**🛡️ DisasterShield - Technical Specifications & Feature Matrix**

**📊 Feature Matrix by User Role**

Feature | Client | Contractor | Admin | Description

--------- | -------- | ------------ | ------- | -------------

\*\*Authentication\*\* | ✅ | ✅ | ✅ | Email/password with role-based access

\*\*Dashboard\*\* | ✅ | ✅ | ✅ | Role-specific dashboard with relevant data

\*\*Project Management\*\* | ✅ | ✅ | ✅ | View and manage assigned projects

\*\*Media Upload\*\* | ✅ | ❌ | ✅ | Upload photos and voice notes

\*\*Contractor Matching\*\* | ✅ | ❌ | ✅ | Automated contractor selection

\*\*Estimate Submission\*\* | ❌ | ✅ | ✅ | Submit detailed cost estimates

\*\*Estimate Review\*\* | ✅ | ❌ | ✅ | Review and accept/reject estimates

\*\*Payment Processing\*\* | ✅ | ❌ | ✅ | Secure Stripe payment integration

\*\*FNOL Generation\*\* | ✅ | ❌ | ✅ | First Notice of Loss document creation

\*\*Insurance Integration\*\* | ✅ | ❌ | ✅ | Direct API submission to insurers

\*\*Notification System\*\* | ✅ | ✅ | ✅ | Real-time in-app notifications

\*\*Email Notifications\*\* | ✅ | ✅ | ✅ | Transactional email system

\*\*SMS Notifications\*\* | ✅ | ✅ | ✅ | Twilio SMS integration

\*\*User Management\*\* | ❌ | ❌ | ✅ | Manage users and roles

\*\*System Analytics\*\* | ❌ | ❌ | ✅ | Business intelligence dashboard

\*\*Insurance Company Management\*\* | ❌ | ❌ | ✅ | Manage insurance provider APIs

---

**🔧 Technical Architecture Details**

**\*\*Frontend Stack\*\***

// Core Technologies  
React 18.2.0 // UI Framework  
TypeScript 5.2.2 // Type Safety  
Vite 5.1.4 // Build Tool  
Tailwind CSS 3.4.1 // Styling  
Radix UI // Component Library  
  
// Key Libraries  
React Router 6.21.3 // Client-side routing  
React Hook Form 7.49.3 // Form management  
Zod 3.22.4 // Schema validation  
Stripe.js // Payment processing  
Supabase JS 2.39.7 // Backend integration

**\*\*Backend Stack\*\***

// Core Technologies  
Supabase // Backend-as-a-Service  
PostgreSQL // Database  
Edge Functions // Serverless functions  
Row Level Security // Database security  
  
// Integrations  
Stripe API // Payment processing  
Resend API // Email service  
Twilio API // SMS service  
Insurance APIs // FNOL submission

**\*\*Database Schema\*\***

-- Core Tables  
auth.users -- Supabase authentication  
public.profiles -- User profiles and roles  
public.projects -- Main project/claim records  
public.contractors -- Contractor information  
public.media -- File attachments  
public.stripe\_customers -- Payment customer mapping  
public.stripe\_orders -- Payment order records  
public.contractor\_estimates -- Dynamic pricing estimates  
public.fnol\_records -- FNOL document records  
public.insurance\_companies -- Insurance provider data  
public.notifications -- In-app notifications  
public.match\_requests -- Contractor matching records

---

**🚀 API Endpoints & Functions**

**\*\*Supabase Edge Functions\*\***

**\*\*Payment Processing\*\***

// POST /functions/v1/stripe-checkout  
// Creates Stripe checkout sessions  
// Parameters: projectId, productKey, userId  
// Returns: checkout URL  
  
// POST /functions/v1/stripe-webhook  
// Handles Stripe webhook events  
// Updates payment status in database  
// Sends completion notifications  
  
// POST /functions/v1/create-dynamic-product  
// Creates dynamic Stripe products  
// Parameters: estimateId, projectId  
// Returns: productId, priceId

**\*\*Communication\*\***

// POST /functions/v1/send-email  
// Sends transactional emails  
// Parameters: to, subject, template, data  
// Returns: success status  
  
// POST /functions/v1/submit-fnol  
// Submits FNOL to insurance companies  
// Parameters: fnolData, insuranceCompanyId  
// Returns: submission status

**\*\*Frontend API Integration\*\***

// Authentication  
supabase.auth.signUp()  
supabase.auth.signIn()  
supabase.auth.signOut()  
supabase.auth.getUser()  
  
// Database Operations  
supabase.from('table').select()  
supabase.from('table').insert()  
supabase.from('table').update()  
supabase.from('table').delete()  
  
// Real-time Subscriptions  
supabase.channel('table').on('\*', callback)

---

**💳 Payment System Architecture**

**\*\*Payment Structure\*\***

interface PaymentGroup {  
 CORE\_PROJECT: {  
 products: ['SECURITY\_DEPOSIT', 'DISASTERSHIELD\_SERVICE\_FEE', 'REPAIR\_COST\_ESTIMATE']  
 required: true  
 }  
 FNOL\_GENERATION: {  
 products: ['FNOL\_GENERATION\_FEE']  
 required: false  
 }  
}  
  
// Payment Amounts  
SECURITY\_DEPOSIT: $500.00 // Refundable  
DISASTERSHIELD\_SERVICE\_FEE: $99.00 // Platform fee  
REPAIR\_COST\_ESTIMATE: Dynamic // Based on contractor estimate  
FNOL\_GENERATION\_FEE: $100.00 // Per FNOL generation

**\*\*Payment Flow\*\***

1. [BOLD]Estimate Acceptance[/BOLD] → Dynamic Stripe product creation  
 2. [BOLD]Checkout Session[/BOLD] → Secure payment processing  
 3. [BOLD]Webhook Processing[/BOLD] → Database updates  
 4. [BOLD]Status Updates[/BOLD] → Real-time UI updates  
 5. [BOLD]Notifications[/BOLD] → Email/SMS alerts

---

**🔐 Security Implementation**

**\*\*Authentication & Authorization\*\***

// Role-based access control  
interface UserRole {  
 client: 'default' | null  
 contractor: 'contractor'  
 admin: 'admin'  
}  
  
// Route protection  
<ClientRoute> // Client-only access  
<AdminRoute> // Admin-only access  
<PublicRoute> // Public access

**\*\*Database Security\*\***

-- Row Level Security policies  
CREATE POLICY "Users can view own data" ON profiles  
 FOR SELECT USING (auth.uid() = id);  
  
CREATE POLICY "Contractors can view own estimates" ON contractor\_estimates  
 FOR SELECT USING (contractor\_id IN (  
 SELECT id FROM contractors WHERE user\_id = auth.uid()  
 ));

**\*\*API Security\*\***

- JWT token validation  
 - CORS configuration  
 - Rate limiting  
 - Input validation  
 - SQL injection prevention

---

**📱 User Interface Components**

**\*\*Core UI Components\*\***

// Layout Components  
<Dashboard> // Role-specific dashboard  
<ProjectPortal> // Project details view  
<PaymentPage> // Payment processing  
<FNOLPage> // Document generation  
  
// Form Components  
<IntakeForm> // Claim filing form  
<EstimateForm> // Contractor estimate form  
<PaymentForm> // Payment processing form  
  
// Display Components  
<StatusTimeline> // Project progress  
<MediaGallery> // File attachments  
<NotificationBell> // Real-time alerts

**\*\*Responsive Design\*\***

- Mobile-first approach  
 - Breakpoints: sm, md, lg, xl  
 - Touch-friendly interfaces  
 - Progressive Web App features

---

**🔄 Workflow Automation**

**\*\*Contractor Matching Algorithm\*\***

interface MatchingCriteria {  
 geographicProximity: number // Distance scoring  
 serviceArea: boolean // Coverage verification  
 expertiseMatch: number // Damage type specialization  
 availability: boolean // Capacity checking  
 rating: number // Performance history  
}  
  
// Scoring system  
const score = (geographic \* 0.4) + (expertise \* 0.3) + (rating \* 0.3)

**\*\*Email Automation\*\***

// Email templates  
- ContractorInvitation // Job opportunity emails  
- JobAcceptedConfirmation // Assignment confirmations  
- PaymentCompleted // Payment notifications  
- EstimateReceived // Estimate notifications  
- FNOLSubmitted // Document status updates

**\*\*Notification System\*\***

// Real-time notifications  
interface Notification {  
 type: 'job\_accepted' | 'payment\_completed' | 'estimate\_received'  
 title: string  
 message: string  
 data: object  
 created\_at: timestamp  
}

---

**📊 Data Flow & State Management**

**\*\*State Management\*\***

// React hooks for state  
const [user, setUser] = useState<User | null>(null)  
const [projects, setProjects] = useState<Project[]>([])  
const [notifications, setNotifications] = useState<Notification[]>([])  
  
// Real-time subscriptions  
useEffect(() => {  
 const subscription = supabase  
 .channel('notifications')  
 .on('postgres\_changes', { event: 'INSERT', schema: 'public', table: 'notifications' },   
 (payload) => setNotifications(prev => [...prev, payload.new])  
 )  
 .subscribe()  
   
 return () => subscription.unsubscribe()  
}, [])

**\*\*Data Validation\*\***

// Zod schemas  
const ProjectSchema = z.object({  
 address: z.string().min(1),  
 city: z.string().min(1),  
 state: z.string().min(2).max(2),  
 peril: z.enum(['water', 'fire', 'wind', 'other']),  
 description: z.string().min(10)  
})

---

**🚀 Deployment & Infrastructure**

**\*\*Frontend Deployment (Vercel)\*\***

{  
 "buildCommand": "npm run build",  
 "outputDirectory": "dist",  
 "framework": "vite",  
 "nodeVersion": "18.x"  
}

**\*\*Backend Deployment (Supabase)\*\***

# Database migrations  
supabase db push  
  
# Edge function deployment  
supabase functions deploy stripe-checkout  
supabase functions deploy stripe-webhook  
supabase functions deploy send-email  
supabase functions deploy create-dynamic-product

**\*\*Environment Configuration\*\***

// Environment variables  
VITE\_SUPABASE\_URL=your\_supabase\_url  
VITE\_SUPABASE\_ANON\_KEY=your\_anon\_key  
VITE\_STRIPE\_PUBLISHABLE\_KEY=your\_stripe\_key  
STRIPE\_SECRET\_KEY=your\_secret\_key  
STRIPE\_WEBHOOK\_SECRET=your\_webhook\_secret  
RESEND\_API\_KEY=your\_resend\_key  
TWILIO\_ACCOUNT\_SID=your\_twilio\_sid  
TWILIO\_AUTH\_TOKEN=your\_twilio\_token

---

**📈 Performance & Scalability**

**\*\*Performance Optimizations\*\***

- Code splitting with React.lazy()  
 - Image optimization and lazy loading  
 - Database query optimization  
 - Caching strategies  
 - CDN integration

**\*\*Scalability Considerations\*\***

- Horizontal scaling with Supabase  
 - Edge function auto-scaling  
 - Database connection pooling  
 - File storage optimization  
 - API rate limiting

---

**🔍 Monitoring & Analytics**

**\*\*Error Tracking\*\***

// Error boundary implementation  
class ErrorBoundary extends React.Component {  
 componentDidCatch(error: Error, errorInfo: ErrorInfo) {  
 // Log error to monitoring service  
 console.error('Application error:', error, errorInfo)  
 }  
}

**\*\*Performance Monitoring\*\***

- Vercel Analytics integration  
 - Supabase performance metrics  
 - Stripe payment analytics  
 - User behavior tracking

---

**🧪 Testing Strategy**

**\*\*Testing Framework\*\***

// Unit tests  
- Component testing with React Testing Library  
- Function testing with Jest  
- API testing with Supertest  
  
// Integration tests  
- End-to-end workflow testing  
- Payment flow testing  
- User authentication testing  
  
// Manual testing  
- Cross-browser compatibility  
- Mobile responsiveness  
- Accessibility compliance

---

**📚 Documentation & Maintenance**

**\*\*Code Documentation\*\***

- TypeScript type definitions  
 - JSDoc comments  
 - API documentation  
 - Component documentation

**\*\*User Documentation\*\***

- User guides and tutorials  
 - FAQ system  
 - Video tutorials  
 - Support documentation

---

[ITALIC]This technical specification provides detailed implementation information for the DisasterShield platform. For specific implementation questions or custom development requests, please contact the development team.[/ITALIC]