BillPaymentApp - Technology Stack Specification

Project: UK Bill Payment Platform
Version: 1.0
Date: August 15, 2025
Document Type: Technical Implementation Guide

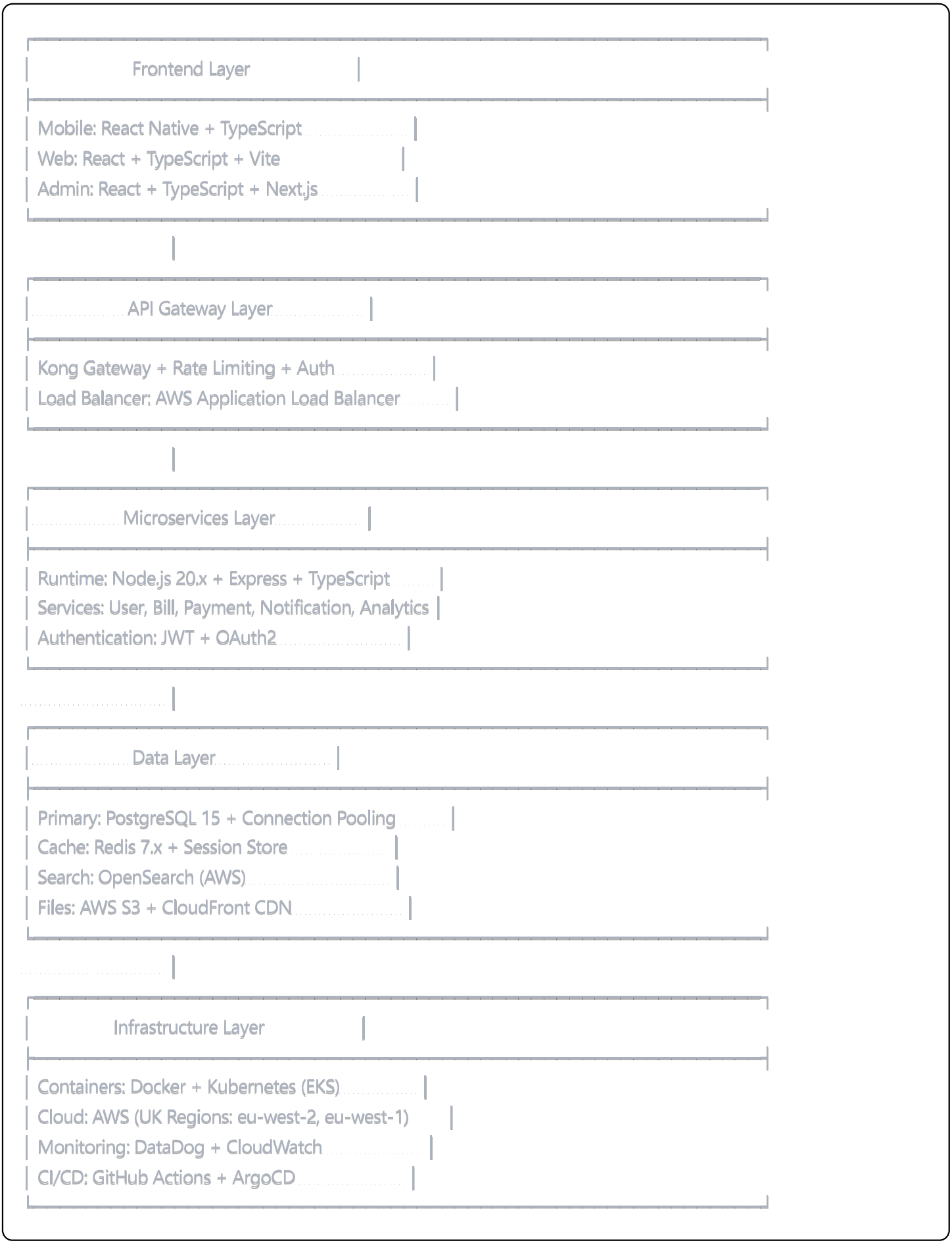
1. Executive Summary

This document defines the complete technology stack for BillPaymentApp, including rationale for technology choices, implementation guidelines, and operational considerations for a production-ready fintech application.

1.1 Technology Philosophy

- **Security First**: All technology choices prioritize security and compliance
- **Scalability**: Architecture designed for 100k+ users from day one
- **Developer Experience**: Tools that enable rapid, reliable development
- **Cost Efficiency**: Balance of performance and operational cost
- **UK Compliance**: Technologies that support regulatory requirements

1.2 Stack Overview



2. Frontend Technologies

2.1 Mobile Application

2.1.1 React Native Setup

```json
{
  "name": "billpayment-mobile",
  "version": "1.0.0",
  "dependencies": {
    "react-native": "0.73.x",
    "typescript": "5.x",
    "@react-navigation/native": "6.x",
    "@react-navigation/stack": "6.x",
    "@reduxjs/toolkit": "2.x",
    "react-redux": "9.x",
    "react-native-keychain": "8.x",
    "react-native-biometrics": "3.x",
    "react-native-document-scanner": "1.x",
    "react-native-push-notification": "10.x"
  }
}
```

2.1.2 Key Mobile Libraries

Library	Purpose	Justification
React Navigation	Navigation	Industry standard, flexible routing
Redux Toolkit	State Management	Predictable state, debugging tools
React Native Keychain	Secure Storage	Biometric authentication support
React Native Camera	OCR Scanning	Bill document capture
Flipper	Development Tools	Advanced debugging capabilities

2.1.3 Mobile Security Implementation

```typescript
```

```
// Secure storage implementation
import { setInternetCredentials, getInternetCredentials } from 'react-native-keychain';

class SecureStorage {
  static async storeToken(token: string): Promise<void> {
    await setInternetCredentials('billpay_auth', 'user', token, {
      accessControl: 'BiometryAny',
      authenticatePrompt: 'Authenticate to access your account',
    });
  }

  static async getToken(): Promise<string | null> {
    try {
      const credentials = await getInternetCredentials('billpay_auth');
      return credentials ? credentials.password : null;
    } catch (error) {
      return null;
    }
  }
}
```

2.2 Web Application

2.2.1 React Web Setup

json

```
{
  "name": "billpayment-web",
  "version": "1.0.0",
  "dependencies": {
    "react": "18.x",
    "typescript": "5.x",
    "vite": "5.x",
    "@tanstack/react-query": "5.x",
    "react-router-dom": "6.x",
    "@headlessui/react": "1.x",
    "tailwindcss": "3.x",
    "react-hook-form": "7.x",
    "zod": "3.x"
  }
}
```

2.2.2 UI Component System

typescript

```
// Design system configuration
const theme = {
  colors: {
    primary: {
      50: '#eff6ff',
      500: '#3b82f6',
      900: '#1e3a8a'
    },
    success: '#10b981',
    error: '#ef4444',
    warning: '#f59e0b'
  },
  spacing: {
    xs: '0.25rem',
    sm: '0.5rem',
    md: '1rem',
    lg: '1.5rem',
    xl: '3rem'
  }
};

// Reusable button component
interface ButtonProps {
  variant: 'primary' | 'secondary' | 'danger';
  size: 'sm' | 'md' | 'lg';
  loading?: boolean;
  children: React.ReactNode;
}

export const Button: React.FC<ButtonProps> = ({ variant, size, loading, children }) => {
  const baseClasses = 'font-medium rounded-lg focus:ring-2 focus:ring-offset-2';
  const variantClasses = {
    primary: 'bg-blue-600 text-white hover:bg-blue-700 focus:ring-blue-500',
    secondary: 'bg-gray-200 text-gray-900 hover:bg-gray-300 focus:ring-gray-500',
    danger: 'bg-red-600 text-white hover:bg-red-700 focus:ring-red-500'
  };

  return (
    <button
      className={`${baseClasses} ${variantClasses[variant]}`}
      disabled={loading}
    >
      {loading ? <Spinner /> : children}
```

```
    </button>
  );
};
```

2.3 Admin Dashboard

2.3.1 Next.js Setup for Admin Portal

```json
{
  "name": "billpayment-admin",
  "dependencies": {
    "next": "14.x",
    "typescript": "5.x",
    "react": "18.x",
    "@tanstack/react-table": "8.x",
    "recharts": "2.x",
    "date-fns": "3.x",
    "react-hook-form": "7.x"
  }
}
```

3. Backend Technologies

3.1 Microservices Architecture

3.1.1 Service Structure

```
src/
├── services/
│   ├── user-service/
│   │   ├── src/
│   │   │   ├── controllers/
│   │   │   ├── models/
│   │   │   ├── middleware/
│   │   │   ├── routes/
│   │   │   └── utils/
│   │   ├── tests/
│   │   └── Dockerfile
│   ├── bill-service/
│   ├── payment-service/
│   └── notification-service/
├── shared/
│   ├── types/
│   ├── utils/
│   └── middleware/
└── infrastructure/
    ├── docker-compose.yml
    └── kubernetes/
```

3.1.2 Node.js Service Template

typescript

```typescript
// Base service structure
import express from 'express';
import helmet from 'helmet';
import cors from 'cors';
import rateLimit from 'express-rate-limit';
import { errorHandler } from './middleware/errorHandler';
import { authMiddleware } from './middleware/auth';
import { validateRequest } from './middleware/validation';

class BaseService {
  private app: express.Application;

  constructor(private serviceName: string) {
    this.app = express();
    this.setupMiddleware();
    this.setupRoutes();
    this.setupErrorHandling();
  }

  private setupMiddleware(): void {
    this.app.use(helmet());
    this.app.use(cors({
      origin: process.env.ALLOWED_ORIGINS?.split(','),
      credentials: true
    }));

    this.app.use(rateLimit({
      windowMs: 15 * 60 * 1000, // 15 minutes
      max: 100 // limit each IP to 100 requests per windowMs
    }));

    this.app.use(express.json({ limit: '10mb' }));
    this.app.use(authMiddleware);
  }

  private setupErrorHandling(): void {
    this.app.use(errorHandler);
  }

  public start(port: number): void {
    this.app.listen(port, () => {
      console.log(`${this.serviceName} running on port ${port}`);
    });
```

```
  }
}
```

3.2 Database Layer

3.2.1 PostgreSQL Configuration

typescript

```typescript
// Database connection with pooling
import { Pool } from 'pg';
import { config } from './config';

const pool = new Pool({
  host: config.database.host,
  port: config.database.port,
  database: config.database.name,
  user: config.database.user,
  password: config.database.password,
  ssl: {
    rejectUnauthorized: false
  },
  max: 20, // maximum number of clients in the pool
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
});

// Database service class
export class DatabaseService {
  static async query<T>(text: string, params?: any[]): Promise<T[]> {
    const start = Date.now();
    const client = await pool.connect();

    try {
      const result = await client.query(text, params);
      const duration = Date.now() - start;

      console.log('Executed query', { text, duration, rows: result.rowCount });
      return result.rows;
    } finally {
      client.release();
    }
  }

  static async transaction<T>(callback: (client: any) => Promise<T>): Promise<T> {
    const client = await pool.connect();

    try {
      await client.query('BEGIN');
      const result = await callback(client);
      await client.query('COMMIT');
      return result;
```

```
    } catch (error) {
      await client.query('ROLLBACK');
      throw error;
    } finally {
      client.release();
    }
  }
}
```

3.2.2 Data Migration System

typescript

```
// Migration interface
interface Migration {
  version: string;
  description: string;
  up: (db: DatabaseService) => Promise<void>;
  down: (db: DatabaseService) => Promise<void>;
}

// Example migration
export const migration_001_create_users: Migration = {
  version: '001',
  description: 'Create users table',
  up: async (db) => {
    await db.query(`
      CREATE TABLE users (
        id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
        email VARCHAR(255) UNIQUE NOT NULL,
        phone VARCHAR(20) UNIQUE NOT NULL,
        password_hash VARCHAR(255) NOT NULL,
        kyc_status VARCHAR(20) DEFAULT 'pending',
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      );

      CREATE INDEX idx_users_email ON users(email);
      CREATE INDEX idx_users_kyc_status ON users(kyc_status);
    `);
  },
  down: async (db) => {
    await db.query('DROP TABLE users;');
  }
};
```

3.3 Authentication & Security

3.3.1 JWT Implementation

typescript

```
import jwt from 'jsonwebtoken';
import bcrypt from 'bcryptjs';
import { config } from './config';

interface JWTPayload {
  userId: string;
  email: string;
  role: string;
}

export class AuthService {
  static async hashPassword(password: string): Promise<string> {
    const saltRounds = 12;
    return bcrypt.hash(password, saltRounds);
  }

  static async validatePassword(password: string, hash: string): Promise<boolean> {
    return bcrypt.compare(password, hash);
  }

  static generateToken(payload: JWTPayload): string {
    return jwt.sign(payload, config.jwt.secret, {
      expiresIn: config.jwt.expiresIn,
      issuer: 'billpayment-app',
      audience: 'billpayment-users'
    });
  }

  static validateToken(token: string): JWTPayload | null {
    try {
      return jwt.verify(token, config.jwt.secret) as JWTPayload;
    } catch (error) {
      return null;
    }
  }
}
```

4. External Integrations

4.1 Payment Processing

4.1.1 Open Banking Integration (TrueLayer)

typescript

```
import { TrueLayerClient } from 'truelayer-client';

class OpenBankingService {
  private client: TrueLayerClient;

  constructor() {
    this.client = new TrueLayerClient({
      clientId: process.env.TRUELAYER_CLIENT_ID!,
      clientSecret: process.env.TRUELAYER_CLIENT_SECRET!,
      environment: process.env.NODE_ENV === 'production' ? 'live' : 'sandbox'
    });
  }

  async linkAccount(userId: string): Promise<string> {
    const authUrl = await this.client.getAuthUrl({
      redirectUri: `${process.env.APP_URL}/auth/callback`,
      scopes: ['accounts', 'balance', 'transactions'],
      state: userId
    });

    return authUrl;
  }

  async initiatePayment(paymentRequest: PaymentRequest): Promise<PaymentResult> {
    const payment = await this.client.createPayment({
      amount: paymentRequest.amount,
      currency: 'GBP',
      paymentMethod: {
        type: 'bank_transfer',
        providerSelection: {},
        beneficiary: {
          type: 'external_account',
          reference: paymentRequest.reference,
          accountIdentifier: {
            type: 'sort_code_account_number',
            sortCode: paymentRequest.sortCode,
            accountNumber: paymentRequest.accountNumber
          }
        }
      },
      user: {
        id: paymentRequest.userId
      }
```

```
    });

    return {
      paymentId: payment.id,
      status: payment.status,
      authorizationUrl: payment.authorizationFlow?.actions?.next?.uri
    };
  }
}
```

4.1.2 Stripe Integration

typescript

```
import Stripe from 'stripe';

class CardPaymentService {
  private stripe: Stripe;

  constructor() {
    this.stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
      apiVersion: '2023-10-16'
    });
  }

  async createPaymentIntent(amount: number, customerId: string): Promise<string> {
    const paymentIntent = await this.stripe.paymentIntents.create({
      amount: Math.round(amount * 100), // Convert to pence
      currency: 'gbp',
      customer: customerId,
      payment_method_types: ['card'],
      metadata: {
        service: 'billpayment'
      }
    });

    return paymentIntent.client_secret!;
  }

  async confirmPayment(paymentIntentId: string): Promise<boolean> {
    const paymentIntent = await this.stripe.paymentIntents.retrieve(paymentIntentId);
    return paymentIntent.status === 'succeeded';
  }
}
```

4.2 Notifications

4.2.1 Multi-Channel Notification Service

typescript

```
interface NotificationChannel {
  send(message: NotificationMessage): Promise<void>;
}

class SMSChannel implements NotificationChannel {
  async send(message: NotificationMessage): Promise<void> {
    // Twilio implementation
    const client = require('twilio')(
      process.env.TWILIO_SID,
      process.env.TWILIO_AUTH_TOKEN
    );

    await client.messages.create({
      body: message.content,
      from: process.env.TWILIO_PHONE_NUMBER,
      to: message.recipient
    });
  }
}

class PushNotificationChannel implements NotificationChannel {
  async send(message: NotificationMessage): Promise<void> {
    // Firebase implementation
    const admin = require('firebase-admin');

    await admin.messaging().send({
      token: message.deviceToken,
      notification: {
        title: message.title,
        body: message.content
      },
      data: message.data
    });
  }
}

class NotificationService {
  private channels: Map<string, NotificationChannel> = new Map();

  constructor() {
    this.channels.set('sms', new SMSChannel());
    this.channels.set('push', new PushNotificationChannel());
  }
```

```
  async sendNotification(type: string, message: NotificationMessage): Promise<void> {
    const channel = this.channels.get(type);
    if (!channel) {
      throw new Error(`Notification channel ${type} not found`);
    }

    await channel.send(message);
  }
}
```

5. Infrastructure & DevOps

5.1 Containerization

5.1.1 Docker Configuration

dockerfile

```
# Multi-stage Dockerfile for Node.js service
FROM node:20-alpine AS builder

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

FROM node:20-alpine AS runtime

RUN addgroup -g 1001 -S nodejs
RUN adduser -S nextjs -u 1001

WORKDIR /app
COPY --from=builder /app/node_modules ./node_modules
COPY --chown=nextjs:nodejs . .

USER nextjs

EXPOSE 3000
ENV PORT 3000

CMD ["npm", "start"]
```

5.1.2 Kubernetes Deployment

yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: billpay-user-service
  labels:
    app: billpay-user-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: billpay-user-service
  template:
    metadata:
      labels:
        app: billpay-user-service
    spec:
      containers:
      - name: user-service
        image: billpay/user-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: billpay-secrets
              key: database-url
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
```

```yaml
        port: 3000
      initialDelaySeconds: 5
      periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: billpay-user-service
spec:
  selector:
    app: billpay-user-service
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: ClusterIP
```

5.2 CI/CD Pipeline

5.2.1 GitHub Actions Workflow

yaml

```yaml
name: Deploy BillPayment Services

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: billpayment-app

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '20'
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm run test:coverage

      - name: Run integration tests
        run: npm run test:integration
        env:
          DATABASE_URL: postgresql://test:test@localhost:5432/test

      - name: Security audit
        run: npm audit --audit-level high

  build:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
```

```
    steps:
      - uses: actions/checkout@v4

      - name: Log in to Container Registry
        uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Build and push Docker image
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }}

  deploy:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
      - name: Deploy to EKS
        run: |
          aws eks update-kubeconfig --name billpay-production
          kubectl set image deployment/billpay-user-service user-service=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ g
          kubectl rollout status deployment/billpay-user-service
```

5.3 Monitoring & Observability

5.3.1 Application Monitoring Setup

typescript

```typescript
// Monitoring middleware
import { Request, Response, NextFunction } from 'express';
import StatsD from 'node-statsd';
import { performance } from 'perf_hooks';

const statsClient = new StatsD({
  host: process.env.STATSD_HOST || 'localhost',
  port: parseInt(process.env.STATSD_PORT || '8125')
});

export const monitoringMiddleware = (req: Request, res: Response, next: NextFunction) => {
  const start = performance.now();

  // Track request count
  statsClient.increment('api.requests.total', 1, {
    method: req.method,
    endpoint: req.route?.path || 'unknown'
  });

  res.on('finish', () => {
    const duration = performance.now() - start;

    // Track response time
    statsClient.timing('api.response_time', duration, {
      method: req.method,
      status_code: res.statusCode.toString(),
      endpoint: req.route?.path || 'unknown'
    });

    // Track status codes
    statsClient.increment('api.responses.total', 1, {
      status_code: res.statusCode.toString(),
      method: req.method
    });
  });

  next();
};
```

5.3.2 Health Check Implementation

typescript

```typescript
// Health check endpoints
export class HealthCheckService {
  static async checkDatabase(): Promise<boolean> {
    try {
      await DatabaseService.query('SELECT 1');
      return true;
    } catch (error) {
      return false;
    }
  }

  static async checkRedis(): Promise<boolean> {
    try {
      await redisClient.ping();
      return true;
    } catch (error) {
      return false;
    }
  }

  static async checkExternalServices(): Promise<Record<string, boolean>> {
    const checks = await Promise.allSettled([
      fetch(`${process.env.TRUELAYER_API_URL}/health`),
      fetch(`${process.env.STRIPE_API_URL}/health`)
    ]);

    return {
      truelayer: checks[0].status === 'fulfilled',
      stripe: checks[1].status === 'fulfilled'
    };
  }

  static async getHealthStatus() {
    const [database, redis, external] = await Promise.all([
      this.checkDatabase(),
      this.checkRedis(),
      this.checkExternalServices()
    ]);

    return {
      status: database && redis ? 'healthy' : 'unhealthy',
      timestamp: new Date().toISOString(),
      checks: {
```

```
        database,
        redis,
        external
      }
    };
  }
}
```

6. Development Tools & Environment

6.1 Development Setup

6.1.1 Local Development Environment

yaml

```yaml
# docker-compose.dev.yml
version: '3.8'
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: billpayment_dev
      POSTGRES_USER: developer
      POSTGRES_PASSWORD: devpassword
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  mailhog:
    image: mailhog/mailhog
    ports:
      - "1025:1025"
      - "8025:8025"

volumes:
  postgres_data:
```

6.1.2 Development Scripts

```json
{
  "scripts": {
    "dev": "concurrently \"npm run dev:services\" \"npm run dev:web\" \"npm run dev:mobile\"",
    "dev:services": "docker-compose -f docker-compose.dev.yml up",
    "dev:web": "cd apps/web && npm run dev",
    "dev:mobile": "cd apps/mobile && npx expo start",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:coverage": "jest --coverage",
    "test:integration": "jest --config jest.integration.config.js",
    "lint": "eslint . --ext .ts,.tsx",
    "lint:fix": "eslint . --ext .ts,.tsx --fix",
    "type-check": "tsc --noEmit",
    "build": "npm run build:services && npm run build:web",
    "migrate": "node scripts/migrate.js",
    "seed": "node scripts/seed.js"
  }
}
```

6.2 Code Quality Tools

6.2.1 ESLint Configuration

```json
```

```
{
  "extends": [
    "@typescript-eslint/recommended",
    "plugin:security/recommended",
    "plugin:import/typescript"
  ],
  "plugins": ["@typescript-eslint", "security", "import"],
  "rules": {
    "@typescript-eslint/no-unused-vars": "error",
    "@typescript-eslint/explicit-function-return-type": "warn",
    "security/detect-object-injection": "error",
    "import/order": ["error", {
      "groups": ["builtin", "external", "internal", "parent", "sibling"],
      "newlines-between": "always"
    }]
  }
}
```

6.2.2 Testing Configuration

typescript

```
// jest.config.js
module.exports = {
  preset: 'ts-jest',
  testEnvironment: 'node',
  collectCoverageFrom: [
    'src/**/*.{ts,tsx}',
    '!src/**/*.d.ts',
    '!src/tests/**/*'
  ],
  coverageThreshold: {
    global: {
      branches: 80,
      functions: 80,
      lines: 80,
      statements: 80
    }
  },
  setupFilesAfterEnv: ['<rootDir>/src/tests/setup.ts']
};

// Example test
describe('PaymentService', () => {
  let paymentService: PaymentService;

  beforeEach(() => {
    paymentService = new PaymentService();
  });

  describe('processPayment', () => {
    it('should process payment successfully', async () => {
      const paymentRequest = {
        amount: 100.50,
        billerAccount: '12345678',
        sortCode: '12-34-56',
        reference: 'BILL001'
      };

      const result = await paymentService.processPayment(paymentRequest);

      expect(result.status).toBe('success');
      expect(result.transactionId).toBeDefined();
    });
```

```
    it('should handle insufficient funds', async () => {
      const paymentRequest = {
        amount: 10000, // Large amount
        billerAccount: '12345678',
        sortCode: '12-34-56',
        reference: 'BILL001'
      };

      await expect(paymentService.processPayment(paymentRequest))
        .rejects.toThrow('Insufficient funds');
    });
  });
});
```

7. Security Implementation

7.1 Application Security

7.1.1 Input Validation

typescript

```
import { z } from 'zod';

// Validation schemas
const BillSchema = z.object({
  billerName: z.string().min(2).max(100),
  amount: z.number().positive().max(10000),
  dueDate: z.date().min(new Date()),
  accountNumber: z.string().regex(/^[0-9]{8,12}$/)
});

const PaymentSchema = z.object({
  billId: z.string().uuid(),
  amount: z.number().positive(),
  paymentMethod: z.enum(['bank_transfer', 'card'])
});

// Validation middleware
export const validateBody = (schema: z.ZodSchema) => {
  return (req: Request, res: Response, next: NextFunction) => {
    try {
      schema.parse(req.body);
      next();
    } catch (error) {
      if (error instanceof z.ZodError) {
        return res.status(400).json({
          error: 'Validation failed',
          details: error.errors
        });
      }
      next(error);
    }
  };
};
```

7.1.2 Rate Limiting & DDoS Protection

typescript

```
import rateLimit from 'express-rate-limit';
import slowDown from 'express-slow-down';

// General API rate limiting
export const apiLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // Limit each IP to 100 requests per windowMs
  message: 'Too many requests from this IP, please try again later.',
  standardHeaders: true,
  legacyHeaders: false,
});

// Strict rate limiting for sensitive endpoints
export const authLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 5, // Limit login attempts
  skipSuccessfulRequests: true,
});

// Payment rate limiting
export const paymentLimiter = rateLimit({
  windowMs: 60 * 1000, // 1 minute
  max: 3, // Max 3 payments per minute
  keyGenerator: (req) => req.user?.id || req.ip,
});

// Speed limiting for suspicious activity
export const speedLimiter = slowDown({
  windowMs: 15 * 60 * 1000,
  delayAfter: 50,
  delayMs: 500
});
```

7.2 Data Encryption

7.2.1 Encryption Service

typescript

```
import crypto from 'crypto';

export class EncryptionService {
  private static readonly algorithm = 'aes-256-gcm';
  private static readonly keyLength = 32;
  private static readonly ivLength = 16;
  private static readonly tagLength = 16;

  static encrypt(text: string, key: string): string {
    const iv = crypto.randomBytes(this.ivLength);
    const cipher = crypto.createCipher(this.algorithm, Buffer.from(key, 'hex'));
    cipher.setAAD(Buffer.from('billpayment', 'utf8'));

    let encrypted = cipher.update(text, 'utf8', 'hex');
    encrypted += cipher.final('hex');

    const tag = cipher.getAuthTag();

    return iv.toString('hex') + encrypted + tag.toString('hex');
  }

  static decrypt(encryptedData: string, key: string): string {
    const iv = Buffer.from(encryptedData.slice(0, this.ivLength * 2), 'hex');
    const tag = Buffer.from(encryptedData.slice(-this.tagLength * 2), 'hex');
    const encrypted = encryptedData.slice(this.ivLength * 2, -this.tagLength * 2);

    const decipher = crypto.createDecipher(this.algorithm, Buffer.from(key, 'hex'));
    decipher.setAAD(Buffer.from('billpayment', 'utf8'));
    decipher.setAuthTag(tag);

    let decrypted = decipher.update(encrypted, 'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return decrypted;
  }
}
```

8. Performance Optimization

8.1 Database Optimization

8.1.1 Connection Pooling & Query Optimization

typescript

```
// Database configuration for production
const poolConfig = {
  host: process.env.DB_HOST,
  port: parseInt(process.env.DB_PORT || '5432'),
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,

  // Connection pool settings
  max: 20, // maximum number of clients in the pool
  min: 5,  // minimum number of clients in the pool
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,

  // Performance settings
  statement_timeout: 30000,
  query_timeout: 30000,

  ssl: process.env.NODE_ENV === 'production' ? {
    rejectUnauthorized: false
  } : false
};

// Optimized queries with proper indexing
class BillRepository {
  static async findUserBills(userId: string, limit = 20, offset = 0) {
    return DatabaseService.query(`
      SELECT
        b.id,
        b.biller_name,
        b.amount,
        b.due_date,
        b.status,
        b.created_at
      FROM bills b
      WHERE b.user_id = $1
        AND b.deleted_at IS NULL
      ORDER BY b.due_date ASC, b.created_at DESC
      LIMIT $2 OFFSET $3
    `, [userId, limit, offset]);
  }

  // Use prepared statements for frequent queries
```

```
  static async findOverdueBills() {
    return DatabaseService.query(`
      SELECT b.*, u.email, u.phone
      FROM bills b
      JOIN users u ON b.user_id = u.id
      WHERE b.due_date < CURRENT_DATE
        AND b.status = 'pending'
        AND b.deleted_at IS NULL
    `);
  }
}
```

8.2 Caching Strategy

8.2.1 Redis Implementation

typescript

```typescript
import Redis from 'ioredis';

class CacheService {
  private redis: Redis;

  constructor() {
    this.redis = new Redis({
      host: process.env.REDIS_HOST,
      port: parseInt(process.env.REDIS_PORT || '6379'),
      password: process.env.REDIS_PASSWORD,
      retryDelayOnFailover: 100,
      maxRetriesPerRequest: 3,
      lazyConnect: true
    });
  }

  async get<T>(key: string): Promise<T | null> {
    try {
      const value = await this.redis.get(key);
      return value ? JSON.parse(value) : null;
    } catch (error) {
      console.error('Cache get error:', error);
      return null;
    }
  }

  async set(key: string, value: any, ttlSeconds = 3600): Promise<void> {
    try {
      await this.redis.setex(key, ttlSeconds, JSON.stringify(value));
    } catch (error) {
      console.error('Cache set error:', error);
    }
  }

  async invalidate(pattern: string): Promise<void> {
    try {
      const keys = await this.redis.keys(pattern);
      if (keys.length > 0) {
        await this.redis.del(...keys);
      }
    } catch (error) {
      console.error('Cache invalidate error:', error);
    }
```

```
  }
}

// Cache decorators for frequently accessed data
export const cache = (ttl = 3600) => {
  return (target: any, propertyName: string, descriptor: PropertyDescriptor) => {
    const method = descriptor.value;

    descriptor.value = async function (...args: any[]) {
      const cacheKey = `${target.constructor.name}:${propertyName}:${JSON.stringify(args)}`;

      let result = await cacheService.get(cacheKey);
      if (result === null) {
        result = await method.apply(this, args);
        await cacheService.set(cacheKey, result, ttl);
      }

      return result;
    };
  };
};
```

9. Cost Estimation

9.1 Infrastructure Costs (Monthly)

Service	Configuration	Monthly Cost (GBP)
EKS Cluster	3 nodes (t3.medium)	£120
RDS PostgreSQL	db.t3.medium Multi-AZ	£180
ElastiCache Redis	cache.t3.micro	£25
Load Balancer	Application LB	£20
S3 Storage	100GB + requests	£5
CloudFront CDN	1TB transfer	£50
Total Infrastructure		**£400**

9.2 Third-Party Service Costs

Service	Usage Model	Monthly Cost (GBP)
TrueLayer	£0.10 per API call	£200 (2k calls)
Stripe	1.4% + 20p per transaction	£350 (£25k volume)
Twilio SMS	£0.04 per SMS	£80 (2k SMS)
DataDog	£12 per host	£36 (3 hosts)
Total Services		**£666**

9.3 Development Tools

Tool	Purpose	Monthly Cost (GBP)
GitHub Actions	CI/CD (3000 minutes)	£12
Sentry	Error monitoring	£20
Auth0	Authentication (1k MAU)	£18
Total Tools		**£50**

Total Monthly Operating Cost: £1,116

Document Approval:

- Chief Technology Officer: _____
- Lead Developer: _____
- DevOps Engineer: _____
- Security Architect: _____