# **SOFTWARE DESIGN DOCUMENT (SDD)**

# **SkillSync Peer Learning Exchange Platform**

# **Document Control**

• Project Name: SkillSync Exchange Platform

• **Version:** 1.0

• **Date:** October 25, 2025

• **Authors:** [Architecture Team]

• Status: Draft for Technical Review

• Distribution: Engineering, DevOps, Technical Leadership

#### TABLE OF CONTENTS

- 1. Introduction
- 2. System Architecture Overview
- 3. Component Design
- 4. Database Design
- 5. API Design
- 6. Security Architecture
- 7. Deployment Architecture
- 8. Performance Considerations
- 9. Testing Strategy
- 10. Appendices

#### 1. INTRODUCTION

#### 1.1 Purpose

This Software Design Document (SDD) provides the technical design and architecture for the SkillSync Peer Learning Exchange Platform. It serves as a blueprint for the development team, detailing system components, interactions, data structures, and implementation strategies.

## 1.2 Scope

#### This document covers:

- High-level system architecture
- Component design and interactions
- Database schema and data models
- API specifications (REST + WebSocket)
- Security architecture
- Deployment and infrastructure
- Performance optimization strategies
- Testing approach

## 1.3 Design Goals

- Scalability: Support 10,000+ concurrent users
- Reliability: 99.9% uptime SLA
- **Performance:** <2s page load, <200ms API response
- Security: OWASP Top 10 compliant, GDPR/CCPA compliant
- Maintainability: Modular, well-documented, testable code
- Extensibility: Easy to add new features without major refactoring

#### 1.4 Technology Stack

#### Frontend:

- Framework: React 18+ with TypeScript
- State Management: Redux Toolkit (global) + React Query (server state)
- UI Library: Tailwind CSS + Headless UI
- Routing: React Router v6
- Real-time: Socket.io-client
- Forms: React Hook Form + Zod validation
- Build Tool: Vite

#### **Backend:**

• Runtime: Node.js 20 LTS

• Framework: Express.js 4.x

• Language: TypeScript

• **ORM:** Prisma or TypeORM

• **Authentication:** JWT + Passport.js

• Real-time: Socket.io

• Job Queue: Bull (Redis-based)

• Validation: Zod

#### **Database:**

• **Primary:** PostgreSQL 15+ (relational data)

• Cache: Redis 7+ (sessions, real-time, job queue)

• Search: PostgreSQL Full-Text Search (or Elasticsearch for scale)

#### AI/ML:

• Matching Algorithm: Python microservice

• Framework: scikit-learn or TensorFlow

• **API:** FastAPI (Python)

• NLP: OpenAI API or Hugging Face (skill extraction)

#### **Storage:**

• Object Storage: AWS S3 or Cloudflare R2

• CDN: CloudFlare

#### **Third-Party Services:**

• Video: Zoom API or Daily.co

• Calendar: Google Calendar API, Microsoft Graph API

• Payments: Stripe

• Email: SendGrid or AWS SES

• Push Notifications: Firebase Cloud Messaging

#### **DevOps:**

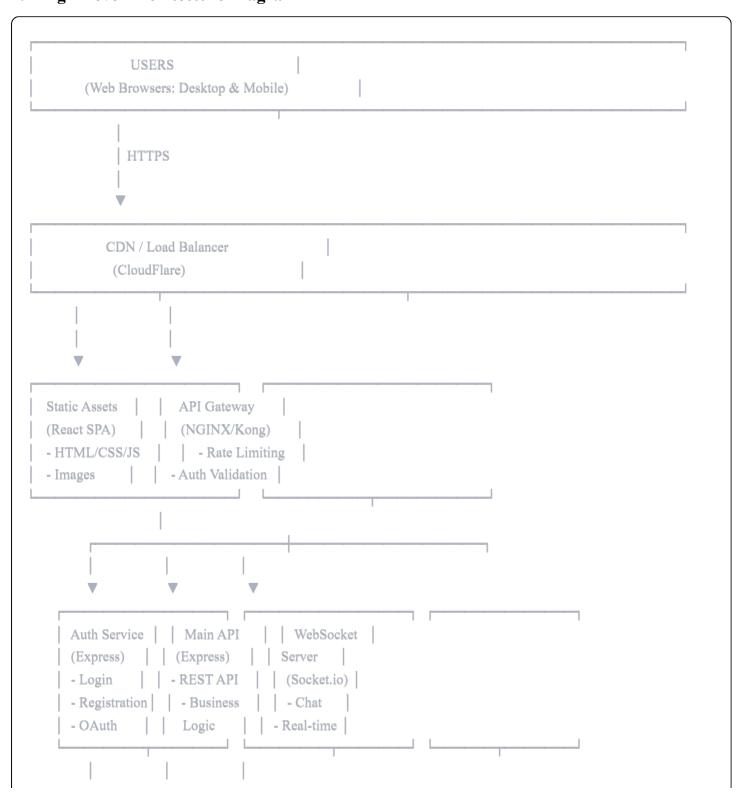
• Hosting: AWS (EC2/ECS) or Google Cloud

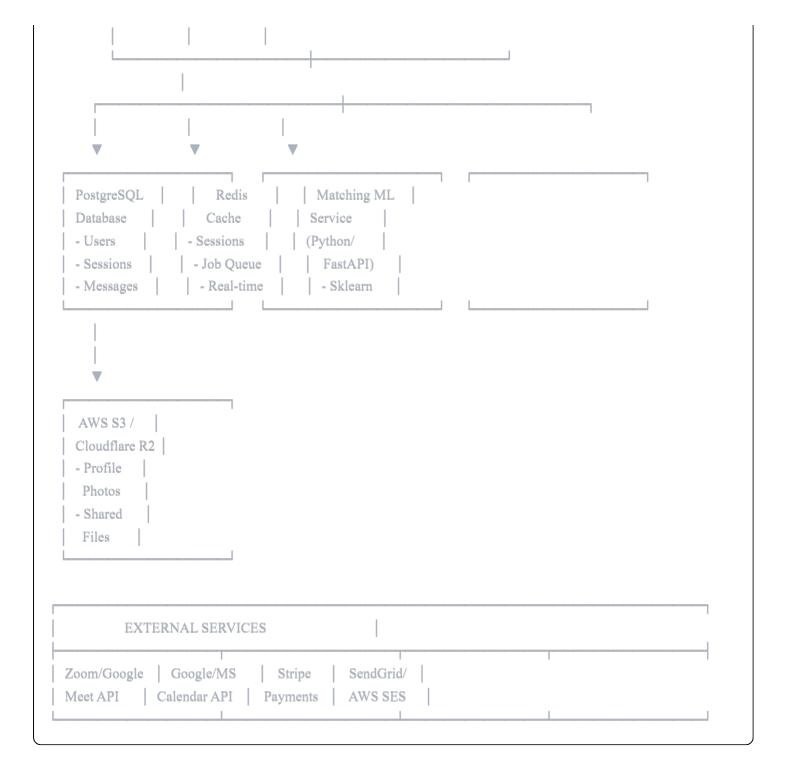
• Containerization: Docker

- Orchestration: Docker Compose (dev) / Kubernetes (prod)
- CI/CD: GitHub Actions
- Monitoring: Datadog or New Relic
- **Logging:** Winston + ELK Stack
- **Version Control:** Git + GitHub

## 2. SYSTEM ARCHITECTURE OVERVIEW

# 2.1 High-Level Architecture Diagram





#### 2.2 Architectural Patterns

# 1. Microservices Architecture (Hybrid)

• Monolithic Core: Main API handles most business logic

#### • Specialized Microservices:

- Matching ML Service (Python)
- Notification Service (Node.js)
- File Processing Service (Node.js)
- Rationale: Balance between simplicity and scalability

#### 2. Event-Driven Architecture

- Event Bus: Redis Pub/Sub or RabbitMQ
- Use Cases:
  - Session state changes trigger notifications
  - Rating submissions trigger credit releases
  - Profile updates trigger match recalculations
- Benefits: Decoupled services, async processing

#### 3. API Gateway Pattern

- Gateway: NGINX or Kong
- Responsibilities:
  - Route requests to appropriate services
  - Rate limiting (100 requests/min per user)
  - JWT validation
  - Request/response logging
- Benefits: Centralized security, monitoring

#### 4. Repository Pattern

- Data Access Layer: Abstract database operations
- Implementation: Prisma ORM or TypeORM repositories
- Benefits: Testability, database independence

#### 5. CQRS (Light Implementation)

- Command: Write operations (mutations)
- Query: Read operations (queries)
- Separation: Different optimizations for reads vs. writes
- Benefits: Performance optimization, scalability

#### 3. COMPONENT DESIGN

#### 3.1 Frontend Architecture

#### 3.1.1 Directory Structure



# 3.1.2 State Management Strategy

#### **Global State (Redux Toolkit):**

typescript		

#### **Server State (React Query):**

```
typescript

// Used for data fetching, caching, synchronization

// Examples: matches, sessions, chat messages

const { data: matches } = useQuery({
   queryKey: ['matches'],
   queryFn: fetchMatches,
   staleTime: 5 * 60 * 1000, // 5 minutes
   cacheTime: 10 * 60 * 1000, // 10 minutes
});
```

#### Local State (useState/useReducer):

- Component-specific UI state
- Form inputs before submission
- Temporary calculations

#### 3.1.3 Key Components

#### **Authentication Components:**

typescript			

```
// LoginForm.tsx
interface LoginFormProps {
  onSuccess: () => void;
}

const LoginForm: React.FC<LoginFormProps> = ({ onSuccess }) => {
  const [credentials, setCredentials] = useState({ email: ", password: " });
  const { mutate: login, isLoading } = useLogin();

const handleSubmit = (e: FormEvent) => {
  e.preventDefault();
  login(credentials, { onSuccess });
};

return (/* form JSX */);
};
```

#### **Real-Time Chat Component:**

```
typescript

// ChatWindow.tsx
const ChatWindow: React.FC<{ conversationId: string }> = ({ conversationId }) => {
  const { messages, sendMessage } = useChat(conversationId);
  const { socket } = useSocket();

useEffect(() => {
    socket.on('new_message', (message) => {
        // Update local state
    });

  return () => socket.off('new_message');
  }, [socket]);

return (/* chat UI */);
};
```

#### **Session Calendar Component:**

```
typescript
```

```
// SessionCalendar.tsx
import FullCalendar from '@fullcalendar/react';

const SessionCalendar: React.FC = () => {
  const { data: sessions } = useSessions();

const events = sessions?.map(session => ({
    id: session.id,
    title: `${session.skill.name} with ${session.partner.name}`,
    start: session.scheduledAt,
    end: new Date(session.scheduledAt.getTime() + session.duration * 60000),
}));

return <FullCalendar events={events} />;
};
```

#### 3.2 Backend Architecture

# 3.2.1 Directory Structure



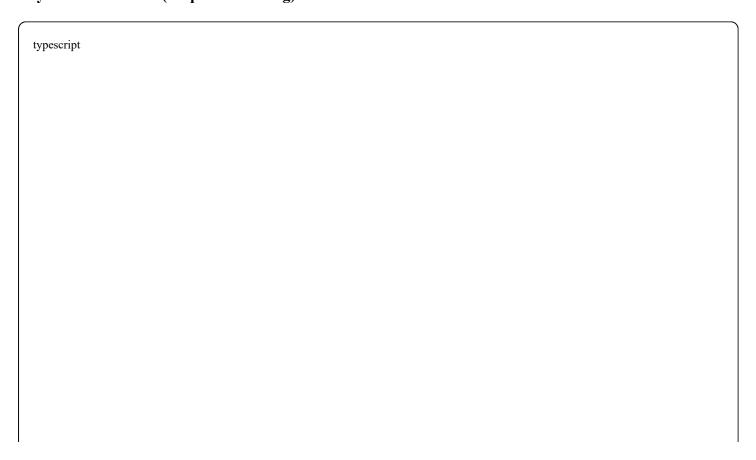
# 3.2.2 Layered Architecture

#### **Layer 1: Routes (API Endpoints)**

typescript			

```
// auth.routes.ts
import { Router } from 'express';
import { AuthController } from './auth.controller';
import { validateRequest } from '../../middleware/validation';
import { registerSchema, loginSchema } from './auth.validation';
const router = Router();
const authController = new AuthController();
router.post('/register',
 validateRequest(registerSchema),
 authController.register
);
router.post('/login',
 validateRequest(loginSchema),
 authController.login
);
router.post('/logout',
 authenticateJWT,
 authController.logout
);
export default router;
```

# **Layer 2: Controllers (Request Handling)**



```
// auth.controller.ts
export class AuthController {
 constructor(private authService: AuthService) {}
 register = async (req: Request, res: Response, next: NextFunction) => {
  try {
   const { email, password, name } = req.body;
   const result = await this.authService.register({ email, password, name });
   res.status(201).json({
     success: true,
     data: result,
   });
  } catch (error) {
   next(error);
 };
 login = async (req: Request, res: Response, next: NextFunction) => {
  try {
   const { email, password } = req.body;
   const result = await this.authService.login(email, password);
   res.cookie('refreshToken', result.refreshToken, {
     httpOnly: true,
     secure: process.env.NODE_ENV ==== 'production',
     maxAge: 30 * 24 * 60 * 60 * 1000, // 30 days
    });
   res.json({
     success: true,
     data: {
      accessToken: result.accessToken,
      user: result.user,
     },
   });
  } catch (error) {
   next(error);
 };
```

**Layer 3: Services (Business Logic)** 

```
// auth.service.ts
import berypt from 'berypt';
import jwt from 'jsonwebtoken';
import { UserRepository } from '.../users/user.repository';
import { EmailService } from '../../shared/services/email.service';
export class AuthService {
 constructor(
  private userRepo: UserRepository,
  private emailService: EmailService
 ) {}
 async register(data: RegisterDTO): Promise<AuthResponse> {
  // Check if user exists
  const existingUser = await this.userRepo.findByEmail(data.email);
  if (existingUser) {
   throw new ConflictError('Email already registered');
  // Hash password
  const hashedPassword = await bcrypt.hash(data.password, 12);
  // Create user
  const user = await this.userRepo.create({
   ...data.
   password: hashedPassword,
   verificationToken: this.generateToken(),
  });
  // Send verification email
  await this.emailService.sendVerificationEmail(
   user.email,
   user.verificationToken
  );
  // Generate JWT tokens
  const accessToken = this.generateAccessToken(user.id);
  const refreshToken = this.generateRefreshToken(user.id);
  return { accessToken, refreshToken, user };
 async login(email: string, password: string): Promise<AuthResponse> {
  // Find user
  const user = await this.userRepo.findByEmail(email);
  if (!user) {
```

```
throw new UnauthorizedError('Invalid credentials');
 // Verify password
 const isValidPassword = await bcrypt.compare(password, user.password);
 if (!isValidPassword) {
  throw new UnauthorizedError('Invalid credentials');
// Check if verified
 if (!user.emailVerified) {
  throw new ForbiddenError('Please verify your email first');
 // Generate tokens
 const accessToken = this.generateAccessToken(user.id);
 const refreshToken = this.generateRefreshToken(user.id);
// Update last login
 await this.userRepo.updateLastLogin(user.id);
 return { accessToken, refreshToken, user };
private generateAccessToken(userId: string): string {
 return jwt.sign(
  { userId, type: 'access' },
  process.env.JWT SECRET!,
  { expiresIn: '7d' }
private generateRefreshToken(userId: string): string {
 return jwt.sign(
  { userId, type: 'refresh' },
  process.env.JWT_REFRESH_SECRET!,
  { expiresIn: '30d' }
 );
```

#### **Layer 4: Repository (Data Access)**

typescript

```
// user.repository.ts
import { PrismaClient } from '@prisma/client';
export class UserRepository {
 constructor(private prisma: PrismaClient) {}
 async findByEmail(email: string) {
  return this.prisma.user.findUnique({
   where: { email },
   select: {
     id: true,
     email: true,
     password: true,
     name: true,
     emailVerified: true,
     profilePhoto: true,
   },
  });
 async create(data: CreateUserDTO) {
  return this.prisma.user.create({
   data: {
     email: data.email,
     password: data.password,
     name: data.name,
     verificationToken: data.verificationToken,
    },
   select: {
     id: true,
     email: true,
     name: true,
     createdAt: true,
   },
  });
 async updateLastLogin(userId: string) {
  return this.prisma.user.update({
   where: { id: userId },
   data: { lastLoginAt: new Date() },
  });
```

# **Socket.io Server Setup:** typescript

```
// websocket/socket.server.ts
import { Server } from 'socket.io';
import { verifySocketAuth } from '../middleware/auth';
import { ChatHandler } from './handlers/chat.handler';
import { NotificationHandler } from './handlers/notification.handler';
export class SocketServer {
 private io: Server;
 constructor(httpServer: any) {
  this.io = new Server(httpServer, {
   cors: {
     origin: process.env.CLIENT_URL,
     credentials: true.
   },
  });
  this.setupMiddleware();
  this.setupHandlers();
 private setupMiddleware() {
  this.io.use(verifySocketAuth);
 private setupHandlers() {
  this.io.on('connection', (socket) => {
   console.log('User connected: ${socket.data.userId}');
   // Join user's personal room
   socket.join('user:${socket.data.userId}');
   // Register handlers
   new ChatHandler(this.io, socket);
   new NotificationHandler(this.io, socket);
   socket.on('disconnect', () => {
     console.log(`User disconnected: ${socket.data.userId}`);
   });
  });
 // Method to emit to specific user from anywhere in app
 emitToUser(userId: string, event: string, data: any) {
  this.io.to('user:${userId}').emit(event, data);
```

<pre>} }</pre>	
Chat Handler:	
typescript	

```
// websocket/handlers/chat.handler.ts
export class ChatHandler {
 constructor(private io: Server, private socket: Socket) {
  this.socket.on('send_message', this.handleSendMessage.bind(this));
  this.socket.on('typing', this.handleTyping.bind(this));
  this.socket.on('read message', this.handleReadMessage.bind(this));
 private async handleSendMessage(data: SendMessageDTO) {
  try {
   const senderId = this.socket.data.userId;
   // Validate users are connected/matched
   const canChat = await this.chatService.canUsersSendMessage(
     senderId,
     data.recipientId
   );
   if (!canChat) {
     this.socket.emit('error', { message: 'Cannot send message to this user' });
     return:
   // Save message to database
   const message = await this.chatService.createMessage({
     senderId,
     recipientId: data.recipientId,
     text: data.text,
     conversationId: data.conversationId,
   });
   // Emit to recipient
   this.io.to('user:${data.recipientId}').emit('new_message', message);
   // Confirm to sender
   this.socket.emit('message sent', { tempId: data.tempId, message });
   // Send push notification if recipient offline
   const isOnline = await this.checkUserOnline(data.recipientId);
   if (!isOnline) {
     await this.notificationService.sendPushNotification(
      data.recipientId,
      'New message',
      `${message.sender.name}: ${message.text.substring(0, 50)}...`
     );
```

```
} catch (error) {
    this.socket.emit('error', { message: 'Failed to send message' });
}

private handleTyping(data: { recipientId: string; isTyping: boolean }) {
    this.io.to('user:${data.recipientId}').emit('user_typing', {
        userId: this.socket.data.userId,
        isTyping: data.isTyping,
    });
}
```

#### 4. DATABASE DESIGN

# 4.1 Entity-Relationship Diagram (ERD)

```
users
id (PK)
email (UNIQUE)
password_hash
name
profile_photo_url
bio
location
timezone
email_verified
phone_verified
last_login_at
created_at
updated_at
      1:N
 user_skills
id (PK)
user_id (FK)
skill_id (FK)
skill_type (ENUM)
 - teach
```

prified			
skills    (PK)	- learn		
skills    (PK)	proficiency (ENUM)		
skills    (PK)	verified		
skills  (PK)  Imme  Integory  Ibeategory  Ibeategory  Ibeategory  Imatches  Integory  Integory  Imatches  Integory	created_at		
skills  (PK)  Imme  Integory  Ibeategory  Ibeategory  Ibeategory  Imatches  Integory  Integory  Imatches  Integory			
skills  (PK)  Imme  Integory  Ibeategory  Ibeategory  Ibeategory  Imatches  Integory  Integory  Imatches  Integory	' I	1	
skills  (PK)  Imme  Integory  Ibeategory  Ibeategory  Ibeategory  Imatches  Integory  Integory  Imatches  Integory			
tegory	19:1		
tegory			
tegory			
ame   tagory   tabcategory   t	skills		
ame   tagory   tabcategory   t			
ame   tagory   tabcategory   t	id (PK)		
abcategory	name		
theategory escription			
matches    (PK)			
matches    PK			
matches    PK			
matches    (PK)	keywords		
l (PK) ser1_id (FK) ser2_id (FK) latch_score latus (ENUM) - suggested - favorited - passed - blocked reated_at  l (PK) loquester_id (FK) scipient_id (FK) cheduled_at laration_minutes	created_at		
l (PK) ser1_id (FK) ser2_id (FK) latch_score latus (ENUM) - suggested - favorited - passed - blocked reated_at  l (PK) loquester_id (FK) scipient_id (FK) cheduled_at laration_minutes			
l (PK) ser1_id (FK) ser2_id (FK) latch_score latus (ENUM) - suggested - favorited - passed - blocked reated_at  l (PK) loquester_id (FK) scipient_id (FK) cheduled_at laration_minutes			
l (PK) ser1_id (FK) ser2_id (FK) latch_score latus (ENUM) - suggested - favorited - passed - blocked reated_at  l (PK) loquester_id (FK) scipient_id (FK) cheduled_at laration_minutes	,		
l (PK) ser1_id (FK) ser2_id (FK) latch_score latus (ENUM) - suggested - favorited - passed - blocked reated_at  l (PK) loquester_id (FK) scipient_id (FK) cheduled_at laration_minutes	matches	1	
ser1_id (FK) ser2_id (FK) ser2_id (FK) ser2_id (FK) substantial (ENUM) substantial (ENUM) substantial (ENUM) substantial (ENUM) substantial (ENUM) substantial (ENUM) sessions	1114101103		
ser1_id (FK) ser2_id (FK) ser2_id (FK) ser2_id (FK) substantial (ENUM) substantial (ENUM) substantial (ENUM) substantial (ENUM) substantial (ENUM) substantial (ENUM) sessions	14 (DE)	I	
ser2_id (FK) latch_score latus (ENUM) - suggested - favorited - passed - blocked leteted_at  sessions    (PK)   cquester_id (FK)       cheduled_at       cheduled_at       cartion_minutes			
attach_score attus (ENUM)  suggested favorited  passed blocked reated_at  sessions  attach_score  attus (ENUM)  suggested  favorited  passed  blocked  reated_at  selection  attach_score  attus (ENUM)  passed  blocked  attus (ENUM)  passed  attus (EN			
atus (ENUM)  suggested  favorited  passed  blocked  reated_at  sessions    I (PK)  cipient_id (FK)    cipient_id (FK)    cheduled_at    curation_minutes	user2_id (FK)		
suggested favorited passed blocked reated_at  sessions  l (PK) requester_id (FK)   reipient_id (FK)   reipient_id (FK)   reipient_id (FK)   reipient_id (FK)   reipient_id (FK)   reipient_id (FK)	match_score		
suggested favorited passed blocked reated_at  sessions  l (PK) requester_id (FK)   reipient_id (FK)   reipient_id (FK)   reipient_id (FK)   reipient_id (FK)   reipient_id (FK)   reipient_id (FK)	status (ENUM)		
favorited   passed   blocked   reated_at    sessions    l (PK)   requester_id (FK)   reipient_id (FK)   clipl_id (FK)   cheduled_at   reation_minutes			
passed   blocked   reated_at			
sessions    (PK)			
sessions    (PK)			
sessions    (PK)			
equester_id (FK)   ccipient_id (FK)   ctil_id (FK)   cheduled_at   curation_minutes	created_at		
equester_id (FK)   ccipient_id (FK)   ctil_id (FK)   cheduled_at   curation_minutes			
equester_id (FK)   ccipient_id (FK)   ctil_id (FK)   cheduled_at   curation_minutes			
equester_id (FK)   ccipient_id (FK)   ctil_id (FK)   cheduled_at   curation_minutes			
equester_id (FK)   ccipient_id (FK)   ctil_id (FK)   cheduled_at   curation_minutes	sessions		
equester_id (FK)   ecipient_id (FK)     still_id (FK)     scheduled_at     suration_minutes	'		
equester_id (FK)   ecipient_id (FK)     still_id (FK)     scheduled_at     suration_minutes	d (BK)		
ccipient_id (FK)			
cill_id (FK)			
cheduled_at     uration_minutes			
uration_minutes	skill_id (FK)		
uration_minutes	scheduled_at		
	- proposed		
confirmed	- confirmed		

- completed			
	_		
ratings	<b>⊣</b>		
communication_rating			
id (PK)	—   —		
sender_id (FK) recipient_id (FK) conversation_id message_text file_url read_at created_at			
credit_transactions	——————————————————————————————————————		
id (PK) user_id (FK) amount balance_after type (ENUM)	I		

- earned			
- spent			
- purchased			
- refunded			
- expired			
- bonus			
related_session_id			
stripe_transaction			
created_at			
notifications	1		
id (PK)	ı		
user_id (FK)			
type			
title			
'			
message			
link			
read_at			
created_at			

# 4.2 Database Schema (Prisma)

prisma	

```
// schema.prisma
datasource db {
 provider = "postgresql"
      = env("DATABASE URL")
generator client {
 provider = "prisma-client-js"
model User {
            String @id @default(uuid())
id
             String @unique
 email
 passwordHash
                  String @map("password_hash")
              String
 name
 profilePhotoUrl String? @map("profile_photo_url")
 bio
            String?
 location
              String?
               String
                      @default("UTC")
 timezone
 emailVerified Boolean @default(false) @map("email_verified")
                Boolean @default(false) @map("phone_verified")
 phoneVerified
 verificationToken String? @map("verification token")
 lastLoginAt
               DateTime? @map("last_login_at")
 createdAt
               DateTime @default(now()) @map("created_at")
 updatedAt
               DateTime @updatedAt @map("updated_at")
// Relations
              UserSkill[]
 skills
 sentMessages
                  Message[]
                                    @relation("SentMessages")
 receivedMessages
                                      @relation("ReceivedMessages")
                    Message[]
 requestedSessions
                                    @relation("RequestedSessions")
                   Session[]
 receivedSessions
                   Session[]
                                    @relation("ReceivedSessions")
 givenRatings
                  Rating[]
                                  @relation("GivenRatings")
 receivedRatings
                                   @relation("ReceivedRatings")
                   Rating[]
 creditTransactions CreditTransaction[]
 notifications
                 Notification[]
 matchesAsUser1
                                     @relation("User1Matches")
                    Match[]
 matchesAsUser2
                    Match[]
                                     @relation("User2Matches")
 (a)(a)map("users")
model Skill {
         String @id @default(uuid())
 name
           String Qunique
```

```
category String
 subcategory String?
 description String?
 keywords String[]
 createdAt DateTime @default(now()) @map("created at")
 // Relations
 userSkills UserSkill[]
 sessions Session[]
 @@index([category])
 @@map("skills")
enum SkillType {
TEACH
LEARN
enum ProficiencyLevel {
 BEGINNER
 INTERMEDIATE
ADVANCED
 EXPERT
model UserSkill {
 id
        String
                    (@id (@default(uuid())
                   @map("user_id")
 userId
          String
 skillId String
                     @map("skill_id")
 skillType SkillType
                        @map("skill_type")
 proficiency ProficiencyLevel
 verified Boolean
                       @default(false)
 createdAt DateTime
                         @default(now()) @map("created_at")
 // Relations
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)
 skill Skill @relation(fields: [skillId], references: [id], onDelete: Cascade)
 @@unique([userId, skillId, skillType])
 @@index([userId])
 @@index([skillId])
 @@map("user_skills")
enum MatchStatus {
 SUGGESTED
```

```
FAVORITED
 PASSED
 BLOCKED
 CONNECTED
model Match {
       String
                @id @default(uuid())
                  @map("user1_id")
 user1Id String
 user2Id String
                  @map("user2 id")
                    @map("match_score")
 matchScore Float
 status MatchStatus @default(SUGGESTED)
 createdAt DateTime @default(now()) @map("created_at")
 // Relations
 user1 User @relation("User1Matches", fields: [user1Id], references: [id], onDelete: Cascade)
 user2 User @relation("User2Matches", fields: [user2Id], references: [id], onDelete: Cascade)
 @@unique([user1Id, user2Id])
 @@index([user1Id])
 @@index([user2Id])
 @@map("matches")
enum SessionStatus {
 PROPOSED
 CONFIRMED
 COMPLETED
 CANCELLED
model Session {
          String
                    @id @default(uuid())
 requesterId String
                        @map("requester id")
 recipientId
             String
                       @map("recipient id")
 skillId
           String
                     @map("skill_id")
 scheduledAt
              DateTime
                           @map("scheduled at")
 durationMinutes Int
                         @map("duration minutes")
           SessionStatus @default(PROPOSED)
 status
 videoLink
              String?
                        @map("video_link")
 creditsCost Int
                      @map("credits_cost")
 agenda
             String?
 createdAt
             DateTime
                          @default(now()) @map("created_at")
 updatedAt
             DateTime
                          @updatedAt @map("updated at")
 // Relations
 requester User @relation("RequestedSessions", fields: [requesterId], references: [id])
```

```
recipient User @relation("ReceivedSessions", fields: [recipientId], references: [id])
 skill Skill @relation(fields: [skillId], references: [id])
 ratings Rating[]
 @@index([requesterId])
 @@index([recipientId])
 @@index([scheduledAt])
 @aindex([status])
 @@map("sessions")
model Rating {
 id
               String @id @default(uuid())
 sessionId
                 String @map("session id")
                String @map("rater_id")
 raterId
 rateeId
                String @map("ratee id")
 overallRating
                   Float @map("overall_rating")
 knowledgeRating
                   Float @map("knowledge rating")
 communicationRating Float @map("communication rating")
 professionalismRating Float @map("professionalism_rating")
 reviewText
                   String? @map("review_text")
 tags
               String[]
 isPublic
                 Boolean @default(true) @map("is public")
 createdAt
                  DateTime @default(now()) @map("created_at")
 // Relations
 session Session @relation(fields: [sessionId], references: [id], onDelete: Cascade)
 rater User @relation("GivenRatings", fields: [raterId], references: [id])
 ratee User @relation("ReceivedRatings", fields: [rateeId], references: [id])
 @@unique([sessionId, raterId])
 @@index([rateeId])
 @@map("ratings")
model Message {
          String @id @default(uuid())
 id
 senderId
             String @map("sender id")
 recipientId String @map("recipient id")
 conversationId String @map("conversation_id")
 messageText String @map("message text")
 fileUrl
           String? @map("file_url")
 readAt
            DateTime? @map("read at")
 createdAt DateTime @default(now()) @map("created at")
 // Relations
 sender User @relation("SentMessages", fields: [senderId], references: [id])
```

```
recipient User @relation("ReceivedMessages", fields: [recipientId], references: [id])
 @@index([conversationId])
 @@index([senderId])
 @@index([recipientId])
 @@map("messages")
enum TransactionType {
 EARNED
 SPENT
 PURCHASED
 REFUNDED
 EXPIRED
 BONUS
 STARTER
model CreditTransaction {
 id
             String
                        @id @default(uuid())
 userId
                          @map("user_id")
              String
               Int
 amount
 balanceAfter
                 Int
                           @map("balance_after")
 type
              TransactionType
 relatedSessionId String?
                              @map("related_session_id")
 stripeTransactionId String?
                               @map("stripe_transaction_id")
 description
                String?
 createdAt
                DateTime
                              @default(now()) @map("created at")
 // Relations
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)
 @@index([userId])
 @@index([createdAt])
 @@map("credit transactions")
model Notification {
       String @id @default(uuid())
 id
 userId String @map("user_id")
 type
        String
 title
      String
 message String
 link
        String?
 readAt DateTime? @map("read_at")
 createdAt DateTime @default(now()) @map("created_at")
```

```
// Relations
user User @relation(fields: [userId], references: [id], onDelete: Cascade)

@@index([userId])
@@index([createdAt])
@@map("notifications")
}
```

# 4.3 Database Indexing Strategy

#### **Performance-Critical Indexes:**

```
sql
-- Users table
CREATE INDEX idx users email ON users(email);
CREATE INDEX idx users created at ON users(created at);
-- User Skills table
CREATE INDEX idx user skills user id ON user skills(user id);
CREATE INDEX idx user skills skill id ON user skills(skill id);
CREATE INDEX idx_user_skills_skill_type ON user_skills(skill_type);
-- Matches table
CREATE INDEX idx matches user1 user2 ON matches(user1 id, user2 id);
CREATE INDEX idx matches status ON matches(status);
CREATE INDEX idx matches score ON matches(match score DESC);
-- Sessions table
CREATE INDEX idx sessions requester ON sessions(requester id, scheduled at);
CREATE INDEX idx sessions recipient ON sessions(recipient id, scheduled at);
CREATE INDEX idx sessions status ON sessions(status);
CREATE INDEX idx sessions scheduled at ON sessions(scheduled at);
-- Messages table
CREATE INDEX idx messages conversation ON messages(conversation id, created at DESC);
CREATE INDEX idx_messages_unread ON messages(recipient_id, read_at) WHERE read_at IS NULL;
-- Ratings table
CREATE INDEX idx ratings ratee ON ratings(ratee id, created at DESC);
-- Full-text search indexes
CREATE INDEX idx skills name trgm ON skills USING gin(name gin trgm ops);
CREATE INDEX idx users bio fts ON users USING gin(to tsvector('english', bio));
```

# 4.4 Data Migration Strategy

#### **Version Control for Schema:**

- Use Prisma Migrate for schema versioning
- All migrations tracked in (prisma/migrations/)
- Never modify migration files after deployment

#### **Migration Process:**

```
# Development
npm run prisma:migrate:dev

# Production (with backup)
npm run db:backup
npm run prisma:migrate:deploy
npm run db:verify
```

#### 5. API DESIGN

# 5.1 REST API Specification

Base URL: (https://api.skillsync.com/v1)

Authentication: Bearer token (JWT) in (Authorization) header

#### **5.1.1 Authentication Endpoints**

POST /auth/login
POST /auth/logout
POST /auth/refresh
POST /auth/verify-email
POST /auth/forgot-password
POST /auth/reset-password
POST /auth/oauth/google
POST /auth/oauth/linkedin

#### **Example: Register User**

http

```
POST /auth/register
Content-Type: application/json

{
    "email": "user@example.com",
    "password": "SecurePass123!",
    "name": "John Doe"
}

Response 201:
{
    "success": true,
    "data": {
        "accessToken": "eyJhbGciOiJIUzIINiIs...",
        "user": {
        "id": "uuid",
        "email": "user@example.com",
        "name": "John Doe",
        "emailVerified": false
    }
}

}
```

#### **5.1.2 User Profile Endpoints**

```
GET /users/me
PATCH /users/me
DELETE /users/me
GET /users/:id
POST /users/me/photo
GET /users/me/stats
```

# **Example: Update Profile**



```
PATCH /users/me
Authorization: Bearer {token}
Content-Type: application/json

{
    "bio": "UX Designer learning Python",
    "location": "San Francisco, CA",
    "timezone": "America/Los_Angeles"
}

Response 200:
{
    "success": true,
    "data": {
        "id": "uuid",
        "bio": "UX Designer learning Python",
        "location": "San Francisco, CA",
        "profileCompleteness": 75
}
}
```

# 5.1.3 Skills Endpoints

```
GET /skills/search?q={query}

POST /skills/request

GET /users/me/skills

POST /users/me/skills

DELETE /users/me/skills/:skillId
```

# **Example: Add Skill**



```
POST /users/me/skills
Authorization: Bearer {token}
Content-Type: application/json

{
    "skillId": "skill-uuid",
    "skiilType": "TEACH",
    "proficiency": "INTERMEDIATE"
}

Response 201:
    {
        "success": true,
        "data": {
        "id": "user-skill-uuid",
        "skill": {
        "name": "Python",
        "category": "Technology"
        },
        "proficiency": "INTERMEDIATE"
}

}
```

#### **5.1.4 Matching Endpoints**

```
GET /matches

GET /matches/suggestions

POST /matches/:matchId/favorite

POST /matches/:matchId/pass

POST /matches/:matchId/block

POST /matches/:matchId/connect
```

# **Example: Get Match Suggestions**

http			

```
GET /matches/suggestions?limit=20
Authorization: Bearer {token}
Response 200:
 "success": true,
 "data": {
  "matches": [
     "id": "match-uuid",
     "user": {
     "id": "user-uuid",
      "name": "Jane Smith",
      "profilePhoto": "https://cdn.../photo.jpg",
      "rating": 4.8,
      "sessionsCompleted": 45
     "matchScore": 92,
     "explanation": "92% match because: You teach Python ↔ They want to learn Python...",
     "matchedSkills": ["Python", "Web Development"],
     "availabilityOverlap": 15
  ],
  "total": 20
```

#### **5.1.5 Session Endpoints**

```
GET /sessions
GET /sessions/:id
POST /sessions
PATCH /sessions/:id
DELETE /sessions/:id
POST /sessions/:id/accept
POST /sessions/:id/decline
POST /sessions/:id/reschedule
POST /sessions/:id/cancel
GET /sessions/upcoming
GET /sessions/past
```

# **Example: Propose Session**

```
POST /sessions
Authorization: Bearer {token}
Content-Type: application/json
 "recipientId": "user-uuid",
 "skillId": "skill-uuid",
 "proposedTimes": [
  "2025-11-01T14:00:00Z",
  "2025-11-02T16:00:00Z"
 ],
 "durationMinutes": 60,
 "agenda": "Introduction to Python basics"
Response 201:
 "success": true,
 "data": {
  "id": "session-uuid",
  "status": "PROPOSED",
  "creditsCost": 10,
  "createdAt": "2025-10-25T12:00:00Z"
```

#### 5.1.6 Chat Endpoints

```
GET /conversations/:id/messages

POST /conversations/:id/messages

POST /conversations/:id/files

GET /conversations/:id/files

DELETE /messages/:id

PATCH /messages/:id/read
```

#### **5.1.7 Rating Endpoints**

```
POST /sessions/:sessionId/ratings
GET /ratings/pending
GET /users/:userId/ratings
```

#### 5.1.8 Credit Endpoints

GET	/credits/balance
GET	/credits/transactions
POST	/credits/purchase
POST	/credits/transfer (admin only)

### **5.2 WebSocket Events**

#### Client $\rightarrow$ Server Events:

```
send_message
typing
read_message
join_conversation
leave_conversation
```

## **Server** → **Client Events:**

```
new_message
user_typing
message_delivered
message_read
notification
match_update
session_reminder
```

Example Event:
javascript

```
// Client sends
socket.emit('send_message', {
 conversationId: 'conv-uuid',
 recipientId: 'user-uuid',
 text: 'Hello!',
 tempId: 'temp-123' // For optimistic UI
});
// Server responds
socket.emit('message_sent', {
 tempId: 'temp-123',
 message: {
  id: 'msg-uuid',
  text: 'Hello!',
  createdAt: '2025-10-25T12:00:00Z'
});
// Recipient receives
socket.emit('new_message', {
 message: { /* full message object */ }
});
```

## 5.3 Error Handling

### **Standard Error Response:**

```
| "success": false,
| "error": {
| "code": "VALIDATION_ERROR",
| "message": "Invalid input data",
| "details": [
| {
| "field": "email",
| "message": "Invalid email format"
| }
| ]
| }
| }
```

#### **HTTP Status Codes:**

• (200) OK - Success

- (201) Created Resource created
- (400) Bad Request Invalid input
- (401) Unauthorized Missing/invalid auth
- (403) Forbidden Insufficient permissions
- (404) Not Found Resource not found
- (409) Conflict Duplicate resource
- (422) Unprocessable Entity Validation failed
- (429) Too Many Requests Rate limit exceeded
- (500) Internal Server Error Server error

### 6. SECURITY ARCHITECTURE

#### 6.1 Authentication & Authorization

#### **JWT Token Structure:**

#### **Token Storage:**

- Access Token: Client-side (localStorage or memory)
- Refresh Token: HTTP-only cookie (secure, sameSite)

### **Role-Based Access Control (RBAC):**

typescript	

```
enum UserRole {
 USER = 'user',
ADMIN = 'admin',
 SUPER_ADMIN = 'super_admin'
// Middleware
const authorize = (roles: UserRole[]) => {
 return (req: Request, res: Response, next: NextFunction) => {
  if (!roles.includes(req.user.role)) {
   return res.status(403).json({ error: 'Forbidden' });
  next();
 };
};
// Usage
router.delete('/users/:id',
 authenticateJWT,
 authorize([UserRole.ADMIN, UserRole.SUPER_ADMIN]),
 userController.deleteUser
);
```

#### **6.2 Data Protection**

### **Encryption:**

• In Transit: TLS 1.3 (HTTPS)

• At Rest: AES-256 for sensitive data

• Passwords: bcrypt (cost factor 12)

• Tokens: Signed with HS256/RS256

### **Sensitive Data Handling:**

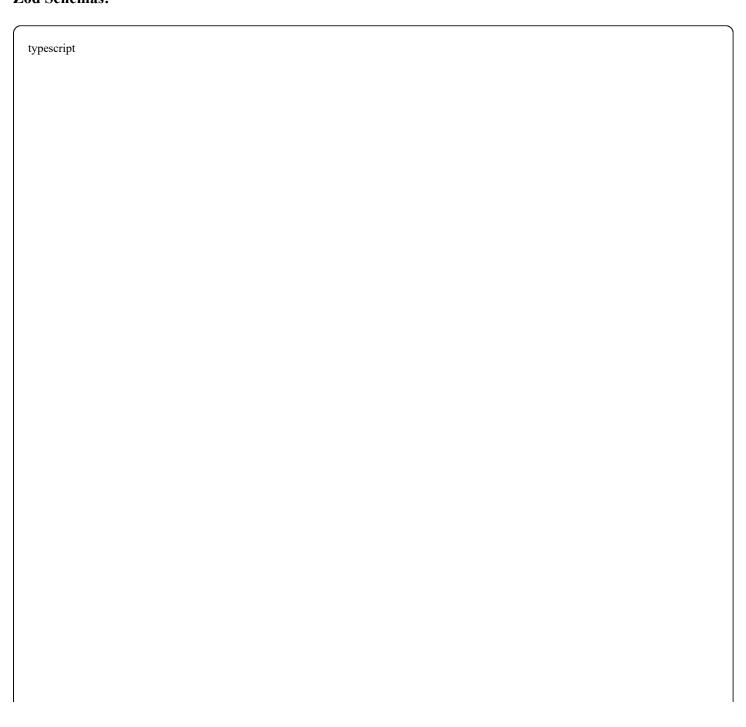
typescript		

```
// Never log passwords, tokens, or PII
logger.info('User login', {
    userId: user.id,
    // ** password: user.password
    // ** email: user.email (use hashed identifier)
});

// Encrypt sensitive fields before storage
const encryptPII = (data: string): string => {
    const cipher = crypto.createCipheriv('aes-256-gcm', key, iv);
    return cipher.update(data, 'utf8', 'hex') + cipher.final('hex');
};
```

## 6.3 Input Validation & Sanitization

### **Zod Schemas:**



```
import { z } from 'zod';
export const registerSchema = z.object({
 email: z.string().email('Invalid email format'),
 password: z.string()
  .min(8, 'Password must be at least 8 characters')
  .regex(/[A-Z]/, 'Must contain uppercase letter')
  .regex(/[a-z]/, 'Must contain lowercase letter')
  .regex(/[0-9]/, 'Must contain number')
  .regex(/[^A-Za-z0-9]/, 'Must contain special character'),
 name: z.string()
  .min(2, 'Name too short')
  .max(50, 'Name too long')
  .regex(/^[a-zA-Z\s]+$/, 'Name can only contain letters'),
});
// Usage
const validateRequest = (schema: z.Schema) => {
 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({
      success: false,
      error: {
       code: 'VALIDATION ERROR',
       details: error.errors
     });
   next(error);
 };
};
```

#### **SQL Injection Prevention:**

- Use Prisma ORM (parameterized queries)
- Never concatenate user input into queries

#### **XSS Prevention:**

• Sanitize HTML input (DOMPurify on client)

- Content-Security-Policy headers
- Escape output in templates

## **6.4 Rate Limiting**

```
typescript
import rateLimit from 'express-rate-limit';
// General API rate limit
const apiLimiter = rateLimit({
 windowMs: 15 * 60 * 1000, // 15 minutes
 max: 100, // 100 requests per window
 message: 'Too many requests, please try again later',
 standardHeaders: true,
 legacyHeaders: false,
});
// Stricter limit for auth endpoints
const authLimiter = rateLimit({
 windowMs: 15 * 60 * 1000,
 max: 5, // 5 attempts per 15 minutes
 skipSuccessfulRequests: true,
});
app.use('/api', apiLimiter);
app.use('/api/auth/login', authLimiter);
```

## 6.5 OWASP Top 10 Mitigation

Vulnerability	Mitigation
A01: Broken Access Control	RBAC, JWT validation, ownership checks
A02: Cryptographic Failures	TLS 1.3, bcrypt, AES-256, secure key storage
A03: Injection	Prisma ORM, input validation, sanitization
A04: Insecure Design	Threat modeling, security reviews
A05: Security Misconfiguration	Helmet.js, secure defaults, no debug in prod
A06: Vulnerable Components	Dependency scanning (Snyk), regular updates
A07: Auth Failures	Strong passwords, MFA, rate limiting
A08: Data Integrity Failures	HTTPS, CORS, CSP headers
A09: Logging Failures	Centralized logging, no PII in logs
A10: SSRF	URL validation, allowlist for external requests
4	•

## 6.6 GDPR/CCPA Compliance

### **Data Subject Rights:**

```
typescript

// Right to Access

GET /users/me/data-export

// Returns all user data in JSON format

// Right to Deletion

DELETE /users/me

// Soft delete, anonymize after 30 days

// Right to Rectification

PATCH /users/me

// Users can update their own data

// Right to Data Portability

GET /users/me/data-export?format=json

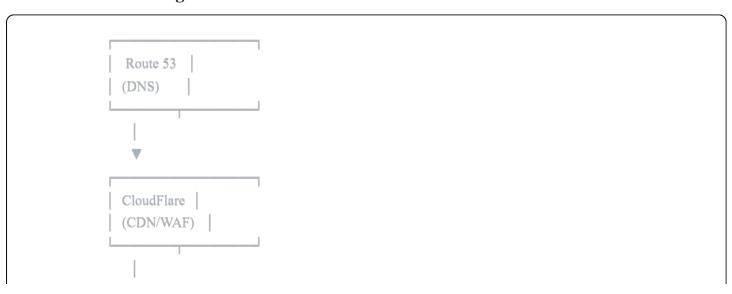
// Exportable in machine-readable format
```

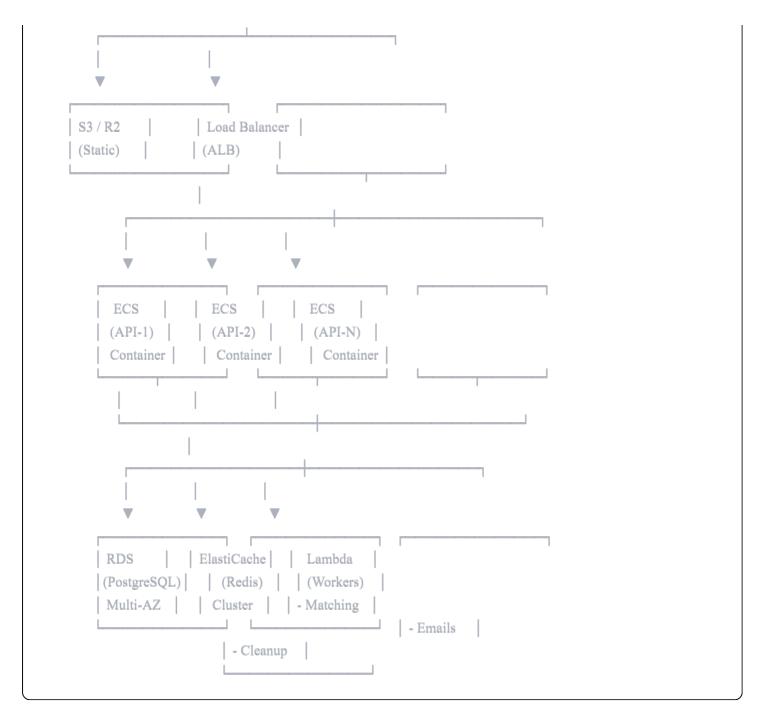
### Privacy by Design:

- Minimal data collection
- Explicit consent for data processing
- Clear privacy policy
- Data retention policies (delete old data)
- Anonymization of analytics data

### 7. DEPLOYMENT ARCHITECTURE

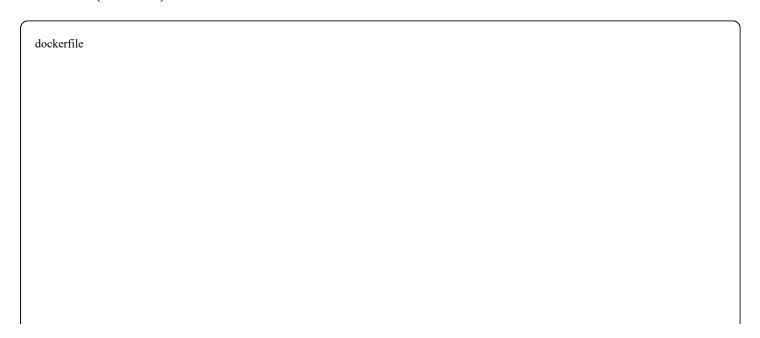
## 7.1 Infrastructure Diagram





# 7.2 Container Strategy (Docker)

## Dockerfile (Backend):



```
# Build stage
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY..
RUN npm run build
RUN npm prune --production
# Production stage
FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package.json ./
EXPOSE 3000
CMD ["node", "dist/server.js"]
```

## **Docker Compose (Development):**

yaml			

```
version: '3.8'
services:
 api:
  build: ./backend
  ports:
   - "3000:3000"
  environment:
   - DATABASE_URL=postgresql://user:pass@db:5432/skillsync
   - REDIS_URL=redis://redis:6379
  depends_on:
   - db
   - redis
 db:
  image: postgres:15-alpine
  environment:
   POSTGRES_DB: skillsync
   POSTGRES_USER: user
   POSTGRES_PASSWORD: pass
  volumes:
   - postgres_data:/var/lib/postgresql/data
 redis:
  image: redis:7-alpine
  ports:
   - "6379:6379"
 frontend:
  build: ./frontend
  ports:
   - "5173:5173"
  volumes:
   - ./frontend:/app
   - /app/node_modules
volumes:
 postgres_data:
```

## 7.3 CI/CD Pipeline (GitHub Actions)

yaml			

```
#.github/workflows/deploy.yml
name: Deploy to Production
on:
 push:
  branches: [main]
jobs:
 test:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v3
   - uses: actions/setup-node@v3
    with:
      node-version: '20'
   - run: npm ci
   - run: npm run test
   - run: npm run lint
 build-and-push:
  needs: test
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v3
   - name: Configure AWS credentials
     uses: aws-actions/configure-aws-credentials@v2
     with:
      aws-access-key-id: ${{ secrets.AWS ACCESS KEY ID }}
      aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      aws-region: us-east-1
   - name: Login to Amazon ECR
     id: login-ecr
     uses: aws-actions/amazon-ecr-login@v1
   - name: Build and push Docker image
     env:
      ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
      IMAGE_TAG: ${{ github.sha }}
     run:
      docker build -t $ECR_REGISTRY/skillsync-api:$IMAGE_TAG.
      docker push $ECR_REGISTRY/skillsync-api:$IMAGE_TAG
 deploy:
  needs: build-and-push
  runs-on: ubuntu-latest
```

```
steps:
- name: Deploy to ECS
run: |
aws ecs update-service \
--cluster skillsync-cluster \
--service skillsync-api \
--force-new-deployment
```

### 7.4 Environment Configuration

```
bash
#.env.production
NODE ENV=production
PORT=3000
# Database
DATABASE_URL=postgresql://user:pass@rds-endpoint:5432/skillsync
DATABASE POOL MIN=5
DATABASE_POOL_MAX=20
# Redis
REDIS URL=redis://elasticache-endpoint:6379
REDIS_TLS=true
#JWT
JWT_SECRET=<generated-secret>
JWT REFRESH SECRET=<generated-secret>
JWT EXPIRY=7d
JWT REFRESH EXPIRY=30d
#AWS
AWS REGION=us-east-1
AWS S3 BUCKET=skillsync-uploads
AWS CLOUDFRONT URL=https://cdn.skillsync.com
# External Services
ZOOM_API_KEY=<key>
STRIPE_SECRET_KEY=<key>
SENDGRID_API_KEY=<key>
GOOGLE_CLIENT_ID=<id>
GOOGLE_CLIENT_SECRET=<secret>
# Monitoring
DATADOG_API_KEY=<key>
SENTRY DSN=<dsn>
```

# 8. PERFORMANCE CONSIDERATIONS

# 8.1 Caching Strategy

ypescript			

```
// L1: Response caching (short-lived)
const getCachedMatches = async (userId: string) => {
 const cacheKey = 'matches:${userId}';
 const cached = await redis.get(cacheKey);
 if (cached) return JSON.parse(cached);
 const matches = await matchingService.getMatches(userId);
 await redis.setex(cacheKey, 300, JSON.stringify(matches)); // 5 min TTL
 return matches;
};
// L2: Database query results (medium-lived)
const getUserProfile = async (userId: string) => {
 const cacheKey = 'user:${userId}';
 const cached = await redis.get(cacheKey);
 if (cached) return JSON.parse(cached);
 const user = await userRepo.findById(userId);
 await redis.setex(cacheKey, 3600, JSON.stringify(user)); // I hour TTL
 return user;
};
// L3: Computed data (long-lived)
const getSkillTaxonomy = async () => {
 const cacheKey = 'skills:taxonomy';
 const cached = await redis.get(cacheKey);
 if (cached) return JSON.parse(cached);
 const skills = await skillRepo.getAllWithHierarchy();
 await redis.setex(cacheKey, 86400, JSON.stringify(skills)); // 24 hours TTL
 return skills;
};
```

#### **Cache Invalidation:**

typescript

```
// Invalidate on user profile update
await redis.del(`user:${userId}`);
await redis.del(`matches:${userId}`); // Dependent data
// Pattern-based invalidation
await redis.keys('sessions:*').then(keys => redis.del(...keys));
```

## 8.2 Database Optimization

### **Query Optimization:**

```
typescript
// X N+1 Query Problem
const sessions = await prisma.session.findMany();
for (const session of sessions) {
 const user = await prisma.user.findUnique({ where: { id: session.requesterId } });
// Z Eager Loading
const sessions = await prisma.session.findMany({
 include: {
  requester: true,
  recipient: true,
  skill: true.
 },
});
// Select only needed fields
const users = await prisma.user.findMany({
 select: {
  id: true,
  name: true,
  profilePhoto: true,
  // Don't select password, email, etc.
 },
});
```

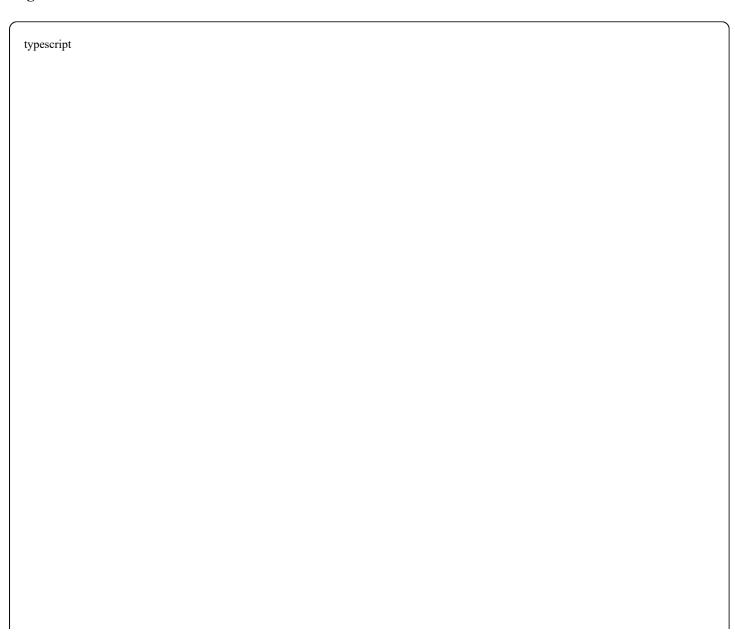
#### **Connection Pooling:**

typescript			

```
// Prisma connection pool
const prisma = new PrismaClient({
    datasources: {
        db: {
            url: process.env.DATABASE_URL,
        },
        // Connection pool configuration
      pool: {
        min: 5,
        max: 20,
        acquireTimeoutMillis: 30000,
        idleTimeoutMillis: 60000,
      },
    });
```

## **8.3** API Performance

## Pagination:



```
// Cursor-based pagination (better for large datasets)
const getMessages = async (conversationId: string, cursor?: string, limit = 50) => {
 const messages = await prisma.message.findMany({
  where: { conversationId },
  take: limit + 1, // Fetch one extra to check if there's more
  cursor: cursor? { id: cursor } : undefined,
  orderBy: { createdAt: 'desc' },
 });
 const hasMore = messages.length > limit;
 const results = hasMore ? messages.slice(0, -1) : messages;
 const nextCursor = hasMore ? results[results.length - 1].id : null;
 return { messages: results, nextCursor, hasMore };
};
// Offset pagination (simpler, for small datasets)
const getUsers = async (page = 1, limit = 20) => {
 const skip = (page - 1) * limit;
 const [users, total] = await Promise.all([
  prisma.user.findMany({ skip, take: limit }),
  prisma.user.count(),
 ]);
 return {
  users,
  pagination: {
   page,
   limit,
   total,
   totalPages: Math.ceil(total / limit),
  },
 };
};
```

#### **Response Compression:**

typescript			

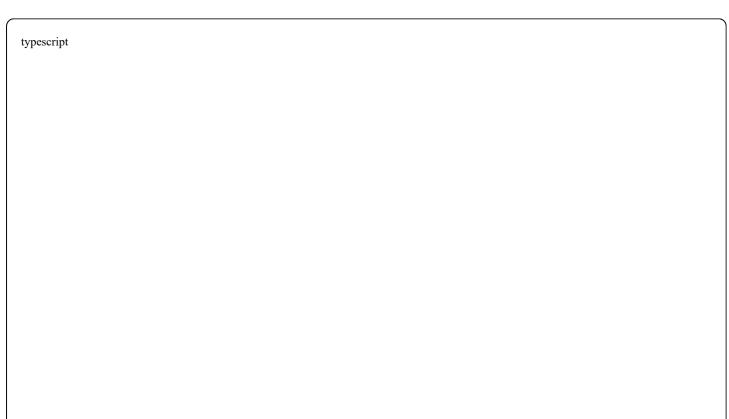
```
import compression from 'compression';

app.use(compression({
    filter: (req, res) => {
      if (req.headers['x-no-compression']) return false;
      return compression.filter(req, res);
    },
    threshold: 1024, // Only compress responses > 1KB
}));
```

### **Request Batching:**

### **8.4 Frontend Performance**

### **Code Splitting:**



### **Image Optimization:**

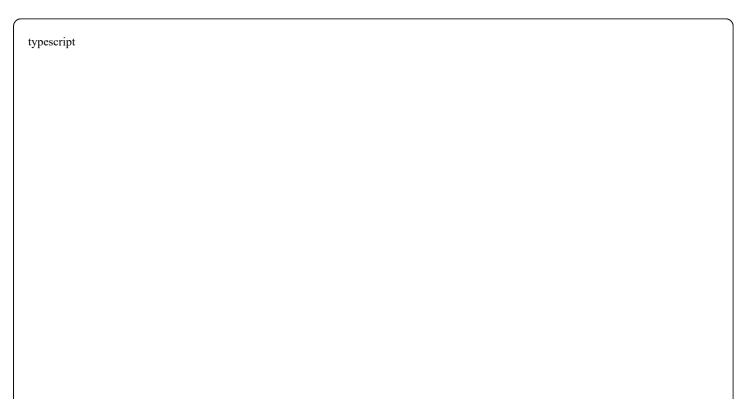
```
typescript
// Next.js Image component (if using Next.js)
import Image from 'next/image';
<Image
 src={user.profilePhoto}
 alt={user.name}
 width=\{200\}
 height=\{200\}
 loading="lazy"
 placeholder="blur"
/>
// Or with standard img + CDN
<img
 src={`${CDN_URL}/${user.profilePhoto}?w=200&h=200&q=80`}
 alt={user.name}
 loading="lazy"
/>
```

#### **Memoization:**

```
import { useMemo, useCallback } from 'react';
const MatchList = ({ matches }) => {
 // Expensive computation
 const sortedMatches = useMemo(() => {
  return matches.sort((a, b) => b.matchScore - a.matchScore);
 }, [matches]);
 // Stable function reference
 const handleConnect = useCallback((matchId) => {
  connectToMatch(matchId);
 }, []);
 return (
  <div>
   {sortedMatches.map(match => (
    <MatchCard
     key={match.id}
     match={match}
     onConnect={handleConnect}
    />
   ))}
  </div>
 );
};
```

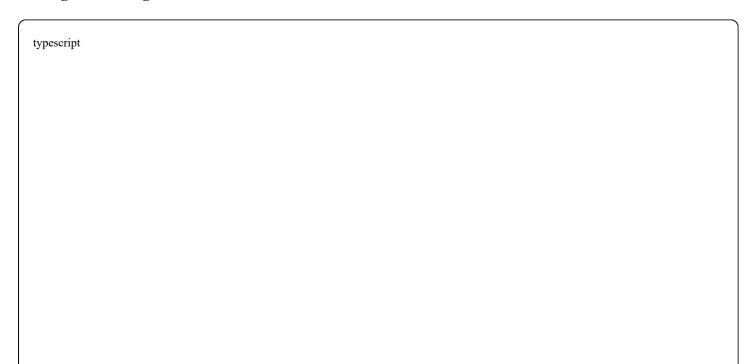
## 8.5 WebSocket Optimization

### **Connection Management:**



```
// Limit connections per user
const MAX_CONNECTIONS_PER_USER = 5;
const userConnections = new Map<string, Set<string>>();
io.on('connection', (socket) => {
 const userId = socket.data.userId;
 if (!userConnections.has(userId)) {
  userConnections.set(userId, new Set());
 const connections = userConnections.get(userId)!;
 if (connections.size >= MAX_CONNECTIONS_PER_USER) {
  socket.emit('error', { message: 'Too many connections' });
  socket.disconnect(true);
  return;
 connections.add(socket.id);
 socket.on('disconnect', () => {
  connections.delete(socket.id);
  if (connections.size === 0) {
   userConnections.delete(userId);
  }
 });
});
```

### **Message Throttling:**



```
// Prevent spam
const messageRateLimiter = new Map<string, number[]>();

socket.on('send_message', (data) => {
    const userId = socket.data.userId;
    const now = Date.now();
    const userMessages = messageRateLimiter.get(userId) || [];

// Keep only messages from last minute
    const recentMessages = userMessages.filter(time => now - time < 60000);

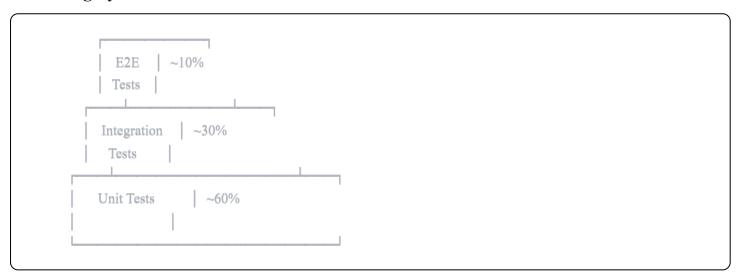
if (recentMessages.length >= 10) {
    socket.emit('error', { message: 'Slow down! Too many messages' });
    return;
}

recentMessages.push(now);
messageRateLimiter.set(userId, recentMessages);

// Process message...
});
```

### 9. TESTING STRATEGY

## 9.1 Testing Pyramid



## 9.2 Unit Testing

### **Backend Unit Tests (Jest):**

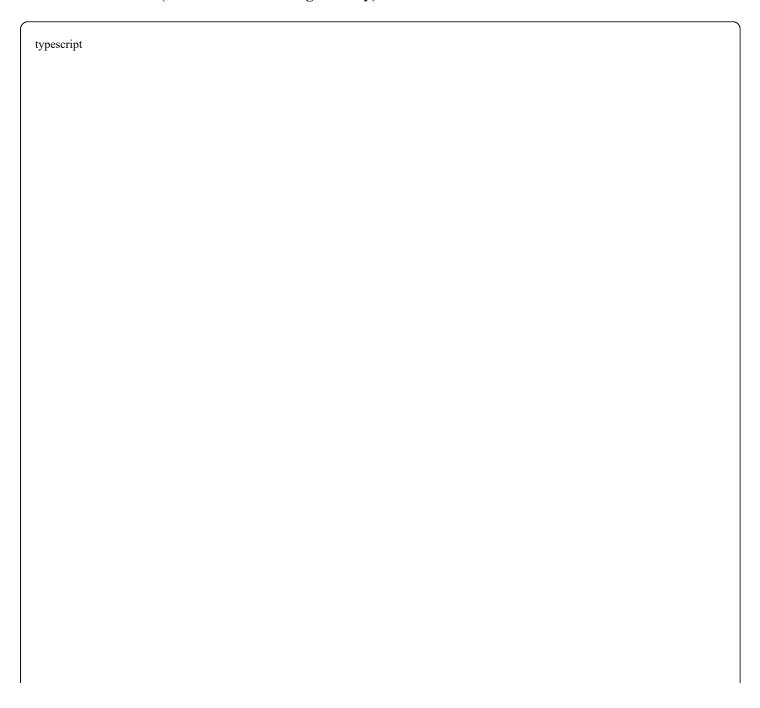
```
typescript
```

```
// auth.service.test.ts
import { AuthService } from './auth.service';
import { UserRepository } from '../users/user.repository';
import { EmailService } from '../../shared/services/email.service';
describe('AuthService', () => {
 let authService: AuthService:
 let userRepo: jest.Mocked<UserRepository>;
 let emailService: jest.Mocked<EmailService>;
 beforeEach(() \Rightarrow \{
  userRepo = {
   findByEmail: jest.fn(),
   create: jest.fn(),
  } as any;
  emailService = {
   sendVerificationEmail: jest.fn(),
  } as any;
  authService = new AuthService(userRepo, emailService);
 });
 describe('register', () => {
  it('should create user and send verification email', async () => {
   userRepo.findByEmail.mockResolvedValue(null);
   userRepo.create.mockResolvedValue({
    id: '123',
     email: 'test@example.com',
     name: 'Test User',
   } as any);
   const result = await authService.register({
     email: 'test@example.com',
     password: 'SecurePass123!',
     name: 'Test User',
   });
   expect(userRepo.create).toHaveBeenCalledWith(
     expect.objectContaining({
      email: 'test@example.com',
      name: 'Test User',
    })
   );
   expect(emailService.sendVerificationEmail).toHaveBeenCalled();
   expect(result.user.id).toBe('123');
```

```
it('should throw error if email already exists', async () => {
  userRepo.findByEmail.mockResolvedValue({ id: '123' } as any);

await expect(
  authService.register({
    email: 'test@example.com',
    password: 'SecurePass123!',
    name: 'Test User',
  })
  ).rejects.toThrow('Email already registered');
  });
});
});
```

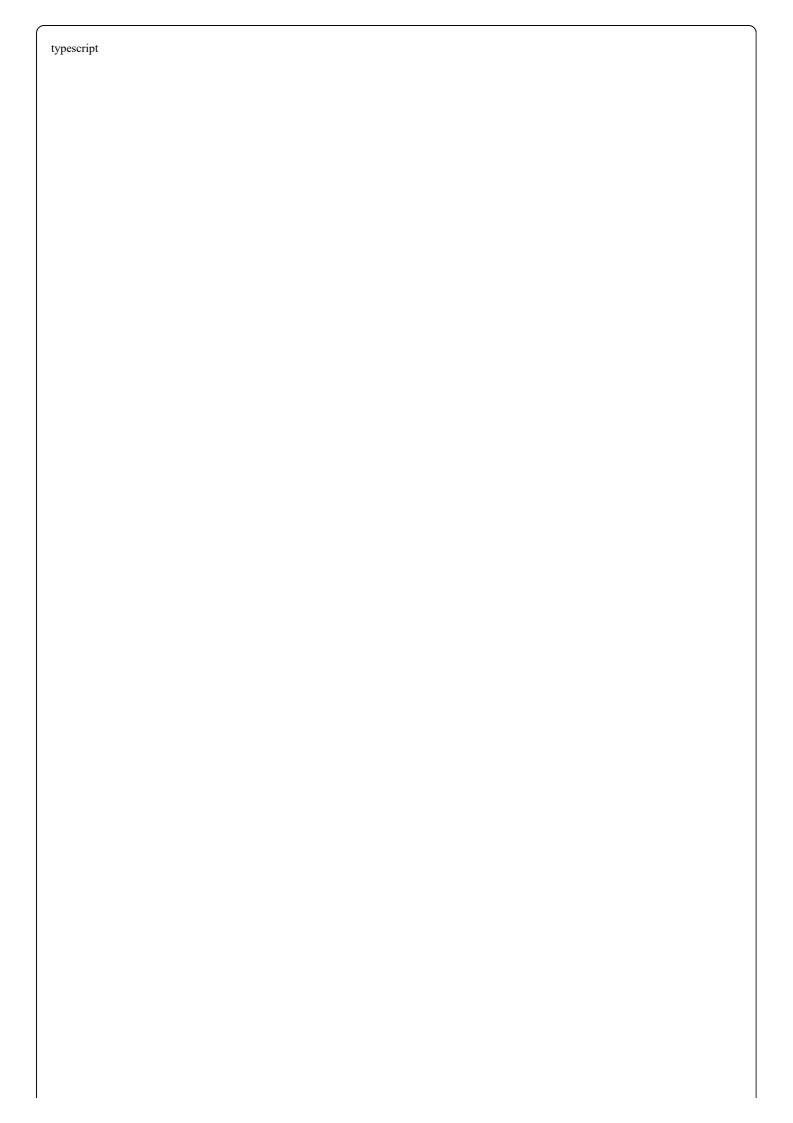
## **Frontend Unit Tests (Vitest + React Testing Library):**



```
// LoginForm.test.tsx
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import { LoginForm } from './LoginForm';
import { useLogin } from '../hooks/useLogin';
jest.mock('../hooks/useLogin');
describe('LoginForm', () => {
 it('should render login form', () => {
  render(<LoginForm onSuccess={jest.fn()} />);
  expect(screen.getByLabelText('Email')).toBeInTheDocument();
  expect(screen.getByLabelText('Password')).toBeInTheDocument();
  expect(screen.getByRole('button', { name: 'Log In' })).toBeInTheDocument();
 });
 it('should call login mutation on submit', async () => {
  const mockLogin = jest.fn();
  (useLogin as jest.Mock).mockReturnValue({
   mutate: mockLogin,
   isLoading: false,
  });
  render(<LoginForm onSuccess={jest.fn()} />);
  fireEvent.change(screen.getByLabelText('Email'), {
   target: { value: 'test@example.com' },
  });
  fireEvent.change(screen.getByLabelText('Password'), {
   target: { value: 'password123' },
  });
  fireEvent.click(screen.getByRole('button', { name: 'Log In' }));
  await waitFor(() \Rightarrow {
   expect(mockLogin).toHaveBeenCalledWith(
     { email: 'test@example.com', password: 'password123' },
     expect.any(Object)
   );
  });
 });
});
```

## 9.3 Integration Testing

#### **API Integration Tests (Supertest):**



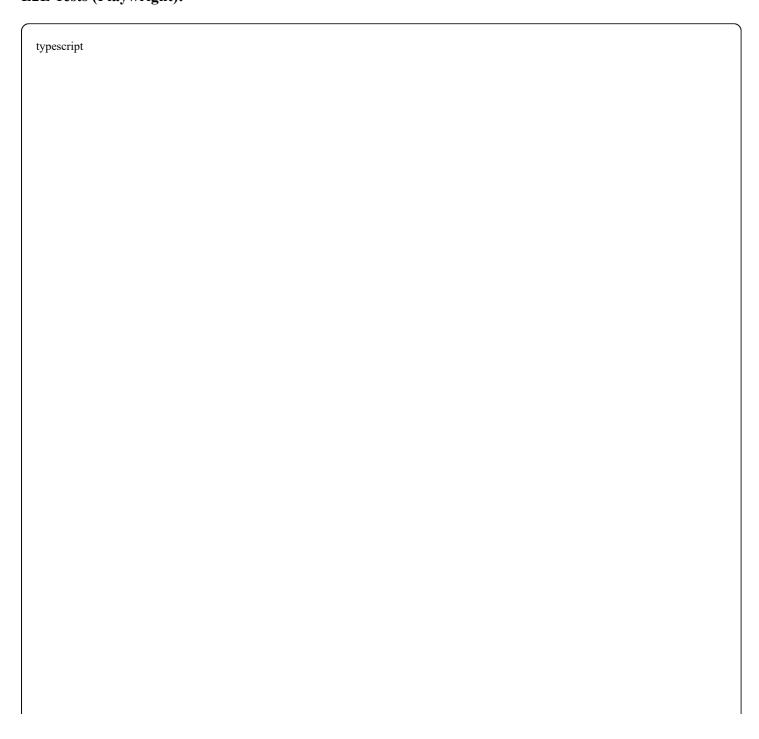
```
// auth.integration.test.ts
import request from 'supertest';
import { app } from '../app';
import { prisma } from '../database/client';
describe('Auth API', () => {
 beforeEach(async () => {
  // Clean database
  await prisma.user.deleteMany();
 });
 describe('POST /auth/register', () => {
  it('should register new user', async () => {
   const response = await request(app)
     .post('/api/auth/register')
     .send({
      email: 'test@example.com',
      password: 'SecurePass123!',
      name: 'Test User',
     })
     .expect(201);
    expect(response.body.success).toBe(true);
   expect(response.body.data.user.email).toBe('test@example.com');
   expect(response.body.data.accessToken).toBeDefined();
   // Verify user in database
   const user = await prisma.user.findUnique({
     where: { email: 'test@example.com' },
   expect(user).toBeDefined();
  });
  it('should return 409 for duplicate email', async () => {
   // Create user first
   await request(app)
     .post('/api/auth/register')
     .send({
      email: 'test@example.com',
      password: 'SecurePass123!',
      name: 'Test User',
     });
   // Try to register again
   const response = await request(app)
     .post('/api/auth/register')
```

```
.send({
    email: 'test@example.com',
    password: 'AnotherPass123!',
    name: 'Another User',
    })
    .expect(409);

expect(response.body.success).toBe(false);
    expect(response.body.error.message).toContain('already registered');
    });
});
```

## 9.4 End-to-End Testing

## **E2E** Tests (Playwright):



```
// registration.e2e.test.ts
import { test, expect } from '@playwright/test';
test.describe('User Registration Flow', () => {
 test('should complete registration and profile setup', async ({ page }) => {
  // Navigate to registration page
  await page.goto('http://localhost:5173/register');
  // Fill registration form
  await page.fill('input[name="email"]', 'newuser@example.com');
  await page.fill('input[name="password"]', 'SecurePass123!');
  await page.fill('input[name="name"]', 'New User');
  await page.click('button[type="submit"]');
  // Should redirect to profile setup
  await expect(page).toHaveURL(/.*\profile\setup/);
  // Complete profile
  await page.fill('textarea[name="bio"]', 'I am a software developer');
  await page.selectOption('select[name="location"]', 'San Francisco, CA');
  // Add skills
  await page.click('button:has-text("Add Skill")');
  await page.fill('input[placeholder="Search skills"]', 'Python');
  await page.click('li:has-text("Python")');
  await page.selectOption('select[name="proficiency"]', 'INTERMEDIATE');
  await page.click('button:has-text("Add")');
  // Submit profile
  await page.click('button:has-text("Complete Profile")');
  // Should redirect to dashboard
  await expect(page).toHaveURL(/.*\/dashboard/);
  // Verify profile completeness
  const completeness = await page.textContent('[data-testid="profile-completeness"]');
  expect(parseInt(completeness!)).toBeGreaterThan(70);
 });
});
```

#### **Session Booking E2E Test:**

typescript

```
test('should book and complete session', async ({ page, context }) => {
 // Create two users
 const teacherPage = await context.newPage();
 const learnerPage = page;
 // Teacher login
 await teacherPage.goto('http://localhost:5173/login');
 await teacherPage.fill('input[name="email"]', 'teacher@example.com');
 await teacherPage.fill('input[name="password"]', 'password123');
 await teacherPage.click('button[type="submit"]');
 // Learner login
 await learnerPage.goto('http://localhost:5173/login');
 await learnerPage.fill('input[name="email"]', 'learner@example.com');
 await learnerPage.fill('input[name="password"]', 'password123');
 await learnerPage.click('button[type="submit"]');
 // Learner browses matches
 await learnerPage.goto('http://localhost:5173/matches');
 await learnerPage.click('button:has-text("Connect"):first');
 // Learner proposes session
 await learnerPage.click('button:has-text("Propose Session")');
 await learnerPage.selectOption('select[name="skill"]', 'Python');
 await learnerPage.click('input[type="date"]');
 // ... select time
 await learnerPage.click('button:has-text("Send Proposal")');
 // Teacher accepts
 await teacherPage.goto('http://localhost:5173/sessions');
 await teacherPage.click('button:has-text("Accept"):first');
 // Verify session confirmed
 await expect(learnerPage.locator('text=Session Confirmed')).toBeVisible();
 // Fast-forward time (mock) and submit ratings
 // ... complete flow
});
```

## 9.5 Performance Testing

#### **Load Testing (k6):**

javascript

```
// load-test.js
import http from 'k6/http';
import { check, sleep } from 'k6';
export const options = {
 stages:
   { duration: '2m', target: 100 }, // Ramp up to 100 users
  { duration: '5m', target: 100 }, // Stay at 100 users
   { duration: '2m', target: 200 }, // Ramp up to 200 users
  { duration: '5m', target: 200 }, // Stay at 200 users
   { duration: '2m', target: 0 }, // Ramp down to 0 users
 ],
 thresholds: {
  http req duration: ['p(95)<500'], // 95% of requests must complete below 500ms
  http_req_failed: ['rate<0.01'], // Error rate must be below 1%
 },
};
export default function () {
 // Login
 const loginRes = http.post('http://api.skillsync.com/auth/login', {
  email: 'test@example.com',
  password: 'password123',
 });
 check(loginRes, {
  'login status is 200': (r) => r.status === 200,
  'login returns token': (r) => r.json('data.accessToken') !== ",
 });
 const token = loginRes.json('data.accessToken');
 // Get matches
 const matchesRes = http.get('http://api.skillsync.com/matches/suggestions', {
  headers: { Authorization: 'Bearer ${token}' },
 });
 check(matchesRes, {
  'matches status is 200': (r) \Rightarrow r.status === 200,
  'matches returned': (r) \Rightarrow r.json('data.matches').length > 0,
 });
 sleep(1);
```

# **9.6 Security Testing**

## **Automated Vulnerability Scanning:**

pash	
# Dependency vulnerabilities	
npm audit	
# OWASP ZAP automated scan	
docker run -t owasp/zap2docker-stable zap-baseline.py \	
-t https://api.skillsync.com \	
-r zap-report.html	
# Snyk security scan	
snyk test	
snyk monitor	

## **Manual Penetration Testing Checklist:**

SQL Injection attempts
XSS payloads in all input fields
CSRF token validation
Authentication bypass attempts
Authorization escalation (access other users' data
Rate limiting effectiveness
Session hijacking attempts
File upload vulnerabilities

## 10. MONITORING & OBSERVABILITY

## **10.1 Application Monitoring**

## **Datadog APM Integration:**

typescript		

```
// server.ts
import { tracer } from 'dd-trace';
tracer.init({
 service: 'skillsync-api',
 env: process.env.NODE_ENV,
 version: process.env.APP_VERSION,
 logInjection: true,
 analytics: true,
});
// Instrument key operations
const span = tracer.startSpan('matching.algorithm');
try {
 const matches = await matchingService.generateMatches(userId);
 span.setTag('match_count', matches.length);
} catch (error) {
 span.setTag('error', true);
 throw error;
} finally {
 span.finish();
```

#### **Custom Metrics:**

```
typescript
import { StatsD } from 'hot-shots';

const statsd = new StatsD({
   host: 'localhost',
   port: 8125,
   prefix: 'skillsync.',
});

// Track business metrics
statsd.increment('sessions.created');
statsd.histogram('sessions.duration', durationMinutes);
statsd.gauge('users.active', activeUserCount);
statsd.timing('matching.algorithm.duration', executionTime);
```

## 10.2 Logging Strategy

#### **Structured Logging (Winston):**

```
import winston from 'winston';
const logger = winston.createLogger({
 level: 'info',
 format: winston.format.combine(
  winston.format.timestamp(),
  winston.format.errors({ stack: true }),
  winston.format.json()
 defaultMeta: { service: 'skillsync-api' },
 transports: [
  new winston.transports.File({ filename: 'error.log', level: 'error' }),
  new winston.transports.File({ filename: 'combined.log' }),
 ],
});
// Usage
logger.info('Session created', {
 sessionId: session.id,
 requesterId: session.requesterId,
 recipientId: session.recipientId,
 creditsCost: session.creditsCost.
});
logger.error('Payment processing failed', {
 userId: user.id,
 amount: amount,
 error: error.message,
 stack: error.stack,
});
```

#### Log Levels:

- ERROR: Application errors, exceptions
- WARN: Degraded functionality, potential issues
- **INFO:** Important business events (session created, user registered)
- **DEBUG:** Detailed diagnostic information
- TRACE: Very detailed, typically disabled in production

### 10.3 Alerting

#### **Alert Rules:**

```
# alerts.yml
alerts:
 - name: HighErrorRate
  condition: error_rate > 5%
  window: 5m
  severity: critical
  channels: [pagerduty, slack]
 - name: SlowAPIResponses
  condition: p95_response_time > 1000ms
  window: 5m
  severity: warning
  channels: [slack]
 - name: DatabaseConnectionPoolExhausted
  condition: db_pool_active >= db_pool_max
  window: 2m
  severity: critical
  channels: [pagerduty, slack]
 - name: LowCreditBalance
  condition: credit_purchase_rate < threshold</pre>
  window: 1h
  severity: info
  channels: [email]
```

### 10.4 Health Checks

typescript	

```
// health.controller.ts
export class HealthController {
 async checkHealth(req: Request, res: Response) {
  const checks = await Promise.all([
   this.checkDatabase(),
   this.checkRedis(),
   this.checkS3(),
   this.checkExternalAPIs(),
  ]);
  const isHealthy = checks.every(check => check.status === 'ok');
  const statusCode = isHealthy ? 200 : 503;
  res.status(statusCode).json({
   status: isHealthy? 'healthy': 'unhealthy',
   timestamp: new Date().toISOString(),
   checks: {
     database: checks[0],
     redis: checks[1],
     storage: checks[2],
     externalAPIs: checks[3],
   },
  });
 private async checkDatabase() {
  try {
   await prisma.$queryRaw'SELECT 1';
   return { status: 'ok', responseTime: 5 };
  } catch (error) {
   return { status: 'error', message: error.message };
```

### 11. APPENDICES

## **Appendix A: Technology Alternatives**

Component	Primary Choice	Alternatives	
Frontend Framework	React	Vue.js, Svelte, Angular	
Backend Framework	Express.js	NestJS, Fastify, Koa	
Database	PostgreSQL	MySQL, MongoDB, CockroachDB	
	•	•	

Component	Primary Choice	Alternatives	
ORM	Prisma	TypeORM, Sequelize, Drizzle	
Cache	Redis	Memcached, KeyDB	
Message Queue	Bull (Redis)	RabbitMQ, AWS SQS, Kafka	
Object Storage	AWS S3	Cloudflare R2, Google Cloud Storage	
Video Platform	Zoom	Google Meet, Daily.co, Agora	
Payment Gateway	Stripe	PayPal, Braintree, Square	
<b>4</b>	<u>-</u>	-	

## **Appendix B: Glossary**

- API Gateway: Single entry point for all client requests
- Circuit Breaker: Pattern to prevent cascading failures
- **CORS:** Cross-Origin Resource Sharing
- DTO: Data Transfer Object
- Idempotency: Operation can be applied multiple times without changing result
- **JWT:** JSON Web Token for authentication
- ORM: Object-Relational Mapping
- Saga Pattern: Manage distributed transactions
- WebSocket: Protocol for real-time bidirectional communication

#### **Appendix C: References**

- Prisma Documentation
- React Documentation
- Socket.io Documentation
- AWS Architecture Best Practices
- OWASP Top 10
- <u>PostgreSQL Performance Tips</u>

### **Appendix D: Change Log**

Version	Date	Author	Changes
1.0	2025-10-25	Architecture Team	Initial SDD creation
4	•	•	<b>•</b>

#### **Document End**

#### **SUMMARY**

This Software Design Document provides a comprehensive technical blueprint for the SkillSync Peer Learning Exchange Platform. Key highlights:

#### **Architecture:**

- Hybrid microservices with monolithic core
- Event-driven for scalability
- RESTful API + WebSocket for real-time features

#### **Technology Stack:**

- Frontend: React 18 + TypeScript + Tailwind CSS
- Backend: Node.js + Express.js + Prisma + PostgreSQL
- Real-time: Socket.io + Redis
- AI/ML: Python FastAPI + scikit-learn

#### **Security:**

- JWT authentication with refresh tokens
- RBAC authorization
- OWASP Top 10 mitigation
- GDPR/CCPA compliance

#### **Performance:**

- Multi-layer caching (Redis)
- Database optimization (indexes, connection pooling)
- CDN for static assets
- Horizontal scalability

#### **Deployment:**

- Docker containers
- AWS/GCP cloud infrastructure
- CI/CD with GitHub Actions
- 99.9% uptime target

## **Testing:**

- 60% unit tests, 30% integration, 10% E2E
- Load testing for 10,000+ concurrent users
- Automated security scanning

This design supports the MVP requirements while providing a scalable foundation for future enhancements including VR/AR features, mobile apps, and enterprise integrations.