backend/
|   |-- src/
|   |   |-- controllers/
|   |   |-- validators/
|   |   |-- services/
|   |   +-- server.js
|   +-- package.json
|-- facial-recognition-service/
|   |-- src/
|   |   |-- api/
|   |   |-- models/
|   |   |-- services/
|   |   +-- app.py
|   +-- requirements.txt
|-- nginx/
|   |-- nginx.conf
|   +-- Dockerfile
|-- certbot/
|   |-- conf/
|   +-- www/
+-- docker-compose.yml
```

Listing 14: Estructura de directorios

```
# Backend API
NODE_ENV=production
PORT=3004
MONGODB_URI=mongodb+srv://...
JWT_SECRET=your-secret-key-here
JWT_EXPIRES_IN=7d

# Google OAuth
GOOGLE_CLIENT_ID=your-client-id
GOOGLE_CLIENT_SECRET=your-client-secret
GOOGLE_CALLBACK_URL=https://travelbrain.ddns.
    net/api/auth/google/callback

# APIs externas
OPENWEATHER_API_KEY=your-api-key
MAPBOX_TOKEN=your-mapbox-token

# CORS
CORS_ORIGINS=https://travelbrain.ddns.net

# Business Rules API
BR_PORT=3005

# Facial Recognition Service
FACE_PORT=8000
FACE_DETECTION_BACKEND=retinaface
FACE_RECOGNITION_MODEL=Facenet512
SIMILARITY_THRESHOLD=0.6
```

Listing 15: .env (ejemplo)

```
# Iniciar todos los servicios
docker-compose up -d --build

# Ver logs
docker-compose logs -f

# Detener servicios
docker-compose down

# Reiniciar un servicio específico
docker-compose restart backend

# Obtener certificados SSL
sh init-letsencrypt.sh

# Ver estado de contenedores
docker-compose ps

# Acceder a un contenedor
docker exec -it travelbrain-backend sh

# Limpiar volúmenes
docker-compose down -v
```

Listing 16: Comandos útiles