🎯 Plan de Acción Completo - Preparación para Entrevistas Frontend

📋 Información General

Objetivo: Preparación integral para entrevistas como Frontend Developer Junior
Duración: 5 semanas
Perfil: Ingeniero Industrial → Frontend Developer (Angular/TypeScript)
Fecha inicio: 26 de agosto, 2025

📊 Resumen del Plan

Fase	Duración	Enfoque Principal	Objetivo
Fase 1	Semanas 1-2	Preparación Técnica	Dominar fundamentos sin IA
Fase 2	Semana 3	Preparación de Discurso	Perfeccionar soft skills
Fase 3	Semana 4	Simulación de Entrevistas	Práctica realista
Fase 4	Semana 5	Optimización Final	Pulir detalles

📝 FASE 1: PREPARACIÓN TÉCNICA (Semanas 1-2)

🔧 Semana 1: Fundamentos Sólidos

Días 1-2: Angular Core

⏰ **Tiempo diario:** 2-3 horas práctica

☑ **Objetivos:**

- Lifecycle hooks (ngOnInit, ngOnDestroy, ngOnChanges)
- Component communication (@Input, @Output, EventEmitter)
- Services y Dependency Injection
- HTTP Client básico

💻 **Ejercicio práctico:**

```
// Crear componente padre-hijo que:
// - Pase datos con @Input
// - Emita eventos con @Output
// - Use service para datos
// - Implemente lifecycle hooks
```

```
@Component({
  selector: 'app-parent',
  template: `
    <app-child
      [userData]="selectedUser"
      (userSelected)="onUserSelected($event)">
    </app-child>
  `
})
export class ParentComponent implements OnInit {
  users: User[] = [];
  selectedUser: User;

  constructor(private userService: UserService) {}

  ngOnInit() {
    this.loadUsers();
  }

  loadUsers() {
    this.userService.getUsers().subscribe(users => {
      this.users = users;
      this.selectedUser = users[0];
    });
  }

  onUserSelected(user: User) {
    this.selectedUser = user;
  }
}
```

Días 3-4: Forms y Routing

⏰ **Tiempo diario:** 2-3 horas

☑ **Objetivos:**

- Reactive Forms con FormBuilder
- Validaciones (built-in y custom)
- Routing básico y avanzado
- Guards (CanActivate, CanDeactivate)

💻 **Ejercicio:**

```
// Login/Register form con:
// - Validaciones complejas
// - Routing protegido
// - Guard para autenticación

export class LoginComponent {
```

```
  loginForm = this.fb.group({
    email: ['', [Validators.required, Validators.email]],
    password: ['', [Validators.required, Validators.minLength(6)]],
    rememberMe: [false]
  });

  constructor(
    private fb: FormBuilder,
    private authService: AuthService,
    private router: Router
  ) {}

  onSubmit() {
    if (this.loginForm.valid) {
      const { email, password } = this.loginForm.value;
      this.authService.login(email, password).subscribe({
        next: () => this.router.navigate(['/dashboard']),
        error: (error) => this.handleError(error)
      });
    } else {
      this.markFormGroupTouched();
    }
  }

  // Custom validator
  static passwordStrength(control: AbstractControl): ValidationErrors | null {
    const value = control.value;
    if (!value) return null;

    const hasNumber = /[0-9]/.test(value);
    const hasUpper = /[A-Z]/.test(value);
    const hasLower = /[a-z]/.test(value);

    const valid = hasNumber && hasUpper && hasLower;
    return valid ? null : { passwordStrength: true };
  }
}
```

Días 5-7: RxJS y State Management

⏰ Tiempo diario: 2-3 horas

☑ Conceptos clave:

- Observables vs Promises
- Operators básicos (map, filter, switchMap)
- Subject vs BehaviorSubject
- Unsubscribe patterns

💻 Ejercicio:

```
// Search component con:
// - Debounce
// - API calls
// - Loading states
// - Error handling

@Component({
  selector: 'app-search',
  template: `
    <input
      type="text"
      placeholder="Search users..."
      [formControl]="searchControl">

    <div *ngIf="loading">Loading...</div>
    <div *ngIf="error">{{ error }}</div>

    <ul>
      <li *ngFor="let user of users$ | async">
        {{ user.name }}
      </li>
    </ul>
  `
})
export class SearchComponent implements OnInit, OnDestroy {
  searchControl = new FormControl('');
  users$ = new BehaviorSubject<User[]>([]);
  loading = false;
  error: string | null = null;

  private destroy$ = new Subject<void>();

  constructor(private userService: UserService) {}

  ngOnInit() {
    this.searchControl.valueChanges.pipe(
      debounceTime(300),
      distinctUntilChanged(),
      switchMap(term => {
        this.loading = true;
        this.error = null;

        if (!term) {
          return of([]);
        }

        return this.userService.searchUsers(term).pipe(
          catchError(error => {
            this.error = 'Error searching users';
            return of([]);
          })
        );
      }),
```

```
      takeUntil(this.destroy$)
    ).subscribe(users => {
      this.users$.next(users);
      this.loading = false;
    });
  }

  ngOnDestroy() {
    this.destroy$.next();
    this.destroy$.complete();
  }
}
```

📆 Semana 2: JavaScript/TypeScript

Días 1-3: JavaScript ES6+

⏰ Tiempo diario: 2-3 horas

☑ Temas clave:

- Array methods (map, filter, reduce, find)
- Destructuring y spread operator
- Async/await vs Promises
- Closures y scope

💻 Ejercicios diarios:

1. FizzBuzz variants:

```
// Clásico FizzBuzz
function fizzBuzz(n) {
  for (let i = 1; i <= n; i++) {
    if (i % 15 === 0) console.log('FizzBuzz');
    else if (i % 3 === 0) console.log('Fizz');
    else if (i % 5 === 0) console.log('Buzz');
    else console.log(i);
  }
}

// Versión con array
function fizzBuzzArray(n) {
  return Array.from({ length: n }, (_, i) => {
    const num = i + 1;
    if (num % 15 === 0) return 'FizzBuzz';
    if (num % 3 === 0) return 'Fizz';
    if (num % 5 === 0) return 'Buzz';
    return num;
  });
}
```

2. Array manipulation problems:

```
const users = [
  { id: 1, name: 'Juan', age: 25, department: 'IT', active: true },
  { id: 2, name: 'Ana', age: 30, department: 'HR', active: false },
  { id: 3, name: 'Carlos', age: 22, department: 'IT', active: true }
];

// 1. Usuarios activos del IT
const activeITUsers = users
  .filter(user => user.active && user.department === 'IT')
  .map(user => user.name);

// 2. Promedio de edad por departamento
const avgAgeByDept = users.reduce((acc, user) => {
  if (!acc[user.department]) {
    acc[user.department] = { total: 0, count: 0 };
  }
  acc[user.department].total += user.age;
  acc[user.department].count++;
  return acc;
}, {});

// 3. Usuario más joven
const youngestUser = users.reduce((youngest, user) =>
  user.age < youngest.age ? user : youngest
);
```

3. Async data processing:

```
// Parallel API calls
async function fetchUserData(userIds) {
  try {
    const promises = userIds.map(id =>
      fetch(`/api/users/${id}`).then(res => res.json())
    );

    const users = await Promise.all(promises);
    return users;
  } catch (error) {
    console.error('Error fetching users:', error);
    throw error;
  }
}

// Sequential processing
async function processUsersSequentially(userIds) {
  const results = [];
```

```
  for (const id of userIds) {
    try {
      const response = await fetch(`/api/users/${id}`);
      const user = await response.json();
      results.push(user);
    } catch (error) {
      console.error(`Error fetching user ${id}:`, error);
    }
  }

  return results;
}

// With timeout
function fetchWithTimeout(url, timeout = 5000) {
  return Promise.race([
    fetch(url),
    new Promise((_, reject) =>
      setTimeout(() => reject(new Error('Timeout')), timeout)
    )
  ]);
}
```

Días 4-7: TypeScript Específico

⏰ **Tiempo diario:** 2-3 horas

☑ **Conceptos para entrevistas:**

- Types vs Interfaces
- Generics básicos
- Union types
- Optional chaining

💻 **Ejercicios TypeScript:**

1. Types vs Interfaces:

```
// Type alias
type UserType = {
  id: number;
  name: string;
  email: string;
};

// Interface
interface UserInterface {
  id: number;
  name: string;
  email: string;
```

```
}

// Extending
interface AdminUser extends UserInterface {
  permissions: string[];
  lastLogin: Date;
}

// Union types
type Status = 'pending' | 'approved' | 'rejected';
type ApiResponse<T> = {
  data: T;
  status: Status;
  message?: string;
};
```

2. Generics básicos:

```
// Generic function
function identity<T>(arg: T): T {
  return arg;
}

// Generic interface
interface Repository<T> {
  findById(id: number): Promise<T | null>;
  findAll(): Promise<T[]>;
  create(entity: T): Promise<T>;
  update(id: number, entity: Partial<T>): Promise<T>;
  delete(id: number): Promise<void>;
}

// Implementation
class UserRepository implements Repository<User> {
  async findById(id: number): Promise<User | null> {
    // Implementation
    return null;
  }

  async findAll(): Promise<User[]> {
    // Implementation
    return [];
  }

  async create(user: User): Promise<User> {
    // Implementation
    return user;
  }

  async update(id: number, user: Partial<User>): Promise<User> {
    // Implementation
    return user as User;
```

```
  }

  async delete(id: number): Promise<void> {
    // Implementation
  }
}
```

📝 Mock interview questions:

❓ "¿Cuál es la diferencia entre any y unknown?"

```
// ❌ any - disables type checking
let value: any = 42;
value = "hello";
value.toUpperCase(); // No error, but runtime error possible

// ☑️ unknown - safer, requires type checking
let value: unknown = 42;
if (typeof value === 'string') {
  value.toUpperCase(); // OK, type narrowed
}
```

❓ "¿Cómo implementarías un generic type?"

```
// Generic API response wrapper
interface ApiResponse<T> {
  data: T;
  success: boolean;
  message?: string;
}

// Usage
const userResponse: ApiResponse<User> = {
  data: { id: 1, name: 'Juan' },
  success: true
};

const usersResponse: ApiResponse<User[]> = {
  data: [{ id: 1, name: 'Juan' }],
  success: true
};
```

💼 FASE 2: PREPARACIÓN DE DISCURSO (Semana 3)

🎤 Días 1-2: Tu Historia de Transición

Elevator Pitch (60 segundos):

```
"Soy Ingeniero Industrial con 12+ años optimizando procesos empresariales.
Durante los últimos 18 meses me especialicé intensivamente en Angular y
TypeScript,
desarrollando 3 aplicaciones web funcionales.

Mi valor único radica en combinar visión de negocio con habilidades técnicas
modernas.
Puedo traducir requisitos complejos de stakeholders en soluciones web escalables,
aplicando mi experiencia en gestión de calidad y metodologías ágiles.

Busco mi primera oportunidad como Frontend Developer donde pueda aportar
tanto código limpio como pensamiento estratégico."
```

Respuestas a preguntas frecuentes:

? "¿Por qué cambiaste de carrera?"

```
☑ "La tecnología me permite escalar el impacto de la optimización de procesos.
   En lugar de mejorar un proceso en una empresa, puedo crear herramientas
   que optimicen procesos para miles de usuarios.

   Mi experiencia gestionando stakeholders y traduciendo necesidades complejas
   en soluciones sistemáticas se traslada perfectamente al desarrollo de
software."
```

? "¿Cómo sabes que te gusta programar?"

```
☑ "Durante 18 meses dediqué 3-4 horas diarias construyendo aplicaciones reales.
   Desarrollé SerTech, que gestiona servicios técnicos, Planeta Predictor para
   pollas deportivas, y Movie App para explorar películas.

   La satisfacción de resolver problemas complejos paso a paso, ver código
   transformarse en soluciones funcionales, y la mejora continua me confirman
   mi pasión por esta disciplina."
```

? "¿Qué aportas que otros developers no?"

```
☑ "Tres elementos únicos:

   1. Experiencia real gestionando stakeholders y traduciendo necesidades
      de negocio en especificaciones técnicas.

   2. Mentalidad de calidad y procesos: sé documentar, aplicar metodologías
      y pensar en escalabilidad desde el diseño.
```

```
3. Comunicación efectiva con equipos multidisciplinarios - puedo hablar
   tanto con CTOs como con usuarios finales."
```

❓ "¿Cuáles son tus debilidades?"

```
☑ "Al venir de otra industria, aún estoy desarrollando algunas habilidades
  técnicas avanzadas como arquitectura de microservicios o DevOps.

  Por eso dedico tiempo diario a aprender nuevas tecnologías y mejores prácticas.
  Mi enfoque es ser honesto sobre mis áreas de crecimiento mientras demuestro
  mi capacidad de aprendizaje acelerado."
```

❓ "¿Expectativas salariales?"

```
☑ "Entiendo que estoy iniciando en tech, pero aporto 12+ años de experiencia
  profesional valiosa. Estoy abierto a discutir un paquete competitivo
  para un role junior que reconozca tanto mi potencial técnico como mi
  experiencia en gestión.

  Mi prioridad es encontrar un equipo donde pueda crecer y aportar valor mutuo."
```

🗣️ Días 3-4: Presentación de Proyectos

Demo Script para SerTech:

```
"Les voy a mostrar SerTech, una aplicación para gestión de servicios técnicos.

🔘 PROBLEMA:
Los técnicos de electrodomésticos necesitaban gestionar solicitudes de manera
eficiente, pero los procesos manuales causaban retrasos y errores.

💡 SOLUCIÓN:
Desarrollé una SPA en Angular que optimiza todo el flujo:

🛠️ TECNOLOGÍAS APLICADAS:
- Componentes reutilizables siguiendo Atomic Design
- Formularios reactivos con validaciones complejas
- Routing protegido con guards personalizados
- Estado manejado con services y observables
- Design responsive con CSS Grid y Flexbox
- TypeScript con tipado estricto

📊 RESULTADO:
Aplicación funcional que demuestra flujo completo de CRUD, comunicación
entre componentes, y mejores prácticas de desarrollo.
```

```
🎯 CÓDIGO DESTACADO:
- 'Aquí implementé un custom validator para validar códigos de servicio...'
- 'Este service maneja el estado global con BehaviorSubject...'
- 'Apliqué principios SOLID en esta estructura de componentes...'

🔗 DEMO EN VIVO: [Mostrar funcionalidades clave]"
```

Demo Script para Planeta Predictor:

```
"Planeta Predictor resuelve la gestión de pollas deportivas.

🏐 PROBLEMA:
Los grupos de amigos necesitaban una forma organizada de hacer predicciones
deportivas y ver rankings en tiempo real.

💡 SOLUCIÓN TÉCNICA:
- Algoritmo de puntuación personalizado
- Tabla de posiciones dinámica con ordenamiento
- Persistencia en Local Storage
- Estado complejo manejado con services

🎯 COMPLEJIDAD TÉCNICA:
- Cálculos matemáticos en tiempo real
- Manejo de arrays complejos y ordenamientos
- Interface responsive para móviles
- Gestión de datos sin backend

📈 VALOR DE NEGOCIO:
Demuestra mi capacidad para crear lógica de negocio compleja y
traducir reglas deportivas en código funcional."
```

🧮 Días 5-7: Preparación para Video Entrevistas

Setup técnico:

```
☑ Hardware:
- Cámara a la altura de los ojos
- Iluminación frontal adecuada (ventana o ring light)
- Auriculares con micrófono de calidad
- Background neutro y profesional

☑ Software:
- Internet estable (test de velocidad: >10 Mbps)
- Backup plan (hotspot móvil)
- VS Code abierto con proyectos listos
- Bookmarks de documentación organizados
- Zoom/Teams/Meet actualizados

☑ Preparación:
```

```
- Portfolio deployed y funcionando
- GitHub profile actualizado
- LinkedIn optimizado
- CV en PDF listo para compartir
```

Práctica de presentación:

```
📞 EJERCICIOS DIARIOS:

Día 5: Graba tu elevator pitch (máximo 90 segundos)
- Objetivo: Fluidez sin muletillas
- Métrica: Completar sin pausas >3 segundos

Día 6: Practica demo de proyectos (3-5 minutos cada uno)
- SerTech: Enfoque en arquitectura y componentes
- Planeta Predictor: Lógica de negocio y algoritmos
- Movie App: API integration y UX

Día 7: Simula live coding explicando en voz alta
- Ejercicio: "Crear lista de usuarios con búsqueda"
- Tiempo: 30 minutos
- Narrar cada decisión técnica
```

🧪 FASE 3: SIMULACIÓN DE ENTREVISTAS (Semana 4)

👨‍💼 Días 1-2: Entrevista HR Simulada

Script de práctica (30 minutos):

❓ "Cuéntame sobre ti"

```
☑ ESTRUCTURA STAR:
Situación: "Soy Ingeniero Industrial con 12+ años..."
Tarea: "Decidí transicionar al desarrollo web porque..."
Acción: "Dediqué 18 meses a especializarme en Angular..."
Resultado: "Desarrollé 3 aplicaciones funcionales y busco..."

⏱ TIEMPO: 90 segundos máximo
```

❓ "¿Por qué quieres trabajar aquí?"

```
☑ FÓRMULA:
1. Research específico de la empresa
2. Conexión con tu background
```

```
3. Valor que puedes aportar

EJEMPLO:
"Investigué [Empresa] y me interesa especialmente [aspecto específico].
Mi background optimizando procesos industriales me permitiría aportar
valor único en [área específica de la empresa], especialmente
considerando que usan [tecnología que manejas]."
```

❓ "¿Cuáles son tus fortalezas?"

```
☑ ESTRUCTURA:
1. Fortaleza específica
2. Ejemplo concreto
3. Aplicación al role

"Mi principal fortaleza es traducir requisitos complejos en soluciones
sistemáticas. Por ejemplo, en la Universidad de Caldas gestioné
procesos de adquisición donde debía entender necesidades técnicas
de múltiples departamentos y crear especificaciones claras.

Esta habilidad se traslada perfectamente al desarrollo frontend,
donde puedo tomar requerimientos de stakeholders y convertirlos
en componentes funcionales y escalables."
```

Preguntas para hacer al entrevistador:

```
🎯 SOBRE EL ROLE:
"¿Cómo es un día típico para un Frontend Developer en el equipo?"
"¿Qué proyectos estaría trabajando en mis primeros 3 meses?"
"¿Cómo maneja el equipo el balance entre features nuevas y refactoring?"

🎯 SOBRE EL EQUIPO:
"¿Cómo está estructurado el equipo de desarrollo?"
"¿Qué oportunidades hay para mentoría y crecimiento técnico?"
"¿Cómo colaboran frontend y backend developers?"

🎯 SOBRE LA EMPRESA:
"¿Cuáles son los principales retos técnicos que enfrenta la empresa?"
"¿Cómo ve la evolución del stack tecnológico en los próximos 2 años?"
"¿Qué métricas usan para medir el éxito del equipo?"
```

💻 Días 3-5: Entrevista Técnica Simulada

Mock Live Coding Session (45 min):

Ejercicio típico: "User List Component"

```
// ⏱️ TIEMPO: 45 minutos
// 🗣️ HABLA EN VOZ ALTA mientras programas

// REQUERIMIENTOS:
// 1. Componente que muestre lista de usuarios
// 2. Consumir API (JSONPlaceholder)
// 3. Implementar búsqueda en tiempo real
// 4. Loading state
// 5. Error handling
// 6. Responsive design

// 📝 PLAN DE EJECUCIÓN (narrar en voz alta):

// "Voy a empezar creando la estructura básica del componente..."
@Component({
  selector: 'app-user-list',
  template: `
    <div class="user-list-container">
      <!-- Search input -->
      <div class="search-section">
        <input
          type="text"
          placeholder="Search users..."
          [formControl]="searchControl"
          class="search-input">
      </div>

      <!-- Loading state -->
      <div *ngIf="loading" class="loading">
        Loading users...
      </div>

      <!-- Error state -->
      <div *ngIf="error" class="error">
        {{ error }}
        <button (click)="loadUsers()">Retry</button>
      </div>

      <!-- User list -->
      <div class="users-grid" *ngIf="!loading && !error">
        <div
          *ngFor="let user of filteredUsers$ | async"
          class="user-card">
          <h3>{{ user.name }}</h3>
          <p>{{ user.email }}</p>
          <p>{{ user.company?.name }}</p>
        </div>
      </div>

      <!-- Empty state -->
      <div
        *ngIf="(filteredUsers$ | async)?.length === 0 && !loading"
        class="empty-state">
```

```
          No users found
        </div>
      </div>
    `,
    styles: [`
      .user-list-container {
        max-width: 1200px;
        margin: 0 auto;
        padding: 20px;
      }

      .search-input {
        width: 100%;
        padding: 12px;
        margin-bottom: 20px;
        border: 1px solid #ddd;
        border-radius: 4px;
        font-size: 16px;
      }

      .users-grid {
        display: grid;
        grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
        gap: 20px;
      }

      .user-card {
        border: 1px solid #eee;
        border-radius: 8px;
        padding: 16px;
        background: white;
        box-shadow: 0 2px 4px rgba(0,0,0,0.1);
      }

      .loading, .error, .empty-state {
        text-align: center;
        padding: 40px;
      }

      .error {
        color: #d32f2f;
      }

      @media (max-width: 768px) {
        .users-grid {
          grid-template-columns: 1fr;
        }
      }
    `]
  })
  export class UserListComponent implements OnInit, OnDestroy {
    // "Primero voy a definir las propiedades que necesito..."
    searchControl = new FormControl('');
    users$ = new BehaviorSubject<User[]>([]);
```

```
  filteredUsers$ = new BehaviorSubject<User[]>([]);
  loading = false;
  error: string | null = null;

  private destroy$ = new Subject<void>();

  constructor(private http: HttpClient) {}

  ngOnInit() {
    // "En ngOnInit voy a cargar los usuarios y configurar la búsqueda..."
    this.loadUsers();
    this.setupSearch();
  }

  ngOnDestroy() {
    // "Importante limpiar subscripciones para evitar memory leaks..."
    this.destroy$.next();
    this.destroy$.complete();
  }

  private loadUsers() {
    // "Método para cargar usuarios de la API con manejo de errores..."
    this.loading = true;
    this.error = null;

    this.http.get<User[]>('https://jsonplaceholder.typicode.com/users')
      .pipe(takeUntil(this.destroy$))
      .subscribe({
        next: (users) => {
          this.users$.next(users);
          this.filteredUsers$.next(users);
          this.loading = false;
        },
        error: (error) => {
          this.error = 'Failed to load users. Please try again.';
          this.loading = false;
          console.error('Error loading users:', error);
        }
      });
  }

  private setupSearch() {
    // "Configurar búsqueda con debounce para mejor performance..."
    this.searchControl.valueChanges.pipe(
      debounceTime(300),
      distinctUntilChanged(),
      takeUntil(this.destroy$)
    ).subscribe(searchTerm => {
      this.filterUsers(searchTerm || '');
    });
  }

  private filterUsers(searchTerm: string) {
    // "Filtrar usuarios por nombre, email o empresa..."
```

```
    const users = this.users$.value;

    if (!searchTerm.trim()) {
      this.filteredUsers$.next(users);
      return;
    }

    const filtered = users.filter(user =>
      user.name.toLowerCase().includes(searchTerm.toLowerCase()) ||
      user.email.toLowerCase().includes(searchTerm.toLowerCase()) ||
      user.company?.name.toLowerCase().includes(searchTerm.toLowerCase())
    );

    this.filteredUsers$.next(filtered);
  }
}

// "También necesito definir la interface User..."
interface User {
  id: number;
  name: string;
  email: string;
  company?: {
    name: string;
  };
}
```

📊 EVALUACIÓN:

```
☑ CRITERIOS DE EVALUACIÓN:

Planificación (20%):
- Explicó el approach antes de codear
- Identificó componentes necesarios
- Consideró edge cases

Código limpio (25%):
- Nombres descriptivos
- Separación de responsabilidades
- Estructura clara

Funcionalidad (25%):
- Cumple todos los requerimientos
- Maneja estados correctamente
- Loading y error handling

Comunicación (20%):
- Explicó decisiones técnicas
- Narró el proceso claramente
- Respondió preguntas efectivamente
```

```
Mejores prácticas (10%):
- Unsubscribe pattern
- TypeScript typing
- Responsive design
```

Preguntas conceptuales preparadas:

❓ "¿Diferencia entre Observable y Promise?"

```
☑ RESPUESTA ESTRUCTURADA:

"Las principales diferencias son:

1. VALORES:
   - Promise: Emite un solo valor y se completa
   - Observable: Puede emitir múltiples valores en el tiempo

2. EJECUCIÓN:
   - Promise: Eager - se ejecuta inmediatamente
   - Observable: Lazy - no se ejecuta hasta subscribe()

3. CANCELACIÓN:
   - Promise: No cancelable
   - Observable: Cancelable con unsubscribe()

4. OPERATORS:
   - Promise: Limitados (.then, .catch, .finally)
   - Observable: Muchos operators (map, filter, switchMap, etc.)

En Angular usamos Observables especialmente para HTTP calls porque
podemos cancelarlos si el componente se destruye, evitando memory leaks."
```

❓ "¿Qué es Change Detection en Angular?"

```
☑ RESPUESTA:

"Change Detection es el mecanismo que Angular usa para detectar cambios
en el modelo de datos y actualizar la vista.

PROCESO:
1. Angular ejecuta un ciclo de detección desde el componente root
2. Verifica cada componente en el árbol para cambios
3. Actualiza el DOM si encuentra diferencias

TRIGGERS:
- Eventos del DOM (click, input)
- HTTP requests
- Timers (setTimeout, setInterval)
- Promises y Observables
```

```
OPTIMIZACIÓN:
- OnPush strategy: Solo revisa cuando cambian @Input o se emite evento
- ChangeDetectorRef para control manual
- Immutable data para mejor performance

En mi experiencia, usar OnPush con immutable data mejora significativamente
la performance en aplicaciones complejas."
```

🤝 Días 6-7: Entrevista con Team Lead

Scenarios de trabajo:

❓ "Un stakeholder quiere cambiar requirements a mitad del sprint"

```
☑ ENFOQUE ESTRUCTURADO:

1. EVALUACIÓN:
   "Primero evaluaría el impacto técnico y de tiempo del cambio,
   consultando con el equipo de desarrollo."

2. COMUNICACIÓN:
   "Explicaría claramente las consecuencias: retraso en delivery,
   posible technical debt, o necesidad de re-priorizar otras features."

3. ALTERNATIVAS:
   "Propondría alternativas como:
   - Implementar el cambio en el siguiente sprint
   - Crear una versión simplificada para este sprint
   - Evaluar si realmente agrega valor al usuario final"

4. DECISIÓN COLABORATIVA:
   "Facilitaría una discusión entre stakeholder, product owner y equipo
   para tomar la mejor decisión basada en datos y prioridades de negocio."

MI EXPERIENCIA:
"En la Universidad de Caldas manejé situaciones similares donde proveedores
cambiaban especificaciones. Mi approach siempre fue transparencia total
sobre impactos y buscar win-win solutions."
```

❓ "¿Cómo documentarías tu código para otros developers?"

```
☑ ESTRATEGIA INTEGRAL:

1. CÓDIGO AUTO-DOCUMENTADO:
   - Nombres descriptivos para variables y funciones
   - Funciones pequeñas con responsabilidad única
   - Estructura clara y consistente
```

````
2. COMENTARIOS ESTRATÉGICOS:
   - JSDoc para métodos públicos complejos
   - Comentarios para lógica de negocio no obvia
   - TODO/FIXME para items pendientes

3. DOCUMENTACIÓN ESTRUCTURAL:
   - README detallado con setup instructions
   - Architecture decision records (ADRs)
   - API documentation para services

4. EJEMPLOS PRÁCTICOS:
   - Unit tests como documentación viva
   - Storybook para componentes reutilizables
   - Code examples en documentación

EJEMPLO CONCRETO:
```typescript
/**
 * Calculates user score based on prediction accuracy
 *
 * @param predictions - User's match predictions
 * @param actualResults - Actual match results
 * @returns Total score with breakdown by category
 *
 * @example
 * const score = calculateUserScore(
 *   [{ matchId: 1, homeGoals: 2, awayGoals: 1 }],
 *   [{ matchId: 1, homeGoals: 2, awayGoals: 0 }]
 * );
 * // Returns: { total: 1, exact: 0, winner: 1, goals: 0 }
 */
function calculateUserScore(predictions: Prediction[], actualResults: Result[]):
ScoreBreakdown {
  // Implementation with clear variable names and logic comments
}
````

? "¿Cómo priorizarías features cuando tienes recursos limitados?"

```
☑ FRAMEWORK DE PRIORIZACIÓN:

1. IMPACTO EN USUARIO:
   - ¿Resuelve un pain point crítico?
   - ¿Afecta a muchos usuarios o pocos?
   - ¿Mejora métricas clave (retention, conversion)?

2. ESFUERZO TÉCNICO:
   - Complejidad de implementación
   - Dependencies con otros sistemas
   - Risk técnico y posibles blockers

3. VALOR DE NEGOCIO:
   - ROI esperado
```
```

```
   - Alignment con objetivos estratégicos
   - Urgencia competitiva

4. TECHNICAL DEBT:
   - Impacto en mantenibilidad
   - Performance implications
   - Security considerations

METODOLOGÍA:
"Usaría una matriz Impact/Effort para categorizar features:
- High Impact, Low Effort: Prioridad 1
- High Impact, High Effort: Evaluar timing
- Low Impact, Low Effort: Quick wins
- Low Impact, High Effort: Eliminar

MI EXPERIENCIA:
En mi background industrial, constantemente priorizaba mejoras de proceso
con recursos limitados. El key es data-driven decisions y stakeholder
alignment claro."
```

🚀 FASE 4: OPTIMIZACIÓN FINAL (Semana 5)

📊 Días 1-3: Análisis y Mejora

Revisión de preparación:

📞 Análisis de mock interviews:

```
☑ CHECKLIST DE REVISIÓN:

Video de Elevator Pitch:
□ Duración bajo 90 segundos
□ Sin muletillas ("eh", "mm", "o sea")
□ Estructura clara: Background → Transición → Valor → Objetivo
□ Tono confiado y entusiasta
□ Llamada a la acción clara

Video de Live Coding:
□ Planificación verbal antes de codear
□ Explicación continua del proceso
□ Manejo apropiado de errores
□ Código limpio y organizado
□ Testing básico del resultado

Video de Demo de Proyectos:
□ Contexto del problema explicado
□ Tecnologías utilizadas claras
□ Valor de negocio articulado
```

```
□ Demo fluida sin errores técnicos
□ Preparado para preguntas deep-dive
```

🔧 **Identificar puntos débiles:**

```
ÁREAS COMUNES A MEJORAR:

Técnicas:
- Sintaxis específica de Angular/TypeScript
- Patterns de manejo de errores
- Best practices de performance
- Testing y debugging skills

Comunicación:
- Nervios iniciales en video calls
- Explicar conceptos técnicos de manera simple
- Hacer preguntas inteligentes al entrevistador
- Manejar preguntas que no sé responder

Presentación:
- Setup técnico durante live coding
- Tiempo management en ejercicios
- Recovery cuando algo no funciona
- Mostrar curiosidad y ganas de aprender
```

⚡ **Plan de refuerzo intensivo:**

```
DÍAS 1-3 PLAN ESPECÍFICO:

Día 1: Refinamiento técnico
- 2 horas: Practicar conceptos donde tuve dudas
- 1 hora: Refactorizar proyecto con mejores prácticas
- 30 min: Memorizar respuestas a preguntas frecuentes

Día 2: Comunicación y soft skills
- 1 hora: Re-grabar elevator pitch hasta perfección
- 1 hora: Practicar explicar conceptos técnicos de manera simple
- 1 hora: Simular Q&A con preguntas difíciles

Día 3: Portfolio y presencia online
- 2 horas: Pulir README de proyectos en GitHub
- 1 hora: Actualizar LinkedIn con keywords específicas
- 30 min: Verificar que todos los links funcionen
```

📈 Optimización de CV y LinkedIn

Keywords específicas para ofertas objetivo:

```
ANGULAR ESPECÍFICO:
☑ Angular 15+, Angular CLI, Angular Material
☑ TypeScript, JavaScript ES6+, RxJS
☑ Reactive Forms, Component Architecture
☑ HTTP Client, Observables, Services
☑ Unit Testing, Jasmine, Karma

FRONTEND GENERAL:
☑ HTML5, CSS3, SCSS/SASS
☑ Responsive Design, Mobile-First
☑ Git, GitHub, Version Control
☑ REST APIs, JSON, HTTP
☑ Cross-browser Compatibility

METODOLOGÍAS:
☑ Agile, Scrum, Kanban
☑ SOLID Principles, Clean Code
☑ Atomic Design, BEM
☑ Code Review, Documentation
☑ Problem Solving, Debugging

SOFT SKILLS ÚNICOS:
☑ Stakeholder Management
☑ Requirements Analysis
☑ Process Optimization
☑ Quality Assurance
☑ Technical Documentation
```

LinkedIn profile optimization:

```
HEADLINE OPTIMIZADO:
"Frontend Developer (Angular/TypeScript) | Ingeniero Industrial con 12+ años
optimizando procesos | Especialista en traducir requisitos complejos en soluciones
web escalables"

SUMMARY ACTUALIZADO:
"Frontend Developer especializado en Angular y TypeScript, con background único en
Ingeniería Industrial y 12+ años optimizando procesos empresariales.

🔧 STACK TÉCNICO:
• Angular 15+, TypeScript, JavaScript ES6+
• HTML5, CSS3, SCSS, Responsive Design
• RxJS, Reactive Forms, HTTP Client
• Git, Testing, Code Review

💼 VALOR DIFERENCIAL:
• Traducción de requisitos complejos en soluciones técnicas
• Gestión efectiva de stakeholders multidisciplinarios
• Aplicación de metodologías de calidad al desarrollo
• Pensamiento sistémico para arquitecturas escalables
```

```
📁 PORTFOLIO:
• SerTech: Gestión de servicios técnicos (Angular + TypeScript)
• Planeta Predictor: App de pollas deportivas (RxJS + Local Storage)
• Movie App: Explorador de películas (API Integration + UX)

🎯 OBJETIVO:
Busco oportunidad como Frontend Developer donde pueda combinar código limpio con
visión de negocio para crear productos que realmente resuelvan problemas de
usuarios."

SKILLS SECTION:
• Angular (Endorsed: solicitar a contactos)
• TypeScript (Endorsed)
• JavaScript (Endorsed)
• Frontend Development (Endorsed)
• Process Optimization (Endorsed)
• Stakeholder Management (Endorsed)
```

🎯 Días 4-5: Preparación Específica por Empresa

Research profundo template:

EMPRESA: [Nombre]

```
📊 INFORMACIÓN BÁSICA:
Fundación: [Año]
Tamaño: [Número empleados]
Industria: [Sector específico]
Funding: [Última ronda, total raised]
Ubicación: [Oficinas principales]

🛠️ TECHNOLOGY STACK:
Frontend: [React, Angular, Vue?]
Backend: [Node.js, Java, Python?]
Database: [MySQL, PostgreSQL, MongoDB?]
Cloud: [AWS, Azure, GCP?]
Tools: [Slack, Jira, Git workflow?]

🧮 PRODUCTOS PRINCIPALES:
1. [Producto 1]: [Descripción breve]
2. [Producto 2]: [Descripción breve]
3. [Producto 3]: [Descripción breve]

🎯 CULTURE & VALUES:
- [Valor 1 de la empresa]
- [Valor 2 de la empresa]
- [Valor 3 de la empresa]

📰 NOTICIAS RECIENTES:
- [Noticia relevante 1]
```

```
- [Noticia relevante 2]
- [Expansión, nuevos productos, etc.]

👥 PERSONAS CLAVE EN LINKEDIN:
- CTO: [Nombre] - [Background]
- Engineering Manager: [Nombre] - [Background]
- HR Lead: [Nombre] - [Background]

❓ PREGUNTAS ESPECÍFICAS PARA HACER:
1. "Veo que recientemente [noticia específica], ¿cómo impacta esto al equipo de
desarrollo?"
2. "Su stack incluye [tecnología específica], ¿cómo ven la evolución hacia
[tecnología relacionada]?"
3. "Me interesa su enfoque en [valor específico de la empresa], ¿cómo se refleja
esto en el día a día del equipo?"
```

Personalización de pitch por empresa:

EJEMPLO: Startup E-commerce

```
"Mi experiencia optimizando procesos de compra en el sector industrial me da
perspectiva única para mejorar user experience en e-commerce. Entiendo los
pain points de usuarios en procesos transaccionales complejos.

Veo que usan Angular en su stack, que es exactamente mi especialización, y
me emociona la oportunidad de contribuir especialmente en la optimización
del checkout flow y la gestión de inventarios en tiempo real."
```

EJEMPLO: Fintech

```
"Mi background gestionando procesos de adquisiciones con normativas estrictas
me dio experiencia valiosa en compliance y documentación detallada - skills
críticos en fintech.

Me interesa especialmente cómo traducen regulaciones complejas en interfaces
intuitivas para usuarios, algo donde mi experiencia bridgeando mundos técnicos
y de negocio sería muy valiosa."
```

EJEMPLO: Consultora de Software

```
"Con 12+ años gestionando stakeholders de diferentes industrias, entiendo los
challenges únicos de cada cliente. Esta experiencia me permitiría ser más
efectivo desarrollando soluciones personalizadas.

Mi combinación de skills técnicos en Angular y experiencia consultiva me
posicionaría perfectamente para roles client-facing donde necesitan alguien
que hable tanto código como negocio."
```

☑ Días 6-7: Checklist Final

Preparación técnica final:

```
☑ PUEDEN HACER SIN AYUDA (TEST FINAL):

Angular Basics:
□ Crear componente con @Input/@Output en 10 minutos
□ Implementar reactive form con validaciones en 15 minutos
□ Configurar routing con guard en 10 minutos
□ Crear service con HTTP call en 10 minutos
□ Manejar observables con RxJS en 15 minutos

JavaScript/TypeScript:
□ Resolver FizzBuzz y explicar el approach
□ Manipular arrays complejos con methods funcionales
□ Implementar async/await con error handling
□ Crear interfaces y types básicos en TypeScript
□ Explicar diferencias clave entre conceptos (Promise vs Observable)

Debugging & Problem Solving:
□ Identificar errores comunes en console
□ Usar browser dev tools efectivamente
□ Explicar approach para debugging paso a paso
□ Manejar situations cuando no sé algo específico
```

Preparación soft skills final:

```
☑ COMUNICACIÓN PERFECCIONADA:

Elevator Pitch:
□ Fluido bajo 90 segundos
□ Adaptable a diferentes audiencias
□ Incluye call-to-action claro
□ Transmite confianza y entusiasmo

Historia de Transición:
□ Convincente y bien estructurada
□ Enfatiza valor agregado, no limitaciones
□ Conecta experiencia pasada con role futuro
□ Demuestra pasión genuina por tecnología

Manejo de Preguntas Difíciles:
□ "No sé esto específicamente, pero mi approach sería..."
□ Convertir debilidades en oportunidades de crecimiento
□ Hacer preguntas inteligentes cuando apropado
□ Mostrar curiosidad y ganas de aprender
```

Setup técnico final:

```
☑ TECNOLOGÍA LISTA:

Internet y Backup:
□ Conexión estable >10 Mbps
□ Hotspot móvil configurado como backup
□ Router ubicado cerca del área de entrevista
□ Speed test realizado día anterior

Hardware:
□ Cámara funcionando a altura de ojos
□ Audio claro (auriculares con micrófono)
□ Iluminación frontal adecuada
□ Background profesional y neutro
□ Laptop cargada + cargador conectado

Software:
□ VS Code actualizado con proyectos abiertos
□ Browser con bookmarks organizados
□ Zoom/Teams/Meet funcionando correctamente
□ Portfolio deployed y links verificados
□ GitHub profile actualizado
```

Documentos y links finales:

```
☑ MATERIALES ORGANIZADOS:

En Desktop de fácil acceso:
□ CV en PDF (versión más reciente)
□ Portfolio links en documento de texto
□ Lista de preguntas para hacer al entrevistador
□ Notas de research por empresa
□ Contact info del recruiter/interviewer

Bookmarks organizados:
□ Angular docs
□ MDN JavaScript reference
□ TypeScript handbook
□ RxJS documentation
□ Company websites de empresas target

Links funcionando:
□ SerTech: https://avila2601.github.io/SerTech/
□ Planeta Predictor: https://avila2601.github.io/planetaPredictor/
□ Movie App: https://avila2601.github.io/movieApp/movies
□ GitHub: https://github.com/avila2601
□ LinkedIn: [tu URL personalizada]
```

CRONOGRAMA DIARIO RECOMENDADO

Lunes a Viernes:

7:00-8:30: Práctica técnica intensiva (1.5h)

- Live coding exercises
- Conceptos específicos identificados como débiles
- Refactoring de proyectos existentes

12:00-12:30: Repaso conceptual (30min)

- Flashcards de preguntas frecuentes
- Review de documentación específica
- Quick coding challenges

19:00-20:00: Soft skills / pitch practice (1h)

- Grabación y análisis de elevator pitch
- Simulación de Q&A
- Research de empresas target

Fines de Semana:

Sábados (3-4 horas):

- Mock interviews completas
- Revisión integral de performance
- Ajustes basados en feedback

Domingos (2-3 horas):

- Research profundo de empresas
- Actualización de CV y LinkedIn
- Planificación de la semana siguiente

MÉTRICAS DE ÉXITO

Semana 1-2: Fundamentos Técnicos

- [ ] Completar ejercicios Angular sin consultar IA
- [ ] Resolver 5+ algoritmos JavaScript básicos
- [ ] Crear 2 proyectos pequeños adicionales
- [ ] Explicar conceptos clave en video de 3 min c/u

Semana 3: Preparación de Discurso

- [ ] Elevator pitch fluido bajo 90 segundos

- ☐ Respuestas preparadas para 15+ preguntas frecuentes
- ☐ Demo scripts para cada proyecto (3-5 min)
- ☐ Setup técnico para video entrevistas optimizado

☑ Semana 4: Simulación de Entrevistas

- ☐ Mock interview HR scoring 8+/10
- ☐ Live coding exercise completado exitosamente
- ☐ Manejo confiado de preguntas técnicas difíciles
- ☐ Lista de 20+ preguntas inteligentes para hacer

☑ Semana 5: Optimización Final

- ☐ Research completado para 10+ empresas target
- ☐ LinkedIn optimizado con keywords específicas
- ☐ Portfolio deployed sin errores
- ☐ Plan de seguimiento post-entrevista definido

📚 RECURSOS DE REFERENCIA

🌐 Links Esenciales:

- **Angular Docs:** https://angular.io/docs
- **MDN JavaScript:** https://developer.mozilla.org/en-US/docs/Web/JavaScript
- **TypeScript Handbook:** https://www.typescriptlang.org/docs/
- **RxJS Documentation:** https://rxjs.dev/guide/overview
- **JSONPlaceholder API:** https://jsonplaceholder.typicode.com

📖 Cheat Sheets:

- **Angular Cheat Sheet:** https://angular.io/guide/cheatsheet
- **TypeScript Cheat Sheet:** https://www.typescriptlang.org/cheatsheets
- **JavaScript ES6+ Reference:** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

🛠 Herramientas:

- **Can I Use:** https://caniuse.com
- **RegExp Tester:** https://regex101.com
- **JSON Formatter:** https://jsonformatter.org
- **Speed Test:** https://speedtest.net

🎉 MENSAJE FINAL

Este plan está diseñado específicamente para tu perfil único como Ingeniero Industrial transitioning hacia Frontend Development. Tu experiencia de 12+ años en gestión de procesos y stakeholders es una **ventaja**

diferencial massive en el mercado tech.

La clave del éxito será:

1. **Consistencia** en la práctica diaria
2. **Honestidad** sobre tu journey y ganas de aprender
3. **Confianza** en el valor único que aportas
4. **Persistencia** - el proceso puede tomar tiempo pero tu perfil es valioso

¡Tu combinación de experiencia empresarial + skills técnicas modernas es exactamente lo que muchas empresas necesitan!

Creado: 26 de agosto, 2025
Versión: 1.0
Para: Diego Fernando Avila Gómez
Objetivo: Primera posición como Frontend Developer Junior