

# TradeSwitch App

## Shared TypeScript Code

Generated: 11/25/2025, 6:42:54 PM

Total Files: 53

### Table of Contents

1. shared\components (1 files)
2. shared\components\birthday-input (1 files)
3. shared\components\create-account-popup (1 files)
4. shared\components\global-alert (1 files)
5. shared\components\loading-spinner (1 files)
6. shared\components\order-summary (1 files)
7. shared\components\password-input (1 files)
8. shared\components\phone-input (1 files)
9. shared\components\plan-banner (1 files)

10. shared\components\plan-limitation-modal (1 files)
11. shared\components\strategy-card (2 files)
12. shared\components\strategy-guide-modal (1 files)
13. shared\components\subscription-processing (1 files)
14. shared\components\text-input (1 files)
15. shared\context (5 files)
16. shared\interfaces (1 files)
17. shared\mobile-header (1 files)
18. shared\pipes (3 files)
19. shared\pop-ups\alert-popup (1 files)
20. shared\pop-ups\confirm-pop-up (1 files)
21. shared\pop-ups\edit-pop-up (1 files)
22. shared\pop-ups\forgot-password (1 files)
23. shared\pop-ups\loading-pop-up (1 files)
24. shared\pop-ups\stripe-loader-popup (1 files)
25. shared\services (20 files)
26. shared\sidebar-menu (1 files)
27. shared\utils (1 files)

## shared\components

### shared\components\index.ts

---

```
1 export { TextInputComponent } from './text-input/text-input.component';
2 export { PhoneInputComponent } from './phone-input/phone-input.component';
3 export { BirthdayInputComponent } from './birthday-input/birthday-input.component';
4 export { PasswordInputComponent } from './password-input/password-input.component';
5 export { StrategyCardComponent } from './strategy-card/strategy-card.component';
6 export type { StrategyCardData } from './strategy-card/strategy-card.interface';
7 export { StrategyGuideModalComponent } from './strategy-guide-modal/strategy-guide-
8 modal.component';
9 export { LoadingSpinnerComponent } from './loading-spinner/loading-spinner.component';
10 export { PlanBannerComponent } from './plan-banner/plan-banner.component';
```

## shared\components\birthday-input

### shared\components\birthday-input\birthday-input.component.ts

---

```
1 import { Component, Input, forwardRef } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { ControlValueAccessor, FormsModule, ReactiveFormsModule, NG_VALUE_ACCESSOR } from
4 '@angular/forms';
5 interface DateValue {
6   month: string;
7   day: string;
8   year: string;
9 }
10
11 /**
12  * Birthday input component with month, day, and year dropdowns.
13  *
14  * This component provides a date input for birthdays using three separate
15  * dropdowns for month, day, and year. It generates valid date ranges and
16  * handles date formatting for form submission.
17  *
18  * Features:
19  * - Angular Forms integration (ControlValueAccessor)
20  * - Three separate dropdowns (month, day, year)
21  * - Year range: 100 years ago to 13 years ago (minimum age validation)
22  * - Day range: 1-31 (adjusts based on month/year)
23  * - Date formatting (YYYY-MM-DD)
24  * - Customizable label
25  * - Touch state tracking
26  *
27  * Date Range:
28  * - Years: Current year - 100 to Current year - 13
29  * - Days: 1-31 (validated based on month)
30  * - Months: January through December
31  *
32  * Usage:
33  * <app-birthday-input
34  *   FormControlName="birthday"
35  *   label="Birthday"
36  *   [required]="true">
37  * </app-birthday-input>
38  *
39  * Relations:
40  * - Used in registration and profile forms
41  * - Integrates with Angular Reactive Forms
42  */
```

```

43 * @component
44 * @selector app-birthday-input
45 * @standalone true
46 */
47 @Component({
48   selector: 'app-birthday-input',
49   standalone: true,
50   imports: [CommonModule, FormsModule, ReactiveFormsModule],
51   templateUrl: './birthday-input.component.html',
52   styleUrls: ['./birthday-input.component.scss'],
53   providers: [
54     {
55       provide: NG_VALUE_ACCESSOR,
56       useExisting: forwardRef(() => BirthdayInputComponent),
57       multi: true
58     }
59   ]
60 })
61 export class BirthdayInputComponent implements ControlValueAccessor {
62   @Input() label: string = 'Birthday';
63   @Input() required: boolean = false;
64   @Input() disabled: boolean = false;
65
66   dateValue: DateValue = { month: '', day: '', year: '' };
67   touched: boolean = false;
68   showMonthDropdown: boolean = false;
69   showDayDropdown: boolean = false;
70   showYearDropdown: boolean = false;
71
72   months = [
73     { value: '01', label: 'January' },
74     { value: '02', label: 'February' },
75     { value: '03', label: 'March' },
76     { value: '04', label: 'April' },
77     { value: '05', label: 'May' },
78     { value: '06', label: 'June' },
79     { value: '07', label: 'July' },
80     { value: '08', label: 'August' },
81     { value: '09', label: 'September' },
82     { value: '10', label: 'October' },
83     { value: '11', label: 'November' },
84     { value: '12', label: 'December' }
85   ];
86
87   days: string[] = [];
88   years: string[] = [];
89
90   onChange = (value: string) => {};
91   onTouched = () => {};
92
93   constructor() {
94     this.generateDays();
95     this.generateYears();
96   }
97
98   private generateDays(): void {
99     for (let i = 1; i <= 31; i++) {
100       this.days.push(i.toString().padStart(2, '0'));
101     }
102   }
103
104   private generateYears(): void {
105     const currentYear = new Date().getFullYear();
106     for (let i = currentYear - 100; i <= currentYear - 13; i++) {
107       this.years.push(i.toString());
108     }
109     this.years.reverse();
110   }
111
112   writeValue(value: string): void {

```

```

113     if (value) {
114         const date = new Date(value);
115         this.dateValue = {
116             month: (date.getMonth() + 1).toString().padStart(2, '0'),
117             day: date.getDate().toString().padStart(2, '0'),
118             year: date.getFullYear().toString()
119         };
120     }
121 }
122
123 registerOnChange(fn: any): void {
124     this.onChange = fn;
125 }
126
127 registerOnTouched(fn: any): void {
128     this.onTouched = fn;
129 }
130
131 setDisabledState(isDisabled: boolean): void {
132     this.disabled = isDisabled;
133 }
134
135 onChange(): void {
136     if (this.dateValue.month && this.dateValue.day && this.dateValue.year) {
137         const dateString = `${this.dateValue.year}-${this.dateValue.month}-${this.dateValue.day}`;
138         this.onChange(dateString);
139     } else {
140         this.onChange('');
141     }
142 }
143
144 onBlur(): void {
145     if (!this.touched) {
146         this.touched = true;
147         this.onTouched();
148     }
149 }
150
151 onMonthFocus(): void {
152     this.showMonthDropdown = true;
153     this.showDayDropdown = false;
154     this.showYearDropdown = false;
155 }
156
157 onDayFocus(): void {
158     this.showMonthDropdown = false;
159     this.showDayDropdown = true;
160     this.showYearDropdown = false;
161 }
162
163 onYearFocus(): void {
164     this.showMonthDropdown = false;
165     this.showDayDropdown = false;
166     this.showYearDropdown = true;
167 }
168
169 onMonthBlur(): void {
170     this.showMonthDropdown = false;
171     this.onBlur();
172 }
173
174 onDayBlur(): void {
175     this.showDayDropdown = false;
176     this.onBlur();
177 }
178
179 onYearBlur(): void {
180     this.showYearDropdown = false;
181     this.onBlur();
182 }

```

```
183 }  
184
```

## Ø=ÜÁ shared\components\create-account-popup

### Ø=ÜÄ shared\components\create-account-popup\create-account-popup.component.ts

---

```
1 import { Component, Input, Output, EventEmitter, OnChanges, SimpleChanges } from '@angular/  
2 import { CommonModule } from '@angular/common';  
3 import { FormsModule } from '@angular/forms';  
4 import { TradeLockerApiService } from '../../../services/tradelocker-api.service';  
5 import { AuthService } from '../../../features/auth/service/authService';  
6 import { AccountData } from '../../../features/auth/models/userModel';  
7 import { Timestamp } from 'firebase/firestore';  
8 import { NumberFormatterService } from '../../../utils/number-formatter.service';  
9  
10 /**  
11  * Component for creating and editing trading accounts.  
12  *  
13  * This component provides a modal interface for adding new trading accounts  
14  * or editing existing ones. It validates account credentials with TradeLocker  
15  * API and handles account creation/update operations.  
16  *  
17  * Features:  
18  * - Create new trading accounts  
19  * - Edit existing trading accounts  
20  * - Account validation with TradeLocker API  
21  * - Formatted balance input with currency formatting  
22  * - Confirmation modals (cancel, success)  
23  * - Form validation  
24  * - Account credential validation  
25  *  
26  * Account Fields:  
27  * - Account name  
28  * - Broker  
29  * - Email (trading account email)  
30  * - Password (broker password)  
31  * - Server  
32  * - Account ID  
33  * - Account number  
34  * - Initial balance (formatted currency input)  
35  *  
36  * Validation:  
37  * - Validates account exists in TradeLocker before creation  
38  * - Checks account credentials with TradeLocker API  
39  * - Form field validation  
40  *  
41  * Relations:  
42  * - AuthService: Creates/updates accounts in Firebase  
43  * - TradeLockerApiService: Validates account credentials  
44  * - NumberFormatterService: Formats balance input  
45  *  
46  * @component  
47  * @selector app-create-account-popup  
48  * @standalone true  
49  */  
50 @Component({  
51   selector: 'app-create-account-popup',  
52   standalone: true,  
53   imports: [CommonModule, FormsModule],  
54   templateUrl: './create-account-popup.component.html',  
55   styleUrls: ['./create-account-popup.component.scss']  
56 })
```

```

57 export class CreateAccountPopupComponent implements OnChanges {
58     @Input() visible = false;
59     @Input() userId: string = '';
60     @Input() currentAccountCount: number = 0;
61     @Input() editMode = false;
62     @Input() accountToEdit: AccountData | null = null;
63     @Output() close = new EventEmitter<void>();
64     @Output() create = new EventEmitter<any>();
65     @Output() update = new EventEmitter<any>();
66     @Output() cancel = new EventEmitter<void>();
67
68     constructor(
69         private authService: AuthService,
70         private tradeLockerApiService: TradeLockerApiService,
71         private numberFormatter: NumberFormatterService
72     ) {}
73
74     ngOnChanges(changes: SimpleChanges) {
75         // When the popup becomes visible, reset or populate the form
76         if (changes['visible'] && changes['visible'].currentValue === true) {
77             if (this.editMode && this.accountToEdit) {
78                 this.populateFormForEdit();
79             } else {
80                 this.resetForm();
81             }
82         }
83     }
84
85     // Form data
86     newAccount = {
87         emailTradingAccount: '',
88         brokerPassword: '',
89         accountName: '',
90         broker: '',
91         server: '',
92         accountID: '',
93         initialBalance: 0,
94         accountNumber: 0,
95     };
96
97     // Properties for formatted balance input
98     initialBalanceInput: string = '';
99     initialBalanceDisplay: string = '';
100
101     // Confirmation modals
102     showCancelConfirm = false;
103     showSuccessModal = false;
104
105     onClose() {
106         this.close.emit();
107     }
108
109     onCancel() {
110         this.showCancelConfirm = true;
111     }
112
113     onConfirmCancel() {
114         this.showCancelConfirm = false;
115         this.cancel.emit();
116     }
117
118     onContinueEditing() {
119         this.showCancelConfirm = false;
120     }
121
122
123
124     async onCreate() {
125         // Basic validation

```

```

127     if (!this.newAccount.accountName || !this.newAccount.broker ||
128         !this.newAccount.emailTradingAccount || !this.newAccount.brokerPassword ||
129         !this.newAccount.server || !this.newAccount.accountID || !
130         this.newAccount.accountNumber || !this.newAccount.initialBalance || !this.newAccount.netPnl || !this.newAccount.profit || !this.newAccount.bestTrade) {
131         return;
132     }
133
134     try {
135         // 1. Validate account exists in TradeLocker
136         const accountExists = await this.validateAccountInTradeLocker();
137         if (!accountExists) {
138             alert('Account does not exist in TradeLocker. Please verify the credentials.');
```

139 return;

140 }

141

142 // 2. Validate account email and ID uniqueness

143 const validationResult = await this.validateAccountUniqueness();

144 if (!validationResult.isValid) {

145 alert(validationResult.message);

146 return;

147 }

148

149 if (this.editMode && this.accountToEdit) {

150 // Update existing account

151 await this.updateAccount();

152 } else {

153 // Create new account

154 await this.createNewAccount();

155 }

156

157 } catch (error) {

158 console.error('Error processing trading account:', error);

159 alert(`Failed to \${this.editMode ? 'update' : 'create'} trading account. Please try again.`);

160 }

161 }

162

163 private async createNewAccount() {

164 // Create account object for Firebase

165 const accountData = this.createAccountObject();

166

167 // Save to Firebase

168 await this.authService.createAccount(accountData);

169

170 // Show success modal

171 this.showSuccessModal = true;

172

173 // Emit the created account data

174 this.create.emit(accountData);

175 }

176

177 private async updateAccount() {

178 if (!this.accountToEdit) return;

179

180 // Create updated account object

181 const updatedAccountData: AccountData = {

182 ...this.accountToEdit,

183 accountName: this.newAccount.accountName,

184 broker: this.newAccount.broker,

185 server: this.newAccount.server,

186 emailTradingAccount: this.newAccount.emailTradingAccount,

187 brokerPassword: this.newAccount.brokerPassword,

188 accountID: this.newAccount.accountID,

189 accountNumber: this.newAccount.accountNumber,

190 initialBalance: this.newAccount.initialBalance,

191 netPnl: this.accountToEdit.netPnl || 0,

192 profit: this.accountToEdit.profit || 0,

193 bestTrade: this.accountToEdit.bestTrade || 0,

194 };

195

196 // Update in Firebase



```

197     await this.authService.updateAccount(this.accountToEdit.id, updatedAccountData);
198
199     // Show success modal
200     this.showSuccessModal = true;
201
202     // Emit the updated account data
203     this.update.emit(updatedAccountData);
204 }
205
206 onGoToList() {
207     this.showSuccessModal = false;
208     this.close.emit();
209 }
210
211 // Balance input event handlers
212 onInitialBalanceInput(event: Event) {
213     const target = event.target as HTMLInputElement;
214     this.initialBalanceInput = target.value;
215 }
216
217 onInitialBalanceFocus() {
218     // When user focuses, show only the number without formatting for editing
219     if (this.newAccount.initialBalance > 0) {
220         this.initialBalanceInput = this.newAccount.initialBalance.toString();
221     } else {
222         this.initialBalanceInput = '';
223     }
224 }
225
226 onInitialBalanceBlur() {
227     // Convert the value to number
228     const numericValue = this.numberFormatter.parseCurrencyValue(this.initialBalanceInput);
229     if (!isNaN(numericValue) && numericValue >= 0) {
230         // Save the unformatted value
231         this.newAccount.initialBalance = numericValue;
232
233         // Show visual format (only for display)
234         this.initialBalanceDisplay = this.numberFormatter.formatCurrencyDisplay(numericValue);
235
236         // Update the input to show the visual format
237         this.initialBalanceInput = this.initialBalanceDisplay;
238     } else {
239         // If not a valid number, clear
240         this.initialBalanceInput = '';
241         this.initialBalanceDisplay = '';
242         this.newAccount.initialBalance = 0;
243     }
244 }
245
246 resetForm() {
247     this.newAccount = {
248         accountName: '',
249         broker: '',
250         server: '',
251         emailTradingAccount: '',
252         brokerPassword: '',
253         accountID: '',
254         accountNumber: 1, // Default to 1 as suggested
255         initialBalance: 0,
256     };
257
258     // Reset balance input properties
259     this.initialBalanceInput = '';
260     this.initialBalanceDisplay = '';
261 }
262
263 populateFormForEdit() {
264     if (this.accountToEdit) {
265         this.newAccount = {
266             accountName: this.accountToEdit.accountName || '',

```

```

267         broker: this.accountToEdit.broker || '',
268         server: this.accountToEdit.server || '',
269         emailTradingAccount: this.accountToEdit.emailTradingAccount || '',
270         brokerPassword: this.accountToEdit.brokerPassword || '',
271         accountID: this.accountToEdit.accountID || '',
272         accountNumber: this.accountToEdit.accountNumber || 1,
273         initialBalance: this.accountToEdit.initialBalance || 0,
274     };
275
276     // Set balance input values for display with currency format
277     if (this.newAccount.initialBalance > 0) {
278         this.initialBalanceDisplay =
279             this.numberFormatter.formatCurrency(this.newAccount.initialBalance); // New formatted value
280     } else {
281         this.initialBalanceInput = '';
282         this.initialBalanceDisplay = '';
283     }
284 }
285
286 private createAccountObject(): AccountData {
287     const timestamp = Date.now().toString(36);
288     const randomPart = Math.random().toString(36).substring(2, 8);
289     const uniqueId = `id_${timestamp}_${randomPart}`;
290
291     return {
292         id: uniqueId,
293         userId: this.userId,
294         emailTradingAccount: this.newAccount.emailTradingAccount,
295         brokerPassword: this.newAccount.brokerPassword,
296         broker: this.newAccount.broker,
297         server: this.newAccount.server,
298         accountName: this.newAccount.accountName,
299         accountID: this.newAccount.accountID,
300         accountNumber: this.newAccount.accountNumber,
301         initialBalance: this.newAccount.initialBalance,
302         createdAt: Timestamp.now(),
303         netPnl: 0,
304         profit: 0,
305         bestTrade: 0,
306     };
307 }
308
309 /**
310  * Validates if the account exists in TradeLocker
311  * Simply tries to get JWT token - if successful, credentials are valid
312  */
313 private async validateAccountInTradeLocker(): Promise<boolean> {
314     try {
315         return await this.tradeLockerApiService.validateAccount({
316             email: this.newAccount.emailTradingAccount,
317             password: this.newAccount.brokerPassword,
318             server: this.newAccount.server
319         });
320     } catch (error) {
321         console.error('Error validating account in TradeLocker:', error);
322         return false;
323     }
324 }
325
326 /**
327  * Validates that broker + server + accountId combination is unique across all accounts
328  * Returns validation result with appropriate message
329  */
330 private async validateAccountUniqueness(): Promise<{isValid: boolean, message: string}> {
331     try {
332         // Check if broker + server + accountId combination already exists
333         const accountExists = await this.authService.checkAccountExists(
334             this.newAccount.broker,
335             this.newAccount.server,

```

```

337         this.newAccount.accountID,
338         this.userId
339     );
340
341     if (accountExists) {
342         return {
343             isValid: false,
344             message: 'This account is already registered. Try with another account or delete
345 the existing trade account it is linked to.'
346         }
347
348         return {
349             isValid: true,
350             message: ''
351         };
352     } catch (error) {
353         return {
354             isValid: false,
355             message: 'This account is already registered. Try with another account or delete the
356 existing trade account it is linked to.'
357         }
358     }
359 }
360

```

## Ø=ÜÄ shared\components\global-alert

### Ø=ÜÄ shared\components\global-alert\global-alert.component.ts

---

```

1  import { Component, OnInit, OnDestroy } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { Subscription } from 'rxjs';
4  import { AlertService, AlertConfig } from '../../services/alert.service';
5  import { AlertPopupComponent } from '../../pop-ups/alert-popup/alert-popup.component';
6
7  /**
8   * Global alert component that displays alerts throughout the application.
9   *
10  * This component subscribes to the AlertService and displays alert dialogs
11  * whenever an alert is triggered anywhere in the application. It provides
12  * a centralized way to show user notifications.
13  *
14  * Features:
15  * - Subscribes to AlertService alert stream
16  * - Displays alerts using AlertPopupComponent
17  * - Handles alert close and confirm actions
18  * - Automatic cleanup on component destroy
19  *
20  * Usage:
21  * Should be included in the root app component template to enable global
22  * alert functionality throughout the application.
23  *
24  * Relations:
25  * - AlertService: Source of alert notifications
26  * - AlertPopupComponent: Displays the actual alert UI
27  *
28  * @component
29  * @selector app-global-alert
30  * @standalone true
31  */
32  @Component({
33     selector: 'app-global-alert',
34     standalone: true,

```

```

35     imports: [CommonModule, AlertPopupComponent],
36     template: `
37         <app-alert-popup
38             [visible]="alertVisible"
39             [title]="alertConfig?.title || ''"
40             [message]="alertConfig?.message || ''"
41             [buttonText]="alertConfig?.buttonText || 'OK'"
42             [type]="alertConfig?.type || 'info'"
43             (close)="onClose()"
44             (confirm)="onConfirm()">
45         </app-alert-popup>
46     `
47 })
48 export class GlobalAlertComponent implements OnInit, OnDestroy {
49     alertVisible = false;
50     alertConfig: AlertConfig | null = null;
51     private subscription: Subscription = new Subscription();
52
53     constructor(private alertService: AlertService) {}
54
55     ngOnInit(): void {
56         this.subscription = this.alertService.alert$.subscribe(alert => {
57             this.alertVisible = alert.visible;
58             this.alertConfig = alert.config;
59         });
60     }
61
62     ngOnDestroy(): void {
63         this.subscription.unsubscribe();
64     }
65
66     onClose(): void {
67         this.alertService.hideAlert();
68     }
69
70     onConfirm(): void {
71         this.alertService.hideAlert();
72     }
73 }
74

```

## Ø=ÜÄ shared\components\loading-spinner

### Ø=ÜÄ shared\components\loading-spinner\loading-spinner.component.ts

---

```

1  import { Component, Input } from '@angular/core';
2
3  /**
4   * Component for displaying a loading spinner with customizable title and description.
5   *
6   * This component provides a reusable loading indicator that can be used throughout
7   * the application to show that data is being loaded or an operation is in progress.
8   *
9   * Features:
10  * - Customizable title and description text
11  * - Reusable across the application
12  * - Standalone component (no module dependencies)
13  *
14  * Relations:
15  * - Used by various components to show loading states
16  *
17  * @component
18  * @selector app-loading-spinner

```

```

19  * @standalone true
20  */
21  @Component({
22    selector: 'app-loading-spinner',
23    standalone: true,
24    imports: [],
25    templateUrl: './loading-spinner.component.html',
26    styleUrls: ['./loading-spinner.component.scss']
27  })
28  export class LoadingSpinnerComponent {
29    @Input() title: string = 'Loading...';
30    @Input() description: string = 'Please wait while we load your data.';
31  }
32

```

## shared\components\order-summary

### shared\components\order-summary\order-summary.component.ts

---

```

1  import { Component, Input, Output, EventEmitter } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3
4  /**
5   * Interface for order details display.
6   *
7   * @interface OrderDetails
8   */
9  export interface OrderDetails {
10    planName: string;
11    price: number;
12    currency: string;
13    billing: string;
14    features: {
15      tradingAccounts: number;
16      strategies: number;
17      consistencyRules: boolean;
18      tradingJournal: boolean;
19      liveStatistics: boolean;
20    };
21    discount?: {
22      code: string;
23      amount: number;
24    };
25    taxes: number;
26    total: number;
27  }
28
29  /**
30   * Interface for order summary configuration.
31   *
32   * @interface OrderSummaryConfig
33   */
34  export interface OrderSummaryConfig {
35    context: 'signup' | 'plan-change';
36    planName: string;
37    price: number;
38    userData?: any;
39  }
40
41  /**
42   * Component for displaying order summary before payment.
43   *
44   * This component shows a detailed summary of the selected plan including

```

```

45 * price, features, discounts, taxes, and total. It's used in both signup
46 * and plan change flows to confirm the order before payment.
47 *
48 * Features:
49 * - Display plan details (name, price, features)
50 * - Calculate discounts (10% for new users)
51 * - Calculate taxes (21% VAT)
52 * - Display total amount
53 * - Progress steps indicator
54 * - Context-aware messaging (signup vs plan-change)
55 *
56 * Calculations:
57 * - Discount: 10% of subtotal (applied automatically)
58 * - Taxes: 21% of (subtotal - discount)
59 * - Total: subtotal - discount + taxes
60 *
61 * Plan Features:
62 * - Trading accounts: Varies by plan (Free: 1, Starter: 2, Pro: 6)
63 * - Strategies: Varies by plan (Free: 1, Starter: 3, Pro: 8)
64 * - All plans include: Consistency rules, Trading journal, Live statistics
65 *
66 * Relations:
67 * - Used in signup flow before payment
68 * - Used in plan change flow before payment
69 *
70 * @component
71 * @selector app-order-summary
72 * @standalone true
73 */
74 @Component({
75   selector: 'app-order-summary',
76   standalone: true,
77   imports: [CommonModule],
78   templateUrl: './order-summary.component.html',
79   styleUrls: ['./order-summary.component.scss']
80 })
81 export class OrderSummaryComponent {
82   @Input() config: OrderSummaryConfig = {
83     context: 'signup',
84     planName: 'Free',
85     price: 0
86   };
87   @Output() continue = new EventEmitter<void>();
88
89   get orderDetails(): OrderDetails {
90     const subtotal = this.config.price;
91     const discountAmount = this.getDiscountAmount(subtotal);
92     const taxes = this.calculateTaxes(subtotal - discountAmount);
93     const total = subtotal - discountAmount + taxes;
94
95     return {
96       planName: this.config.planName,
97       price: subtotal,
98       currency: 'USD',
99       billing: 'Monthly',
100       features: {
101         tradingAccounts: this.getTradingAccountsCount(this.config.planName),
102         strategies: this.getStrategiesCount(this.config.planName),
103         consistencyRules: true,
104         tradingJournal: true,
105         liveStatistics: true
106       },
107       discount: discountAmount > 0 ? {
108         code: '10RSTORDER',
109         amount: discountAmount
110       } : undefined,
111       taxes: taxes,
112       total: total
113     };
114   }

```

```

115
116 private getTradingAccountsCount(planName: string): number {
117     switch (planName.toLowerCase()) {
118         case 'free': return 1;
119         case 'starter': return 2;
120         case 'pro': return 6;
121         default: return 1;
122     }
123 }
124
125 private getStrategiesCount(planName: string): number {
126     switch (planName.toLowerCase()) {
127         case 'free': return 1;
128         case 'starter': return 3;
129         case 'pro': return 8;
130         default: return 1;
131     }
132 }
133
134 private getDiscountAmount(subtotal: number): number {
135     // Aplicar descuento del 10% para nuevos usuarios o cambios de plan
136     return subtotal > 0 ? Math.round(subtotal * 0.1 * 100) / 100 : 0;
137 }
138
139 private calculateTaxes(subtotal: number): number {
140     // Calcular impuestos del 21% (IVA)
141     return Math.round(subtotal * 0.21 * 100) / 100;
142 }
143
144 onContinue(): void {
145     this.continue.emit();
146 }
147
148 getContextTitle(): string {
149     switch (this.config.context) {
150         case 'signup':
151             return 'Review your plan details.';
152         case 'plan-change':
153             return 'Review your new plan details.';
154         default:
155             return 'Review your plan details.';
156     }
157 }
158
159 getProgressSteps(): { step: number; label: string; status: 'completed' | 'active' }[] {
160     switch (this.config.context) {
161         case 'signup':
162             return [
163                 { step: 1, label: 'Plan Selection', status: 'completed' },
164                 { step: 2, label: 'Payment', status: 'completed' },
165                 { step: 3, label: 'Summary', status: 'active' }
166             ];
167         case 'plan-change':
168             return [
169                 { step: 1, label: 'Plan Selection', status: 'completed' },
170                 { step: 2, label: 'Payment', status: 'completed' },
171                 { step: 3, label: 'Summary', status: 'active' }
172             ];
173         default:
174             return [
175                 { step: 1, label: 'Plan Selection', status: 'completed' },
176                 { step: 2, label: 'Payment', status: 'completed' },
177                 { step: 3, label: 'Summary', status: 'active' }
178             ];
179     }
180 }
181 }
182

```

## shared\components\password-input

### shared\components\password-input\password-input.component.ts

---

```
1 import { Component, Input, forwardRef } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { ControlValueAccessor, FormsModule, ReactiveFormsModule, NG_VALUE_ACCESSOR,
4 Validator, AbstractControl, ValidationErrors, NG_VALIDATORS } from '@angular/forms';
5 /**
6  * Password input component with validation and strength indicator.
7  *
8  * This component provides a password input field with comprehensive validation
9  * and visual strength feedback. It integrates with Angular Forms and includes
10 * built-in password strength validation.
11 *
12 * Features:
13 * - Angular Forms integration (ControlValueAccessor and Validator)
14 * - Password visibility toggle
15 * - Password strength indicator (weak, medium, strong)
16 * - Real-time validation feedback
17 * - Validation rules:
18 *   - Minimum 8 characters
19 *   - Must contain uppercase letter
20 *   - Must contain lowercase letter
21 *   - Must contain number or symbol
22 *   - Cannot contain user's name or email
23 * - Color-coded strength indicator
24 * - Customizable label and placeholder
25 *
26 * Password Strength:
27 * - Weak: Less than 3 validations pass
28 * - Medium: 3-4 validations pass
29 * - Strong: All 5 validations pass
30 *
31 * Usage:
32 * <app-password-input
33 *   formControlName="password"
34 *   label="Password"
35 *   [showValidation]="true"
36 *   [userEmail]="userEmail"
37 *   [userName]="userName">
38 * </app-password-input>
39 *
40 * Relations:
41 * - Used in registration and password change forms
42 * - Integrates with Angular Reactive Forms
43 *
44 * @component
45 * @selector app-password-input
46 * @standalone true
47 */
48 @Component({
49   selector: 'app-password-input',
50   standalone: true,
51   imports: [CommonModule, FormsModule, ReactiveFormsModule],
52   templateUrl: './password-input.component.html',
53   styleUrls: ['./password-input.component.scss'],
54   providers: [
55     {
56       provide: NG_VALUE_ACCESSOR,
57       useExisting: forwardRef(() => PasswordInputComponent),
58       multi: true
59     },
60     {
61       provide: NG_VALIDATORS,
```



```

62         useExisting: forwardRef(() => PasswordInputComponent),
63         multi: true
64     }
65 }
66 })
67 export class PasswordInputComponent implements ControlValueAccessor, Validator {
68     @Input() label: string = 'Password';
69     @Input() placeholder: string = '*****';
70     @Input() required: boolean = false;
71     @Input() disabled: boolean = false;
72     @Input() showValidation: boolean = false;
73     @Input() userEmail: string = '';
74     @Input() userName: string = '';
75
76     value: string = '';
77     touched: boolean = false;
78     showPassword: boolean = false;
79
80     // Validaciones de contraseña
81     passwordValidations = {
82         noNameOrEmail: false,
83         minLength: false,
84         hasNumberOrSymbol: false,
85         hasUppercase: false,
86         hasLowercase: false
87     };
88
89     get passwordStrength(): string {
90         const validCount = Object.values(this.passwordValidations).filter(Boolean).length;
91         if (validCount < 3) return 'weak';
92         if (validCount < 5) return 'medium';
93         return 'strong';
94     }
95
96     get strengthColor(): string {
97         const strength = this.passwordStrength;
98         switch (strength) {
99             case 'weak': return '#EF4444';
100             case 'medium': return '#F59E0B';
101             case 'strong': return '#22C55E';
102             default: return '#6B7280';
103         }
104     }
105
106     onChange = (value: string) => {};
107     onTouched = () => {};
108
109     writeValue(value: string): void {
110         this.value = value;
111     }
112
113     registerOnChange(fn: any): void {
114         this.onChange = fn;
115     }
116
117     registerOnTouched(fn: any): void {
118         this.onTouched = fn;
119     }
120
121     setDisabledState(isDisabled: boolean): void {
122         this.disabled = isDisabled;
123     }
124
125     onInput(event: Event): void {
126         const value = (event.target as HTMLInputElement).value;
127         this.value = value;
128         this.validatePassword(value);
129         this.onChange(value);
130     }
131

```

```

132 private validatePassword(password: string): void {
133     // No puede contener el nombre o email
134     this.passwordValidations.noNameOrEmail = !this.containsNameOrEmail(password);
135
136     // Al menos 8 caracteres
137     this.passwordValidations.minLength = password.length >= 8;
138
139     // Al menos 1 número o símbolo
140     this.passwordValidations.hasNumberOrSymbol = /[0-9!@#%&*( )_+ \- = \[ \] { } ; ' : " \
141     \ | , . < > \ / ? ] / . test(password);
142     // Al menos 1 letra mayúscula
143     this.passwordValidations.hasUppercase = /[A-Z]/.test(password);
144
145     // Al menos 1 letra minúscula
146     this.passwordValidations.hasLowercase = /[a-z]/.test(password);
147 }
148
149 private containsNameOrEmail(password: string): boolean {
150     const lowerPassword = password.toLowerCase();
151     const lowerName = this.userName.toLowerCase();
152     const lowerEmail = this.userEmail.toLowerCase();
153
154     return lowerPassword.includes(lowerName) || lowerPassword.includes(lowerEmail);
155 }
156
157 onBlur(): void {
158     if (!this.touched) {
159         this.touched = true;
160         this.onTouched();
161     }
162 }
163
164 togglePasswordVisibility(): void {
165     this.showPassword = !this.showPassword;
166 }
167
168 // Implementación de Validator
169 validate(control: AbstractControl): ValidationErrors | null {
170     if (!control.value || !this.showValidation) {
171         return null;
172     }
173
174     const password = control.value;
175     const errors: ValidationErrors = {};
176
177     // Validar que no contenga nombre o email
178     if (this.containsNameOrEmail(password)) {
179         errors['containsNameOrEmail'] = true;
180     }
181
182     // Validar longitud mínima
183     if (password.length < 8) {
184         errors['minLength'] = true;
185     }
186
187     // Validar que tenga número o símbolo
188     if (!/[0-9!@#%&*( )_+ \- = \[ \] { } ; ' : " \ | , . < > \ / ? ] / . test(password)) {
189         errors['hasNumberOrSymbol'] = true;
190     }
191
192     // Validar que tenga mayúscula
193     if (!/[A-Z]/.test(password)) {
194         errors['hasUppercase'] = true;
195     }
196
197     // Validar que tenga minúscula
198     if (!/[a-z]/.test(password)) {
199         errors['hasLowercase'] = true;
200     }
201 }

```

```

202     return Object.keys(errors).length > 0 ? errors : null;
203   }
204 }
205

```

## Ø=ÜÄ shared\components\phone-input

### Ø=ÜÄ shared\components\phone-input\phone-input.component.ts

---

```

1  import { Component, Input, forwardRef, OnInit } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { ControlValueAccessor, FormsModule, ReactiveFormsModule, NG_VALUE_ACCESSOR } from
4  @angular/forms;
5  import { CountryOption, CountryService } from '../services/countryService';
6
7  interface CountryCode {
8    code: string;
9    flag: string;
10   dialCode: string;
11 }
12
13 /**
14  * Phone number input component with country code selection.
15  *
16  * This component provides a phone number input field with country code selection
17  * dropdown. It loads countries from CountryService and allows users to select
18  * a country code, which is then combined with the phone number.
19  *
20  * Features:
21  * - Angular Forms integration (ControlValueAccessor)
22  * - Country code dropdown with flags
23  * - Searchable country list
24  * - Default country selection (Colombia, fallback to US)
25  * - Phone number formatting
26  * - Combines dial code with phone number
27  * - Customizable label and placeholder
28  *
29  * Usage:
30  * <app-phone-input
31  *   formControlName="phoneNumber"
32  *   label="Phone Number"
33  *   [required]="true">
34  * </app-phone-input>
35  *
36  * Relations:
37  * - CountryService: Provides country data with dial codes and flags
38  * - Used in registration and profile forms
39  *
40  * @component
41  * @selector app-phone-input
42  * @standalone true
43  */
44 @Component({
45   selector: 'app-phone-input',
46   standalone: true,
47   imports: [CommonModule, FormsModule, ReactiveFormsModule],
48   templateUrl: './phone-input.component.html',
49   styleUrls: ['./phone-input.component.scss'],
50   providers: [
51     {
52       provide: NG_VALUE_ACCESSOR,
53       useExisting: forwardRef(() => PhoneInputComponent),
54       multi: true
55     }
56   ]
57 })

```

```

55     ]
56   })
57   export class PhoneInputComponent implements ControlValueAccessor, OnInit {
58     @Input() label: string = 'Phone number';
59     @Input() placeholder: string = '(000) 00-0000';
60     @Input() required: boolean = false;
61     @Input() disabled: boolean = false;
62
63     selectedCountry: CountryCode = { code: 'US', flag: 'https://flagcdn.com/us.svg', dialCode:
64     '+1' };
65     phoneNumber: string = '';
66     showCountryDropdown: boolean = false;
67     touched: boolean = false;
68     searchTerm: string = '';
69     filteredCountries: CountryCode[] = [];
70     countries: CountryCode[] = [];
71     loading: boolean = true;
72
73     constructor(private countryService: CountryService) {}
74
75     ngOnInit(): void {
76       this.loadCountries();
77     }
78
79     private loadCountries(): void {
80       this.countryService.getCountries().subscribe({
81         next: (countries) => {
82           this.countries = countries;
83           this.filteredCountries = countries;
84           this.loading = false;
85
86           // Buscar Colombia por defecto, si no existe usar US
87           const colombia = countries.find(c => c.code === 'CO');
88           if (colombia) {
89             this.selectedCountry = colombia;
90           }
91         },
92         error: (error) => {
93           console.error('Error loading countries:', error);
94           this.loading = false;
95           // Mantener el país por defecto en caso de error
96         }
97       });
98
99       onChange = (value: string) => {};
100       onTouched = () => {};
101
102       writeValue(value: string): void {
103         if (value) {
104           // Si el valor ya incluye un código de país, extraer solo el número
105           const phoneMatch = value.match(/^(\+\d{1,4})\s(.+)$/);
106           if (phoneMatch) {
107             const [, dialCode, phoneNumber] = phoneMatch;
108             // Buscar el país correspondiente al código de marcación
109             const country = this.countries.find(c => c.dialCode === dialCode);
110             if (country) {
111               this.selectedCountry = country;
112             }
113             this.phoneNumber = phoneNumber;
114           } else {
115             this.phoneNumber = value;
116           }
117         } else {
118           this.phoneNumber = '';
119         }
120       }
121
122       registerOnChange(fn: any): void {
123         this.onChange = fn;
124       }

```

```

125
126 registerOnTouched(fn: any): void {
127     this.onTouched = fn;
128 }
129
130 setDisabledState(isDisabled: boolean): void {
131     this.disabled = isDisabled;
132 }
133
134 onPhoneInput(event: Event): void {
135     const value = (event.target as HTMLInputElement).value;
136     this.phoneNumber = value;
137     this.onChange(this.formatPhoneNumber(value));
138 }
139
140 private formatPhoneNumber(phoneNumber: string): string {
141     if (!phoneNumber) return '';
142     return `${this.selectedCountry.dialCode} ${phoneNumber}`;
143 }
144
145 onBlur(): void {
146     if (!this.touched) {
147         this.touched = true;
148         this.onTouched();
149     }
150 }
151
152 selectCountry(country: CountryCode): void {
153     this.selectedCountry = country;
154     this.showCountryDropdown = false;
155     this.onChange(this.formatPhoneNumber(this.phoneNumber));
156 }
157
158 toggleCountryDropdown(): void {
159     if (!this.disabled && !this.loading) {
160         this.showCountryDropdown = !this.showCountryDropdown;
161         if (this.showCountryDropdown) {
162             this.filteredCountries = this.countries;
163             this.searchTerm = '';
164         }
165     }
166 }
167
168 onSearchInput(event: Event): void {
169     const value = (event.target as HTMLInputElement).value;
170     this.searchTerm = value;
171     this.filterCountries();
172 }
173
174 private filterCountries(): void {
175     if (!this.searchTerm) {
176         this.filteredCountries = this.countries;
177     } else {
178         const searchLower = this.searchTerm.toLowerCase();
179         this.filteredCountries = this.countries.filter(country =>
180             country.code.toLowerCase().includes(searchLower) ||
181             country.dialCode.includes(searchLower)
182         );
183     }
184 }
185 }
186

```

## shared\components\plan-banner

### shared\components\plan-banner\plan-banner.component.ts

---

```
1 import { Component, Input, Output, EventEmitter } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 /**
5  * Component for displaying plan limitation banners.
6  *
7  * This component displays informational or warning banners related to plan
8  * limitations, such as approaching account/strategy limits or needing to
9  * upgrade. It provides actions to upgrade plans or dismiss the banner.
10 *
11 * Features:
12 * - Customizable message and type (info, warning, error)
13 * - Upgrade plan action
14 * - Dismiss banner action
15 * - Visibility control
16 *
17 * Banner Types:
18 * - info: Informational banner (blue)
19 * - warning: Warning banner (yellow/orange)
20 * - error: Error banner (red)
21 *
22 * Usage:
23 * <app-plan-banner
24 *   [visible]="showBanner"
25 *   [message]="bannerMessage"
26 *   [type]='warning'
27 *   (upgradePlan)="onUpgradePlan()"
28 *   (closeBanner)="onCloseBanner()">
29 * </app-plan-banner>
30 *
31 * Relations:
32 * - Used by components that need to display plan limitation warnings
33 * - Often used with PlanLimitationsGuard
34 *
35 * @component
36 * @selector app-plan-banner
37 * @standalone true
38 */
39 @Component({
40   selector: 'app-plan-banner',
41   standalone: true,
42   imports: [CommonModule],
43   templateUrl: './plan-banner.component.html',
44   styleUrls: ['./plan-banner.component.scss']
45 })
46 export class PlanBannerComponent {
47   @Input() visible: boolean = false;
48   @Input() message: string = '';
49   @Input() type: string = 'info';
50
51   @Output() upgradePlan = new EventEmitter<void>();
52   @Output() closeBanner = new EventEmitter<void>();
53
54   onUpgradePlan() {
55     this.upgradePlan.emit();
56   }
57
58   onCloseBanner() {
59     this.closeBanner.emit();
60   }
61 }
```

## Ø=ÜÁ shared\components\plan-limitation-modal

### Ø=ÜÄ shared\components\plan-limitation-modal\plan-limitation-modal.component.ts

---

```

1  import { Component, Input, Output, EventEmitter } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { Router } from '@angular/router';
4
5  /**
6   * Interface for plan limitation modal data.
7   *
8   * @interface PlanLimitationModalData
9   */
10 export interface PlanLimitationModalData {
11   showModal: boolean;
12   modalType: 'upgrade' | 'blocked';
13   title: string;
14   message: string;
15   primaryButtonText: string;
16   secondaryButtonText?: string;
17   onPrimaryAction: () => void;
18   onSecondaryAction?: () => void;
19 }
20
21 /**
22  * Component for displaying plan limitation modals.
23  *
24  * This component displays modals when users hit plan limitations, either
25  * requiring an upgrade or being blocked from a feature. It supports two
26  * modal types: upgrade (suggests upgrading) and blocked (access denied).
27  *
28  * Features:
29  * - Two modal types: upgrade and blocked
30  * - Customizable title, message, and button texts
31  * - Primary and secondary actions
32  * - Navigation to account page or signup
33  * - Close modal functionality
34  *
35  * Modal Types:
36  * - upgrade: User has reached limit, suggests upgrading plan
37  * - blocked: User is banned/cancelled, access denied
38  *
39  * Usage:
40  * <app-plan-limitation-modal
41  *   [modalData]="limitationModalData"
42  *   (closeModal)="onCloseModal()">
43  * </app-plan-limitation-modal>
44  *
45  * Relations:
46  * - PlanLimitationsGuard: Generates modal data
47  * - Used by components that check plan limitations
48  *
49  * @component
50  * @selector app-plan-limitation-modal
51  * @standalone true
52  */
53 @Component({
54   selector: 'app-plan-limitation-modal',
55   standalone: true,
56   imports: [CommonModule],
57   templateUrl: './plan-limitation-modal.component.html',

```

```

58   styleUrls: ['./plan-limitation-modal.component.scss']
59 })
60 export class PlanLimitationModalComponent {
61   @Input() modalData: PlanLimitationModalData = {
62     showModal: false,
63     modalType: 'upgrade',
64     title: '',
65     message: '',
66     primaryButtonText: '',
67     onPrimaryAction: () => {}
68   };
69
70   @Output() closeModal = new EventEmitter<void>();
71
72   constructor(private router: Router) {}
73
74   onCloseModal(): void {
75     this.closeModal.emit();
76   }
77
78   onPrimaryAction(): void {
79     this.modalData.onPrimaryAction();
80     this.onCloseModal();
81   }
82
83   onSecondaryAction(): void {
84     if (this.modalData.onSecondaryAction) {
85       this.modalData.onSecondaryAction();
86     }
87     this.onCloseModal();
88   }
89
90   navigateToAccount(): void {
91     this.router.navigate(['/account']);
92   }
93
94   navigateToSignup(): void {
95     this.router.navigate(['/signup']);
96   }
97 }
98

```

## shared\components\strategy-card

### shared\components\strategy-card\strategy-card.component.ts

---

```

1  import { Component, Input, Output, EventEmitter, HostListener, OnInit, OnDestroy, Inject }
2  from '@angular/core';
3  import { FormsModule } from '@angular/forms';
4  import { StrategyCardData } from './strategy-card.interface';
5  import { NumberFormatterService } from '../../utils/number-formatter.service';
6  import { StrategyDaysUpdaterService } from '../../services/strategy-days-updater.service';
7  import { interval, Subscription } from 'rxjs';
8  import { getFirestore, doc, getDoc, updateDoc } from 'firebase/firestore';
9  import { firebaseApp } from '../../firebase/firebase.init';
10 import { isPlatformBrowser } from '@angular/common';
11 import { PLATFORM_ID } from '@angular/core';
12
13 /**
14  * Component for displaying a strategy card with details and actions.
15  *
16  * This component displays a single strategy in card format with its
17  * status, statistics, and action buttons. It supports inline name editing,

```



```

18  * favorite toggling, and various strategy operations.
19  *
20  * Features:
21  * - Display strategy information (name, status, rules, days active, win rate)
22  * - Inline name editing with Firebase sync
23  * - Favorite toggle
24  * - More options menu (edit, duplicate, delete)
25  * - Customize button
26  * - Active rules calculation from Firebase
27  * - Days active calculation and update
28  * - Status indicators (active/inactive)
29  * - Win rate visualization
30  *
31  * Strategy Information:
32  * - Name: Editable inline
33  * - Status: Active/Inactive indicator
34  * - Rules: Count of active rules
35  * - Days Active: Calculated from creation date
36  * - Win Rate: Percentage display
37  * - Favorite: Star indicator
38  *
39  * Actions:
40  * - Edit: Opens strategy editor
41  * - Favorite: Toggles favorite status
42  * - More Options: Shows menu (edit, duplicate, delete)
43  * - Customize: Opens customization
44  *
45  * Relations:
46  * - StrategyDaysUpdaterService: Updates days active
47  * - NumberFormatterService: Formats win rate percentage
48  * - Used by StrategyComponent for strategy list display
49  *
50  * @component
51  * @selector app-strategy-card
52  * @standalone true
53  */
54  @Component({
55    selector: 'app-strategy-card',
56    standalone: true,
57    imports: [CommonModule, FormsModule],
58    templateUrl: './strategy-card.component.html',
59    styleUrls: ['./strategy-card.component.scss']
60  })
61  export class StrategyCardComponent implements OnInit, OnDestroy {
62    @Input() strategy: StrategyCardData = {
63      id: '',
64      name: '',
65      status: false,
66      lastModified: '',
67      rules: 0,
68      days_active: 0,
69      winRate: 0,
70      isFavorite: false,
71      created_at: null,
72      updated_at: null,
73      userId: '',
74      configurationId: '',
75      dateActive: [],
76      dateInactive: []
77    };
78
79    @Input() showCustomizeButton: boolean = true;
80    @Input() customizeButtonText: string = 'Customize strategy';
81
82    @Output() edit = new EventEmitter<string>();
83    @Output() favorite = new EventEmitter<string>();
84    @Output() moreOptions = new EventEmitter<string>();
85    @Output() customize = new EventEmitter<string>();
86    @Output() editStrategy = new EventEmitter<string>();
87    @Output() duplicate = new EventEmitter<string>();

```

```

88     @Output() delete = new EventEmitter<string>();
89     @Output() nameChanged = new EventEmitter<{id: string, newName: string}>();
90
91     showOptionsMenu = false;
92     isEditingName = false;
93     editingStrategyName = '';
94     isSavingName = false;
95
96     private numberFormatter = new NumberFormatterService();
97     private updateSubscription?: Subscription;
98     private userId?: string;
99     private db: any;
100
101     constructor(
102         private daysUpdaterService: StrategyDaysUpdaterService,
103         @Inject(PLATFORM_ID) private platformId: Object
104     ) {
105         if (isPlatformBrowser(this.platformId)) {
106             this.db = getFirestore(firebaseApp);
107         }
108     }
109
110     ngOnInit(): void {
111         this.calculateActiveRules();
112         this.updateDaysActive();
113     }
114
115     ngOnDestroy(): void {
116         // Limpiar suscripciones si las hay
117     }
118
119     /**
120     * Actualiza los días activos de la estrategia
121     */
122     private async updateDaysActive(): Promise<void> {
123         if (!this.strategy.id || !this.strategy.userId) {
124             return;
125         }
126
127         try {
128             // Calcular días activos localmente para mostrar inmediatamente
129             if (this.strategy.created_at) {
130                 const calculatedDays =
131                     this.daysUpdaterService.getDaysActive(this.strategy.created_at);
132             }
133
134             // Actualizar en Firebase
135             await this.daysUpdaterService.updateStrategyDaysActive(this.strategy.id,
136                 this.strategy.userId);
137             console.error('Error updating days active:', error);
138         }
139     }
140
141     onEdit() {
142         this.startEditName();
143     }
144
145     startEditName() {
146         this.isEditingName = true;
147         this.editingStrategyName = this.strategy.name;
148         // Focus en el input después de que se renderice
149         setTimeout(() => {
150             const input = document.querySelector('.strategy-name-input') as HTMLInputElement;
151             if (input) {
152                 input.focus();
153                 input.select();
154             }
155         }, 0);
156     }
157

```

```

158     async saveStrategyName() {
159         if (this.editingStrategyName.trim() && this.editingStrategyName.trim() !==
160             this.strategy.name && this.editingStrategyName.trim() !==
161                 this.strategy.newName) {
162             this.isSavingName = true;
163             try {
164                 // Actualizar en Firebase
165                 await this.updateStrategyNameInFirebase(newName);
166
167                 // Actualizar el nombre localmente
168                 this.strategy.name = newName;
169
170                 // Emitir evento para notificar al componente padre
171                 this.nameChanged.emit({
172                     id: this.strategy.id,
173                     newName: newName
174                 });
175             } catch (error) {
176                 // No actualizamos el nombre local si falla en Firebase
177                 console.error('Error updating the name of the strategy:', error);
178             } finally {
179                 this.isSavingName = false;
180             }
181         }
182
183         this.isEditingName = false;
184         this.editingStrategyName = '';
185     }
186
187     cancelEditName() {
188         this.isEditingName = false;
189         this.editingStrategyName = '';
190     }
191
192     onFavorite() {
193         this.favorite.emit(this.strategy.id);
194     }
195
196     onMoreOptions(event: Event) {
197         event.preventDefault();
198         event.stopPropagation();
199         this.showOptionsMenu = !this.showOptionsMenu;
200         // Removed moreOptions.emit() to prevent Google alert
201     }
202
203     onCustomize() {
204         this.customize.emit(this.strategy.id);
205     }
206
207     onEditStrategy() {
208         this.editStrategy.emit(this.strategy.id);
209     }
210
211     onDuplicate() {
212         this.showOptionsMenu = false;
213         this.duplicate.emit(this.strategy.id);
214     }
215
216     onDelete() {
217         this.showOptionsMenu = false;
218         this.delete.emit(this.strategy.id);
219     }
220
221     onCloseOptionsMenu() {
222         this.showOptionsMenu = false;
223     }
224
225     @HostListener('document:click', ['$event'])
226     onDocumentClick(event: Event) {
227         // Solo cerrar si el click no es en el botón de more options o en el menú

```

```

228     const target = event.target as HTMLElement;
229     const moreBtn = target.closest('.more-btn');
230     const optionsMenu = target.closest('.options-menu');
231
232     if (this.showOptionsMenu && !moreBtn && !optionsMenu) {
233         this.showOptionsMenu = false;
234     }
235 }
236
237 getStatusClass(): string {
238     return this.strategy.status ? 'active' : 'inactive';
239 }
240
241 getStatusText(): string {
242     return this.strategy.status ? 'Active' : 'Inactive';
243 }
244
245 getWinRateWidth(): string {
246     return `${this.strategy.winRate}%`;
247 }
248
249 formatPercentage(value: number): string {
250     return this.numberFormatter.formatPercentage(value);
251 }
252
253 /**
254  * Actualiza el nombre de la estrategia en Firebase
255  */
256 private async updateStrategyNameInFirebase(newName: string): Promise<void> {
257     if (!this.db || !this.strategy.id) {
258         throw new Error('Cannot update: missing database information or strategy ID');
259     }
260
261     try {
262         // Intentar actualizar en la colección 'strategies'
263         const strategyRef = doc(this.db, 'configuration-overview', this.strategy.id);
264         const strategyDoc = await getDoc(strategyRef);
265
266         if (strategyDoc.exists()) {
267             // Si existe, actualizar el documento
268             await updateDoc(strategyRef, {
269                 name: newName,
270                 updated_at: new Date()
271             });
272         } else {
273             // Si no existe, mostrar error específico
274             const errorMsg = `The strategy with ID "${this.strategy.id}" does not exist in
275 Firebase`;
276             console.error(errorMsg);
277             throw new Error(errorMsg);
278         }
279     } catch (error) {
280         console.error('Error updating the name in Firebase:', error);
281         throw error;
282     }
283
284     /**
285      * Calcula las reglas activas desde la colección configurations
286      */
287     private async calculateActiveRules(): Promise<void> {
288         if (!this.db || !this.strategy.configurationId) {
289             return;
290         }
291
292         try {
293             const configDoc = await getDoc(doc(this.db, 'configurations',
294 this.strategy.configurationId));
295             if (configDoc.exists()) {
296                 const configData = configDoc.data();
297                 let activeRulesCount = 0;

```

```

298
299         // Contar reglas activas
300         if (configData['maxDailyTrades']?.isActive) activeRulesCount++;
301         if (configData['riskReward']?.isActive) activeRulesCount++;
302         if (configData['riskPerTrade']?.isActive) activeRulesCount++;
303         if (configData['daysAllowed']?.isActive) activeRulesCount++;
304         if (configData['hoursAllowed']?.isActive) activeRulesCount++;
305         if (configData['assetsAllowed']?.isActive) activeRulesCount++;
306
307         this.strategy.rules = activeRulesCount;
308     }
309     } catch (error) {
310         console.error('Error calculating active rules:', error);
311     }
312 }
313 }
314

```

## shared\components\strategy-card\strategy-card.interface.ts

---

```

1  /**
2   * Interface for strategy card display data.
3   *
4   * This interface defines the data structure used by StrategyCardComponent
5   * to display strategy information in card format.
6   *
7   * @interface StrategyCardData
8   */
9  export interface StrategyCardData {
10     id: string;
11     name: string;
12     status: boolean; // true/false como en Firebase
13     lastModified: string;
14     rules: number; // Se calculará dinámicamente
15     days_active: number; // Viene de Firebase
16     winRate: number; // Se calculará dinámicamente
17     isFavorite?: boolean;
18     created_at: any; // Timestamp de Firebase
19     updated_at: any; // Timestamp de Firebase
20     userId: string;
21     configurationId: string;
22     dateActive?: string[]; // ISO 8601 strings - Array de fechas cuando se activó la estrategia
23     dateInactive?: string[]; // ISO 8601 strings - Array de fechas cuando se desactivó la
24     estrategia
25

```

## shared\components\strategy-guide-modal

### shared\components\strategy-guide-modal\strategy-guide-modal.component.ts

---

```

1  import { Component, EventEmitter, Output } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3
4  /**
5   * Component for displaying a strategy creation guide modal.
6   *
7   * This component provides a step-by-step guide for new users on how to
8   * create and manage trading strategies. It includes three steps with
9   * images and descriptions, and allows users to skip the guide in the future.

```

```

10  *
11  * Features:
12  * - Three-step guide (Create, Select, Edit)
13  * - Step navigation (next/previous)
14  * - Image and description for each step
15  * - "Don't show again" option
16  * - Direct navigation to strategy editing
17  * - Close modal functionality
18  *
19  * Guide Steps:
20  * 1. Create a strategy: Design trading plan with custom rules
21  * 2. Select strategy: Pick strategy for each account
22  * 3. Edit a strategy: Fine-tune rules anytime
23  *
24  * Usage:
25  * Shown to first-time users when they access the strategy page.
26  * Users can opt to not see it again.
27  *
28  * Relations:
29  * - Used by StrategyComponent for first-time user guidance
30  *
31  * @component
32  * @selector app-strategy-guide-modal
33  * @standalone true
34  */
35  @Component({
36    selector: 'app-strategy-guide-modal',
37    standalone: true,
38    imports: [CommonModule],
39    templateUrl: './strategy-guide-modal.component.html',
40    styleUrls: ['./strategy-guide-modal.component.scss']
41  })
42  export class StrategyGuideModalComponent {
43    @Output() closeModal = new EventEmitter<void>();
44    @Output() dontShowAgain = new EventEmitter<void>();
45    @Output() editStrategy = new EventEmitter<void>();
46
47    currentStep = 0;
48    totalSteps = 3;
49
50    steps = [
51      {
52        title: 'Create a strategy',
53        description: 'Design your trading plan and set personalized rules that fit you.',
54        image: 'assets/images/strategy/Create.webp',
55        content: 'Create your own trading strategy with custom rules and parameters.'
56      },
57      {
58        title: 'Select strategy',
59        description: 'Pick the perfect strategy for each account with ease.',
60        image: 'assets/images/strategy/Select.webp',
61        content: 'Choose from your created strategies and apply them to your trading accounts.'
62      },
63      {
64        title: 'Edit a strategy',
65        description: 'Fine tune rules anytime to activate or disable specific ones.',
66        image: 'assets/images/strategy/Edit.webp',
67        content: 'Modify your strategies, adjust parameters, and optimize your trading rules.'
68      }
69    ];
70
71    onNext(): void {
72      if (this.currentStep < this.totalSteps - 1) {
73        this.currentStep++;
74      } else {
75        // En el último paso, emitir evento para editar estrategia
76        this.editStrategy.emit();
77      }
78    }
79  }

```

```

80     onPrevious(): void {
81         if (this.currentStep > 0) {
82             this.currentStep--;
83         }
84     }
85
86     onClose(): void {
87         this.closeModal.emit();
88     }
89
90     onDontShowAgain(): void {
91         this.dontShowAgain.emit();
92     }
93
94     getCurrentStep() {
95         return this.steps[this.currentStep];
96     }
97
98     getButtonText(): string {
99         return this.currentStep === this.totalSteps - 1 ? 'Edit strategy' : 'Next';
100     }
101 }
102

```

## Ø=ÜÁ shared\components\subscription-processing

### Ø=ÜÄ shared\components\subscription-processing\subscription-processing.component.ts

---

```

1  import { Component, Input, Output, EventEmitter, OnInit, OnDestroy, inject } from '@angular/
2  import { CommonModule } from '@angular/common';
3  import { SubscriptionService } from '../../../services/subscription-service';
4  import { UserStatus } from '../../../features/overview/models/overview';
5
6  /**
7   * Interface for subscription processing configuration.
8   *
9   * @interface SubscriptionProcessingConfig
10  */
11  export interface SubscriptionProcessingConfig {
12      paymentId: string;
13      userId: string;
14      context: 'signup' | 'plan-change';
15      planName?: string;
16  }
17
18  /**
19   * Component for displaying subscription payment processing status.
20   *
21   * This component shows a processing screen while a subscription payment is
22   * being verified. It polls the subscription status and displays success
23   * or error states based on the payment result.
24   *
25   * Features:
26   * - Payment status polling (every 2 seconds)
27   * - Success state display
28   * - Error state display with retry option
29   * - Timeout handling (30 seconds)
30   * - Context-aware messaging (signup vs plan-change)
31   * - Go back functionality
32   *
33   * Status States:
34   * - processing: Payment is being verified
35   * - success: Payment confirmed (PURCHASED status)

```

```

36 * - error: Payment failed or timeout
37 *
38 * Polling:
39 * - Checks subscription status every 2 seconds
40 * - Stops when status changes or timeout occurs
41 * - Supports retry on error
42 *
43 * Relations:
44 * - SubscriptionService: Checks subscription status
45 * - Used in signup and plan change flows
46 *
47 * @component
48 * @selector app-subscription-processing
49 * @standalone true
50 */
51 @Component({
52   selector: 'app-subscription-processing',
53   standalone: true,
54   imports: [CommonModule],
55   templateUrl: './subscription-processing.component.html',
56   styleUrls: ['./subscription-processing.component.scss']
57 })
58 export class SubscriptionProcessingComponent implements OnInit, OnDestroy {
59   @Input() config: SubscriptionProcessingConfig = {
60     paymentId: '',
61     userId: '',
62     context: 'signup'
63   };
64   @Output() paymentSuccess = new EventEmitter<void>();
65   @Output() paymentError = new EventEmitter<void>();
66   @Output() goBack = new EventEmitter<void>();
67
68   processingStatus = 'processing';
69   errorMessage = '';
70   private statusCheckInterval: any;
71   private subscriptionService = inject(SubscriptionService);
72
73   ngOnInit(): void {
74     this.startStatusListener();
75   }
76
77   ngOnDestroy(): void {
78     if (this.statusCheckInterval) {
79       clearInterval(this.statusCheckInterval);
80     }
81   }
82
83   // TODO: IMPLEMENTAR ENDPOINT DE VERIFICACIÓN DE PAGO - Reemplazar polling con API real
84   private startStatusListener(): void {
85     // Verificar estado del pago cada 2 segundos
86     this.statusCheckInterval = setInterval(async () => {
87       try {
88         const subscription = await
89 this.subscriptionService.getSubscriptionById(this.config.userId, this.config.paymentId);
90         if (subscription) {
91           switch (subscription.status) {
92             case UserStatus.PURCHASED:
93               this.processingStatus = 'success';
94               this.paymentSuccess.emit();
95               this.clearInterval();
96               break;
97             case UserStatus.CREATED:
98               // Aún procesando
99               break;
100             default:
101               this.processingStatus = 'error';
102               this.errorMessage = 'Error processing the subscription';
103               this.paymentError.emit();
104               this.clearInterval();
105               break;

```



```

106         }
107     } else {
108         this.processingStatus = 'error';
109         this.errorMessage = 'Subscription not found';
110         this.paymentError.emit();
111         this.clearInterval();
112     }
113 } catch (error) {
114     console.error('Error verifying subscription status:', error);
115     this.processingStatus = 'error';
116     this.errorMessage = 'Connection error';
117     this.paymentError.emit();
118     this.clearInterval();
119 }
120 }, 2000);
121
122 // Timeout after 30 seconds
123 setTimeout(() => {
124     if (this.processingStatus === 'processing') {
125         this.processingStatus = 'error';
126         this.errorMessage = 'Timeout';
127         this.paymentError.emit();
128         this.clearInterval();
129     }
130 }, 30000);
131 }
132
133 private clearInterval(): void {
134     if (this.statusCheckInterval) {
135         clearInterval(this.statusCheckInterval);
136         this.statusCheckInterval = null;
137     }
138 }
139
140 onGoBack(): void {
141     this.clearInterval();
142     this.goBack.emit();
143 }
144
145 onRetry(): void {
146     this.processingStatus = 'processing';
147     this.errorMessage = '';
148     this.startStatusListener();
149 }
150
151 getContextTitle(): string {
152     switch (this.config.context) {
153         case 'signup':
154             return 'Almost there! Redirecting to finalize your payment.';
155         case 'plan-change':
156             return 'Almost there! Processing your plan change.';
157         default:
158             return 'Almost there! Redirecting to finalize your payment.';
159     }
160 }
161
162 getSuccessMessage(): string {
163     switch (this.config.context) {
164         case 'signup':
165             return 'Your subscription has been activated successfully.';
166         case 'plan-change':
167             return `Your plan has been successfully changed to ${this.config.planName} || 'the
168 new plan default:
169             return 'Your subscription has been activated successfully.';
170     }
171 }
172 }
173

```

## shared\components\text-input

### shared\components\text-input\text-input.component.ts

---

```
1 import { Component, Input, forwardRef } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { ControlValueAccessor, FormsModule, ReactiveFormsModule, NG_VALUE_ACCESSOR } from
4 '@angular/forms';
5 /**
6  * Reusable text input component with Angular Forms integration.
7  *
8  * This component provides a customizable text input that integrates with
9  * Angular Reactive Forms and Template-driven Forms through ControlValueAccessor.
10 * It supports labels, placeholders, validation states, and disabled states.
11 *
12 * Features:
13 * - Angular Forms integration (ControlValueAccessor)
14 * - Customizable label and placeholder
15 * - Support for different input types (text, email, password, etc.)
16 * - Required field indicator
17 * - Disabled state support
18 * - Touch state tracking
19 * - Validation error display
20 *
21 * Usage:
22 * <app-text-input
23 *   FormControlName="email"
24 *   label="Email"
25 *   placeholder="Enter your email"
26 *   type="email"
27 *   [required]="true">
28 * </app-text-input>
29 *
30 * Relations:
31 * - Used in forms throughout the application
32 * - Integrates with Angular Reactive Forms and Template-driven Forms
33 *
34 * @component
35 * @selector app-text-input
36 * @standalone true
37 */
38 @Component({
39   selector: 'app-text-input',
40   standalone: true,
41   imports: [CommonModule, FormsModule, ReactiveFormsModule],
42   templateUrl: './text-input.component.html',
43   styleUrls: ['./text-input.component.scss'],
44   providers: [
45     {
46       provide: NG_VALUE_ACCESSOR,
47       useExisting: forwardRef(() => TextInputComponent),
48       multi: true
49     }
50   ]
51 })
52 export class TextInputComponent implements ControlValueAccessor {
53   @Input() label: string = '';
54   @Input() placeholder: string = '';
55   @Input() type: string = 'text';
56   @Input() required: boolean = false;
57   @Input() disabled: boolean = false;
58
59   value: string = '';
60   touched: boolean = false;
61 }
```

```

62     onChange = (value: string) => {};
63     onTouched = () => {};
64
65     writeValue(value: string): void {
66         this.value = value;
67     }
68
69     registerOnChange(fn: any): void {
70         this.onChange = fn;
71     }
72
73     registerOnTouched(fn: any): void {
74         this.onTouched = fn;
75     }
76
77     setDisabledState(isDisabled: boolean): void {
78         this.disabled = isDisabled;
79     }
80
81     onInput(event: Event): void {
82         const value = (event.target as HTMLInputElement).value;
83         this.value = value;
84         this.onChange(value);
85     }
86
87     onBlur(): void {
88         if (!this.touched) {
89             this.touched = true;
90             this.onTouched();
91         }
92     }
93 }
94

```

## Ö=ÜÁ shared\context

### Ö=ÜÄ shared\context\constants.ts

---

```

1  // Constantes para el contexto de la aplicación
2
3  export const CACHE_CONFIG = {
4      // TTL en milisegundos
5      TRADE_LOCKER_TTL: 5 * 60 * 1000, // 5 minutos
6      API_TTL: 10 * 60 * 1000, // 10 minutos
7      USER_DATA_TTL: 30 * 60 * 1000, // 30 minutos
8
9      // Tamaños máximos
10     MAX_CACHE_SIZE: 100,
11     MAX_TRADE_LOCKER_ACCOUNTS: 50,
12     MAX_PLUGIN_HISTORY: 1000,
13
14     // Intervalos de actualización
15     AUTO_REFRESH_INTERVAL: 60 * 1000, // 1 minuto
16     STALE_DATA_THRESHOLD: 2 * 60 * 1000, // 2 minutos
17 } as const;
18
19 export const LOADING_STATES = {
20     USER: 'user',
21     ACCOUNTS: 'accounts',
22     STRATEGIES: 'strategies',
23     PLAN: 'plan',
24     PLUGIN_HISTORY: 'pluginHistory',
25     TRADE_LOCKER: 'tradeLocker',

```

```

26 } as const;
27
28 export const ERROR_TYPES = {
29   USER: 'user',
30   ACCOUNTS: 'accounts',
31   STRATEGIES: 'strategies',
32   PLAN: 'plan',
33   PLUGIN_HISTORY: 'pluginHistory',
34   TRADE_LOCKER: 'tradeLocker',
35 } as const;
36
37 export const CONTEXT_EVENTS = {
38   USER_LOGIN: 'user:login',
39   USER_LOGOUT: 'user:logout',
40   USER_UPDATE: 'user:update',
41   ACCOUNTS_ADD: 'accounts:add',
42   ACCOUNTS_UPDATE: 'accounts:update',
43   ACCOUNTS_REMOVE: 'accounts:remove',
44   STRATEGIES_ADD: 'strategies:add',
45   STRATEGIES_UPDATE: 'strategies:update',
46   STRATEGIES_REMOVE: 'strategies:remove',
47   STRATEGIES_ACTIVATE: 'strategies:activate',
48   PLAN_UPDATE: 'plan:update',
49   PLUGIN_HISTORY_ADD: 'pluginHistory:add',
50   CACHE_CLEAR: 'cache:clear',
51   ERROR_SET: 'error:set',
52   ERROR_CLEAR: 'error:clear',
53 } as const;
54
55 export const DEFAULT_PLAN = {
56   planId: 'free',
57   planName: 'Free',
58   maxAccounts: 1,
59   maxStrategies: 1,
60   features: ['basic_trading'],
61   isActive: true,
62 } as const;
63
64 export const PLUGIN_ACTIONS = {
65   ACCOUNT_CREATED: 'account_created',
66   ACCOUNT_UPDATED: 'account_updated',
67   ACCOUNT_DELETED: 'account_deleted',
68   STRATEGY_CREATED: 'strategy_created',
69   STRATEGY_UPDATED: 'strategy_updated',
70   STRATEGY_DELETED: 'strategy_deleted',
71   STRATEGY_ACTIVATED: 'strategy_activated',
72   STRATEGY_DEACTIVATED: 'strategy_deactivated',
73   PLAN_CHANGED: 'plan_changed',
74   USER_LOGIN: 'user_login',
75   USER_LOGOUT: 'user_logout',
76 } as const;
77

```

## shared\context\context.ts

---

```

1 import { Injectable, signal, computed, effect } from '@angular/core';
2 import { BehaviorSubject, Observable, combineLatest, map, distinctUntilChanged } from 'rxjs';
3 import { User } from '../../features/overview/models/overview';
4 import { AccountData } from '../../features/auth/models/userModel';
5 import { ConfigurationOverview } from '../../features/strategy/models/strategy.model';
6 import { BalanceData, GroupedTradeFinal } from '../../features/report/models/report.model';
7 import { Plan } from '../services/planService';
8 import { TradeLockerApiService } from '../services/tradelocker-api.service';
9
10 // Interfaces para datos de API externa
11 export interface TradeLockerAccountData {

```

```

12     accountId: string;
13     balance: BalanceData;
14     lastUpdated: number;
15     isValid: boolean;
16 }
17
18 export interface PluginHistoryData {
19     id: string;
20     userId: string;
21     action: string;
22     timestamp: number;
23     details: any;
24 }
25
26 export interface UserPlanData {
27     planId: string;
28     planName: string;
29     maxAccounts: number;
30     maxStrategies: number;
31     features: string[];
32     isActive: boolean;
33     expiresAt?: number;
34 }
35
36 export interface AppContextState {
37     // Usuario autenticado
38     currentUser: User | null;
39     isAuthenticated: boolean;
40
41     // Datos del usuario
42     userAccounts: AccountData[];
43     userStrategies: ConfigurationOverview[];
44     userPlan: UserPlanData | null;
45     pluginHistory: PluginHistoryData[];
46
47     // Planes globales (cargados una vez al login)
48     globalPlans: Plan[];
49     planLimits: { [planName: string]: { tradingAccounts: number; strategies: number } };
50
51     // Datos de reportes
52     reportData: {
53         accountHistory: any[];
54         stats: any;
55         balanceData: any;
56         monthlyReports: any[];
57     };
58
59     // Trading history por cuenta (nuevo)
60     tradingHistoryByAccount: Map<string, {
61         accountHistory: GroupedTradeFinal[];
62         stats: any;
63         balanceData: any;
64         lastUpdated: number;
65     }>;
66
67     // Datos de overview (para admins)
68     overviewData: {
69         allUsers: User[];
70         subscriptions: any[];
71         monthlyReports: any[];
72         allAccounts: AccountData[];
73         allStrategies: ConfigurationOverview[];
74     };
75
76     // Datos de API externa (con caché)
77     tradeLockerData: Map<string, TradeLockerAccountData>;
78     apiCache: Map<string, { data: any; timestamp: number; ttl: number }>;
79
80     // Estados de carga
81     isLoading: {

```

```

82     user: boolean;
83     accounts: boolean;
84     strategies: boolean;
85     plan: boolean;
86     pluginHistory: boolean;
87     tradeLocker: boolean;
88     report: boolean;
89     overview: boolean;
90     globalPlans: boolean;
91 };
92
93 // Estados de error
94 errors: {
95     user: string | null;
96     accounts: string | null;
97     strategies: string | null;
98     plan: string | null;
99     pluginHistory: string | null;
100    tradeLocker: string | null;
101    report: string | null;
102    overview: string | null;
103    globalPlans: string | null;
104 };
105
106 // Configuración de caché
107 cacheConfig: {
108     tradeLockerTtl: number; // 5 minutos
109     apiTtl: number; // 10 minutos
110     maxCacheSize: number; // 100 elementos
111 };
112 }
113
114 /**
115  * Global application context service for managing shared state.
116  *
117  * This service provides a centralized state management system using Angular signals
118  * and RxJS observables. It manages user data, accounts, strategies, plans, trading
119  * history, and API caching across the entire application.
120  *
121  * Features:
122  * - User state management (current user, authentication status)
123  * - Account management (CRUD operations)
124  * - Strategy management (CRUD operations, activation)
125  * - Plan management (user plan, global plans, limits)
126  * - Trading history per account (with localStorage persistence)
127  * - API caching (TradeLocker data, general API cache)
128  * - Report data management
129  * - Overview data management (for admins)
130  * - Loading states per component
131  * - Error states per component
132  * - Computed signals for derived data
133  *
134  * State Management:
135  * - Signals: For reactive UI updates (Angular signals)
136  * - Observables: For RxJS-based subscriptions
137  * - BehaviorSubject: Internal state management
138  * - localStorage: Persistence for trading history
139  *
140  * Caching:
141  * - TradeLocker data: 5-minute TTL
142  * - General API cache: 10-minute TTL
143  * - Automatic cache eviction (20% oldest entries when max size reached)
144  *
145  * Relations:
146  * - TradeLockerApiService: Fetches trading data
147  * - All feature modules: Consume and update context data
148  * - localStorage: Persists trading history
149  *
150  * @service
151  * @injectable

```

```

152 * @providedIn root
153 */
154 @Injectable({
155     providedIn: 'root'
156 })
157 export class AppContextService {
158     // Estado inicial
159     private initialState: AppContextState = {
160         currentUser: null,
161         isAuthenticated: false,
162         userAccounts: [],
163         userStrategies: [],
164         userPlan: null,
165         pluginHistory: [],
166         globalPlans: [],
167         planLimits: {},
168         reportData: {
169             accountHistory: [],
170             stats: null,
171             balanceData: null,
172             monthlyReports: []
173         },
174         overviewData: {
175             allUsers: [],
176             subscriptions: [],
177             monthlyReports: [],
178             allAccounts: [],
179             allStrategies: []
180         },
181         tradeLockerData: new Map(),
182         apiCache: new Map(),
183         tradingHistoryByAccount: new Map(),
184         isLoading: {
185             user: false,
186             accounts: false,
187             strategies: false,
188             plan: false,
189             pluginHistory: false,
190             tradeLocker: false,
191             report: false,
192             overview: false,
193             globalPlans: false
194         },
195         errors: {
196             user: null,
197             accounts: null,
198             strategies: null,
199             plan: null,
200             pluginHistory: null,
201             tradeLocker: null,
202             report: null,
203             overview: null,
204             globalPlans: null
205         },
206         cacheConfig: {
207             tradeLockerTtl: 5 * 60 * 1000, // 5 minutos
208             apiTtl: 10 * 60 * 1000, // 10 minutos
209             maxCacheSize: 100
210         }
211     };
212
213     // Estado reactivo principal
214     private stateSubject = new BehaviorSubject<AppContextState>(this.initialState);
215     public state$ = this.stateSubject.asObservable();
216
217     // Signals para datos específicos (más eficientes para UI)
218     public currentUser = signal<User | null>(null);
219     public isAuthenticated = signal<boolean>(false);
220     public userAccounts = signal<AccountData[]>([]);
221     public userStrategies = signal<ConfigurationOverview[]>([]);

```

```

222     public userPlan = signal<UserPlanData | null>(null);
223     public pluginHistory = signal<PluginHistoryData[]>([]);
224
225     // Signals para planes globales
226     public globalPlans = signal<Plan[]>([]);
227     public planLimits = signal<{ [planName: string]: { tradingAccounts: number; strategies:
228 number } }>({});
229
230     // Signals para datos de reportes
231     public reportData = signal<{
232         accountHistory: any[];
233         stats: any;
234         balanceData: any;
235         monthlyReports: any[];
236     }>({
237         accountHistory: [],
238         stats: null,
239         balanceData: null,
240         monthlyReports: []
241     });
242
243     // Signals para datos de overview
244     public overviewData = signal<{
245         allUsers: User[];
246         subscriptions: any[];
247         monthlyReports: any[];
248         allAccounts: AccountData[];
249         allStrategies: ConfigurationOverview[];
250     }>({
251         allUsers: [],
252         subscriptions: [],
253         monthlyReports: [],
254         allAccounts: [],
255         allStrategies: []
256     });
257
258     // Computed signals para datos derivados
259     public activeStrategies = computed(() =>
260         this.userStrategies().filter(strategy => strategy.status === true)
261     );
262
263     public totalAccounts = computed(() =>
264         this.userAccounts().length
265     );
266
267     public totalStrategies = computed(() =>
268         this.userStrategies().length
269     );
270
271     public canCreateAccount = computed(() => {
272         const plan = this.userPlan();
273         const currentCount = this.totalAccounts();
274         return plan ? currentCount < plan.maxAccounts : false;
275     });
276
277     public canCreateStrategy = computed(() => {
278         const plan = this.userPlan();
279         const currentCount = this.totalStrategies();
280         return plan ? currentCount < plan.maxStrategies : false;
281     });
282
283     public planLimitations = computed(() => ({
284         maxAccounts: this.userPlan()?.maxAccounts || 0,
285         currentAccounts: this.totalAccounts(),
286         maxStrategies: this.userPlan()?.maxStrategies || 0,
287         currentStrategies: this.totalStrategies(),
288         canCreateAccount: this.canCreateAccount(),
289         canCreateStrategy: this.canCreateStrategy()
290     }));
291
292     // Computed signals para planes globales

```



```

292 public orderedPlans = computed(() => {
293     const plans = this.globalPlans();
294
295     const orderedPlanNames = ['Free', 'Starter', 'Pro'];
296     const orderedPlans: Plan[] = [];
297
298     orderedPlanNames.forEach(planName => {
299         const plan = plans.find(p => p.name.toLowerCase() === planName.toLowerCase());
300         if (plan) {
301             orderedPlans.push(plan);
302         }
303     });
304
305     return orderedPlans;
306 });
307
308 // Observables específicos para suscripciones
309 public currentUser$ = this.state$.pipe(
310     map(state => state.currentUser),
311     distinctUntilChanged()
312 );
313
314 public userAccounts$ = this.state$.pipe(
315     map(state => state.userAccounts),
316     distinctUntilChanged()
317 );
318
319 public userStrategies$ = this.state$.pipe(
320     map(state => state.userStrategies),
321     distinctUntilChanged()
322 );
323
324 public userPlan$ = this.state$.pipe(
325     map(state => state.userPlan),
326     distinctUntilChanged()
327 );
328
329 public pluginHistory$ = this.state$.pipe(
330     map(state => state.pluginHistory),
331     distinctUntilChanged()
332 );
333
334 public isLoading$ = this.state$.pipe(
335     map(state => state.isLoading),
336     distinctUntilChanged()
337 );
338
339 public errors$ = this.state$.pipe(
340     map(state => state.errors),
341     distinctUntilChanged()
342 );
343
344 public reportData$ = this.state$.pipe(
345     map(state => state.reportData),
346     distinctUntilChanged()
347 );
348
349 public overviewData$ = this.state$.pipe(
350     map(state => state.overviewData),
351     distinctUntilChanged()
352 );
353
354 public globalPlans$ = this.state$.pipe(
355     map(state => state.globalPlans),
356     distinctUntilChanged()
357 );
358
359 public planLimits$ = this.state$.pipe(
360     map(state => state.planLimits),
361     distinctUntilChanged()

```

```

362     };
363
364     constructor(private tradeLockerApi: TradeLockerApiService) {
365         // Sincronizar signals con el estado principal
366         effect(() => {
367             this.updateState({
368                 currentUser: this.currentUser(),
369                 isAuthenticated: this.isAuthenticated(),
370                 userAccounts: this.userAccounts(),
371                 userStrategies: this.userStrategies(),
372                 userPlan: this.userPlan(),
373                 pluginHistory: this.pluginHistory(),
374                 globalPlans: this.globalPlans(),
375                 planLimits: this.planLimits(),
376                 reportData: this.reportData(),
377                 overviewData: this.overviewData()
378             });
379         });
380     }
381
382     // ===== MÉTODOS DE ACTUALIZACIÓN DE ESTADO =====
383
384     private updateState(updates: Partial<AppContextState>): void {
385         const currentState = this.stateSubject.value;
386         const newState = { ...currentState, ...updates };
387         this.stateSubject.next(newState);
388     }
389
390     // ===== MÉTODOS DE USUARIO =====
391
392     setCurrentUser(user: User | null): void {
393         this.currentUser.set(user);
394         this.isAuthenticated.set(user !== null);
395
396         if (user) {
397             this.updateState({
398                 currentUser: user,
399                 isAuthenticated: true
400             });
401
402             // NO cargar trading history automáticamente para evitar peticiones repetidas
403             // this.loadTradingHistoryAfterLogin(user.id);
404         } else {
405             this.clearUserData();
406         }
407     }
408
409     updateUserData(userData: Partial<User>): void {
410         const currentUser = this.currentUser();
411         if (currentUser) {
412             const updatedUser = { ...currentUser, ...userData };
413             this.currentUser.set(updatedUser);
414         }
415     }
416
417     clearUserData(): void {
418         this.currentUser.set(null);
419         this.isAuthenticated.set(false);
420         this.userAccounts.set([]);
421         this.userStrategies.set([]);
422         this.userPlan.set(null);
423         this.pluginHistory.set([]);
424         // No limpiar globalPlans y planLimits ya que son globales
425
426         this.updateState({
427             currentUser: null,
428             isAuthenticated: false,
429             userAccounts: [],
430             userStrategies: [],
431             userPlan: null,

```

```

432     pluginHistory: []
433   });
434 }
435
436 // ===== MÉTODOS DE CUENTAS =====
437
438 setUserAccounts(accounts: AccountData[]): void {
439   this.userAccounts.set(accounts);
440 }
441
442 addAccount(account: AccountData): void {
443   const currentAccounts = this.userAccounts();
444   this.userAccounts.set([...currentAccounts, account]);
445 }
446
447 updateAccount(accountId: string, updates: Partial<AccountData>): void {
448   const currentAccounts = this.userAccounts();
449   const updatedAccounts = currentAccounts.map(account =>
450     account.id === accountId ? { ...account, ...updates } : account
451   );
452   this.userAccounts.set(updatedAccounts);
453 }
454
455 removeAccount(accountId: string): void {
456   const currentAccounts = this.userAccounts();
457   const filteredAccounts = currentAccounts.filter(account => account.id !== accountId);
458   this.userAccounts.set(filteredAccounts);
459 }
460
461 // ===== MÉTODOS DE ESTRATEGIAS =====
462
463 setUserStrategies(strategies: ConfigurationOverview[]): void {
464   this.userStrategies.set(strategies);
465 }
466
467 addStrategy(strategy: ConfigurationOverview & { id: string }): void {
468   const currentStrategies = this.userStrategies();
469   this.userStrategies.set([...currentStrategies, strategy]);
470 }
471
472 updateStrategy(strategyId: string, updates: Partial<ConfigurationOverview>): void {
473   const currentStrategies = this.userStrategies();
474   const updatedStrategies = currentStrategies.map(strategy =>
475     (strategy as any).id === strategyId ? { ...strategy, ...updates } : strategy
476   );
477   this.userStrategies.set(updatedStrategies);
478 }
479
480 removeStrategy(strategyId: string): void {
481   const currentStrategies = this.userStrategies();
482   const filteredStrategies = currentStrategies.filter(strategy => (strategy as any).id !==
483 strategyId);
484   this.userStrategies.set(filteredStrategies);
485 }
486
487 activateStrategy(strategyId: string): void {
488   const currentStrategies = this.userStrategies();
489   const updatedStrategies = currentStrategies.map(strategy => ({
490     ...strategy,
491     status: (strategy as any).id === strategyId
492   }));
493   this.userStrategies.set(updatedStrategies);
494 }
495
496 // ===== MÉTODOS DE PLAN =====
497
498 setUserPlan(plan: UserPlanData | null): void {
499   this.userPlan.set(plan);
500 }
501
502 updateUserPlan(planUpdates: Partial<UserPlanData>): void {

```

```

502     const currentPlan = this.userPlan();
503     if (currentPlan) {
504         const updatedPlan = { ...currentPlan, ...planUpdates };
505         this.userPlan.set(updatedPlan);
506     }
507 }
508
509 // ===== MÉTODOS DE PLANES GLOBALES =====
510
511 setGlobalPlans(plans: Plan[]): void {
512     this.globalPlans.set(plans);
513
514     // Crear mapa de límites automáticamente
515     const limits: { [planName: string]: { tradingAccounts: number; strategies: number } } =
516     {}; plans.forEach(plan => {
517         // Usar el nombre original del plan como clave
518         limits[plan.name] = {
519             tradingAccounts: plan.tradingAccounts,
520             strategies: plan.strategies
521         };
522     });
523     this.planLimits.set(limits);
524 }
525
526 getPlanByName(planName: string): Plan | undefined {
527     const plans = this.globalPlans();
528     return plans.find(plan => plan.name.toLowerCase() === planName.toLowerCase());
529 }
530
531 getPlanById(planId: string): Plan | undefined {
532     const plans = this.globalPlans();
533     return plans.find(plan => plan.id === planId);
534 }
535
536 getPlanLimits(planName: string): { tradingAccounts: number; strategies: number } | null {
537     const limits = this.planLimits();
538     // Buscar por nombre exacto primero
539     if (limits[planName]) {
540         return limits[planName];
541     }
542
543     // Si no se encuentra, buscar case-insensitive
544     const planKey = Object.keys(limits).find(key =>
545         key.toLowerCase() === planName.toLowerCase()
546     );
547
548     return planKey ? limits[planKey] : null;
549 }
550
551 // ===== MÉTODOS DE HISTORIAL DE PLUGINS =====
552
553 setPluginHistory(history: PluginHistoryData[]): void {
554     this.pluginHistory.set(history);
555 }
556
557 addPluginHistoryEntry(entry: PluginHistoryData): void {
558     const currentHistory = this.pluginHistory();
559     this.pluginHistory.set([entry, ...currentHistory]);
560 }
561
562 // ===== MÉTODOS DE CACHE DE API =====
563
564 setTradeLockerData(accountId: string, data: TradeLockerAccountData): void {
565     const currentState = this.stateSubject.value;
566     const newTradeLockerData = new Map(currentState.tradeLockerData);
567     newTradeLockerData.set(accountId, data);
568
569     this.updateState({
570         tradeLockerData: newTradeLockerData
571     });

```

```

572     }
573
574     getTradeLockerData(accountId: string): TradeLockerAccountData | null {
575         const currentState = this.stateSubject.value;
576         return currentState.tradeLockerData.get(accountId) || null;
577     }
578
579     isTradeLockerDataValid(accountId: string): boolean {
580         const data = this.getTradeLockerData(accountId);
581         if (!data) return false;
582
583         const now = Date.now();
584         const ttl = this.stateSubject.value.cacheConfig.tradeLockerTtl;
585         return (now - data.lastUpdated) < ttl;
586     }
587
588     setApiCache(key: string, data: any, ttl?: number): void {
589         const currentState = this.stateSubject.value;
590         const newApiCache = new Map(currentState.apiCache);
591         const cacheTtl = ttl || currentState.cacheConfig.apiTtl;
592
593         newApiCache.set(key, {
594             data,
595             timestamp: Date.now(),
596             ttl: cacheTtl
597         });
598
599         // Limpiar caché si excede el tamaño máximo
600         if (newApiCache.size > currentState.cacheConfig.maxCacheSize) {
601             const entries = Array.from(newApiCache.entries());
602             entries.sort((a, b) => a[1].timestamp - b[1].timestamp);
603
604             // Eliminar el 20% más antiguo
605             const toRemove = Math.floor(entries.length * 0.2);
606             for (let i = 0; i < toRemove; i++) {
607                 newApiCache.delete(entries[i][0]);
608             }
609         }
610
611         this.updateState({
612             apiCache: newApiCache
613         });
614     }
615
616     getApiCache(key: string): any | null {
617         const currentState = this.stateSubject.value;
618         const cached = currentState.apiCache.get(key);
619
620         if (!cached) return null;
621
622         const now = Date.now();
623         if ((now - cached.timestamp) > cached.ttl) {
624             // Expirar caché
625             const newApiCache = new Map(currentState.apiCache);
626             newApiCache.delete(key);
627             this.updateState({ apiCache: newApiCache });
628             return null;
629         }
630
631         return cached.data;
632     }
633
634     clearApiCache(): void {
635         this.updateState({
636             tradeLockerData: new Map(),
637             apiCache: new Map()
638         });
639     }
640
641     // ===== MÉTODOS DE ESTADO DE CARGA =====

```

```

642
643 setLoading(component: keyof AppContextState['isLoading'], loading: boolean): void {
644     const currentState = this.stateSubject.value;
645     this.updateState({
646         isLoading: {
647             ...currentState.isLoading,
648             [component]: loading
649         }
650     });
651 }
652
653 setError(component: keyof AppContextState['errors'], error: string | null): void {
654     const currentState = this.stateSubject.value;
655     this.updateState({
656         errors: {
657             ...currentState.errors,
658             [component]: error
659         }
660     });
661 }
662
663 clearErrors(): void {
664     this.updateState({
665         errors: {
666             user: null,
667             accounts: null,
668             strategies: null,
669             plan: null,
670             pluginHistory: null,
671             tradeLocker: null,
672             report: null,
673             overview: null,
674             globalPlans: null
675         }
676     });
677 }
678
679 // ===== MÉTODOS DE UTILIDAD =====
680
681 getCurrentState(): AppContextState {
682     return this.stateSubject.value;
683 }
684
685 resetState(): void {
686     this.stateSubject.next(this.initialState);
687     this.currentUser.set(null);
688     this.isAuthenticated.set(false);
689     this.userAccounts.set([]);
690     this.userStrategies.set([]);
691     this.userPlan.set(null);
692     this.pluginHistory.set([]);
693     this.globalPlans.set([]);
694     this.planLimits.set({});
695 }
696
697 // ===== MÉTODOS DE VALIDACIÓN =====
698
699 isDataStale(component: keyof AppContextState['isLoading'], maxAge: number = 5 * 60 *
700 1000): boolean {
701     // Implementar lógica para verificar si los datos están obsoletos
702     // Esto se puede usar para decidir si hacer nuevas peticiones
703     return true; // Placeholder
704 }
705
706 // ===== MÉTODOS DE SUSCRIPCIÓN =====
707
708 subscribeToUserChanges(): Observable<User | null> {
709     return this.currentUser$;
710 }
711
712 subscribeToAccountsChanges(): Observable<AccountData[]> {

```

```

712     return this.userAccounts$;
713 }
714
715 subscribeToStrategiesChanges(): Observable<ConfigurationOverview[]> {
716     return this.userStrategies$;
717 }
718
719 subscribeToPlanChanges(): Observable<UserPlanData | null> {
720     return this.userPlan$;
721 }
722
723 subscribeToPluginHistoryChanges(): Observable<PluginHistoryData[]> {
724     return this.pluginHistory$;
725 }
726
727 subscribeToGlobalPlansChanges(): Observable<Plan[]> {
728     return this.globalPlans$;
729 }
730
731 subscribeToPlanLimitsChanges(): Observable<{ [planName: string]: { tradingAccounts:
732     number, sharePrices: number } }> {
733     return this.planLimits$;
734 }
735 // ===== MÉTODOS DE DATOS DE REPORTES =====
736
737 setReportData(data: {
738     accountHistory?: any[];
739     stats?: any;
740     balanceData?: any;
741     monthlyReports?: any[];
742 }): void {
743     const currentData = this.reportData();
744     this.reportData.set({ ...currentData, ...data });
745 }
746
747 updateReportStats(stats: any): void {
748     const currentData = this.reportData();
749     this.reportData.set({ ...currentData, stats });
750 }
751
752 updateReportBalance(balanceData: any): void {
753     const currentData = this.reportData();
754     this.reportData.set({ ...currentData, balanceData });
755 }
756
757 updateReportHistory(accountHistory: any[]): void {
758     const currentData = this.reportData();
759     this.reportData.set({ ...currentData, accountHistory });
760 }
761
762 // ===== MÉTODOS DE DATOS DE OVERVIEW =====
763
764 setOverviewData(data: {
765     allUsers?: User[];
766     subscriptions?: any[];
767     monthlyReports?: any[];
768     allAccounts?: AccountData[];
769     allStrategies?: ConfigurationOverview[];
770 }): void {
771     const currentData = this.overviewData();
772     this.overviewData.set({ ...currentData, ...data });
773 }
774
775 updateOverviewUsers(allUsers: User[]): void {
776     const currentData = this.overviewData();
777     this.overviewData.set({ ...currentData, allUsers });
778 }
779
780 updateOverviewSubscriptions(subscriptions: any[]): void {
781     const currentData = this.overviewData();

```

```

782     this.overviewData.set({ ...currentData, subscriptions });
783 }
784
785 updateOverviewAccounts(allAccounts: AccountData[]): void {
786     const currentData = this.overviewData();
787     this.overviewData.set({ ...currentData, allAccounts });
788 }
789
790 updateOverviewStrategies(allStrategies: ConfigurationOverview[]): void {
791     const currentData = this.overviewData();
792     this.overviewData.set({ ...currentData, allStrategies });
793 }
794
795 // ===== MÉTODOS DE TRADING HISTORY POR CUENTA =====
796
797 /**
798  * Cargar trading history de la primera cuenta después del login
799  */
800 private async loadTradingHistoryAfterLogin(userId: string): Promise<void> {
801     try {
802         // Esperar a que las cuentas estén cargadas
803         const accounts = this.userAccounts();
804         if (accounts.length === 0) {
805             // Si no hay cuentas, intentar cargarlas
806             await this.loadUserAccountsIfNeeded(userId);
807             return;
808         }
809
810         // Cargar trading history de la primera cuenta
811         const firstAccount = accounts[0];
812         await this.loadTradingHistoryForAccount(firstAccount);
813     } catch (error) {
814         console.error('Error loading trading history after login:', error);
815     }
816 }
817
818 /**
819  * Cargar cuentas del usuario si no están cargadas
820  */
821 private async loadUserAccountsIfNeeded(userId: string): Promise<void> {
822     // Este método se implementará cuando se integre con AuthService
823     // Por ahora, solo logueamos que necesitamos cargar las cuentas
824     console.log('Loading user accounts for userId:', userId);
825 }
826
827 /**
828  * Cargar trading history para una cuenta específica
829  */
830 async loadTradingHistoryForAccount(account: AccountData): Promise<void> {
831     try {
832         this.setLoading('tradeLocker', true);
833         this.setError('tradeLocker', null);
834
835         // Obtener userKey
836         const userKey = await this.tradeLockerApi.getUserKey(
837             account.emailTradingAccount,
838             account.brokerPassword,
839             account.server
840         ).toPromise();
841
842         if (!userKey) {
843             throw new Error('No se pudo obtener userKey');
844         }
845
846         // Obtener balance data
847         const balanceData = await this.tradeLockerApi.getAccountBalance(
848             account.accountID,
849             userKey,
850             1
851         ).toPromise();

```



```

852
853 // Obtener trading history
854 const tradingHistory = await this.tradeLockerApi.getTradingHistory(
855     userKey,
856     account.accountID,
857     1
858 ).toPromise();
859
860 // Calcular estadísticas
861 const stats = this.calculateStatsFromTrades(tradingHistory || []);
862
863 // Guardar datos por cuenta
864 const accountData = {
865     accountHistory: tradingHistory || [],
866     stats: stats,
867     balanceData: balanceData,
868     lastUpdated: Date.now()
869 };
870
871 this.setTradingHistoryForAccount(account.id, accountData);
872
873 // Guardar en localStorage
874 this.saveTradingHistoryToLocalStorage(account.id, accountData);
875
876 this.setLoading('tradeLocker', false);
877 } catch (error) {
878     console.error('Error loading trading history for account:', error);
879     this.setLoading('tradeLocker', false);
880     this.setError('tradeLocker', 'Error al cargar trading history');
881 }
882 }
883
884 /**
885  * Calcular estadísticas desde los trades
886  */
887 private calculateStatsFromTrades(trades: GroupedTradeFinal[]): any {
888     if (!trades || trades.length === 0) {
889         return {
890             netPnl: 0,
891             tradeWinPercent: 0,
892             profitFactor: 0,
893             avgWinLossTrades: 0,
894             totalTrades: 0,
895             activePositions: 0
896         };
897     }
898
899     const normalizedTrades = trades.map(trade => ({
900         ...trade,
901         pnl: trade.pnl ?? 0
902     }));
903
904     const totalGains = normalizedTrades
905         .filter(t => t.pnl > 0)
906         .reduce((sum, t) => sum + t.pnl, 0);
907
908     const totalLosses = Math.abs(normalizedTrades
909         .filter(t => t.pnl < 0)
910         .reduce((sum, t) => sum + t.pnl, 0));
911
912     const winningTrades = normalizedTrades.filter(t => t.pnl > 0).length;
913     const totalTrades = normalizedTrades.length;
914     const winPercent = totalTrades > 0 ? (winningTrades / totalTrades) * 100 : 0;
915     const profitFactor = totalLosses > 0 ? totalGains / totalLosses : (totalGains > 0 ?
916 999.99 : 0);
917     return {
918         netPnl: Math.round((totalGains - totalLosses) * 100) / 100,
919         tradeWinPercent: Math.round(winPercent * 100) / 100,
920         profitFactor: Math.round(profitFactor * 100) / 100,
921         avgWinLossTrades: 0, // Se calculará si es necesario

```

```

922         totalTrades: totalTrades,
923         activePositions: trades.filter(trade => trade.isOpen === true).length
924     };
925 }
926
927 /**
928  * Establecer trading history para una cuenta específica
929  */
930 setTradingHistoryForAccount(accountId: string, data: {
931     accountHistory: GroupedTradeFinal[];
932     stats: any;
933     balanceData: any;
934     lastUpdated: number;
935 }): void {
936     const currentState = this.stateSubject.value;
937     const newTradingHistory = new Map(currentState.tradingHistoryByAccount);
938     newTradingHistory.set(accountId, data);
939
940     this.updateState({
941         tradingHistoryByAccount: newTradingHistory
942     });
943 }
944
945 /**
946  * Obtener trading history para una cuenta específica
947  */
948 getTradingHistoryForAccount(accountId: string): {
949     accountHistory: GroupedTradeFinal[];
950     stats: any;
951     balanceData: any;
952     lastUpdated: number;
953 } | null {
954     const currentState = this.stateSubject.value;
955     return currentState.tradingHistoryByAccount.get(accountId) || null;
956 }
957
958 /**
959  * Guardar trading history en localStorage por cuenta
960  */
961 saveTradingHistoryToLocalStorage(accountId: string, data: {
962     accountHistory: GroupedTradeFinal[];
963     stats: any;
964     balanceData: any;
965     lastUpdated: number;
966 }): void {
967     try {
968         const key = `tradeSwitch_tradingHistory_${accountId}`;
969
970         // Cargar datos existentes para preservar balanceData si es null
971         let existingBalanceData = null;
972         try {
973             const existingData = localStorage.getItem(key);
974             if (existingData) {
975                 const parsed = JSON.parse(existingData);
976                 existingBalanceData = parsed.balanceData;
977             }
978         } catch (error) {
979             // Si no hay datos existentes, continuar
980         }
981
982         const dataToSave = {
983             accountHistory: data.accountHistory,
984             stats: data.stats,
985             // Solo sobrescribir balanceData si no es null, de lo contrario preservar el
986             // existente
987             balanceData: data.balanceData !== null && data.balanceData !== undefined
988                 ? data.balanceData
989                 : existingBalanceData,
990             lastUpdated: data.lastUpdated
991         };

```

```

992     localStorage.setItem(key, JSON.stringify(dataToSave));
993 } catch (error) {
994     console.error('Error saving trading history to localStorage:', error);
995 }
996 }
997
998 /**
999  * Cargar trading history desde localStorage por cuenta
1000 */
1001 loadTradingHistoryFromLocalStorage(accountId: string): {
1002     accountHistory: GroupedTradeFinal[];
1003     stats: any;
1004     balanceData: any;
1005     lastUpdated: number;
1006 } | null {
1007     try {
1008         const key = `tradeSwitch_tradingHistory_${accountId}`;
1009         const data = localStorage.getItem(key);
1010         if (data) {
1011             return JSON.parse(data);
1012         }
1013     } catch (error) {
1014         console.error('Error loading trading history from localStorage:', error);
1015     }
1016     return null;
1017 }
1018
1019 /**
1020  * Limpiar trading history de una cuenta específica
1021 */
1022 clearTradingHistoryForAccount(accountId: string): void {
1023     const currentState = this.stateSubject.value;
1024     const newTradingHistory = new Map(currentState.tradingHistoryByAccount);
1025     newTradingHistory.delete(accountId);
1026
1027     this.updateState({
1028         tradingHistoryByAccount: newTradingHistory
1029     });
1030
1031     // Limpiar también del localStorage
1032     try {
1033         const key = `tradeSwitch_tradingHistory_${accountId}`;
1034         localStorage.removeItem(key);
1035     } catch (error) {
1036         console.error('Error clearing trading history from localStorage:', error);
1037     }
1038 }
1039
1040 /**
1041  * Cargar datos de reporte desde localStorage por accountID
1042 */
1043 loadReportDataFromLocalStorage(accountID: string): {
1044     tradingAccount: AccountData;
1045     accountHistory: GroupedTradeFinal[];
1046     stats: any;
1047     balanceData: any;
1048     lastUpdated: number;
1049 } | null {
1050     try {
1051         const key = `tradeSwitch_reportData_${accountID}`;
1052         const data = localStorage.getItem(key);
1053         if (data) {
1054             return JSON.parse(data);
1055         }
1056     } catch (error) {
1057         console.error('Error loading report data from localStorage:', error);
1058     }
1059     return null;
1060 }
1061

```

```

1062  /**
1063   * Guardar datos de reporte en localStorage por accountID
1064   */
1065   saveReportDataToLocalStorage(accountID: string, tradingAccount: AccountData, reportData: {
1066     accountHistory: GroupedTradeFinal[];
1067     stats: any;
1068     balanceData: any;
1069     lastUpdated: number;
1070 }): void {
1071   try {
1072     const key = `tradeSwitch_reportData_${accountID}`;
1073
1074     // Cargar datos existentes para preservar balanceData si es null
1075     let existingBalanceData = null;
1076     try {
1077       const existingData = localStorage.getItem(key);
1078       if (existingData) {
1079         const parsed = JSON.parse(existingData);
1080         existingBalanceData = parsed.balanceData;
1081       }
1082     } catch (error) {
1083       // Si no hay datos existentes, continuar
1084     }
1085
1086     const dataToSave = {
1087       tradingAccount: tradingAccount,
1088       accountHistory: reportData.accountHistory,
1089       stats: reportData.stats,
1090       // Solo sobrescribir balanceData si no es null, de lo contrario preservar el
1091       // existente
1092       balanceData: reportData.balanceData !== null && reportData.balanceData !== undefined
1093         ? reportData.balanceData
1094         : existingBalanceData,
1095       lastUpdated: reportData.lastUpdated
1096     };
1097
1098     localStorage.setItem(key, JSON.stringify(dataToSave));
1099   } catch (error) {
1100     console.error('Error saving report data to localStorage:', error);
1101   }
1102 }
1103 /**
1104 * Limpiar datos de reporte de localStorage por accountID
1105 */
1106 clearReportDataFromLocalStorage(accountID: string): void {
1107   try {
1108     const key = `tradeSwitch_reportData_${accountID}`;
1109     localStorage.removeItem(key);
1110   } catch (error) {
1111     console.error('Error clearing report data from localStorage:', error);
1112   }
1113 }
1114 }
1115

```

## Ø=ÜÄ shared\context\index.ts

---

```

1  // Exportaciones del contexto de la aplicación
2
3  export { AppContextService } from './context';
4  export * from './types';
5  export * from './constants';
6  export * from './utils';
7
8  // Re-exportar tipos importantes para facilitar el uso
9  export type {

```

```

10     ConfigurationOverviewWithId,
11     AccountDataWithId,
12     LoadingState,
13     ErrorState,
14     CacheConfig,
15     TradeLockerAccountData,
16     PluginHistoryData,
17     UserPlanData,
18     PlanLimitations,
19     CachedData,
20     ContextEvent,
21     ContextEventListener
22 } from './types';
23

```

## shared\context\types.ts

---

```

1  // Tipos extendidos para el contexto de la aplicación
2
3  import { AccountData } from "../../features/auth/models/userModel";
4  import { BalanceData } from "../../features/report/models/report.model";
5  import { ConfigurationOverview } from "../../features/strategy/models/strategy.model";
6
7  export interface ConfigurationOverviewWithId extends ConfigurationOverview {
8      id: string;
9  }
10
11  export interface AccountDataWithId extends AccountData {
12      id: string;
13  }
14
15  // Tipos para estados de carga
16  export interface LoadingState {
17      user: boolean;
18      accounts: boolean;
19      strategies: boolean;
20      plan: boolean;
21      pluginHistory: boolean;
22      tradeLocker: boolean;
23  }
24
25  // Tipos para estados de error
26  export interface ErrorState {
27      user: string | null;
28      accounts: string | null;
29      strategies: string | null;
30      plan: string | null;
31      pluginHistory: string | null;
32      tradeLocker: string | null;
33  }
34
35  // Tipos para configuración de caché
36  export interface CacheConfig {
37      tradeLockerTtl: number;
38      apiTtl: number;
39      maxCacheSize: number;
40  }
41
42  // Tipos para datos de API externa
43  export interface TradeLockerAccountData {
44      accountId: string;
45      balance: BalanceData;
46      lastUpdated: number;
47      isValid: boolean;
48  }
49

```

```

50 export interface PluginHistoryData {
51   id: string;
52   userId: string;
53   action: string;
54   timestamp: number;
55   details: any;
56 }
57
58 export interface UserPlanData {
59   planId: string;
60   planName: string;
61   maxAccounts: number;
62   maxStrategies: number;
63   features: string[];
64   isActive: boolean;
65   expiresAt?: number;
66 }
67
68 // Tipos para limitaciones de plan
69 export interface PlanLimitations {
70   maxAccounts: number;
71   currentAccounts: number;
72   maxStrategies: number;
73   currentStrategies: number;
74   canCreateAccount: boolean;
75   canCreateStrategy: boolean;
76 }
77
78 // Tipos para datos de caché de API
79 export interface CachedData<T = any> {
80   data: T;
81   timestamp: number;
82   ttl: number;
83 }
84
85 // Tipos para eventos del contexto
86 export type ContextEvent =
87   | 'user:login'
88   | 'user:logout'
89   | 'user:update'
90   | 'accounts:add'
91   | 'accounts:update'
92   | 'accounts:remove'
93   | 'strategies:add'
94   | 'strategies:update'
95   | 'strategies:remove'
96   | 'strategies:activate'
97   | 'plan:update'
98   | 'pluginHistory:add'
99   | 'cache:clear'
100   | 'error:set'
101   | 'error:clear';
102
103 // Tipos para listeners de eventos
104 export type ContextEventListener = (event: ContextEvent, data?: any) => void;
105

```

## **shared\context\utils.ts**

---

```

1 import { CACHE_CONFIG, PLUGIN_ACTIONS } from './constants';
2 import { TradeLockerAccountData, PluginHistoryData, UserPlanData } from './types';
3
4 // Utilidades para el contexto de la aplicación
5
6 /**
7  * Verifica si los datos están obsoletos basándose en su timestamp

```

```

8  */
9  export function isDataStale(timestamp: number, ttl: number = CACHE_CONFIG.API_TTL): boolean {
10     const now = Date.now();
11     return (now - timestamp) > ttl;
12 }
13
14 /**
15  * Verifica si los datos de TradeLocker están obsoletos
16  */
17 export function isTradeLockerDataStale(data: TradeLockerAccountData): boolean {
18     return isDataStale(data.lastUpdated, CACHE_CONFIG.TRADE_LOCKER_TTL);
19 }
20
21 /**
22  * Crea una entrada de historial de plugin
23  */
24 export function createPluginHistoryEntry(
25     userId: string,
26     action: string,
27     details: any = {}
28 ): PluginHistoryData {
29     return {
30         id: generateId(),
31         userId,
32         action,
33         timestamp: Date.now(),
34         details
35     };
36 }
37
38 /**
39  * Genera un ID único
40  */
41 export function generateId(): string {
42     return `${Date.now()}_${Math.random().toString(36).substr(2, 9)}`;
43 }
44
45 /**
46  * Valida si un plan permite crear más cuentas
47  */
48 export function canCreateAccount(plan: UserPlanData | null, currentCount: number): boolean {
49     if (!plan) return false;
50     return currentCount < plan.maxAccounts;
51 }
52
53 /**
54  * Valida si un plan permite crear más estrategias
55  */
56 export function canCreateStrategy(plan: UserPlanData | null, currentCount: number): boolean {
57     if (!plan) return false;
58     return currentCount < plan.maxStrategies;
59 }
60
61 /**
62  * Calcula las limitaciones del plan
63  */
64 export function calculatePlanLimitations(
65     plan: UserPlanData | null,
66     currentAccounts: number,
67     currentStrategies: number
68 ) {
69     return {
70         maxAccounts: plan?.maxAccounts || 0,
71         currentAccounts,
72         maxStrategies: plan?.maxStrategies || 0,
73         currentStrategies,
74         canCreateAccount: canCreateAccount(plan, currentAccounts),
75         canCreateStrategy: canCreateStrategy(plan, currentStrategies),
76         accountsRemaining: Math.max(0, (plan?.maxAccounts || 0) - currentAccounts),
77         strategiesRemaining: Math.max(0, (plan?.maxStrategies || 0) - currentStrategies)

```

```

78     };
79 }
80
81 /**
82  * Limpia datos obsoletos del caché
83  */
84 export function cleanStaleCache<T>(  

85     cache: Map<string, { data: T; timestamp: number; ttl: number }>  

86 ): Map<string, { data: T; timestamp: number; ttl: number }> {  

87     const now = Date.now();  

88     const cleanedCache = new Map();  

89
90     for (const [key, value] of cache.entries()) {  

91         if ((now - value.timestamp) <= value.ttl) {  

92             cleanedCache.set(key, value);  

93         }  

94     }  

95
96     return cleanedCache;  

97 }
98
99 /**
100  * Limita el tamaño del caché eliminando las entradas más antiguas
101  */
102 export function limitCacheSize<T>(  

103     cache: Map<string, T>,  

104     maxSize: number = CACHE_CONFIG.MAX_CACHE_SIZE  

105 ): Map<string, T> {  

106     if (cache.size <= maxSize) {  

107         return cache;  

108     }  

109
110     const entries = Array.from(cache.entries());  

111     entries.sort((a, b) => {  

112         // Asumir que T tiene una propiedad timestamp  

113         const aTime = (a[1] as any).timestamp || 0;  

114         const bTime = (b[1] as any).timestamp || 0;  

115         return aTime - bTime;  

116     });  

117
118     const toRemove = entries.length - maxSize;  

119     const limitedCache = new Map(cache);  

120
121     for (let i = 0; i < toRemove; i++) {  

122         limitedCache.delete(entries[i][0]);  

123     }  

124
125     return limitedCache;  

126 }
127
128 /**
129  * Debounce para evitar actualizaciones excesivas
130  */
131 export function debounce<T extends (...args: any[]) => any>(  

132     func: T,  

133     wait: number  

134 ): (...args: Parameters<T>) => void {  

135     let timeout: NodeJS.Timeout;  

136
137     return (...args: Parameters<T>) => {  

138         clearTimeout(timeout);  

139         timeout = setTimeout(() => func(...args), wait);  

140     };  

141 }
142
143 /**
144  * Throttle para limitar la frecuencia de actualizaciones
145  */
146 export function throttle<T extends (...args: any[]) => any>(  

147     func: T,  


```



```

148     limit: number
149 ): (...args: Parameters<T>) => void {
150     let inThrottle: boolean;
151
152     return (...args: Parameters<T>) => {
153         if (!inThrottle) {
154             func(...args);
155             inThrottle = true;
156             setTimeout(() => inThrottle = false, limit);
157         }
158     };
159 }
160
161 /**
162  * Valida si un objeto es válido para el contexto
163  */
164 export function isValidContextData(data: any): boolean {
165     return data !== null && data !== undefined && typeof data === 'object';
166 }
167
168 /**
169  * Clona profundamente un objeto para evitar mutaciones
170  */
171 export function deepClone<T>(obj: T): T {
172     if (obj === null || typeof obj !== 'object') {
173         return obj;
174     }
175
176     if (obj instanceof Date) {
177         return new Date(obj.getTime()) as T;
178     }
179
180     if (obj instanceof Array) {
181         return obj.map(item => deepClone(item)) as T;
182     }
183
184     if (typeof obj === 'object') {
185         const cloned = {} as T;
186         for (const key in obj) {
187             if (obj.hasOwnProperty(key)) {
188                 cloned[key] = deepClone(obj[key]);
189             }
190         }
191         return cloned;
192     }
193
194     return obj;
195 }
196
197 /**
198  * Crea un hash simple para un objeto
199  */
200 export function createHash(obj: any): string {
201     const str = JSON.stringify(obj);
202     let hash = 0;
203
204     for (let i = 0; i < str.length; i++) {
205         const char = str.charCodeAt(i);
206         hash = ((hash << 5) - hash) + char;
207         hash = hash & hash; // Convertir a 32-bit integer
208     }
209
210     return hash.toString(36);
211 }
212
213 /**
214  * Verifica si dos objetos son iguales (shallow comparison)
215  */
216 export function shallowEqual(obj1: any, obj2: any): boolean {
217     if (obj1 === obj2) return true;

```

```

218
219     if (obj1 == null || obj2 == null) return false;
220
221     if (typeof obj1 !== 'object' || typeof obj2 !== 'object') return false;
222
223     const keys1 = Object.keys(obj1);
224     const keys2 = Object.keys(obj2);
225
226     if (keys1.length !== keys2.length) return false;
227
228     for (let key of keys1) {
229         if (obj1[key] !== obj2[key]) return false;
230     }
231
232     return true;
233 }
234

```

## Ø=ÜÁ shared\interfaces

### Ø=ÜÄ shared\interfaces\plan-limitation-modal.interface.ts

---

```

1  export interface PlanLimitationModalData {
2      showModal: boolean;
3      modalType: 'upgrade' | 'blocked';
4      title: string;
5      message: string;
6      primaryButtonText: string;
7      secondaryButtonText?: string;
8      onPrimaryAction: () => void;
9      onSecondaryAction?: () => void;
10 }
11

```

## Ø=ÜÁ shared\mobile-header

### Ø=ÜÄ shared\mobile-header\mobile-header.component.ts

---

```

1  import { Component, OnDestroy } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { RouterModule, Router } from '@angular/router';
4  import { Store } from '@ngrx/store';
5  import { selectUser } from '../../../features/auth/store/user.selectios';
6  import { setUserData } from '../../../features/auth/store/user.actions';
7  import { UserStatus } from '../../../features/overview/models/overview';
8  import { AuthService } from '../../../features/auth/service/authService';
9
10 /**
11  * Mobile header component for responsive navigation.
12  *
13  * This component provides a mobile-friendly navigation header with a hamburger
14  * menu, user information, and logout functionality. It's designed for smaller
15  * screens where the sidebar is replaced by a collapsible header menu.
16  *
17  * Features:
18  * - Mobile menu toggle (hamburger menu)

```

```

19  * - User information display (name, initials, admin status, ban status)
20  * - Logout functionality
21  * - Responsive design for mobile devices
22  * - Menu open/close state management
23  *
24  * Relations:
25  * - AuthService: Handles logout functionality
26  * - Store (NgRx): Gets current user data
27  * - Router: Navigation after logout
28  *
29  * @component
30  * @selector app-mobile-header
31  * @standalone true
32  */
33  @Component({
34      selector: 'app-mobile-header',
35      standalone: true,
36      imports: [CommonModule, RouterModule],
37      templateUrl: './mobile-header.component.html',
38      styleUrls: ['./mobile-header.component.scss']
39  })
40  export class MobileHeaderComponent implements OnDestroy {
41      isMobileMenuOpen = false;
42      userName: string = '';
43      lastName: string = '';
44      isAdmin: boolean = false;
45      userToken: string = '';
46      isBanned: boolean = false;
47
48      constructor(
49          private authService: AuthService,
50          private router: Router,
51          private store: Store
52      ) {
53          this.store.select(selectUser).subscribe((user) => {
54              this.userName = user?.user?.firstName || '';
55              this.lastName = user?.user?.lastName || '';
56              this.isAdmin = user?.user?.isAdmin || false;
57              this.userToken = user?.user?.tokenId || '';
58              this.isBanned = user?.user?.status === UserStatus.BANNED;
59          });
60      }
61
62      ngOnDestroy() {
63          // Cleanup si es necesario
64      }
65
66      toggleMobileMenu() {
67          this.isMobileMenuOpen = !this.isMobileMenuOpen;
68      }
69
70      closeMobileMenu() {
71          this.isMobileMenuOpen = false;
72      }
73
74      onlyNameInitials(): string {
75          if (!this.userName) return '';
76          return this.userName.charAt(0) + this.lastName.charAt(1);
77      }
78
79      logout() {
80          this.closeMobileMenu();
81          this.authService
82              .logout()
83              .then(() => {
84                  // Limpiar todo el localStorage
85                  localStorage.clear();
86                  this.store.dispatch(setUserData({ user: null }));
87                  this.router.navigate(['/login']);
88              })

```

```

89         .catch((error) => {
90             alert('Logout failed. Please try again.');
```

## shared\pipes

### shared\pipes\currency-format.pipe.ts

---

```

1  import { Pipe, PipeTransform } from '@angular/core';
2  import { NumberFormatterService } from '../utils/number-formatter.service';
3
4  /**
5   * Pipe for formatting numbers as currency.
6   *
7   * This pipe transforms numeric values into formatted currency strings using
8   * the NumberFormatterService. It handles null/undefined values gracefully
9   * and formats values as USD currency with proper separators.
10  *
11  * Features:
12  * - Formats numbers as currency (USD)
13  * - Handles null/undefined values (returns '$0.00')
14  * - Handles string inputs (converts to number)
15  * - Uses NumberFormatterService for consistent formatting
16  *
17  * Usage:
18  * {{ value | currencyFormat }}
19  *
20  * Relations:
21  * - NumberFormatterService: Provides the actual formatting logic
22  *
23  * @pipe
24  * @name currencyFormat
25  * @standalone true
26  */
27  @Pipe({
28      name: 'currencyFormat',
29      standalone: true
30  })
31  export class CurrencyFormatPipe implements PipeTransform {
32
33      constructor(private numberFormatter: NumberFormatterService) {}
34
35      transform(value: number | string | null | undefined): string {
36          return this.numberFormatter.formatCurrency(value);
37      }
38  }
39
```

### shared\pipes\number-format.pipe.ts

---

```

1  import { Pipe, PipeTransform } from '@angular/core';
2  import { NumberFormatterService } from '../utils/number-formatter.service';
3
4  /**
5   * Pipe for formatting numbers with thousand separators.
6   *
```

```

7  * This pipe transforms numeric values into formatted number strings with
8  * proper thousand separators and decimal places. It uses the NumberFormatterService
9  * for consistent formatting across the application.
10 *
11 * Features:
12 * - Formats numbers with thousand separators
13 * - Configurable decimal places (default: 2)
14 * - Handles null/undefined values (returns '0.00')
15 * - Handles string inputs (converts to number)
16 *
17 * Usage:
18 * {{ value | numberFormat }}           // 2 decimals (default)
19 * {{ value | numberFormat:0 }}         // 0 decimals
20 * {{ value | numberFormat:4 }}         // 4 decimals
21 *
22 * Relations:
23 * - NumberFormatterService: Provides the actual formatting logic
24 *
25 * @pipe
26 * @name numberFormat
27 * @standalone true
28 */
29 @Pipe({
30   name: 'numberFormat',
31   standalone: true
32 })
33 export class NumberFormatPipe implements PipeTransform {
34
35   constructor(private numberFormatter: NumberFormatterService) {}
36
37   transform(value: number | string | null | undefined, decimals: number = 2): string {
38     return this.numberFormatter.formatNumber(value, decimals);
39   }
40 }
41

```

## shared\pipes\percentage-format.pipe.ts

---

```

1  import { Pipe, PipeTransform } from '@angular/core';
2  import { NumberFormatterService } from '../utils/number-formatter.service';
3
4  /**
5   * Pipe for formatting numbers as percentages.
6   *
7   * This pipe transforms numeric values into formatted percentage strings with
8   * the % symbol. It uses the NumberFormatterService for consistent formatting
9   * and handles the conversion from decimal to percentage format.
10  *
11  * Features:
12  * - Formats numbers as percentages with % symbol
13  * - Handles null/undefined values (returns '0.00%')
14  * - Handles string inputs (converts to number)
15  * - Converts decimal values to percentage (e.g., 0.5 !' 50.00%)
16  *
17  * Usage:
18  * {{ value | percentageFormat }}
19  *
20  * Relations:
21  * - NumberFormatterService: Provides the actual formatting logic
22  *
23  * @pipe
24  * @name percentageFormat
25  * @standalone true
26  */
27  @Pipe({
28    name: 'percentageFormat',

```

```

29     standalone: true
30   })
31   export class PercentageFormatPipe implements PipeTransform {
32
33     constructor(private numberFormatter: NumberFormatterService) {}
34
35     transform(value: number | string | null | undefined): string {
36       return this.numberFormatter.formatPercentage(value);
37     }
38   }
39

```

## Ø=ÜÁ shared\pop-ups\alert-popup

### Ø=ÜÄ shared\pop-ups\alert-popup\alert-popup.component.ts

---

```

1  import { Component, Input, Output, EventEmitter } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3
4  /**
5   * Component for displaying alert dialogs.
6   *
7   * This component provides a reusable alert dialog that can display different
8   * types of messages (info, warning, error, success) with customizable title,
9   * message, and button text.
10  *
11  * Features:
12  * - Multiple alert types (info, warning, error, success)
13  * - Customizable title and message
14  * - Customizable button text
15  * - Visibility control
16  * - Close and confirm events
17  *
18  * Usage:
19  * <app-alert-popup
20  *   [visible]="showAlert"
21  *   [title]="alertTitle"
22  *   [message]="alertMessage"
23  *   [type]="'error'"
24  *   [buttonText]="'OK'"
25  *   (close)="onCloseAlert()">
26  * </app-alert-popup>
27  *
28  * Relations:
29  * - AlertService: Often used together to show alerts
30  *
31  * @component
32  * @selector app-alert-popup
33  * @standalone true
34  */
35  @Component({
36    selector: 'app-alert-popup',
37    standalone: true,
38    imports: [CommonModule],
39    templateUrl: './alert-popup.component.html',
40    styleUrls: ['./alert-popup.component.scss']
41  })
42  export class AlertPopupComponent {
43    @Input() visible: boolean = false;
44    @Input() title: string = 'Alert';
45    @Input() message: string = '';
46    @Input() buttonText: string = 'OK';
47    @Input() type: 'info' | 'warning' | 'error' | 'success' = 'info';

```

```

48
49     @Output() close = new EventEmitter<void>();
50     @Output() confirm = new EventEmitter<void>();
51
52     onClose(): void {
53         this.close.emit();
54     }
55
56     onConfirm(): void {
57         this.confirm.emit();
58         this.close.emit();
59     }
60 }
61

```

## shared\pop-ups\confirm-pop-up

### shared\pop-ups\confirm-pop-up\confirm-popup.component.ts

---

```

1  import { Component, Input } from '@angular/core';
2
3  import { CommonModule } from '@angular/common';
4
5  /**
6   * Component for displaying confirmation dialogs.
7   *
8   * This component provides a reusable confirmation dialog for actions that
9   * require user confirmation. It supports both regular and dangerous actions
10  * with customizable messages and button texts.
11  *
12  * Features:
13  * - Customizable title and message
14  * - Customizable confirm and cancel button texts
15  * - Dangerous action styling (for destructive actions)
16  * - Close and cancel callbacks
17  * - Visibility control
18  *
19  * Usage:
20  * <app-confirm-popup
21  *   [visible]="showConfirm"
22  *   [title]=" 'Confirm Action' "
23  *   [message]=" 'Are you sure?' "
24  *   [confirmButtonText]=" 'Confirm' "
25  *   [cancelButtonText]=" 'Cancel' "
26  *   [isDangerous]="false"
27  *   [close]="onClose"
28  *   [cancel]="onCancel">
29  * </app-confirm-popup>
30  *
31  * Relations:
32  * - Used by components that require user confirmation before actions
33  *
34  * @component
35  * @selector app-confirm-popup
36  * @standalone true
37  */
38  @Component({
39     selector: 'app-confirm-popup',
40     imports: [CommonModule],
41     templateUrl: './confirm-popup.component.html',
42     styleUrls: ['./confirm-popup.component.scss'],
43     standalone: true,
44 })

```

```

45 export class ConfirmPopupComponent {
46   @Input() visible = false;
47   @Input() close!: () => void;
48   @Input() cancel!: () => void;
49   @Input() title: string = 'Confirm changes to your strategy?';
50   @Input() message: string = 'These updates will modify how your trading rules are applied.';
51   @Input() confirmButtonText: string = 'Apply changes';
52   @Input() cancelButtonText: string = 'Cancel';
53   @Input() isDangerous: boolean = false; // Para acciones destructivas (eliminar, etc.)
54 }
55

```

## Ø=ÜÄ shared\pop-ups\edit-pop-up

### Ø=ÜÄ shared\pop-ups\edit-pop-up\edit-popup.component.ts

---

```

1  import { Component, Input } from '@angular/core';
2
3  /**
4   * Component for displaying an edit popup overlay.
5   *
6   * This component provides a simple popup overlay that can be used to display
7   * edit forms or content. It includes a close callback for handling dismissal.
8   *
9   * Features:
10  * - Visibility control
11  * - Close callback function
12  * - Reusable popup overlay
13  *
14  * Usage:
15  * <app-edit-popup
16  *   [visible]="showEditPopup"
17  *   [close]="onCloseEditPopup">
18  * </app-edit-popup>
19  *
20  * Relations:
21  * - Used by components that need edit popup overlays
22  *
23  * @component
24  * @selector app-edit-popup
25  * @standalone true
26  */
27  @Component({
28    selector: 'app-edit-popup',
29    imports: [],
30    templateUrl: './edit-popup.component.html',
31    styleUrls: ['./edit-popup.component.scss'],
32    standalone: true,
33  })
34  export class EditPopupComponent {
35    @Input() visible = false;
36    @Input() close!: () => void;
37  }
38

```



## Ø=ÜÁ shared\pop-ups\forgot-password

### Ø=ÜÄ shared\pop-ups\forgot-password\forgot-password.component.ts

---

```
1 import { Component, EventEmitter, Input, Output } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';
4 import { AuthService } from '../../services/auth.service';
5
6 /**
7  * Component for password reset functionality.
8  *
9  * This component provides a modal interface for users to request a password
10 * reset email. It includes form validation, error handling, and success
11 * messaging.
12 *
13 * Features:
14 * - Email input with validation
15 * - Password reset email sending
16 * - Success and error message display
17 * - Loading state management
18 * - Form validation
19 * - Close/back functionality
20 *
21 * Validation:
22 * - Email is required
23 * - Email must be valid format
24 *
25 * Error Handling:
26 * - Handles user-not-found errors
27 * - Handles invalid-email errors
28 * - Generic error messages for other failures
29 *
30 * Relations:
31 * - AuthService: Sends password reset email
32 *
33 * @component
34 * @selector app-forgot-password-popup
35 * @standalone true
36 */
37 @Component({
38   selector: 'app-forgot-password-popup',
39   standalone: true,
40   imports: [CommonModule, ReactiveFormsModule],
41   templateUrl: './forgot-password.component.html',
42   styleUrls: ['./forgot-password.component.scss']
43 })
44 export class ForgotPasswordPopupComponent {
45   @Input() visible: boolean = false;
46   @Output() close = new EventEmitter<void>();
47
48   form: FormGroup;
49   submitted = false;
50   loading = false;
51   successMessage = '';
52   errorMessage = '';
53
54   constructor(private fb: FormBuilder, private authService: AuthService) {
55     this.form = this.fb.group({
56       email: ['', [Validators.required, Validators.email]]
57     });
58   }
59
60   onBack(): void {
61     this.resetFeedback();
62     this.close.emit();
63   }
64 }
```

```

65     async onSubmit(): Promise<void> {
66         this.submitted = true;
67         this.successMessage = '';
68         this.errorMessage = '';
69         if (this.form.invalid || this.loading) return;
70
71         this.loading = true;
72         const email = this.form.value.email as string;
73         try {
74             await this.authService.sendPasswordReset(email);
75             this.successMessage = 'We have sent you an email with instructions.';
76         } catch (error: any) {
77             if (error?.code === 'auth/user-not-found') {
78                 this.errorMessage = 'No account found with this email.';
79             } else if (error?.code === 'auth/invalid-email') {
80                 this.errorMessage = 'Invalid email format.';
81             } else {
82                 this.errorMessage = 'Failed to send reset email. Please try again later.';
83             }
84         } finally {
85             this.loading = false;
86         }
87     }
88
89     private resetFeedback(): void {
90         this.submitted = false;
91         this.successMessage = '';
92         this.errorMessage = '';
93         this.loading = false;
94     }
95 }
96
97
98

```

## Ø=ÜÄ shared\pop-ups\loading-pop-up

### Ø=ÜÄ shared\pop-ups\loading-pop-up\loading-popup.component.ts

---

```

1  import { Component, Input } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3
4  /**
5   * Component for displaying a full-screen loading overlay.
6   *
7   * This component provides a modal-style loading indicator that covers the entire
8   * screen with a semi-transparent overlay. It displays a spinner and loading message
9   * to indicate that an operation is in progress.
10  *
11  * Features:
12  * - Full-screen overlay with semi-transparent background
13  * - Animated spinner
14  * - Loading message
15  * - Visibility control via Input property
16  * - Fixed positioning with high z-index
17  *
18  * Usage:
19  * <app-loading-popup [visible]="isLoading"></app-loading-popup>
20  *
21  * Relations:
22  * - Used by components that need to block user interaction during loading
23  *
24  * @component

```

```

25 * @selector app-loading-popup
26 * @standalone true
27 */
28 @Component({
29   selector: 'app-loading-popup',
30   standalone: true,
31   imports: [CommonModule],
32   template: `
33     <div class="loading-overlay" *ngIf="visible">
34       <div class="loading-box">
35         <div class="spinner"></div>
36         <p>Loading data...</p>
37       </div>
38     </div>
39   `,
40   styles: [
41     `
42     .loading-overlay {
43       position: fixed;
44       top: 0;
45       left: 0;
46       right: 0;
47       bottom: 0;
48       background: rgba(0, 0, 0, 0.5);
49       display: flex;
50       align-items: center;
51       justify-content: center;
52       z-index: 2000;
53     }
54     .loading-box {
55       background: #23252b;
56       padding: 24px 36px;
57       border-radius: 12px;
58       color: #fff;
59       display: flex;
60       flex-direction: column;
61       align-items: center;
62     }
63     .spinner {
64       border: 4px solid rgba(255, 255, 255, 0.2);
65       border-top: 4px solid #c8fc00;
66       border-radius: 50%;
67       width: 40px;
68       height: 40px;
69       animation: spin 1s linear infinite;
70       margin-bottom: 12px;
71     }
72     @keyframes spin {
73       0% {
74         transform: rotate(0deg);
75       }
76       100% {
77         transform: rotate(360deg);
78       }
79     }
80   `,
81   ],
82 })
83 export class LoadingPopupComponent {
84   @Input() visible = false;
85 }
86

```

## shared\pop-ups\stripe-loader-popup

### shared\pop-ups\stripe-loader-popup\stripe-loader-popup.component.ts

---

```
1 import { Component, Input } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 /**
5  * Component for displaying a loading overlay during Stripe redirect.
6  *
7  * This component provides a full-screen loading overlay that is shown
8  * when redirecting users to Stripe for payment processing. It displays
9  * a loading message to inform users that they are being redirected.
10  *
11  * Features:
12  * - Full-screen overlay
13  * - Customizable loading message
14  * - Visibility control
15  *
16  * Usage:
17  * <app-stripe-loader-popup
18  *   [visible]="isRedirecting"
19  *   [message]="Redirecting to Stripe...">
20  * </app-stripe-loader-popup>
21  *
22  * Relations:
23  * - Used by PlanSettingsComponent during Stripe checkout redirect
24  * - Used when opening Stripe customer portal
25  *
26  * @component
27  * @selector app-stripe-loader-popup
28  * @standalone true
29  */
30 @Component({
31   selector: 'app-stripe-loader-popup',
32   standalone: true,
33   imports: [CommonModule],
34   templateUrl: './stripe-loader-popup.component.html',
35   styleUrls: ['./stripe-loader-popup.component.scss']
36 })
37 export class StripeLoaderPopupComponent {
38   @Input() visible: boolean = false;
39   @Input() message: string = 'Redirecting to Stripe...';
40 }
41
```

## shared\services

### shared\services\account-deletion.service.ts

---

```
1 import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2 import { isPlatformBrowser } from '@angular/common';
3 import {
4   getFirestore,
5   collection,
6   query,
7   where,
8   getDocs,
```

```

9     deleteDoc,
10    doc,
11    writeBatch
12 } from 'firebase/firestore';
13 import { firebaseApp } from '../../firebase/firebase.init';
14
15 /**
16  * Service for comprehensive user account deletion.
17  *
18  * This service handles the complete deletion of all user data from Firebase
19  * when a user requests account deletion. It uses batch operations for atomic
20  * deletion of all related data across multiple collections.
21  *
22  * Features:
23  * - Delete all user data atomically (batch operations)
24  * - Delete user accounts
25  * - Delete user strategies (configuration-overview and configurations)
26  * - Delete monthly reports
27  * - Delete plugin history
28  * - Delete link tokens
29  * - Delete user subscriptions
30  * - Delete user document
31  *
32  * Deletion Process:
33  * 1. Collects all user data to delete
34  * 2. Uses Firestore batch for atomic operations
35  * 3. Deletes in order: accounts !' strategies !' reports !' plugin history !' tokens !' subscription
36  * Returns success/failure status
37  *
38  * Data Deleted:
39  * - `accounts`: All user trading accounts
40  * - `configuration-overview`: Strategy metadata
41  * - `configurations`: Strategy rules
42  * - `monthly_reports`: Monthly trading reports
43  * - `plugin_history`: Plugin activation history
44  * - `tokens`: Link tokens
45  * - `users/{userId}/subscription`: Subscription subcollection
46  * - `users/{userId}`: User document
47  *
48  * Relations:
49  * - Used by ProfileDetailsComponent for account deletion
50  * - Ensures complete data removal for GDPR compliance
51  *
52  * @service
53  * @injectable
54  * @providedIn root
55  */
56 @Injectable({
57   providedIn: 'root'
58 })
59 export class AccountDeletionService {
60   private isBrowser: boolean;
61   private db: ReturnType<typeof getFirestore> | null = null;
62
63   constructor(@Inject(PLATFORM_ID) private platformId: Object) {
64     this.isBrowser = isPlatformBrowser(this.platformId);
65     if (this.isBrowser) {
66       this.db = getFirestore(firebaseApp);
67     }
68   }
69
70   /**
71    * Deletes all data associated with a user from Firebase
72    * @param userId - ID of the user to delete
73    * @returns Promise<boolean> - true if deleted successfully, false if there was an error
74    */
75   async deleteUserData(userId: string): Promise<boolean> {
76     if (!this.db) {
77       console.warn('Firestore not available in SSR');
78       return false;

```

```

79     }
80
81     try {
82
83         // Use batch for atomic operations
84         const batch = writeBatch(this.db);
85         let operationsCount = 0;
86
87         // 1. Delete accounts
88         const accountsDeleted = await this.deleteUserAccounts(userId, batch);
89         operationsCount += accountsDeleted;
90
91         // 2. Delete configuration-overview and associated configurations
92         const configsDeleted = await this.deleteUserConfigurations(userId, batch);
93         operationsCount += configsDeleted;
94
95         // 3. Delete monthly_reports
96         const reportsDeleted = await this.deleteUserMonthlyReports(userId, batch);
97         operationsCount += reportsDeleted;
98
99         // 4. Delete plugin_history
100        const pluginHistoryDeleted = await this.deleteUserPluginHistory(userId, batch);
101        operationsCount += pluginHistoryDeleted;
102
103        // 5. Delete tokens
104        const tokensDeleted = await this.deleteUserTokens(userId, batch);
105        operationsCount += tokensDeleted;
106
107        // 6. Delete user subscription subcollection
108        const subscriptionsDeleted = await this.deleteUserSubscriptions(userId, batch);
109        operationsCount += subscriptionsDeleted;
110
111        // 7. Delete user from users collection
112        const userDeleted = await this.deleteUser(userId, batch);
113        operationsCount += userDeleted;
114
115        // Execute all operations in batch
116        if (operationsCount > 0) {
117            await batch.commit();
118            return true;
119        } else {
120            console.log(`& p No data found to delete for user ${userId}`);
121            return true; // Not an error if no data exists
122        }
123
124    } catch (error) {
125        console.error('L Error deleting user data:', error);
126        return false;
127    }
128 }
129
130 /**
131  * Deletes all user accounts
132  */
133 private async deleteUserAccounts(userId: string, batch: any): Promise<number> {
134     try {
135         const accountsRef = collection(this.db!, 'accounts');
136         const q = query(accountsRef, where('userId', '==', userId));
137         const querySnapshot = await getDocs(q);
138
139         let count = 0;
140         querySnapshot.forEach((docSnapshot) => {
141             batch.delete(docSnapshot.ref);
142             count++;
143         });
144
145         console.log(`Ø-ÜÊ Deleting ${count} accounts for user ${userId}`);
146         return count;
147     } catch (error) {
148         console.error('Error deleting accounts:', error);

```

```

149         return 0;
150     }
151 }
152
153 /**
154  * Deletes configuration-overview and associated configurations
155  */
156 private async deleteUserConfigurations(userId: string, batch: any): Promise<number> {
157     try {
158         const configOverviewRef = collection(this.db!, 'configuration-overview');
159         const q = query(configOverviewRef, where('userId', '==', userId));
160         const querySnapshot = await getDocs(q);
161
162         let count = 0;
163         const configIds: string[] = [];
164
165         // First collect configurationId to delete configurations
166         querySnapshot.forEach((docSnapshot) => {
167             const data = docSnapshot.data();
168             if (data['configurationId']) {
169                 configIds.push(data['configurationId']);
170             }
171             batch.delete(docSnapshot.ref);
172             count++;
173         });
174
175         // Delete associated configurations
176         for (const configId of configIds) {
177             const configRef = doc(this.db!, 'configurations', configId);
178             batch.delete(configRef);
179             count++;
180         }
181
182         console.log(`\n Deleting ${count} configurations for user ${userId}`);
183         return count;
184     } catch (error) {
185         console.error('Error deleting configurations:', error);
186         return 0;
187     }
188 }
189
190 /**
191  * Deletes user monthly_reports
192  */
193 private async deleteUserMonthlyReports(userId: string, batch: any): Promise<number> {
194     try {
195         const reportsRef = collection(this.db!, 'monthly_reports');
196         const q = query(reportsRef, where('id', '==', userId));
197         const querySnapshot = await getDocs(q);
198
199         let count = 0;
200         querySnapshot.forEach((docSnapshot) => {
201             batch.delete(docSnapshot.ref);
202             count++;
203         });
204
205         console.log(`\n Deleting ${count} monthly reports for user ${userId}`);
206         return count;
207     } catch (error) {
208         console.error('Error deleting monthly reports:', error);
209         return 0;
210     }
211 }
212
213 /**
214  * Deletes user plugin_history
215  */
216 private async deleteUserPluginHistory(userId: string, batch: any): Promise<number> {
217     try {
218         const pluginHistoryRef = collection(this.db!, 'plugin_history');

```

```

219     const q = query(pluginHistoryRef, where('id', '=', userId));
220     const querySnapshot = await getDocs(q);
221
222     let count = 0;
223     querySnapshot.forEach((docSnapshot) => {
224         batch.delete(docSnapshot.ref);
225         count++;
226     });
227
228     console.log(`ðŸ“œ Deleting ${count} plugin histories for user ${userId}`);
229     return count;
230 } catch (error) {
231     console.error('Error deleting plugin history:', error);
232     return 0;
233 }
234 }
235
236 /**
237  * Deletes user tokens
238  */
239 private async deleteUserTokens(userId: string, batch: any): Promise<number> {
240     try {
241         const tokensRef = collection(this.db!, 'tokens');
242         const q = query(tokensRef, where('userId', '=', userId));
243         const querySnapshot = await getDocs(q);
244
245         let count = 0;
246         querySnapshot.forEach((docSnapshot) => {
247             batch.delete(docSnapshot.ref);
248             count++;
249         });
250
251         console.log(`ðŸ“œ Deleting ${count} tokens for user ${userId}`);
252         return count;
253     } catch (error) {
254         console.error('Error deleting tokens:', error);
255         return 0;
256     }
257 }
258
259 /**
260  * Deletes user subscription subcollection
261  */
262 private async deleteUserSubscriptions(userId: string, batch: any): Promise<number> {
263     try {
264         const subscriptionsRef = collection(this.db!, 'users', userId, 'subscription');
265         const querySnapshot = await getDocs(subscriptionsRef);
266
267         let count = 0;
268         querySnapshot.forEach((docSnapshot) => {
269             batch.delete(docSnapshot.ref);
270             count++;
271         });
272
273         console.log(`ðŸ“œ Deleting ${count} subscriptions for user ${userId}`);
274         return count;
275     } catch (error) {
276         console.error('Error deleting subscriptions:', error);
277         return 0;
278     }
279 }
280
281 /**
282  * Deletes user from users collection
283  */
284 private async deleteUser(userId: string, batch: any): Promise<number> {
285     try {
286         const userRef = doc(this.db!, 'users', userId);
287         batch.delete(userRef);
288         console.log(`ðŸ“œ Deleting user ${userId} from users collection`);

```



```

289         return 1;
290     } catch (error) {
291         console.error('Error deleting user:', error);
292         return 0;
293     }
294 }
295 }
296

```

## Ø=ÜÄ shared\services\accounts-operations.service.ts

---

```

1  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2  import { getFirestore, doc, setDoc, getDoc, collection, query, where, getDocs, deleteDoc,
3  updateDoc, isPlatformBrowser, firebase } from '@angular/common';
4  import { AccountData } from '../../../features/auth/models/userModel';
5
6  /**
7   * Service for trading account operations in Firebase.
8   *
9   * This service provides CRUD operations for trading accounts, including
10  * creation, retrieval, updates, and deletion. It also includes validation
11  * methods to check for duplicate emails and account IDs.
12  *
13  * Features:
14  * - Create trading account
15  * - Get user accounts
16  * - Get all accounts
17  * - Check if email exists (for validation)
18  * - Check if account ID exists (for validation)
19  * - Update account
20  * - Delete account (returns userId for cache invalidation)
21  *
22  * Account Validation:
23  * - Checks for duplicate email addresses across users
24  * - Checks for duplicate account IDs across users
25  * - Excludes current user's accounts from duplicate checks
26  *
27  * Data Structure:
28  * - Stored in: `accounts/{accountId}`
29  * - Contains: Account credentials, broker info, balance, trading stats
30  *
31  * Relations:
32  * - Used by AuthService for account management
33  * - Used by TradingAccountsComponent for account operations
34  * - Used by CreateAccountPopupComponent for account creation
35  *
36  * @service
37  * @injectable
38  * @providedIn root
39  */
40  @Injectable({
41      providedIn: 'root'
42  })
43  export class AccountsOperationsService {
44      private isBrowser: boolean;
45      private db: ReturnType<typeof getFirestore> | null = null;
46
47      constructor(@Inject(PLATFORM_ID) private platformId: Object) {
48          this.isBrowser = isPlatformBrowser(this.platformId);
49          if (this.isBrowser) {
50              const { firebaseApp } = require('../../../firebase/firebase.init.ts');
51              this.db = getFirestore(firebaseApp);
52          }
53      }
54
55      /**

```

```

56     * Crear cuenta de trading
57     */
58     async createAccount(account: AccountData): Promise<void> {
59         if (!this.db) {
60             console.warn('Firestore not available in SSR');
61             return;
62         }
63         await setDoc(doc(this.db, 'accounts', account.id), account);
64     }
65
66     /**
67     * Obtener cuentas de un usuario
68     */
69     async getUserAccounts(userId: string): Promise<AccountData[] | null> {
70         if (!this.db) {
71             console.warn('Firestore not available in SSR');
72             return null;
73         }
74         const accountsCollection = collection(this.db, 'accounts');
75         const q = query(accountsCollection, where('userId', '==', userId));
76         const querySnapshot = await getDocs(q);
77         const accounts: AccountData[] = [];
78         querySnapshot.forEach((doc) => {
79             accounts.push(doc.data() as AccountData);
80         });
81         return accounts.length > 0 ? accounts : null;
82     }
83
84     /**
85     * Obtener todas las cuentas
86     */
87     async getAllAccounts(): Promise<AccountData[] | null> {
88         if (!this.db) {
89             console.warn('Firestore not available in SSR');
90             return null;
91         }
92         const accountsCollection = collection(this.db, 'accounts');
93         const querySnapshot = await getDocs(accountsCollection);
94         const accounts: AccountData[] = [];
95         querySnapshot.forEach((doc) => {
96             accounts.push(doc.data() as AccountData);
97         });
98         return accounts.length > 0 ? accounts : null;
99     }
100
101     /**
102     * Verificar si un email de trading ya existe
103     */
104     async checkEmailExists(emailTradingAccount: string, currentUserId: string):
105     Promise<boolean> {
106         console.warn('Firestore not available in SSR');
107         return false;
108     }
109     const accountsCollection = collection(this.db, 'accounts');
110     const q = query(
111         accountsCollection,
112         where('emailTradingAccount', '==', emailTradingAccount),
113         where('userId', '!=', currentUserId) // Exclude current user's accounts
114     );
115     const querySnapshot = await getDocs(q);
116     return !querySnapshot.empty; // Returns true if email exists for another user
117 }
118
119 /**
120 * Verificar si un accountID ya existe
121 */
122 async checkAccountIdExists(accountID: string, currentUserId: string): Promise<boolean> {
123     if (!this.db) {
124         console.warn('Firestore not available in SSR');
125         return false;

```

```

126     }
127     const accountsCollection = collection(this.db, 'accounts');
128     const q = query(
129         accountsCollection,
130         where('accountID', '==', accountID),
131         where('userId', '!=', currentUserId) // Exclude current user's accounts
132     );
133     const querySnapshot = await getDocs(q);
134     return !querySnapshot.empty; // Returns true if accountID exists for another user
135 }
136
137 /**
138  * Actualizar cuenta
139  */
140 async updateAccount(accountId: string, accountData: AccountData): Promise<void> {
141     if (!this.db) {
142         console.warn('Firestore not available in SSR');
143         return;
144     }
145     const accountDoc = doc(this.db, 'accounts', accountId);
146     await updateDoc(accountDoc, {
147         accountName: accountData.accountName,
148         broker: accountData.broker,
149         server: accountData.server,
150         emailTradingAccount: accountData.emailTradingAccount,
151         brokerPassword: accountData.brokerPassword,
152         accountID: accountData.accountID,
153         accountNumber: accountData.accountNumber,
154         balance: accountData.balance,
155         initialBalance: accountData.initialBalance,
156         netPnl: accountData.netPnl,
157         profit: accountData.profit,
158         bestTrade: accountData.bestTrade,
159     });
160 }
161
162 /**
163  * Eliminar cuenta
164  */
165 async deleteAccount(accountId: string): Promise<string | null> {
166     if (!this.db) {
167         console.warn('Firestore not available in SSR');
168         return null;
169     }
170
171     // Obtener el userId antes de eliminar la cuenta
172     const accountDoc = doc(this.db, 'accounts', accountId);
173     const accountSnap = await getDoc(accountDoc);
174
175     if (!accountSnap.exists()) {
176         throw new Error('Account not found');
177     }
178
179     const accountData = accountSnap.data() as AccountData;
180     const userId = accountData.userId || null;
181
182     // Eliminar la cuenta
183     await deleteDoc(accountDoc);
184
185     // Retornar el userId para poder actualizar los conteos
186     return userId;
187 }
188
189 /**
190  * Verificar unicidad de cuenta (email y accountID)
191  */
192 async validateAccountUniqueness(emailTradingAccount: string, accountID: string,
193     currentUserID: string): Promise<{ isValid: boolean; message: string }> {
194     const [emailExists, accountIdExists] = await Promise.all([
195         this.checkEmailExists(emailTradingAccount, currentUserId),

```

```

196         this.checkAccountIdExists(accountID, currentUserID)
197     });
198
199     if (emailExists || accountIdExists) {
200         return {
201             isValid: false,
202             message: 'This account is already registered, try with another account or delete
203 this trade account first'
204         }
205
206         return {
207             isValid: true,
208             message: 'Account creation/update successful'
209         };
210     } catch (error) {
211         console.error('Error validating account uniqueness:', error);
212         return {
213             isValid: false,
214             message: 'Error validating account uniqueness'
215         };
216     }
217 }
218
219 /**
220  * Obtener el número total de cuentas de trading de un usuario
221  */
222 async getAllLengthUserAccounts(userId: string): Promise<number> {
223     if (!this.db) {
224         console.warn('Firestore not available in SSR');
225         return 0;
226     }
227
228     try {
229         const accountsCollection = collection(this.db, 'accounts');
230         const q = query(accountsCollection, where('userId', '==', userId));
231         const querySnapshot = await getDocs(q);
232         return querySnapshot.size;
233     } catch (error) {
234         console.error('Error getting accounts count:', error);
235         return 0;
236     }
237 }
238
239 /**
240  * Verificar si existe una cuenta con la combinación broker + server + accountID
241  */
242 async checkAccountExists(broker: string, server: string, accountID: string, currentUserID:
243 string): Promise<boolean> {
244     console.warn('Firestore not available in SSR');
245     return false;
246 }
247
248 try {
249     const accountsRef = collection(this.db, 'accounts');
250     const q = query(
251         accountsRef,
252         where('broker', '==', broker),
253         where('server', '==', server),
254         where('accountID', '==', accountID),
255         where('userId', '!=', currentUserID) // Excluir la cuenta actual si estamos editando
256     );
257
258     const querySnapshot = await getDocs(q);
259     return !querySnapshot.empty;
260 } catch (error) {
261     console.error('Error checking account existence:', error);
262     return false;
263 }
264 }
265 }

```

## shared\services\alert.service.ts

---

```

1  import { Injectable } from '@angular/core';
2  import { BehaviorSubject } from 'rxjs';
3
4  /**
5   * Configuration interface for alert dialogs.
6   *
7   * @interface AlertConfig
8   */
9  export interface AlertConfig {
10     title: string;
11     message: string;
12     buttonText?: string;
13     type?: 'info' | 'warning' | 'error' | 'success';
14 }
15
16 /**
17  * Service for displaying alert dialogs throughout the application.
18  *
19  * This service provides a centralized way to show alert messages with different
20  * types (info, warning, error, success). It uses RxJS BehaviorSubject to manage
21  * alert state and provides convenience methods for common alert types.
22  *
23  * Features:
24  * - Show alerts with custom title, message, and type
25  * - Convenience methods for error, warning, success, and info alerts
26  * - Observable stream for alert state changes
27  * - Hide alerts programmatically
28  *
29  * Usage:
30  * Components can subscribe to `alert$` observable to display alerts, or use
31  * the convenience methods that automatically show alerts.
32  *
33  * Relations:
34  * - Used by components throughout the application for user notifications
35  * - AlertPopupComponent: Displays the actual alert UI
36  *
37  * @service
38  * @injectable
39  * @providedIn root
40  */
41 @Injectable({
42     providedIn: 'root'
43 })
44 export class AlertService {
45     private alertSubject = new BehaviorSubject<{ visible: boolean; config: AlertConfig | null }
46 >({ visible: false,
47     config: null
48 });
49
50     public alert$ = this.alertSubject.asObservable();
51
52     showAlert(config: AlertConfig): void {
53         this.alertSubject.next({
54             visible: true,
55             config: {
56                 title: config.title,
57                 message: config.message,
58                 buttonText: config.buttonText || 'OK',
59                 type: config.type || 'info'
60             }
61         });
62     }
63 }

```

```

63
64     hideAlert(): void {
65         this.alertSubject.next({
66             visible: false,
67             config: null
68         });
69     }
70
71     // Métodos de conveniencia
72     showError(message: string, title: string = 'Error'): void {
73         this.showAlert({
74             title,
75             message,
76             type: 'error'
77         });
78     }
79
80     showWarning(message: string, title: string = 'Warning'): void {
81         this.showAlert({
82             title,
83             message,
84             type: 'warning'
85         });
86     }
87
88     showSuccess(message: string, title: string = 'Success'): void {
89         this.showAlert({
90             title,
91             message,
92             type: 'success'
93         });
94     }
95
96     showInfo(message: string, title: string = 'Information'): void {
97         this.showAlert({
98             title,
99             message,
100             type: 'info'
101         });
102     }
103 }
104

```

## shared\services\auth.service.ts

---

```

1  import { isPlatformBrowser } from '@angular/common';
2  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
3  import {
4      createUserWithEmailAndPassword,
5      getAuth,
6      GoogleAuthProvider,
7      OAuthProvider,
8      onAuthStateChanged,
9      signInWithEmailAndPassword,
10     signInWithPopup,
11 } from 'firebase/auth';
12 import { BehaviorSubject, filter, Observable } from 'rxjs';
13 import { AppContextService } from '../context';
14 import { PlanService, Plan } from '../planService';
15 import { SubscriptionService, Subscription } from '../subscription-service';
16 import { UserStatus } from '../features/overview/models/overview';
17 import { UsersOperationsService } from '../users-operations.service';
18 import { AccountsOperationsService } from '../accounts-operations.service';
19 import { StrategyOperationsService } from '../strategy-operations.service';
20 import { TokensOperationsService, LinkToken } from '../tokens-operations.service';
21 import { User } from '../features/overview/models/overview';

```

```

22 import { AccountData, UserCredentials } from '../../features/auth/models/userModel';
23
24 /**
25  * Authentication service for Firebase Auth and user management.
26  *
27  * This service provides comprehensive authentication functionality including
28  * user registration, login (email/password, Google, Apple), logout, password
29  * reset, and user data management. It also manages user plan subscriptions
30  * and integrates with AppContextService for global state management.
31  *
32  * Features:
33  * - User registration and login (email/password, Google, Apple)
34  * - Logout functionality
35  * - Password reset
36  * - Authentication state observables
37  * - User data CRUD operations
38  * - Account management (trading accounts)
39  * - Strategy management
40  * - Token management (link tokens)
41  * - User plan subscription management
42  * - Global plans loading
43  * - Real-time subscription listener
44  *
45  * Plan Management:
46  * - Listens to user subscription changes
47  * - Updates AppContextService with plan data
48  * - Handles banned, cancelled, and active subscription states
49  * - Loads global plans on authentication
50  *
51  * Relations:
52  * - UsersOperationsService: User data operations
53  * - AccountsOperationsService: Trading account operations
54  * - StrategyOperationsService: Strategy operations
55  * - TokensOperationsService: Link token operations
56  * - SubscriptionService: Subscription management
57  * - PlanService: Plan information
58  * - AppContextService: Global state management
59  *
60  * @service
61  * @injectable
62  * @providedIn root
63  */
64 @Injectable({ providedIn: 'root' })
65 export class AuthService {
66   private isBrowser: boolean;
67   private authStateSubject = new BehaviorSubject<boolean | null>(null);
68   authStateChanged = this.authStateSubject.asObservable();
69
70   constructor(
71     @Inject(PLATFORM_ID) private platformId: Object,
72     private usersOperationsService: UsersOperationsService,
73     private accountsOperationsService: AccountsOperationsService,
74     private strategyOperationsService: StrategyOperationsService,
75     private tokensOperationsService: TokensOperationsService,
76     private appContext: AppContextService,
77     private planService: PlanService,
78     private subscriptionService: SubscriptionService
79   ) {
80     this.isBrowser = isPlatformBrowser(this.platformId);
81
82     if (this.isBrowser) {
83       onAuthStateChanged(getAuth(), async (user) => {
84         this.authStateSubject.next(user !== null);
85         if (user?.uid) {
86           await this.startUserPlanListener(user.uid);
87         } else {
88           this.stopUserPlanListener();
89           this.appContext.setUserPlan(null);
90         }
91       });

```

```

92     } else {
93         this.authStateSubject.next(false);
94     }
95 }
96
97 private subscriptionUnsubscribe: (() => void) | null = null;
98
99 private async startUserPlanListener(userId: string): Promise<void> {
100     this.stopUserPlanListener();
101
102     // Cargar planes globales si no están cargados
103     await this.loadGlobalPlansIfNeeded();
104
105     this.subscriptionUnsubscribe = this.subscriptionService.listenToUserLatestSubscription(
106         userId,
107         async (subscription) => {
108             await this.updateUserPlanFromSubscription(subscription);
109         }
110     );
111 }
112
113 private stopUserPlanListener(): void {
114     if (this.subscriptionUnsubscribe) {
115         this.subscriptionUnsubscribe();
116         this.subscriptionUnsubscribe = null;
117     }
118 }
119
120 private async loadGlobalPlansIfNeeded(): Promise<void> {
121     // Verificar si los planes globales ya están cargados
122     const currentPlans = this.appContext.globalPlans();
123     if (currentPlans.length > 0) {
124         return;
125     }
126
127     try {
128         const plans = await this.planService.getAllPlans();
129         this.appContext.setGlobalPlans(plans);
130     } catch (error) {
131         console.error('Error cargando planes globales:', error);
132     }
133 }
134
135 private async updateUserPlanFromSubscription(subscription: Subscription | null):
136 Promise<void> {
137     if (!subscription) {
138         this.appContext.setUserPlan(null);
139         return;
140     }
141
142     const status = subscription.status;
143
144     // Estado baneado: bloquear uso
145     if (status === UserStatus.BANNED) {
146         this.appContext.setUserPlan({
147             planId: subscription.planId,
148             planName: 'Banned',
149             maxAccounts: 0,
150             maxStrategies: 0,
151             features: [],
152             isActive: false,
153             expiresAt: subscription.periodEnd ? (subscription.periodEnd as any).toMillis?.
154             () ?? undefined : undefined, // Extensiones especiales
155             status: UserStatus.BANNED,
156             price: '0'
157         } as any);
158         return;
159     }
160
161     // Estado cancelado: plan Free

```



```

162         if (status === UserStatus.CANCELLED) {
163             this.appContext.setUserPlan({
164                 planId: 'free',
165                 planName: 'Free',
166                 maxAccounts: 1,
167                 maxStrategies: 1,
168                 features: [],
169                 isActive: true,
170                 status: UserStatus.CANCELLED,
171                 price: '0'
172             } as any);
173             return;
174         }
175
176         // Activo: cargar plan y mapear límites
177         const plan: Plan | undefined = await this.planService.getPlanById(subscription.planId);
178         if (!plan) {
179             // Si el plan no existe, tratar como sin plan
180             this.appContext.setUserPlan(null);
181             return;
182         }
183
184         this.appContext.setUserPlan({
185             planId: plan.id,
186             planName: plan.name,
187             maxAccounts: plan.tradingAccounts ?? 1,
188             maxStrategies: plan.strategies ?? 1,
189             features: [],
190             isActive: true,
191             status,
192             price: plan.price
193         } as any);
194     } catch (error) {
195         console.error('Error actualizando user plan desde subscription:', error);
196         this.appContext.setUserPlan(null);
197     }
198 }
199
200 // Firebase Auth primitives
201 getAuth() { return getAuth(); }
202 register(user: UserCredentials) { return createUserWithEmailAndPassword(getAuth(),
203     user.email, user.password); }
204 signInWithEmailAndPassword() { return signInWithEmailAndPassword(getAuth(), user.email,
205     user.password); }
206 signInWithApple() { const provider = new OAuthProvider('apple.com'); return
207     signInWithPopup(getAuth(), provider); }
208
209 async sendPasswordReset(email: string): Promise<void> {
210     const { sendPasswordResetEmail } = await import('firebase/auth');
211     return sendPasswordResetEmail(getAuth(), email);
212 }
213
214 // Observabilidad de sesión
215 isAuthenticated(): Observable<boolean> {
216     return this.authStateSubject.asObservable().pipe(filter((state): state is boolean =>
217         state !== null));
218 }
219
220 getCurrentUser(): any {
221     try {
222         if (!this.isBrowser) return null;
223         return getAuth().currentUser;
224     } catch (error) {
225         console.error('Error obteniendo usuario actual:', error);
226         return null;
227     }
228 }
229
230 async getBearerTokenFirebase(userId: string): Promise<string> {
231     const token = await getAuth().currentUser?.getIdToken();
232     if (!token) throw new Error('Token not found');
233     return token;

```

```

232     }
233
234     // Users collection operations
235     async getUserData(uid: string): Promise<User> {
236         this.appContext.setLoading('user', true);
237         this.appContext.setError('user', null);
238         try {
239             const userData = await this.usersOperationsService.getUserData(uid);
240
241             // Actualizar conteos de trading_accounts y strategies
242             await this.updateUserCounts(uid);
243
244             // Obtener nuevamente los datos actualizados después de actualizar los conteos
245             const updatedUserData = await this.usersOperationsService.getUserData(uid);
246
247             this.appContext.setCurrentUser(updatedUserData);
248             this.appContext.setLoading('user', false);
249             return updatedUserData;
250         } catch (error) {
251             this.appContext.setLoading('user', false);
252             this.appContext.setError('user', 'Error al obtener datos del usuario');
253             throw error;
254         }
255     }
256
257     async getUserById(userId: string): Promise<User | null> {
258         return this.usersOperationsService.getUserById(userId);
259     }
260
261     async updateUser(userId: string, userData: Partial<User>): Promise<void> {
262         return this.usersOperationsService.updateUser(userId, userData);
263     }
264
265     async createUser(user: User) { return this.usersOperationsService.createUser(user); }
266     async createLinkToken(token: LinkToken) { return
this.usersOperationsService.createLinkToken(token); }
267 this.usersOperationsService.createLinkToken(token); }
268 this.usersOperationsService.createLinkToken(token); }
269 this.usersOperationsService.createLinkToken(token); }
270
271     // Accounts operations
272     async createAccount(account: AccountData) {
273         this.appContext.setLoading('accounts', true);
274         this.appContext.setError('accounts', null);
275         try {
276             await this.accountsOperationsService.createAccount(account);
277             this.appContext.addAccount(account);
278
279             // Actualizar conteos del usuario
280             if (account.userId) {
281                 await this.updateUserCounts(account.userId);
282             }
283
284             this.appContext.setLoading('accounts', false);
285         } catch (error) {
286             this.appContext.setLoading('accounts', false);
287             this.appContext.setError('accounts', 'Error al crear cuenta');
288             throw error;
289         }
290     }
291
292     async getUserAccounts(userId: string): Promise<AccountData[] | null> {
293         this.appContext.setLoading('accounts', true);
294         this.appContext.setError('accounts', null);
295         try {
296             const accounts = await this.accountsOperationsService.getUserAccounts(userId);
297             this.appContext.setUserAccounts(accounts || []);
298             this.appContext.setLoading('accounts', false);
299             return accounts;
300         } catch (error) {
301             this.appContext.setLoading('accounts', false);
302             this.appContext.setError('accounts', 'Error al obtener cuentas del usuario');

```

```

302         throw error;
303     }
304 }
305
306 async getAllAccounts(): Promise<AccountData[] | null> { return
307 this.accountsOperationsService.getAllAccounts(string, currentUserId: string):
308 Promise<boolean> {
309 this.accountsOperationsService.checkEmailExists(email: string, currentUserId:
310 string): Promise<boolean> {
311 this.accountsOperationsService.createAccount(accountId: string, currentUserId:
312 string): Promise<boolean> {
313 const userId = await this.accountsOperationsService.deleteAccount(accountId);
314
315 // Actualizar conteos del usuario después de eliminar la cuenta
316 if (userId) {
317     await this.updateUserCounts(userId);
318 }
319
320 // Verificar si un email de usuario ya está registrado
321 async getUserByEmail(email: string): Promise<User | null> {
322     try {
323         return await this.usersOperationsService.getUserByEmail(email);
324     } catch (error) {
325         console.error('Error checking if email exists:', error);
326         return null;
327     }
328 }
329
330 // Método para obtener datos del usuario para validaciones (cuentas y estrategias)
331 async getUserDataForValidation(userId: string): Promise<{
332     accounts: AccountData[];
333     strategies: any[];
334 }> {
335     try {
336         const [accounts, strategies] = await Promise.all([
337             this.accountsOperationsService.getUserAccounts(userId),
338             this.strategyOperationsService.getUserStrategyViews(userId)
339         ]);
340
341         return {
342             accounts: accounts || [],
343             strategies: strategies || []
344         };
345     } catch (error) {
346         console.error('Error getting user data for validation:', error);
347         return {
348             accounts: [],
349             strategies: []
350         };
351     }
352 }
353
354 /**
355  * Actualizar los conteos de trading_accounts y strategies del usuario
356  */
357 async updateUserCounts(userId: string): Promise<void> {
358     try {
359         const [tradingAccountsCount, strategiesCount] = await Promise.all([
360             this.accountsOperationsService.getAllLengthUserAccounts(userId),
361             this.strategyOperationsService.getAllLengthConfigurationsOverview(userId)
362         ]);
363
364         await this.updateUser(userId, {
365             trading_accounts: tradingAccountsCount,
366             strategies: strategiesCount
367         });
368     } catch (error) {
369         console.error('Error updating user counts:', error);
370     }
371 }

```

```
372 }
373
374
375
```

## Ø=ÜÄ shared\services\countryService.ts

---

```
1
2 import { Injectable } from '@angular/core';
3 import { HttpClient } from '@angular/common/http';
4 import { Observable, map, shareReplay, tap } from 'rxjs';
5
6 /**
7  * Interface for country data from REST Countries API.
8  *
9  * @interface Country
10 */
11 export interface Country {
12   name: {
13     common: string;
14     official: string;
15   };
16   flags: {
17     png: string;
18     svg: string;
19     alt: string;
20   };
21   idd: {
22     root: string;
23     suffixes: string[];
24   };
25 }
26
27 /**
28  * Interface for formatted country option.
29  *
30  * @interface CountryOption
31 */
32 export interface CountryOption {
33   code: string;
34   name: string;
35   flag: string;
36   dialCode: string;
37 }
38
39 /**
40  * Service for fetching and managing country data.
41  *
42  * This service fetches country data from the REST Countries API and formats
43  * it for use in phone number inputs. It includes country codes, flags, and
44  * dial codes, with caching to avoid repeated API calls.
45  *
46  * Features:
47  * - Fetch all countries from REST Countries API
48  * - Format countries with codes, names, flags, and dial codes
49  * - Cache countries data in memory
50  * - Share replay for multiple subscribers
51  * - Filter countries with valid dial codes
52  * - Sort countries alphabetically
53  * - Map special country names to ISO codes
54  *
55  * API:
56  * - Source: https://restcountries.com/v3.1/all
57  * - Fields: idd (dial codes), flags, name
58  *
59  * Caching:
```

```

60 * - Caches countries after first fetch
61 * - Uses shareReplay for multiple subscribers
62 * - Prevents duplicate API calls
63 *
64 * Relations:
65 * - Used by PhoneInputComponent for country selection
66 * - Used by UserModalComponent for country detection
67 *
68 * @service
69 * @injectable
70 * @providedIn root
71 */
72 @Injectable({
73   providedIn: 'root'
74 })
75 export class CountryService {
76   private apiUrl = 'https://restcountries.com/v3.1/all?fields=idd,flags,name';
77   private cachedCountries: CountryOption[] | null = null;
78   private inFlight$: Observable<CountryOption[]>;
79
80   constructor(private http: HttpClient) {}
81
82   getCountries(): Observable<CountryOption[]> {
83     if (this.cachedCountries) {
84       return new Observable<CountryOption[]>((subscriber) => {
85         subscriber.next(this.cachedCountries as CountryOption[]);
86         subscriber.complete();
87       });
88     }
89
90     if (this.inFlight$) {
91       return this.inFlight$;
92     }
93
94     this.inFlight$ = this.http.get<Country[]>(this.apiUrl).pipe(
95       map(countries =>
96         countries
97           .filter(country => country.idd && country.idd.root && country.idd.suffixes)
98           .map(country => ({
99             code: this.extractCountryCode(country.name.common),
100             name: country.name.common,
101             flag: country.flags.svg,
102             dialCode: this.formatDialCode(country.idd.root, country.idd.suffixes[0])
103           })))
104       .sort((a, b) => a.name.localeCompare(b.name))
105     ),
106     tap(list => {
107       this.cachedCountries = list;
108     }),
109     shareReplay(1)
110   );
111
112   return this.inFlight$;
113 }
114
115 private extractCountryCode(countryName: string): string {
116   // Mapeo de nombres de países a códigos ISO para casos especiales
117   const countryCodeMap: { [key: string]: string } = {
118     'United States': 'US',
119     'United Kingdom': 'GB',
120     'South Korea': 'KR',
121     'North Korea': 'KP',
122     'United Arab Emirates': 'AE',
123     'Czech Republic': 'CZ',
124     'Dominican Republic': 'DO',
125     'Central African Republic': 'CF',
126     'Republic of the Congo': 'CG',
127     'Democratic Republic of the Congo': 'CD',
128     'South Africa': 'ZA',
129     'New Zealand': 'NZ',

```

130 'Papua New Guinea': 'PG',  
131 'Saudi Arabia': 'SA',  
132 'Costa Rica': 'CR',  
133 'El Salvador': 'SV',  
134 'Puerto Rico': 'PR',  
135 'Trinidad and Tobago': 'TT',  
136 'Saint Vincent and the Grenadines': 'VC',  
137 'Saint Kitts and Nevis': 'KN',  
138 'Antigua and Barbuda': 'AG',  
139 'Saint Lucia': 'LC',  
140 'Dominica': 'DM',  
141 'Grenada': 'GD',  
142 'Barbados': 'BB',  
143 'Bahamas': 'BS',  
144 'Bermuda': 'BM',  
145 'Cayman Islands': 'KY',  
146 'British Virgin Islands': 'VG',  
147 'US Virgin Islands': 'VI',  
148 'Turks and Caicos Islands': 'TC',  
149 'Anguilla': 'AI',  
150 'Montserrat': 'MS',  
151 'Sint Maarten': 'SX',  
152 'Aruba': 'AW',  
153 'Curaçao': 'CW',  
154 'Bonaire': 'BQ',  
155 'Saba': 'BQ',  
156 'Sint Eustatius': 'BQ',  
157 'Falkland Islands': 'FK',  
158 'South Georgia and the South Sandwich Islands': 'GS',  
159 'British Indian Ocean Territory': 'IO',  
160 'Pitcairn Islands': 'PN',  
161 'Saint Helena': 'SH',  
162 'Ascension Island': 'AC',  
163 'Tristan da Cunha': 'TA',  
164 'Norfolk Island': 'NF',  
165 'Christmas Island': 'CX',  
166 'Cocos Islands': 'CC',  
167 'Heard Island and McDonald Islands': 'HM',  
168 'Bouvet Island': 'BV',  
169 'Svalbard and Jan Mayen': 'SJ',  
170 'Åland Islands': 'AX',  
171 'Faroe Islands': 'FO',  
172 'Greenland': 'GL',  
173 'French Guiana': 'GF',  
174 'Guadeloupe': 'GP',  
175 'Martinique': 'MQ',  
176 'Mayotte': 'YT',  
177 'Réunion': 'RE',  
178 'Saint Barthélemy': 'BL',  
179 'Saint Martin': 'MF',  
180 'Saint Pierre and Miquelon': 'PM',  
181 'Wallis and Futuna': 'WF',  
182 'French Polynesia': 'PF',  
183 'New Caledonia': 'NC',  
184 'Cook Islands': 'CK',  
185 'Niue': 'NU',  
186 'Tokelau': 'TK',  
187 'American Samoa': 'AS',  
188 'Guam': 'GU',  
189 'Northern Mariana Islands': 'MP',  
190 'Marshall Islands': 'MH',  
191 'Micronesia': 'FM',  
192 'Palau': 'PW',  
193 'Solomon Islands': 'SB',  
194 'Vanuatu': 'VU',  
195 'Fiji': 'FJ',  
196 'Tonga': 'TO',  
197 'Kiribati': 'KI',  
198 'Nauru': 'NR',  
199 'Tuvalu': 'TV',

200 'Samoa': 'WS',  
201 'Timor-Leste': 'TL',  
202 'Brunei': 'BN',  
203 'Maldives': 'MV',  
204 'Sri Lanka': 'LK',  
205 'Bangladesh': 'BD',  
206 'Bhutan': 'BT',  
207 'Nepal': 'NP',  
208 'Myanmar': 'MM',  
209 'Thailand': 'TH',  
210 'Cambodia': 'KH',  
211 'Vietnam': 'VN',  
212 'Malaysia': 'MY',  
213 'Indonesia': 'ID',  
214 'Philippines': 'PH',  
215 'Taiwan': 'TW',  
216 'Hong Kong': 'HK',  
217 'Macau': 'MO',  
218 'Mongolia': 'MN',  
219 'Kazakhstan': 'KZ',  
220 'Uzbekistan': 'UZ',  
221 'Turkmenistan': 'TM',  
222 'Tajikistan': 'TJ',  
223 'Kyrgyzstan': 'KG',  
224 'Afghanistan': 'AF',  
225 'Pakistan': 'PK',  
226 'India': 'IN',  
227 'China': 'CN',  
228 'Japan': 'JP',  
229 'Russia': 'RU',  
230 'Belarus': 'BY',  
231 'Ukraine': 'UA',  
232 'Moldova': 'MD',  
233 'Romania': 'RO',  
234 'Bulgaria': 'BG',  
235 'Greece': 'GR',  
236 'Turkey': 'TR',  
237 'Cyprus': 'CY',  
238 'Lebanon': 'LB',  
239 'Syria': 'SY',  
240 'Iraq': 'IQ',  
241 'Iran': 'IR',  
242 'Israel': 'IL',  
243 'Palestine': 'PS',  
244 'Jordan': 'JO',  
245 'Kuwait': 'KW',  
246 'Qatar': 'QA',  
247 'Bahrain': 'BH',  
248 'Oman': 'OM',  
249 'Yemen': 'YE',  
250 'Egypt': 'EG',  
251 'Libya': 'LY',  
252 'Tunisia': 'TN',  
253 'Algeria': 'DZ',  
254 'Morocco': 'MA',  
255 'Western Sahara': 'EH',  
256 'Mauritania': 'MR',  
257 'Mali': 'ML',  
258 'Burkina Faso': 'BF',  
259 'Niger': 'NE',  
260 'Chad': 'TD',  
261 'Sudan': 'SD',  
262 'South Sudan': 'SS',  
263 'Ethiopia': 'ET',  
264 'Eritrea': 'ER',  
265 'Djibouti': 'DJ',  
266 'Somalia': 'SO',  
267 'Kenya': 'KE',  
268 'Uganda': 'UG',  
269 'Tanzania': 'TZ',

270 'Rwanda': 'RW',  
271 'Burundi': 'BI',  
272 'Cameroon': 'CM',  
273 'Nigeria': 'NG',  
274 'Benin': 'BJ',  
275 'Togo': 'TG',  
276 'Ghana': 'GH',  
277 'Ivory Coast': 'CI',  
278 'Liberia': 'LR',  
279 'Sierra Leone': 'SL',  
280 'Guinea': 'GN',  
281 'Guinea-Bissau': 'GW',  
282 'Senegal': 'SN',  
283 'Gambia': 'GM',  
284 'Cape Verde': 'CV',  
285 'São Tomé and Príncipe': 'ST',  
286 'Equatorial Guinea': 'GQ',  
287 'Gabon': 'GA',  
288 'Angola': 'AO',  
289 'Zambia': 'ZM',  
290 'Zimbabwe': 'ZW',  
291 'Botswana': 'BW',  
292 'Namibia': 'NA',  
293 'Lesotho': 'LS',  
294 'Eswatini': 'SZ',  
295 'Madagascar': 'MG',  
296 'Mauritius': 'MU',  
297 'Seychelles': 'SC',  
298 'Comoros': 'KM',  
299 'Malawi': 'MW',  
300 'Mozambique': 'MZ',  
301 'Iceland': 'IS',  
302 'Ireland': 'IE',  
303 'Norway': 'NO',  
304 'Sweden': 'SE',  
305 'Finland': 'FI',  
306 'Denmark': 'DK',  
307 'Estonia': 'EE',  
308 'Latvia': 'LV',  
309 'Lithuania': 'LT',  
310 'Poland': 'PL',  
311 'Germany': 'DE',  
312 'Netherlands': 'NL',  
313 'Belgium': 'BE',  
314 'Luxembourg': 'LU',  
315 'France': 'FR',  
316 'Monaco': 'MC',  
317 'Liechtenstein': 'LI',  
318 'Switzerland': 'CH',  
319 'Austria': 'AT',  
320 'Slovakia': 'SK',  
321 'Hungary': 'HU',  
322 'Slovenia': 'SI',  
323 'Croatia': 'HR',  
324 'Bosnia and Herzegovina': 'BA',  
325 'Serbia': 'RS',  
326 'Montenegro': 'ME',  
327 'North Macedonia': 'MK',  
328 'Albania': 'AL',  
329 'Kosovo': 'XK',  
330 'Italy': 'IT',  
331 'San Marino': 'SM',  
332 'Vatican City': 'VA',  
333 'Malta': 'MT',  
334 'Spain': 'ES',  
335 'Portugal': 'PT',  
336 'Andorra': 'AD',  
337 'Canada': 'CA',  
338 'Mexico': 'MX',  
339 'Guatemala': 'GT',



```

340         'Belize': 'BZ',
341         'Honduras': 'HN',
342         'Panama': 'PA',
343         'Cuba': 'CU',
344         'Jamaica': 'JM',
345         'Haiti': 'HT',
346         'Colombia': 'CO',
347         'Venezuela': 'VE',
348         'Guyana': 'GY',
349         'Suriname': 'SR',
350         'Brazil': 'BR',
351         'Ecuador': 'EC',
352         'Peru': 'PE',
353         'Bolivia': 'BO',
354         'Paraguay': 'PY',
355         'Uruguay': 'UY',
356         'Argentina': 'AR',
357         'Chile': 'CL'
358     };
359
360     return countryCodeMap[countryName] || countryName.substring(0, 2).toUpperCase();
361 }
362
363 private formatDialCode(root: string, suffix: string): string {
364     return root + suffix;
365 }
366 }

```

## shared\services\firebase-data.service.ts

---

```

1 import { getFirestore, collection, getDocs, doc, getDoc, setDoc, updateDoc, deleteDoc,
2   addDoc, orderBy, query, where, limit, startAfter, endAt, Timestamp, Firestore,
3   FirestoreError } from '@angular/fire/firestore';
4
5 /**
6  * Generic service for Firebase Firestore operations.
7  *
8  * This service provides generic CRUD operations for any Firestore collection,
9  * making it a utility service for common database operations. It abstracts
10  * away the Firebase API details and provides a simple interface.
11  *
12  * Features:
13  * - Get all documents from a collection
14  * - Get document by ID
15  * - Create document (with or without custom ID)
16  * - Update document
17  * - Delete document
18  * - Query documents by field
19  * - Check if document exists
20  *
21  * Usage:
22  * Useful for simple operations that don't require specialized services.
23  * For complex operations, use specific services (e.g., StrategyOperationsService).
24  *
25  * Relations:
26  * - Used as a utility service throughout the application
27  * - Provides generic database access patterns
28  *
29  * @service
30  * @injectable
31  * @providedIn root
32  */
33 @Injectable({
34     providedIn: 'root'
35 })
36 export class FirebaseDataService {

```

```

37 private isBrowser: boolean;
38 private db: ReturnType<typeof getFirestore> | null = null;
39
40 constructor(@Inject(PLATFORM_ID) private platformId: Object) {
41   this.isBrowser = isPlatformBrowser(this.platformId);
42   if (this.isBrowser) {
43     const { firebaseApp } = require('../../firebase/firebase.init.ts');
44     this.db = getFirestore(firebaseApp);
45   }
46 }
47
48 /**
49  * Generic method to get all documents from a collection
50  */
51 async getCollection(collectionName: string): Promise<any[]> {
52   if (!this.db) {
53     console.warn('Firestore not available in SSR');
54     return [];
55   }
56
57   try {
58     const snapshot = await getDocs(collection(this.db, collectionName));
59     const documents: any[] = [];
60
61     snapshot.forEach((doc) => {
62       const data = doc.data();
63       (data as any).id = doc.id;
64       documents.push(data);
65     });
66
67     return documents;
68   } catch (error) {
69     console.error(`Error getting collection ${collectionName}:`, error);
70     return [];
71   }
72 }
73
74 /**
75  * Generic method to get a document by ID
76  */
77 async getDocument(collectionName: string, docId: string): Promise<any | null> {
78   if (!this.db) {
79     console.warn('Firestore not available in SSR');
80     return null;
81   }
82
83   try {
84     const docRef = doc(this.db, collectionName, docId);
85     const docSnap = await getDoc(docRef);
86
87     if (docSnap.exists()) {
88       const data = docSnap.data();
89       (data as any).id = docSnap.id;
90       return data;
91     }
92     return null;
93   } catch (error) {
94     console.error(`Error getting document ${docId} from ${collectionName}:`, error);
95     return null;
96   }
97 }
98
99 /**
100  * Generic method to create a document
101  */
102 async createDocument(collectionName: string, data: any, docId?: string): Promise<string> {
103   if (!this.db) {
104     console.warn('Firestore not available in SSR');
105     throw new Error('Firestore not available');
106   }

```

```

107
108     try {
109         if (docId) {
110             await setDoc(doc(this.db, collectionName, docId), data);
111             return docId;
112         } else {
113             const docRef = await addDoc(collection(this.db, collectionName), data);
114             return docRef.id;
115         }
116     } catch (error) {
117         console.error(`Error creating document in ${collectionName}:`, error);
118         throw error;
119     }
120 }
121
122 /**
123  * Generic method to update a document
124  */
125 async updateDocument(collectionName: string, docId: string, data: any): Promise<void> {
126     if (!this.db) {
127         console.warn('Firestore not available in SSR');
128         return;
129     }
130
131     try {
132         await updateDoc(doc(this.db, collectionName, docId), data);
133     } catch (error) {
134         console.error(`Error updating document ${docId} in ${collectionName}:`, error);
135         throw error;
136     }
137 }
138
139 /**
140  * Generic method to delete a document
141  */
142 async deleteDocument(collectionName: string, docId: string): Promise<void> {
143     if (!this.db) {
144         console.warn('Firestore not available in SSR');
145         return;
146     }
147
148     try {
149         await deleteDoc(doc(this.db, collectionName, docId));
150     } catch (error) {
151         console.error(`Error deleting document ${docId} from ${collectionName}:`, error);
152         throw error;
153     }
154 }
155
156 /**
157  * Generic method to query documents
158  */
159 async queryDocuments(collectionName: string, field: string, operator: any, value: any):
160 Promise<any[]> {
161     if (!this.db) {
162         console.warn('Firestore not available in SSR');
163         return [];
164     }
165
166     try {
167         const q = query(
168             collection(this.db, collectionName),
169             where(field, operator, value)
170         );
171
172         const snapshot = await getDocs(q);
173         const documents: any[] = [];
174
175         snapshot.forEach((doc) => {
176             const data = doc.data();
177             (data as any).id = doc.id;

```

```

177         documents.push(data);
178     });
179
180     return documents;
181 } catch (error) {
182     console.error(`Error querying documents from ${collectionName}:`, error);
183     return [];
184 }
185 }
186
187 /**
188  * Check if a document exists
189  */
190 async documentExists(collectionName: string, docId: string): Promise<boolean> {
191     if (!this.db) {
192         console.warn('Firestore not available in SSR');
193         return false;
194     }
195
196     try {
197         const docRef = doc(this.db, collectionName, docId);
198         const docSnap = await getDoc(docRef);
199         return docSnap.exists();
200     } catch (error) {
201         console.error(`Error checking if document ${docId} exists in ${collectionName}:`,
202 error);return false;
203     }
204 }
205 }
206

```

## Ø=ÜÄ shared\services\global-strategy-updater.service.ts

---

```

1  import { Injectable, Inject, PLATFORM_ID } from '@angular/core';
2  import { isPlatformBrowser } from '@angular/common';
3  import { StrategyDaysUpdaterService } from './strategy-days-updater.service';
4
5  /**
6   * Service for global strategy updates.
7   *
8   * This service provides a high-level interface for updating strategy active
9   * days. It acts as a wrapper around StrategyDaysUpdaterService, providing
10  * convenient methods for updating all strategies or a single strategy.
11  *
12  * Features:
13  * - Update all strategies for a user
14  * - Update single strategy by ID
15  * - Error handling and logging
16  *
17  * Relations:
18  * - StrategyDaysUpdaterService: Performs the actual updates
19  * - Used by components that need to refresh strategy days active
20  *
21  * @service
22  * @injectable
23  * @providedIn root
24  */
25  @Injectable({
26      providedIn: 'root'
27  })
28  export class GlobalStrategyUpdaterService {
29      private isBrowser: boolean;
30      private updateInterval?: any;
31
32      constructor(
33          @Inject(PLATFORM_ID) private platformId: Object,

```

```

34     private strategyDaysUpdater: StrategyDaysUpdaterService
35   ) {
36     this.isBrowser = isPlatformBrowser(this.platformId);
37   }
38
39   /**
40    * Actualiza los días activos de la estrategia activa del usuario
41    * @param userId - ID del usuario
42    */
43   async updateAllStrategies(userId: string): Promise<void> {
44     if (!this.isBrowser) {
45       return;
46     }
47
48     try {
49       await this.strategyDaysUpdater.updateActiveStrategyDaysActive(userId);
50     } catch (error) {
51       console.error('GlobalStrategyUpdaterService: Error al actualizar estrategia activa:',
52 error);
53     }
54
55     /**
56      * Actualiza una estrategia específica
57      * @param strategyId - ID de la estrategia
58      * @param userId - ID del usuario
59      */
60     async updateSingleStrategy(strategyId: string, userId: string): Promise<void> {
61       try {
62         await this.strategyDaysUpdater.updateStrategyDaysActive(strategyId, userId);
63       } catch (error) {
64         console.error('GlobalStrategyUpdaterService: Error al actualizar estrategia:', error);
65       }
66     }
67   }
68

```

## shared\services\monthly-reports.service.ts

---

```

1  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2  import { getFirestore, collection, doc, getDoc, setDoc, updateDoc, deleteDoc, query, where,
3  orderBy } from '@angular/firestore';
4  import { MonthlyReport } from '../../features/report/models/report.model';
5  import { newDataId } from '../../features/report/utils/firebase-data-utils';
6
7  /**
8   * Service for managing monthly trading reports in Firebase.
9   *
10  * This service provides CRUD operations for monthly trading reports that
11  * aggregate trading statistics by month. Reports are used for historical
12  * analysis and performance tracking.
13  *
14  * Features:
15  * - Update monthly report
16  * - Get monthly report by ID
17  * - Get all monthly reports
18  * - Get monthly reports by user ID
19  * - Get monthly report by user, month, and year
20  * - Delete monthly report
21  *
22  * Report Structure:
23  * - Stored in: `monthly_reports/{reportId}`
24  * - Report ID format: Generated using `newDataId()` utility
25  * - Contains: Monthly aggregated trading statistics
26  *
27  * Relations:
28  * - Used by ReportService for saving monthly summaries

```

```

29  * - Used by ReportComponent for historical data
30  * - Uses `newDataId` utility for unique ID generation
31  *
32  * @service
33  * @injectable
34  * @providedIn root
35  */
36  @Injectable({
37    providedIn: 'root'
38  })
39  export class MonthlyReportsService {
40    private isBrowser: boolean;
41    private db: ReturnType<typeof getFirestore> | null = null;
42
43    constructor(@Inject(PLATFORM_ID) private platformId: Object) {
44      this.isBrowser = isPlatformBrowser(this.platformId);
45      if (this.isBrowser) {
46        const { firebaseApp } = require('../../firebase/firebase.init.ts');
47        this.db = getFirestore(firebaseApp);
48      }
49    }
50
51    /**
52     * Update monthly report
53     */
54    async updateMonthlyReport(monthlyReport: MonthlyReport): Promise<void> {
55      if (!this.db) {
56        console.warn('Firestore not available in SSR');
57        return;
58      }
59
60      try {
61        const id = newDataId(
62          monthlyReport.id,
63          monthlyReport.month,
64          monthlyReport.year
65        );
66
67        await setDoc(doc(this.db, 'monthly_reports', id), monthlyReport);
68      } catch (error) {
69        console.error('Error updating monthly report:', error);
70        throw error;
71      }
72    }
73
74    /**
75     * Get monthly report by ID
76     */
77    async getMonthlyReport(reportId: string): Promise<MonthlyReport | null> {
78      if (!this.db) {
79        console.warn('Firestore not available in SSR');
80        return null;
81      }
82
83      try {
84        const docRef = doc(this.db, 'monthly_reports', reportId);
85        const docSnap = await getDoc(docRef);
86
87        if (docSnap.exists()) {
88          return docSnap.data() as MonthlyReport;
89        }
90        return null;
91      } catch (error) {
92        console.error('Error getting monthly report:', error);
93        return null;
94      }
95    }
96
97    /**
98     * Get all monthly reports

```

```

99     */
100     async getAllMonthlyReports(): Promise<MonthlyReport[]> {
101         if (!this.db) {
102             console.warn('Firestore not available in SSR');
103             return [];
104         }
105
106         try {
107             const snapshot = await getDoc(doc(this.db, 'monthly_reports'));
108             const reports: MonthlyReport[] = [];
109
110             snapshot.data()?.['forEach']((doc: any) => {
111                 const data = doc.data() as MonthlyReport;
112                 (data as any).id = doc.id;
113                 reports.push(data);
114             });
115
116             return reports;
117         } catch (error) {
118             console.error('Error getting all monthly reports:', error);
119             return [];
120         }
121     }
122
123     /**
124      * Get monthly reports by user ID
125      */
126     async getMonthlyReportsByUserId(userId: string): Promise<MonthlyReport[]> {
127         if (!this.db) {
128             console.warn('Firestore not available in SSR');
129             return [];
130         }
131
132         try {
133             const q = query(
134                 collection(this.db, 'monthly_reports'),
135                 where('userId', '==', userId),
136                 orderBy('year', 'desc'),
137                 orderBy('month', 'desc')
138             );
139
140             const snapshot = await getDoc(doc(this.db, 'monthly_reports'));
141             const reports: MonthlyReport[] = [];
142
143             snapshot.data()?.['forEach']((doc: any) => {
144                 const data = doc.data() as MonthlyReport;
145                 (data as any).id = doc.id;
146                 reports.push(data);
147             });
148
149             return reports;
150         } catch (error) {
151             console.error('Error getting monthly reports by user ID:', error);
152             return [];
153         }
154     }
155
156     /**
157      * Get monthly report by user, month and year
158      */
159     async getMonthlyReportByUserMonthYear(userId: string, month: number, year: number):
160     Promise<MonthlyReport | null> {
161         console.warn('Firestore not available in SSR');
162         return null;
163     }
164
165     try {
166         const q = query(
167             collection(this.db, 'monthly_reports'),
168             where('userId', '==', userId),

```

```

169         where('month', '==', month),
170         where('year', '==', year),
171         limit(1)
172     );
173
174     const snapshot = await getDoc(doc(this.db, 'monthly_reports'));
175
176     if (snapshot.exists()) {
177         const doc = snapshot.data()?.[0];
178         const data = doc.data() as MonthlyReport;
179         (data as any).id = doc.id;
180         return data;
181     }
182
183     return null;
184 } catch (error) {
185     console.error('Error getting monthly report by user, month and year:', error);
186     return null;
187 }
188 }
189
190 /**
191  * Delete monthly report
192  */
193 async deleteMonthlyReport(reportId: string): Promise<void> {
194     if (!this.db) {
195         console.warn('Firestore not available in SSR');
196         return;
197     }
198
199     try {
200         await deleteDoc(doc(this.db, 'monthly_reports', reportId));
201     } catch (error) {
202         console.error('Error deleting monthly report:', error);
203         throw error;
204     }
205 }
206 }
207

```

## shared\services\overview-data.service.ts

---

```

1 import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2 import { getFirestore, collection, getDocs, query, where, orderBy, limit, startAfter, doc,
3 Snapshot } from '@angular/firestore';
4
5 /**
6  * Service for fetching overview dashboard data.
7  *
8  * This service provides methods to fetch data for the admin overview dashboard,
9  * including users, subscriptions, monthly reports, strategies, and accounts.
10 * It supports pagination for large datasets.
11 *
12 * Features:
13 * - Get overview subscription data
14 * - Get users data (with pagination)
15 * - Get user accounts (with pagination)
16 * - Get monthly reports data
17 * - Get configuration overview data
18 * - Get accounts data
19 *
20 * Pagination:
21 * - Supports cursor-based pagination
22 * - Orders users by subscription_date (descending)
23 * - Orders accounts by accountID (descending)
24 *

```



```

25  * Relations:
26  * - Used by OverviewService for data aggregation
27  * - Used by OverviewComponent for dashboard display
28  *
29  * @service
30  * @injectable
31  * @providedIn root
32  */
33  @Injectable({
34    providedIn: 'root'
35  })
36  export class OverviewDataService {
37    private isBrowser: boolean;
38    private db: ReturnType<typeof getFirestore> | null = null;
39
40    constructor(@Inject(PLATFORM_ID) private platformId: Object) {
41      this.isBrowser = isPlatformBrowser(this.platformId);
42      if (this.isBrowser) {
43        const { firebaseApp } = require('../../firebase/firebase.init.ts');
44        this.db = getFirestore(firebaseApp);
45      }
46    }
47
48    /**
49     * Get overview subscription data
50     */
51    async getOverviewSubscriptionData() {
52      if (!this.db) {
53        console.warn('Firestore not available in SSR');
54        return null;
55      }
56
57      try {
58        const snapshot = await getDocs(collection(this.db, 'overview-subscriptions'));
59        return snapshot;
60      } catch (error) {
61        console.error('Error getting overview subscription data:', error);
62        return null;
63      }
64    }
65
66    /**
67     * Get users data for overview
68     */
69    async getUsersData() {
70      if (!this.db) {
71        console.warn('Firestore not available in SSR');
72        return null;
73      }
74
75      try {
76        const snapshot = await getDocs(collection(this.db, 'users'));
77        return snapshot;
78      } catch (error) {
79        console.error('Error getting users data:', error);
80        return null;
81      }
82    }
83
84    /**
85     * Paginación de usuarios para la tabla de Overview
86     * Ordena por subscription_date desc (fallback: lastUpdated desc)
87     */
88    async getUsersPage(pageSize: number, startAfterDocId?: string) {
89      if (!this.db) {
90        console.warn('Firestore not available in SSR');
91        return { docs: [], lastDocId: undefined };
92      }
93
94      try {

```

```

195     const usersCol = collection(this.db, 'users');
196     let qRef: any = query(usersCol, orderBy('subscription_date', 'desc'), limit(pageSize));
197     if (startAfterDocId) {
198         const cursor = await getDoc(doc(this.db, 'users', startAfterDocId));
199         if (cursor.exists()) {
200             qRef = query(usersCol, orderBy('subscription_date', 'desc'), startAfter(cursor),
201 limit(pageSize));
202         }
203         const snapshot = await getDocs(qRef);
204         const lastDoc = snapshot.docs[snapshot.docs.length - 1];
205         return { docs: snapshot.docs, lastDocId: lastDoc?.id };
206     } catch (error) {
207         console.error('Error getting users page:', error);
208         return { docs: [], lastDocId: undefined };
209     }
210 }
211
212 /**
213  * Paginación de cuentas por usuario para la tabla (si se requiere desplegar cuentas)
214  */
215 async getUserAccountsPage(userId: string, pageSize: number, startAfterAccountId?: string) {
216     if (!this.db) {
217         console.warn('Firestore not available in SSR');
218         return { docs: [], lastDocId: undefined };
219     }
220
221     try {
222         const accountsCol = collection(this.db, 'accounts');
223         let qRef: any = query(
224             accountsCol,
225             where('userId', '==', userId),
226             orderBy('accountID', 'desc'),
227             limit(pageSize)
228         );
229         if (startAfterAccountId) {
230             const cursor = await getDoc(doc(this.db, 'accounts', startAfterAccountId));
231             if (cursor.exists()) {
232                 qRef = query(
233                     accountsCol,
234                     where('userId', '==', userId),
235                     orderBy('accountID', 'desc'),
236                     startAfter(cursor),
237                     limit(pageSize)
238                 );
239             }
240         }
241         const snapshot = await getDocs(qRef);
242         const lastDoc = snapshot.docs[snapshot.docs.length - 1];
243         return { docs: snapshot.docs, lastDocId: lastDoc?.id };
244     } catch (error) {
245         console.error('Error getting user accounts page:', error);
246         return { docs: [], lastDocId: undefined };
247     }
248 }
249
250 /**
251  * Get monthly reports data
252  */
253 async getMonthlyReportsData() {
254     if (!this.db) {
255         console.warn('Firestore not available in SSR');
256         return null;
257     }
258
259     try {
260         const snapshot = await getDocs(collection(this.db, 'monthly_reports'));
261         return snapshot;
262     } catch (error) {
263         console.error('Error getting monthly reports data:', error);
264         return null;

```

```

165     }
166   }
167
168   /**
169    * Get configuration overview data
170    */
171   async getConfigurationOverviewData() {
172     if (!this.db) {
173       console.warn('Firestore not available in SSR');
174       return null;
175     }
176
177     try {
178       const snapshot = await getDocs(collection(this.db, 'configuration-overview'));
179       return snapshot;
180     } catch (error) {
181       console.error('Error getting configuration overview data:', error);
182       return null;
183     }
184   }
185
186   /**
187    * Get accounts data
188    */
189   async getAccountsData() {
190     if (!this.db) {
191       console.warn('Firestore not available in SSR');
192       return null;
193     }
194
195     try {
196       const snapshot = await getDocs(collection(this.db, 'accounts'));
197       return snapshot;
198     } catch (error) {
199       console.error('Error getting accounts data:', error);
200       return null;
201     }
202   }
203 }
204

```

## Ø=ÜÄ shared\services\planService.ts

---

```

1 import { Injectable } from '@angular/core';
2 import { getFirestore, collection, query, getDocs, updateDoc, doc, Timestamp, setDoc,
3   orderBy, deleteDoc, deleteDocFromDatabase, deleteDocFromDatabaseSnapshot } from 'firebase/firestore';
4
5 /**
6  * Interface for subscription plan data.
7  */
8  @interface Plan
9  */
10 export interface Plan {
11   id: string;
12   name: string;
13   price: string;
14   strategies: number;
15   tradingAccounts: number;
16   createdAt?: any;
17   updatedAt?: any;
18   planPriceId?: string;
19 }
20
21 /**
22  * Service for managing subscription plans in Firebase.
23  */

```

```

24 * This service provides CRUD operations for subscription plans, including
25 * creating, reading, updating, and deleting plans. Plans define the features
26 * and limits available to users (e.g., number of strategies, trading accounts).
27 *
28 * Features:
29 * - Create new plans
30 * - Get all plans
31 * - Get plan by ID
32 * - Update existing plans
33 * - Delete plans
34 * - Query plans by name
35 *
36 * Plan Structure:
37 * - Stored in: `plan/{planId}`
38 * - Contains: name, price, strategies limit, trading accounts limit
39 * - Includes Stripe price ID for payment integration
40 *
41 * Relations:
42 * - Used by AuthService for loading global plans
43 * - Used by PlanSettingsComponent for displaying available plans
44 * - Used by PlanLimitationsGuard for checking plan limits
45 * - Used by AppContextService for caching global plans
46 *
47 * @service
48 * @injectable
49 * @providedIn root
50 */
51 @Injectable({
52   providedIn: 'root'
53 })
54 export class PlanService {
55
56   /**
57    * CREAR: Crear un nuevo plan
58    * @param plan Datos del plan a crear
59    * @returns Promise con el ID del documento creado
60    */
61   async createPlan(plan: Plan): Promise<string> {
62     try {
63       const planData = {
64         ...plan,
65         createdAt: new Date(),
66         updatedAt: new Date()
67       };
68
69       await setDoc(doc(db, 'plan', planData.id), planData);
70       return planData.id;
71     } catch (error) {
72       console.error('Error al crear plan:', error);
73       throw error;
74     }
75   }
76
77   /**
78    * LEER: Obtener todos los planes
79    * @returns Promise con array de todos los planes
80    */
81   async getAllPlans(): Promise<Plan[]> {
82     try {
83       const plansRef = collection(db, 'plan');
84
85       // Intentar sin orderBy primero para ver si ese es el problema
86       const querySnapshot = await getDocs(plansRef);
87
88       if (querySnapshot.empty) {
89         console.log('& p No se encontraron planes en la colección "plan"');
90         return [];
91       }
92
93       const plans = querySnapshot.docs.map((doc) => {

```

```

94         const data = doc.data();
95         return { id: doc.id, ...data };
96     });
97
98     return plans as Plan[];
99 } catch (error) {
100     console.error('L Error al obtener planes:', error);
101     return [];
102 }
103 }
104
105 /**
106  * LEER: Obtener un plan por ID
107  * @param id ID del plan
108  * @returns Promise con el plan específico
109  */
110 async getPlanById(id: string): Promise<Plan | undefined> {
111     return getDoc(doc(db, 'plan', id))
112         .then((doc) => ({ id: doc.id, ...doc.data() } as Plan))
113         .catch((error) => {
114             console.error('L Error al obtener el plan por ID:', error);
115             return undefined;
116         })
117 }
118
119 /**
120  * ACTUALIZAR: Actualizar un plan existente
121  * @param id ID del plan a actualizar
122  * @param plan Datos actualizados del plan
123  * @returns Promise que se resuelve cuando se completa la actualización
124  */
125 async updatePlan(id: string, plan: Partial<Omit<Plan, 'id' | 'createdAt'>>): Promise<void>
126 {
127     try {
128         const updateData = {
129             ...plan,
130             updatedAt: new Date()
131         };
132
133         await updateDoc(doc(db, 'plan', id), updateData);
134     } catch (error) {
135         console.error('L Error al actualizar plan:', error);
136         throw error;
137     }
138 }
139
140 /**
141  * ELIMINAR: Eliminar un plan
142  * @param id ID del plan a eliminar
143  * @returns Promise que se resuelve cuando se completa la eliminación
144  */
145 async deletePlan(id: string): Promise<void> {
146     try {
147         await deleteDoc(doc(db, 'plan', id));
148     } catch (error) {
149         console.error('L Error al eliminar plan:', error);
150         throw error;
151     }
152 }
153
154 /**
155  * LEER: Obtener un plan por nombre exacto
156  * @param name Nombre exacto del plan
157  * @returns Promise con el plan específico o undefined si no se encuentra
158  */
159 async getPlanByName(name: string): Promise<Plan | undefined> {
160     try {
161         const plansRef = collection(db, 'plan');
162         const querySnapshot = await getDocs(plansRef);
163
164         if (querySnapshot.empty) {

```

```

164         return undefined;
165     }
166
167     // Buscar plan con nombre exacto
168     const matchingDoc = querySnapshot.docs.find(doc => {
169         const data = doc.data();
170         return data['name'] === name;
171     });
172
173     if (matchingDoc) {
174         const data = matchingDoc.data();
175         return { id: matchingDoc.id, ...data } as Plan;
176     }
177
178     return undefined;
179 } catch (error) {
180     console.error('L Error al obtener plan por nombre:', error);
181     return undefined;
182 }
183 }
184
185 /**
186  * LEER: Buscar planes por nombre
187  * @param name Nombre o parte del nombre a buscar
188  * @returns Promise con planes que coinciden con la búsqueda
189  */
190 async searchPlansByName(name: string): Promise<Plan[]> {
191     try {
192
193         const plansRef = collection(db, 'plan');
194
195         // Primero obtener todos los planes para debug
196         const allPlansSnapshot = await getDocs(plansRef);
197
198         if (allPlansSnapshot.empty) {
199             console.log('& p La colección "plan" está vacía');
200             return [];
201         }
202
203         // Mostrar todos los planes disponibles
204         allPlansSnapshot.docs.forEach((doc, index) => {
205             const data = doc.data();
206         });
207
208         // Buscar planes que coincidan con el nombre (búsqueda simple)
209         const matchingPlans = allPlansSnapshot.docs.filter(doc => {
210             const data = doc.data();
211             const planName = data['name']?.toLowerCase() || '';
212             const searchName = name.toLowerCase();
213             return planName.includes(searchName) || searchName.includes(planName);
214         });
215
216         const plans = matchingPlans.map((doc) => {
217             const data = doc.data();
218             console.log('Plan encontrado:', { id: doc.id, name: data['name'] });
219             return { id: doc.id, ...data };
220         });
221
222         return plans as Plan[];
223     } catch (error) {
224         console.error('L Error en searchPlansByName:', error);
225         return [];
226     }
227 }
228 }
229
230 /**
231  * UTILIDAD: Verificar si un plan existe
232  * @param id ID del plan a verificar
233  * @returns Promise que se resuelve con true si existe, false si no

```

```

234     */
235     async planExists(id: string): Promise<boolean> {
236         try {
237             const document = await getDoc(doc(db, 'plan', id)) as DocumentSnapshot<Plan>;
238             return document.exists() || false;
239         } catch (error) {
240             console.error('L Error al verificar existencia del plan:', error);
241             return false;
242         }
243     }
244
245     /**
246     * UTILIDAD: Obtener el conteo total de planes
247     * @returns Promise con el número total de planes
248     */
249     async getPlansCount(): Promise<number> {
250         return getDocs(query(collection(db, 'plan')))
251             .then((snapshot) => snapshot.docs.length)
252             .catch((error) => {
253                 return 0;
254             })
255     }
256 }

```

## shared\services\plugin-history.service.ts

---

```

1  import { isPlatformBrowser } from '@angular/common';
2  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
3  import {
4      collection,
5      getDocs,
6      getFirestore,
7      where,
8      query,
9      onSnapshot,
10     doc,
11     getDoc,
12 } from 'firebase/firestore';
13 import { Observable, from } from 'rxjs';
14 import { firebaseApp } from '../firebase/firebase.init';
15 import { TimezoneService } from './timezone.service';
16
17 /**
18  * Interface for plugin history data.
19  *
20  * @interface PluginHistory
21  */
22 export interface PluginHistory {
23     id: string;
24     isActive: boolean;
25     updatedOn: string;
26     tokenNeeded: boolean;
27     dateActive: string[];
28     dateInactive: string[];
29 }
30
31 /**
32  * Service for managing plugin activation history.
33  *
34  * This service tracks when the trading plugin was activated and deactivated
35  * for users. It's used to determine if trades were executed while the plugin
36  * was active, which is essential for strategy adherence calculations.
37  *
38  * Features:
39  * - Get plugin usage history for a user
40  * - Determine if plugin is currently active based on dates

```

```

41  * - Real-time listener for plugin history changes
42  * - Timezone-aware date comparisons
43  *
44  * Plugin Status Logic:
45  * - If `dateActive` has more elements than `dateInactive`: plugin is active
46  * - If same count: compare last dates (last active > last inactive = active)
47  * - Uses UTC conversion for accurate date comparisons
48  *
49  * Data Structure:
50  * - Stored in: `plugin_history/plugin_{userId}`
51  * - Contains: activation/deactivation date arrays
52  *
53  * Relations:
54  * - Used by CalendarComponent for strategy adherence checks
55  * - Used by ReportComponent for determining if trades followed strategies
56  * - TimezoneService: For accurate date comparisons
57  *
58  * @service
59  * @injectable
60  * @providedIn root
61  */
62  @Injectable({
63    providedIn: 'root'
64  })
65  export class PluginHistoryService {
66
67    private isBrowser: boolean;
68    private db: ReturnType<typeof getFirestore> | null = null;
69
70    constructor(
71      @Inject(PLATFORM_ID) private platformId: Object,
72      private timezoneService: TimezoneService
73    ) {
74      this.isBrowser = isPlatformBrowser(this.platformId);
75      if (this.isBrowser) {
76        this.db = getFirestore(firebaseApp);
77      }
78    }
79
80    async getPluginUsageHistory(userId: string): Promise<PluginHistory[]> {
81      if (!this.db) {
82        console.warn('Firestore not available in SSR');
83        return [];
84      }
85
86      try {
87        // NUEVA LÓGICA: Buscar por document ID = plugin_{userId}
88        const pluginDocId = `plugin_${userId}`;
89        const docRef = doc(this.db, 'plugin_history', pluginDocId);
90        const docSnap = await getDoc(docRef);
91
92        if (!docSnap.exists()) {
93          return [];
94        }
95
96        const data = docSnap.data();
97        const pluginHistory = { id: docSnap.id, ...data };
98
99        return [pluginHistory] as PluginHistory[];
100
101      } catch (error) {
102        console.error('Error getting plugin usage history:', error);
103        return [];
104      }
105    }
106  }
107
108  /**
109   * MÉTODO NUEVO: Determinar si el plugin está activo basándose en las fechas
110   * LÓGICA MEJORADA CON ZONA HORARIA:

```



```

111 * - Si dateActive tiene más elementos que dateInactive: está activo
112 * - Si tienen la misma cantidad: comparar la última fecha de cada array
113 * - Si la última fecha de dateActive > última fecha de dateInactive: está activo
114 * - Si la última fecha de dateInactive > última fecha de dateActive: está inactivo
115 * - Usa conversión UTC para comparaciones precisas
116 */
117 isPluginActiveByDates(pluginHistory: PluginHistory): boolean {
118     if (!pluginHistory.dateActive || !pluginHistory.dateInactive) {
119         // Fallback al campo isActive si no hay fechas
120         return pluginHistory.isActive;
121     }
122
123     const activeDates = pluginHistory.dateActive;
124     const inactiveDates = pluginHistory.dateInactive;
125
126     // Si dateActive tiene más elementos que dateInactive, está activo
127     if (activeDates.length > inactiveDates.length) {
128         return true;
129     }
130
131     // Si tienen la misma cantidad, comparar las últimas fechas
132     if (activeDates.length === inactiveDates.length) {
133         if (activeDates.length === 0) {
134             return false; // No hay fechas, asumir inactivo
135         }
136
137         // MEJORA: Usar conversión UTC para comparaciones precisas
138         const lastActiveDate =
139             this.timezonesService.convertToUTC(activeDates[activeDates.length - 1]);
140         const lastInactiveDate =
141             this.timezonesService.convertToUTC(inactiveDates[inactiveDates.length - 1]);
142
143         // Si la última fecha de active es mayor que la de inactive, está activo
144         return lastActiveDate > lastInactiveDate;
145     }
146
147     // Si dateInactive tiene más elementos que dateActive, está inactivo
148     return false;
149 }
150 /**
151  * MÉTODO NUEVO: Listener en tiempo real para plugin history
152  * FLUJO DINÁMICO:
153  * - Retorna un Observable que emite cambios en tiempo real
154  * - Filtra por userId específico
155  * - El componente se suscribe y recibe actualizaciones automáticas
156  * - Maneja errores y limpieza de recursos
157  */
158 getPluginHistoryRealtime(userId: string): Observable<PluginHistory[]> {
159     if (!this.db) {
160         console.warn('Firestore not available in SSR');
161         return from([]);
162     }
163
164     return new Observable<PluginHistory[]>(subscriber => {
165         // NUEVA LÓGICA: Buscar por document ID = plugin_{userId}
166         const pluginDocId = `plugin_${userId}`;
167         const pluginHistoryRef = collection(this.db!, 'plugin_history');
168         const q = query(pluginHistoryRef, where('__name__', '==', pluginDocId));
169
170         const unsubscribe = onSnapshot(q,
171             (snapshot) => {
172                 const pluginHistory = snapshot.docs.map(doc => ({
173                     id: doc.id,
174                     ...doc.data()
175                 }) as PluginHistory));
176
177                 subscriber.next(pluginHistory);
178             },
179             (error) => {
180                 console.error('Error in plugin history listener:', error);

```

```

181             subscriber.error(error);
182         }
183     });
184
185     // Retornar función de limpieza
186     return () => {
187         unsubscribe();
188     };
189 });
190 }
191
192 }

```

## shared\services\reasons.service.ts

---

```

1  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2  import { isPlatformBrowser } from '@angular/common';
3  import {
4      getFirestore,
5      collection,
6      addDoc,
7      updateDoc,
8      getDocs,
9      doc,
10     query,
11     where,
12     orderBy,
13     limit,
14     serverTimestamp,
15 } from 'firebase/firestore';
16
17 /**
18  * Interface for ban reason record data.
19  *
20  * @interface BanReasonRecord
21  */
22 export interface BanReasonRecord {
23     id?: string;
24     reason: string;
25     dateBan: any; // serverTimestamp
26     dateUnban: any | null; // serverTimestamp or null
27 }
28
29 /**
30  * Service for managing user ban reasons.
31  *
32  * This service handles the creation and tracking of ban reasons for users.
33  * It stores ban records in a subcollection under each user's document,
34  * allowing administrators to track why users were banned and when they
35  * were unbanned.
36  *
37  * Features:
38  * - Create ban reason record
39  * - Update ban reason (e.g., add unban date)
40  * - Get latest open ban reason
41  *
42  * Data Structure:
43  * - Stored in: `users/{userId}/reasons/{reasonId}`
44  * - Tracks: ban reason, ban date, unban date
45  *
46  * Relations:
47  * - Used by UsersDetailsComponent for ban/unban operations
48  * - Used by AuthGuard for checking ban status
49  *
50  * @service
51  * @injectable

```

```

52 * @providedIn root
53 */
54 @Injectable({ providedIn: 'root' })
55 export class ReasonsService {
56   private isBrowser: boolean;
57   private db: ReturnType<typeof getFirestore> | null = null;
58
59   constructor(@Inject(PLATFORM_ID) private platformId: Object) {
60     this.isBrowser = isPlatformBrowser(this.platformId);
61     if (this.isBrowser) {
62       const { firebaseApp } = require('../..../firebase/firebase.init.ts');
63       this.db = getFirestore(firebaseApp);
64     }
65   }
66
67   private reasonsCollectionPath(userId: string) {
68     if (!this.db) throw new Error('Firestore not available in SSR');
69     return collection(this.db, 'users', userId, 'reasons');
70   }
71
72   async createReason(userId: string, reason: string): Promise<string> {
73     const colRef = this.reasonsCollectionPath(userId);
74     const docRef = await addDoc(colRef, {
75       reason,
76       dateBan: serverTimestamp(),
77       dateUnban: null,
78     } as BanReasonRecord);
79     return docRef.id;
80   }
81
82   async updateReason(userId: string, reasonId: string, data: Partial<BanReasonRecord>):
83   Promise<void> {
84     if (!this.db) throw new Error('Firestore not available in SSR');
85     await updateDoc(doc(this.db, 'users', userId, 'reasons', reasonId), data as any);
86   }
87
88   async getOpenLatestReason(userId: string): Promise<BanReasonRecord | null> {
89     // Para evitar índices compuestos, ordenamos por dateBan y tomamos el más reciente
90     const colRef = this.reasonsCollectionPath(userId);
91     const qRef = query(colRef, orderBy('dateBan', 'desc'), limit(1));
92     const snapshot = await getDocs(qRef);
93     if (snapshot.empty) return null;
94     const docSnap = snapshot.docs[0];
95     const data = docSnap.data() as BanReasonRecord;
96     return { ...data, id: docSnap.id };
97   }
98 }
99
100

```

## **shared\services\strategy-days-updater.service.ts**

```

1 import { Injectable, Inject, PLATFORM_ID } from '@angular/core';
2 import { isPlatformBrowser } from '@angular/common';
3 import { getFirestore, collection, query, getDocs, updateDoc, doc, Timestamp, where } from
4   'firebase/firestore';
5
6 /**
7  * Service for updating strategy active days in Firebase.
8  *
9  * This service calculates and updates the number of days a strategy has
10  * been active based on its creation date. It supports updating all user
11  * strategies or a specific strategy.
12  *
13  * Features:
14  * - Update active days for all user strategies

```

```

15  * - Update active days for active strategy only
16  * - Update active days for specific strategy
17  * - Calculate days active from creation date
18  *
19  * Days Active Calculation:
20  * - Calculates days from `created_at` timestamp to current date
21  * - Updates `days_active` field in configuration-overview
22  * - Automatically updates `updated_at` timestamp
23  *
24  * Relations:
25  * - Used by GlobalStrategyUpdaterService for batch updates
26  * - Used by StrategyCardComponent for displaying days active
27  * - Updates `configuration-overview` collection
28  *
29  * @service
30  * @injectable
31  * @providedIn root
32  */
33  @Injectable({
34    providedIn: 'root'
35  })
36  export class StrategyDaysUpdaterService {
37    private isBrowser: boolean;
38    private db: ReturnType<typeof getFirestore> | null = null;
39
40    constructor(@Inject(PLATFORM_ID) private platformId: Object) {
41      this.isBrowser = isPlatformBrowser(this.platformId);
42      if (this.isBrowser) {
43        this.db = getFirestore(firebaseApp);
44      }
45    }
46
47    /**
48     * Updates active days for all user strategies
49     * @param userId - User ID
50     */
51    async updateAllStrategiesDaysActive(userId: string): Promise<void> {
52      if (!this.isBrowser || !this.db) {
53        console.warn('StrategyDaysUpdaterService: Cannot execute on server');
54        return;
55      }
56
57      try {
58        // Get all user strategies
59        const strategiesRef = collection(this.db, 'configuration-overview');
60        const q = query(strategiesRef);
61        const querySnapshot = await getDocs(q);
62
63        const strategiesToUpdate: { id: string; daysActive: number }[] = [];
64
65        querySnapshot.forEach((docSnapshot) => {
66          const data = docSnapshot.data();
67
68          // Verify that the strategy belongs to the user
69          if (data['userId'] === userId && data['created_at']) {
70            const daysActive = this.calculateDaysActive(data['created_at']);
71
72            // Always update to keep synchronized
73            strategiesToUpdate.push({
74              id: docSnapshot.id,
75              daysActive: daysActive
76            });
77          }
78        });
79
80        // Update all strategies
81        const updatePromises = strategiesToUpdate.map(strategy =>
82          updateDoc(doc(this.db!, 'configuration-overview', strategy.id), {
83            days_active: strategy.daysActive,
84            updated_at: Timestamp.now()

```

```

85         })
86     );
87
88     if (updatePromises.length > 0) {
89         await Promise.all(updatePromises);
90     }
91
92     } catch (error) {
93         console.error('StrategyDaysUpdaterService: Error updating active days:', error);
94         throw error;
95     }
96 }
97
98 /**
99  * Updates active days for the user's active strategy
100  * @param userId - User ID
101  */
102 async updateActiveStrategyDaysActive(userId: string): Promise<void> {
103     if (!this.isBrowser || !this.db) {
104         console.warn('StrategyDaysUpdaterService: Cannot execute on server');
105         return;
106     }
107
108     try {
109         // Find the user's active strategy
110         const strategiesRef = collection(this.db, 'configuration-overview');
111         const q = query(
112             strategiesRef,
113             where('userId', '==', userId),
114             where('status', '==', true)
115         );
116         const querySnapshot = await getDocs(q);
117
118         if (querySnapshot.empty) {
119             console.log('StrategyDaysUpdaterService: No active strategy found for user:',
120 userId);return;
121         }
122
123         // There should only be one active strategy
124         const activeStrategyDoc = querySnapshot.docs[0];
125         const data = activeStrategyDoc.data();
126
127         if (!data['created_at']) {
128             console.warn('StrategyDaysUpdaterService: Active strategy without creation date');
129             return;
130         }
131
132         const daysActive = this.calculateDaysActive(data['created_at']);
133
134         // Only update if days have changed
135         if (data['days_active'] !== daysActive) {
136             await updateDoc(activeStrategyDoc.ref, {
137                 days_active: daysActive,
138                 updated_at: Timestamp.now()
139             });
140             console.log(`StrategyDaysUpdaterService: Updated active strategy
141 ${activeStrategyDoc.id} with ${daysActive} active days`);
142
143         } catch (error) {
144             console.error('StrategyDaysUpdaterService: Error updating active strategy days:',
145 error);throw error;
146         }
147     }
148
149     /**
150     * Updates active days for a specific strategy
151     * @param strategyId - Strategy ID
152     * @param userId - User ID (for security verification)
153     */
154     async updateStrategyDaysActive(strategyId: string, userId: string): Promise<void> {

```

```

155     if (!this.isBrowser || !this.db) {
156         console.warn('StrategyDaysUpdaterService: Cannot execute on server');
157         return;
158     }
159
160     try {
161         const strategyRef = doc(this.db, 'configuration-overview', strategyId);
162         const strategyDoc = await getDocs(query(collection(this.db, 'configuration-
163 overview')));
164         let strategyData: any = null;
165         strategyDoc.forEach(docSnapshot => {
166             if (docSnapshot.id === strategyId && docSnapshot.data()['userId'] === userId) {
167                 strategyData = docSnapshot.data();
168             }
169         });
170
171         if (!strategyData || !strategyData['created_at']) {
172             console.warn('StrategyDaysUpdaterService: Strategy not found or without creation
173 date'); return;
174         }
175
176         const daysActive = this.calculateDaysActive(strategyData['created_at']);
177
178         // Only update if days have changed
179         if (strategyData['days_active'] !== daysActive) {
180             await updateDoc(strategyRef, {
181                 days_active: daysActive,
182                 updated_at: Timestamp.now()
183             });
184         }
185
186     } catch (error) {
187         console.error('StrategyDaysUpdaterService: Error updating strategy active days:',
188 error); throw error;
189     }
190 }
191
192 /**
193  * Calculates active days since creation date
194  * @param createdAt - Firebase timestamp or creation date
195  * @returns Number of active days
196  */
197 private calculateDaysActive(createdAt: any): number {
198     let createdAtDate: Date;
199
200     // Handle different Firebase timestamp types
201     if (createdAt && typeof createdAt.toDate === 'function') {
202         // It's a Firebase Timestamp
203         createdAtDate = createdAt.toDate();
204     } else if (createdAt && createdAt.seconds) {
205         // It's an object with seconds
206         createdAtDate = new Date(createdAt.seconds * 1000);
207     } else if (createdAt instanceof Date) {
208         // Already a date
209         createdAtDate = createdAt;
210     } else if (typeof createdAt === 'string') {
211         // It's a date string
212         createdAtDate = new Date(createdAt);
213     } else {
214         console.warn('StrategyDaysUpdaterService: Unrecognized date format:', createdAt);
215         return 0;
216     }
217
218     // Get current date and creation date in YYYY-MM-DD format (without hours)
219     const now = new Date();
220     const today = new Date(now.getFullYear(), now.getMonth(), now.getDate());
221     const createdDay = new Date(createdAtDate.getFullYear(), createdAtDate.getMonth(),
222 createdAtDate.getDate());
223
224     // Calculate difference in complete days
225     const diffTime = today.getTime() - createdDay.getTime();

```

```

225     const diffDays = Math.floor(diffTime / (1000 * 60 * 60 * 24));
226
227     // If it's the same day, return 0
228     // If complete days have passed, return the difference
229     return Math.max(0, diffDays);
230 }
231
232 /**
233  * Gets active days of a strategy without updating in Firebase
234  * @param createdAt - Firebase timestamp or creation date
235  * @returns Number of active days
236  */
237 getDaysActive(createdAt: any): number {
238     return this.calculateDaysActive(createdAt);
239 }
240 }
241

```

## shared\services\strategy-operations.service.ts

---

```

1  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2  import { getFirestore, doc, setDoc, getDoc, collection, query, where, getDocs, deleteDoc,
3  updateDoc, orderBy, limit, addDoc } from '@angular/firestore';
4  import { Timestamp } from 'firebase/firestore';
5  import { MaxDailyTradesConfig, StrategyState, ConfigurationOverview } from '../../features/
6  strategy/models/strategy.model';
7  /**
8   * Service for strategy operations in Firebase.
9   *
10  * This service provides comprehensive CRUD operations for trading strategies,
11  * managing both strategy metadata (configuration-overview) and strategy rules
12  * (configurations). It handles the complete strategy lifecycle.
13  *
14  * Features:
15  * - Configuration Overview Operations:
16  *   - Create, read, update, delete strategy metadata
17  *   - Get all strategies for a user
18  *   - Get active strategies
19  *   - Soft delete strategies (mark as deleted)
20  * - Configuration Operations:
21  *   - Create, read, update strategy rules
22  *   - Get configuration by user ID
23  *   - Update individual rule configurations
24  * - Strategy Management:
25  *   - Activate/deactivate strategies
26  *   - Copy strategies
27  *   - Generate unique strategy IDs
28  *
29  * Data Structure:
30  * - `configuration-overview`: Strategy metadata (name, status, dates, etc.)
31  * - `configurations`: Strategy rules (maxDailyTrades, riskReward, etc.)
32  * - Strategies are linked by `configurationId` field
33  *
34  * Relations:
35  * - Used by StrategyService for strategy operations
36  * - Used by StrategyComponent for strategy management
37  * - Used by EditStrategyComponent for rule updates
38  *
39  * @service
40  * @injectable
41  * @providedIn root
42  */
43  @Injectable({
44     providedIn: 'root'
45  })
46  export class StrategyOperationsService {

```

```

47 private isBrowser: boolean;
48 private db: ReturnType<typeof getFirestore> | null = null;
49
50 constructor(@Inject(PLATFORM_ID) private platformId: Object) {
51   this.isBrowser = isPlatformBrowser(this.platformId);
52   if (this.isBrowser) {
53     const { firebaseApp } = require('../../firebase/firebase.init.ts');
54     this.db = getFirestore(firebaseApp);
55   }
56 }
57
58 // ===== CONFIGURATION-OVERVIEW (colección de metadatos) =====
59
60 /**
61  * Crear configuration-overview (solo metadatos)
62  */
63 async createConfigurationOverview(userId: string, name: string): Promise<string> {
64   if (!this.db) {
65     console.warn('Firestore not available in SSR');
66     return '';
67   }
68
69   const overviewId = this.generateOverviewId();
70   const now = Timestamp.now();
71
72   const configurationOverview: ConfigurationOverview = {
73     userId: userId,
74     name: name,
75     status: false, // Inicialmente inactiva
76     created_at: now,
77     updated_at: now,
78     days_active: 0,
79     configurationId: '' // Se establecerá después de crear la configuración
80   };
81
82   await setDoc(doc(this.db, 'configuration-overview', overviewId), configurationOverview);
83   return overviewId;
84 }
85
86 /**
87  * Obtener configuration-overview por ID (solo metadatos)
88  */
89 async getConfigurationOverview(overviewId: string): Promise<ConfigurationOverview | null> {
90   if (!this.db) {
91     console.warn('Firestore not available in SSR');
92     return null;
93   }
94
95   try {
96     const docRef = doc(this.db, 'configuration-overview', overviewId);
97     const docSnap = await getDoc(docRef);
98
99     if (docSnap.exists()) {
100       const data = docSnap.data() as ConfigurationOverview;
101       (data as any).id = docSnap.id; // Agregar ID del documento
102       return data;
103     }
104     return null;
105   } catch (error) {
106     console.error('Error getting configuration overview:', error);
107     return null;
108   }
109 }
110
111 /**
112  * Actualizar configuration-overview
113  */
114 async updateConfigurationOverview(overviewId: string, updates:
Partial<ConfigurationOverview>): Promise<void> {
115   if (!this.db) {
116     console.warn('Firestore not available in SSR');

```



```

117         return;
118     }
119
120     const docRef = doc(this.db, 'configuration-overview', overviewId);
121     const updateData = {
122         ...updates,
123         updated_at: Timestamp.now()
124     };
125
126     await updateDoc(docRef, updateData);
127 }
128
129 /**
130  * Eliminar configuration-overview
131  */
132 async deleteConfigurationOverview(overviewId: string): Promise<void> {
133     if (!this.db) {
134         console.warn('Firestore not available in SSR');
135         return;
136     }
137
138     const docRef = doc(this.db, 'configuration-overview', overviewId);
139     await deleteDoc(docRef);
140 }
141
142 // ===== CONFIGURATIONS (colección de reglas) =====
143
144 /**
145  * Crear configuración (solo reglas + IDs)
146  */
147 async createConfiguration(userId: string, configurationOverviewId: string, configuration:
148 StrategyState): Promise<void> {
149     console.warn('Firestore not available in SSR');
150     return;
151 }
152
153 const configData = {
154     ...configuration,
155     configurationOverviewId: configurationOverviewId,
156     userId: userId
157 };
158
159 await setDoc(doc(this.db, 'configurations', userId), configData);
160 }
161
162 /**
163  * Obtener configuración por userId
164  */
165 async getConfiguration(userId: string): Promise<StrategyState | null> {
166     if (!this.db) {
167         console.warn('Firestore not available in SSR');
168         return null;
169     }
170
171     try {
172         const docRef = doc(this.db, 'configurations', userId);
173         const docSnap = await getDoc(docRef);
174
175         if (docSnap.exists()) {
176             return docSnap.data() as StrategyState;
177         }
178         return null;
179     } catch (error) {
180         console.error('Error getting configuration:', error);
181         return null;
182     }
183 }
184
185 /**
186  * Actualizar configuración

```

```

187     */
188     async updateConfiguration(userId: string, configuration: StrategyState): Promise<void> {
189         if (!this.db) {
190             console.warn('Firestore not available in SSR');
191             return;
192         }
193
194         await updateDoc(doc(this.db, 'configurations', userId), configuration as any);
195     }
196
197     /**
198     * Crear solo configuration (sin userId ni configurationOverviewId)
199     */
200     async createConfigurationOnly(configuration: StrategyState): Promise<string> {
201         if (!this.db) {
202             console.warn('Firestore not available in SSR');
203             throw new Error('Firestore not available');
204         }
205
206         try {
207             const docRef = await addDoc(collection(this.db, 'configurations'), configuration as
208 any); return docRef.id;
209         } catch (error) {
210             console.error('Error creating configuration:', error);
211             throw error;
212         }
213     }
214
215     /**
216     * Crear configuration-overview con configurationId
217     */
218     async createConfigurationOverviewWithConfigId(userId: string, name: string,
219 configurationId: string, shouldBeActive: boolean): Promise<string> {
220         if (!this.db) {
221             console.warn('Firestore not available in SSR');
222             throw new Error('Firestore not available');
223         }
224
225         try {
226             const now = new Date();
227             const overviewData: ConfigurationOverview = {
228                 userId,
229                 name,
230                 status: shouldBeActive, // Activa solo si no hay otra activa
231                 created_at: now,
232                 updated_at: now,
233                 days_active: 0,
234                 configurationId,
235                 dateActive: [now.toISOString()],
236             };
237
238             const docRef = await addDoc(collection(this.db, 'configuration-overview'),
239 overviewData as any);
240         } catch (error) {
241             console.error('Error creating configuration overview:', error);
242             throw error;
243         }
244
245         /**
246         * Actualizar configuration por ID
247         */
248         async updateConfigurationById(configurationId: string, configuration: StrategyState):
249 Promise<void> {
250             if (!this.db) {
251                 console.warn('Firestore not available in SSR');
252                 throw new Error('Firestore not available');
253             }
254
255             try {
256                 const docRef = doc(this.db, 'configurations', configurationId);
257                 await updateDoc(docRef, configuration as any);

```

```

257     } catch (error) {
258         console.error('Error updating configuration by ID:', error);
259         throw error;
260     }
261 }
262
263 /**
264  * Obtener configuración por ID
265  */
266 async getConfigurationById(configurationId: string): Promise<StrategyState | null> {
267     if (!this.db) {
268         console.warn('Firestore not available in SSR');
269         return null;
270     }
271
272     try {
273         const docRef = doc(this.db, 'configurations', configurationId);
274         const docSnap = await getDoc(docRef);
275         if (docSnap.exists()) {
276             return docSnap.data() as StrategyState;
277         }
278         return null;
279     } catch (error) {
280         console.error('Error getting configuration by ID:', error);
281         return null;
282     }
283 }
284
285 /**
286  * Obtener configuración por configurationOverviewId (método legacy para compatibilidad)
287  */
288 async getConfigurationByOverviewId(overviewId: string): Promise<StrategyState | null> {
289     if (!this.db) {
290         console.warn('Firestore not available in SSR');
291         return null;
292     }
293
294     try {
295         // Buscar en configurations donde configurationOverviewId coincida
296         const q = query(
297             collection(this.db, 'configurations'),
298             where('configurationOverviewId', '==', overviewId),
299             limit(1)
300         );
301
302         const querySnapshot = await getDocs(q);
303         if (!querySnapshot.empty) {
304             const doc = querySnapshot.docs[0];
305             return doc.data() as StrategyState;
306         }
307
308         return null;
309     } catch (error) {
310         console.error('Error getting configuration by overview ID:', error);
311         return null;
312     }
313 }
314
315 /**
316  * Obtener todas las estrategias de un usuario
317  */
318 async getUserStrategyViews(userId: string): Promise<ConfigurationOverview[]> {
319     if (!this.db) {
320         console.warn('Firestore not available in SSR');
321         return [];
322     }
323
324     try {
325         // 1. Primero obtener todos los configuration-overview del usuario
326         const overviewQuery = query(

```

```

327     collection(this.db, 'configuration-overview'),
328     where('userId', '==', userId)
329 );
330
331 const overviewSnapshot = await getDocs(overviewQuery);
332
333 const strategies: ConfigurationOverview[] = [];
334
335 overviewSnapshot.forEach((doc) => {
336     const data = doc.data() as ConfigurationOverview;
337     (data as any).id = doc.id; // Agregar ID del documento
338     strategies.push(data);
339 });
340
341 // Filtrar estrategias que no estén marcadas como deleted
342 // Mostrar solo las que:
343 // 1. No tienen el campo 'deleted' (estrategias antiguas)
344 // 2. Tienen 'deleted: false' (explícitamente no eliminadas)
345 const activeStrategies = strategies.filter(strategy =>
346     strategy.deleted === undefined || strategy.deleted === false
347 );
348
349 // Ordenar manualmente por updated_at descendente
350 activeStrategies.sort((a, b) => {
351     const dateA = a.updated_at.toDate();
352     const dateB = b.updated_at.toDate();
353     return dateB.getTime() - dateA.getTime();
354 });
355
356 return activeStrategies;
357 } catch (error) {
358     console.error('Error getting user strategies:', error);
359     return [];
360 }
361 }
362
363 /**
364  * Obtener configuración activa (método legacy para compatibilidad)
365  */
366 async getActiveConfiguration(userId: string): Promise<ConfigurationOverview | null> {
367     if (!this.db) {
368         console.warn('Firestore not available in SSR');
369         return null;
370     }
371
372     try {
373         // Buscar estrategia activa en configuration-overview
374         const q = query(
375             collection(this.db, 'configuration-overview'),
376             where('userId', '==', userId),
377             where('status', '==', true),
378             limit(1)
379         );
380
381         const querySnapshot = await getDocs(q);
382         if (!querySnapshot.empty) {
383             const doc = querySnapshot.docs[0];
384             const data = doc.data() as ConfigurationOverview;
385             (data as any).id = doc.id;
386             return data;
387         }
388
389         return null;
390     } catch (error) {
391         console.error('Error getting active configuration:', error);
392         return null;
393     }
394 }
395
396 /**

```

```

397     * Activar una estrategia
398     */
399     async activateStrategyView(userId: string, strategyId: string): Promise<void> {
400         if (!this.db) {
401             console.warn('Firestore not available in SSR');
402             return;
403         }
404
405         try {
406             // 1. Desactivar todas las estrategias del usuario
407             const q = query(
408                 collection(this.db, 'configuration-overview'),
409                 where('userId', '==', userId),
410                 where('status', '==', true)
411             );
412
413             const querySnapshot = await getDocs(q);
414             const batch = [];
415
416             querySnapshot.forEach((doc) => {
417                 batch.push(updateDoc(doc.ref, { status: false }));
418             });
419
420             // 2. Activar la estrategia seleccionada
421             batch.push(updateDoc(doc(this.db, 'configuration-overview', strategyId), {
422                 status: true,
423                 updated_at: Timestamp.now()
424             }));
425
426             // Ejecutar todas las actualizaciones
427             await Promise.all(batch);
428         } catch (error) {
429             console.error('Error activating strategy:', error);
430             throw error;
431         }
432     }
433
434     /**
435     * Actualizar fechas de activación/desactivación de estrategias
436     */
437     async updateStrategyDates(userId: string, strategyId: string, dateActive?: Date,
438     dateInactive?: Date): Promise<void> {
439         if (!this.db) {
440             console.warn('Firestore not available in SSR');
441             return;
442         }
443
444         try {
445             const strategyRef = doc(this.db, 'configuration-overview', strategyId);
446             const strategyDoc = await getDoc(strategyRef);
447
448             if (!strategyDoc.exists()) {
449                 throw new Error('Strategy not found');
450             }
451
452             const currentData = strategyDoc.data();
453             const updateData: any = {};
454
455             // Solo actualizar si se proporciona un valor válido
456             if (dateActive !== undefined && dateActive !== null) {
457                 const currentDateActive = currentData['dateActive'] || [];
458                 const newDateActive = [...currentDateActive, dateActive.toISOString()];
459                 updateData.dateActive = newDateActive;
460
461                 // Si se está activando, cambiar status a true
462                 updateData.status = true;
463             }
464
465             // Solo actualizar si se proporciona un valor válido
466             if (dateInactive !== undefined && dateInactive !== null) {
467                 const currentDateInactive = currentData['dateInactive'] || [];

```

```

467         const newDateInactive = [...currentDateInactive, dateInactive.toISOString()];
468         updateData.dateInactive = newDateInactive;
469
470         // Si se está desactivando, cambiar status a false
471         updateData.status = false;
472     }
473
474     // Actualizar timestamp solo si hay cambios
475     if (Object.keys(updateData).length > 0) {
476         updateData.updated_at = Timestamp.now();
477         await updateDoc(strategyRef, updateData);
478     }
479     } catch (error) {
480         console.error('Error updating strategy dates:', error);
481         throw error;
482     }
483 }
484
485 /**
486  * Eliminar una estrategia
487  */
488 async deleteStrategyView(strategyId: string): Promise<void> {
489     if (!this.db) {
490         console.warn('Firestore not available in SSR');
491         return;
492     }
493
494     try {
495         // 1. Obtener el configurationId del overview
496         const strategy = await this.getConfigurationOverview(strategyId);
497         if (!strategy) {
498             throw new Error('Strategy not found');
499         }
500
501         // 2. Eliminar configuration-overview
502         await this.deleteConfigurationOverview(strategyId);
503
504         // 3. Eliminar configuration usando configurationId
505         if (strategy.configurationId) {
506             try {
507                 const configDocRef = doc(this.db, 'configurations', strategy.configurationId);
508                 await deleteDoc(configDocRef);
509             } catch (error) {
510                 console.warn('Configuration not found for deletion:', error);
511             }
512         }
513     } catch (error) {
514         console.error('Error deleting strategy:', error);
515         throw error;
516     }
517 }
518
519 /**
520  * Marcar una estrategia como deleted (soft delete)
521  */
522 async markStrategyAsDeleted(strategyId: string): Promise<void> {
523     if (!this.db) {
524         console.warn('Firestore not available in SSR');
525         return;
526     }
527
528     try {
529         // 1. Obtener el configurationId del overview
530         const strategy = await this.getConfigurationOverview(strategyId);
531         if (!strategy) {
532             throw new Error('Strategy not found');
533         }
534
535         const currentTimestamp = new Date();
536

```

```

537 // 2. Marcar configuration-overview como deleted y agregar dateInactive
538 const overviewDocRef = doc(this.db, 'configuration-overview', strategyId);
539 const overviewDoc = await getDoc(overviewDocRef);
540
541 if (overviewDoc.exists()) {
542   const currentData = overviewDoc.data();
543   const updateData: any = {
544     deleted: true,
545     deleted_at: Timestamp.now(),
546     updated_at: Timestamp.now(),
547     status: false // Marcar como inactiva
548   };
549
550   // Agregar dateInactive si la estrategia estaba activa
551   if (currentData['status'] === true) {
552     const currentDateInactive = currentData['dateInactive'] || [];
553     const newDateInactive = [...currentDateInactive,
554 Timestamp.now().toDate(), Timestamp.now().toDate()];
555     updateData['dateInactive'] = newDateInactive;
556   }
557   await updateDoc(overviewDocRef, updateData);
558 }
559
560 // 3. Marcar configuration como deleted usando configurationId
561 if (strategy.configurationId) {
562   try {
563     const configDocRef = doc(this.db, 'configurations', strategy.configurationId);
564     await updateDoc(configDocRef, {
565       deleted: true,
566       deleted_at: Timestamp.now(),
567       updated_at: Timestamp.now()
568     });
569   } catch (error) {
570     console.warn('Configuration not found for soft deletion:', error);
571   }
572 }
573 } catch (error) {
574   console.error('Error marking strategy as deleted:', error);
575   throw error;
576 }
577 }
578
579 /**
580  * Obtener el número total de estrategias de un usuario (solo no eliminadas)
581  */
582 async getAllLengthConfigurationsOverview(userId: string): Promise<number> {
583   if (!this.db) {
584     console.warn('Firestore not available in SSR');
585     return 0;
586   }
587
588   try {
589     // Obtener todas las estrategias del usuario
590     const overviewQuery = query(
591       collection(this.db, 'configuration-overview'),
592       where('userId', '==', userId)
593     );
594
595     const overviewSnapshot = await getDocs(overviewQuery);
596
597     let count = 0;
598     overviewSnapshot.forEach((doc) => {
599       const data = doc.data() as ConfigurationOverview;
600       // Solo contar las que no estén marcadas como deleted (deleted !== true)
601       // Las que tienen deleted === false o no tienen el campo deleted se cuentan
602       if (data.deleted === undefined || data.deleted === false) {
603         count++;
604       }
605     });
606

```

```

607         return count;
608     } catch (error) {
609         console.error('Error getting strategies count:', error);
610         return 0;
611     }
612 }
613
614 /**
615  * Generar ID único para configuration-overview
616  */
617 private generateOverviewId(): string {
618     return 'overview_' + Date.now() + '_' + Math.random().toString(36).substr(2, 9);
619 }
620 }
621

```

## shared\services\subscription-service.ts

---

```

1
2 import { Injectable } from '@angular/core';
3 import {
4     collection,
5     doc,
6     getDocs,
7     getDoc,
8     addDoc,
9     updateDoc,
10    deleteDoc,
11    query,
12    orderBy,
13    Timestamp,
14    onSnapshot,
15    limit
16 } from 'firebase/firestore';
17 import { db } from '../../firebase/firebase.init';
18 import { UserStatus } from '../../features/overview/models/overview';
19
20 /**
21  * Interface for user subscription data.
22  *
23  * @interface Subscription
24  */
25 export interface Subscription {
26     id?: string;
27     planId: string;
28     status: UserStatus;
29     created_at: Timestamp;
30     updated_at: Timestamp;
31     userId: string;
32     transactionId?: string;
33     periodStart?: Timestamp;
34     periodEnd?: Timestamp;
35     cancelAtPeriodEnd?: boolean;
36 }
37
38 /**
39  * Service for managing user subscriptions in Firebase.
40  *
41  * This service provides CRUD operations for user subscriptions, including
42  * creating, reading, updating, and listening to subscription changes.
43  * It manages subscription data in the Firestore subcollection
44  * `users/{userId}/subscription`.
45  *
46  * Features:
47  * - Get user's latest subscription
48  * - Listen to subscription changes in real-time

```



```

49  * - Create new subscriptions
50  * - Update existing subscriptions
51  * - Delete subscriptions
52  * - Get subscription by ID
53  * - Get all user subscriptions
54  *
55  * Subscription Structure:
56  * - Stored in: `users/{userId}/subscription/{subscriptionId}`
57  * - Ordered by `created_at` descending
58  * - Only one active subscription per user expected
59  *
60  * Relations:
61  * - Used by AuthService for plan management
62  * - Used by PlanSettingsComponent for subscription display
63  * - Used by PlanLimitationsGuard for plan validation
64  *
65  * @service
66  * @injectable
67  * @providedIn root
68  */
69  @Injectable({
70    providedIn: 'root'
71  })
72  export class SubscriptionService {
73    constructor() {}
74
75    /**
76     * Obtiene la última suscripción de un usuario (único documento esperado)
77     * @param userId ID del usuario
78     * @returns Promise con la suscripción o null si no existe
79     */
80    async getUserLatestSubscription(userId: string): Promise<Subscription | null> {
81      try {
82        const paymentsRef = collection(db, 'users', userId, 'subscription');
83        const q = query(paymentsRef, orderBy('created_at', 'desc'), limit(1));
84        const querySnapshot = await getDocs(q);
85
86        if (querySnapshot.empty) {
87          return null;
88        }
89
90        const latestDoc = querySnapshot.docs[0];
91        const data = latestDoc.data();
92        return data as unknown as Subscription;
93      } catch (error) {
94        console.error('L Error al obtener suscripción del usuario:', error);
95        throw error;
96      }
97    }
98
99    /**
100     * Escucha cambios en la última suscripción del usuario (único documento esperado)
101     * Devuelve una función para desuscribirse
102     */
103    listenToUserLatestSubscription(
104      userId: string,
105      handler: (subscription: Subscription | null) => void
106    ): () => void {
107      const paymentsRef = collection(db, 'users', userId, 'subscription');
108      const q = query(paymentsRef, orderBy('created_at', 'desc'), limit(1));
109      const unsubscribe = onSnapshot(q, (snapshot) => {
110        if (snapshot.empty) {
111          handler(null);
112          return;
113        }
114        const latestDoc = snapshot.docs[0];
115        const data = latestDoc.data();
116        handler({ id: latestDoc.id, ...data } as Subscription);
117      }, (error) => {
118        console.error('L Error en listener de suscripción:', error);

```

```

119     handler(null);
120   });
121   return unsubscribe;
122 }
123
124 // TODO: IMPLEMENTAR ENDPOINT DE VERIFICACIÓN DE PAGO - Reemplazar Firebase con API real
125 /**
126  * Obtiene un pago específico por ID
127  * @param userId ID del usuario
128  * @param paymentId ID del pago
129  * @returns Promise con el pago o null si no existe
130  */
131 async getSubscriptionById(userId: string, paymentId: string): Promise<Subscription | null>
132 {   try {
133     const paymentRef = doc(db, 'users', userId, 'subscription', paymentId);
134     const paymentSnap = await getDoc(paymentRef);
135
136     if (paymentSnap.exists()) {
137       return {
138         id: paymentSnap.id,
139         ...paymentSnap.data()
140       } as Subscription;
141     } else {
142       return null;
143     }
144   } catch (error) {
145     console.error('Error al obtener pago:', error);
146     throw error;
147   }
148 }
149
150 // TODO: IMPLEMENTAR ENDPOINT DE CREACIÓN DE PAGO - Reemplazar Firebase con API real
151 /**
152  * Crea un nuevo pago
153  * @param userId ID del usuario
154  * @param paymentData Datos del pago (sin id)
155  * @returns Promise con el ID del pago creado
156  */
157 async createSubscription(userId: string, paymentData: Omit<Subscription, 'id' |
158 'created_at' | 'updated_at'>): Promise<string> {
159   const paymentsRef = collection(db, 'users', userId, 'subscription');
160   const now = Timestamp.now();
161
162   const newPayment = {
163     ...paymentData,
164     created_at: now,
165     updated_at: now
166   };
167
168   const docRef = await addDoc(paymentsRef, newPayment);
169   return docRef.id;
170 } catch (error) {
171   console.error('Error al crear pago:', error);
172   throw error;
173 }
174 }
175
176 /**
177  * Actualiza un pago existente
178  * @param userId ID del usuario
179  * @param paymentId ID del pago
180  * @param updateData Datos a actualizar
181  * @returns Promise void
182  */
183 async updateSubscription(userId: string, paymentId: string, updateData:
184 Partial<Omit<Subscription, 'id' | 'created_at' | 'updated_at' | 'userId'>>): Promise<void> {
185   const paymentRef = doc(db, 'users', userId, 'subscription', paymentId);
186   const updatePayload = {
187     ...updateData,
188     updated_at: Timestamp.now()

```

```

189     };
190
191     await updateDoc(paymentRef, updatePayload);
192   } catch (error) {
193     console.error('Error al actualizar pago:', error);
194     throw error;
195   }
196 }
197
198 /**
199  * Elimina un pago
200  * @param userId ID del usuario
201  * @param paymentId ID del pago
202  * @returns Promise void
203  */
204 async deleteSubscription(userId: string, paymentId: string): Promise<void> {
205   try {
206     const paymentRef = doc(db, 'users', userId, 'subscription', paymentId);
207     await deleteDoc(paymentRef);
208   } catch (error) {
209     console.error('Error al eliminar pago:', error);
210     throw error;
211   }
212 }
213
214 /**
215  * Obtiene pagos filtrados por estado
216  * @param userId ID del usuario
217  * @param status Estado del pago
218  * @returns Promise con array de pagos filtrados
219  */
220 async getSubscriptionsByStatus(userId: string, status: Subscription['status']):
221 Promise<Subscription[]> {
222   try {
223     const paymentsRef = collection(db, 'users', userId, 'subscription');
224     const q = query(
225       paymentsRef,
226       orderBy('created_at', 'desc')
227     );
228     const querySnapshot = await getDocs(q);
229
230     return querySnapshot.docs
231       .map(doc => ({
232         id: doc.id,
233         ...doc.data()
234       } as Subscription))
235       .filter(payment => payment.status === status);
236   } catch (error) {
237     console.error('Error al obtener pagos por estado:', error);
238     throw error;
239   }
240 }
241
242 /**
243  * Obtiene el total de pagos de un usuario
244  * @param userId ID del usuario
245  * @returns Promise con el número total de pagos
246  */
247 async getTotalSubscriptionsCount(userId: string): Promise<number> {
248   try {
249     const paymentsRef = collection(db, 'users', userId, 'subscription');
250     const querySnapshot = await getDocs(paymentsRef);
251     return querySnapshot.size;
252   } catch (error) {
253     console.error('Error al obtener conteo de pagos:', error);
254     throw error;
255   }
256 }
257
258 /**
259  * Método de debug para verificar la estructura de la base de datos

```

```

259     * @param userId ID del usuario
260     */
261 }

```

## shared\services\timezone.service.ts

---

```

1  import { Injectable } from '@angular/core';
2  import * as moment from 'moment-timezone';
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class TimezoneService {
8
9    constructor() { }
10
11    /**
12     * Obtener la zona horaria del usuario desde el navegador
13     */
14    getUserTimezone(): string {
15      try {
16        return Intl.DateTimeFormat().resolvedOptions().timeZone;
17      } catch (error) {
18        console.warn('No se pudo detectar la zona horaria del usuario, usando UTC');
19        return 'UTC';
20      }
21    }
22
23    /**
24     * Convertir una fecha local del usuario a UTC
25     * MEJORA: Manejo correcto de fechas que ya están en UTC
26     * @param localDate Fecha en zona horaria local del usuario
27     * @param userTimezone Zona horaria del usuario (opcional, se detecta automáticamente)
28     * @returns Fecha en UTC
29     */
30    convertToUTC(localDate: Date | string, userTimezone?: string): Date {
31      const timezone = userTimezone || this.getUserTimezone();
32
33      try {
34        // Si la fecha ya es un objeto Date, verificar si ya está en UTC
35        if (localDate instanceof Date) {
36          // Verificar si la fecha ya está en UTC (sin offset de zona horaria)
37          const dateString = localDate.toISOString();
38          if (dateString.includes('Z') || dateString.includes('+00:00')) {
39            // Ya está en UTC, devolver tal como está
40            return localDate;
41          }
42        }
43
44        // Si es string, verificar si ya tiene indicador UTC
45        if (typeof localDate === 'string') {
46          if (localDate.includes('Z') || localDate.includes('+00:00') ||
47              localDate.includes('UTC')) { convertir directamente
48            return new Date(localDate);
49          }
50        }
51
52        // Crear momento en la zona horaria del usuario
53        const momentInUserTz = moment.tz(localDate, timezone);
54
55        // Convertir a UTC
56        const utcMoment = momentInUserTz.utc();
57
58        return utcMoment.toDate();
59      } catch (error) {
60        console.error('Error convirtiendo fecha a UTC:', error);

```

```

61         // Fallback: usar la fecha original
62         return new Date(localDate);
63     }
64 }
65
66 /**
67  * Convertir una fecha UTC a la zona horaria del usuario
68  * @param utcDate Fecha en UTC
69  * @param userTimezone Zona horaria del usuario (opcional, se detecta automáticamente)
70  * @returns Fecha en zona horaria del usuario
71  */
72 convertFromUTC(utcDate: Date | string, userTimezone?: string): Date {
73     const timezone = userTimezone || this.getUserTimezone();
74
75     try {
76         // Crear momento UTC
77         const utcMoment = moment.utc(utcDate);
78
79         // Convertir a la zona horaria del usuario
80         const localMoment = utcMoment.tz(timezone);
81
82         return localMoment.toDate();
83     } catch (error) {
84         console.error('Error convirtiendo fecha desde UTC:', error);
85         // Fallback: usar la fecha original
86         return new Date(utcDate);
87     }
88 }
89
90 /**
91  * Verificar si una fecha está dentro de un rango de horas específico
92  * @param date Fecha a verificar
93  * @param startTime Hora de inicio (formato HH:mm)
94  * @param endTime Hora de fin (formato HH:mm)
95  * @param timezone Zona horaria para la comparación
96  * @returns true si la fecha está dentro del rango
97  */
98 isWithinTimeRange(
99     date: Date | string,
100     startTime: string,
101     endTime: string,
102     timezone: string
103 ): boolean {
104     try {
105         const momentDate = moment.tz(date, timezone);
106         const currentTime = momentDate.format('HH:mm');
107
108         // Si el rango cruza medianoche (ej: 22:00 - 06:00)
109         if (startTime > endTime) {
110             return currentTime >= startTime || currentTime <= endTime;
111         } else {
112             // Rango normal (ej: 09:00 - 17:00)
113             return currentTime >= startTime && currentTime <= endTime;
114         }
115     } catch (error) {
116         console.error('Error verificando rango de tiempo:', error);
117         return false;
118     }
119 }
120
121 /**
122  * Obtener la fecha actual en UTC
123  * @returns Fecha actual en UTC
124  */
125 getCurrentUTC(): Date {
126     return moment.utc().toDate();
127 }
128
129 /**
130  * Obtener la fecha actual en la zona horaria del usuario

```

```

131     * @param userTimezone Zona horaria del usuario (opcional)
132     * @returns Fecha actual en zona horaria del usuario
133     */
134     getCurrentLocal(userTimezone?: string): Date {
135         const timezone = userTimezone || this.getUserTimezone();
136         return moment.tz(timezone).toDate();
137     }
138
139     /**
140     * Formatear fecha para mostrar en la zona horaria del usuario
141     * @param date Fecha a formatear
142     * @param format Formato deseado (por defecto: 'YYYY-MM-DD HH:mm:ss')
143     * @param userTimezone Zona horaria del usuario (opcional)
144     * @returns Fecha formateada
145     */
146     formatDate(
147         date: Date | string,
148         format: string = 'YYYY-MM-DD HH:mm:ss',
149         userTimezone?: string
150     ): string {
151         const timezone = userTimezone || this.getUserTimezone();
152
153         try {
154             return moment.tz(date, timezone).format(format);
155         } catch (error) {
156             console.error('Error formateando fecha:', error);
157             return new Date(date).toString();
158         }
159     }
160
161     /**
162     * Comparar dos fechas considerando zona horaria
163     * @param date1 Primera fecha
164     * @param date2 Segunda fecha
165     * @param timezone Zona horaria para la comparación
166     * @returns -1 si date1 < date2, 0 si son iguales, 1 si date1 > date2
167     */
168     compareDates(
169         date1: Date | string,
170         date2: Date | string,
171         timezone?: string
172     ): number {
173         const tz = timezone || this.getUserTimezone();
174
175         try {
176             const moment1 = moment.tz(date1, tz);
177             const moment2 = moment.tz(date2, tz);
178
179             if (moment1.isBefore(moment2)) return -1;
180             if (moment1.isAfter(moment2)) return 1;
181             return 0;
182         } catch (error) {
183             console.error('Error comparando fechas:', error);
184             return 0;
185         }
186     }
187
188     /**
189     * Verificar si una fecha está en el día de la semana especificado
190     * @param date Fecha a verificar
191     * @param dayOfWeek Día de la semana (0=domingo, 1=lunes, ..., 6=sábado)
192     * @param timezone Zona horaria para la verificación
193     * @returns true si la fecha está en el día especificado
194     */
195     isDayOfWeek(
196         date: Date | string,
197         dayOfWeek: number,
198         timezone?: string
199     ): boolean {
200         const tz = timezone || this.getUserTimezone();

```

```

201
202     try {
203         const momentDate = moment.tz(date, tz);
204         return momentDate.day() === dayOfWeek;
205     } catch (error) {
206         console.error('Error verificando día de la semana:', error);
207         return false;
208     }
209 }
210
211 /**
212  * Convertir fecha de trade del servidor a UTC
213  * MÉTODO ESPECÍFICO: Para fechas que vienen del servidor y pueden estar en diferentes
214  * formatos. Param tradeDate Fecha del trade (puede ser timestamp, string, o Date)
215  * @returns Fecha en UTC
216  */
217 convertTradeDateToUTC(tradeDate: any): Date {
218     try {
219         let utcDate: Date;
220
221         // Si es un timestamp numérico (milisegundos)
222         if (typeof tradeDate === 'number') {
223             utcDate = new Date(tradeDate);
224             // Verificar si es válido
225             if (isNaN(utcDate.getTime())) {
226                 throw new Error('Timestamp inválido');
227             }
228         }
229         // Si es un string
230         else if (typeof tradeDate === 'string') {
231             // Si contiene 'Z' o '+00:00', ya está en UTC
232             if (tradeDate.includes('Z') || tradeDate.includes('+00:00')) {
233                 utcDate = new Date(tradeDate);
234             }
235             // Si es un timestamp en string
236             else {
237                 const numericValue = parseInt(tradeDate);
238                 if (!isNaN(numericValue)) {
239                     utcDate = new Date(numericValue);
240                 } else {
241                     utcDate = new Date(tradeDate);
242                 }
243             }
244         }
245         // Si ya es un Date
246         else if (tradeDate instanceof Date) {
247             utcDate = tradeDate;
248         }
249         // Fallback
250         else {
251             utcDate = new Date(tradeDate);
252         }
253
254         // FORZAR UTC: Crear una nueva fecha usando UTC para asegurar que esté en UTC
255         const utcTimestamp = Date.UTC(
256             utcDate.getUTCFullYear(),
257             utcDate.getUTCMonth(),
258             utcDate.getUTCDate(),
259             utcDate.getUTCHours(),
260             utcDate.getUTCMinutes(),
261             utcDate.getUTCSeconds(),
262             utcDate.getUTCMilliseconds()
263         );
264
265         return new Date(utcTimestamp);
266     } catch (error) {
267         console.error('Error convirtiendo fecha de trade a UTC:', error);
268         return new Date();
269     }
270 }

```

```

271
272 /**
273  * Obtener información de debug sobre zona horaria
274  * @param date Fecha a analizar
275  * @param userTimezone Zona horaria del usuario (opcional)
276  * @returns Información de debug
277  */
278 getTimezoneDebugInfo(date: Date | string, userTimezone?: string): any {
279     const timezone = userTimezone || this.getUserTimezone();
280
281     try {
282         const momentDate = moment.tz(date, timezone);
283         const utcDate = moment.utc(date);
284
285         return {
286             originalDate: date,
287             userTimezone: timezone,
288             localTime: momentDate.format('YYYY-MM-DD HH:mm:ss'),
289             utcTime: utcDate.format('YYYY-MM-DD HH:mm:ss'),
290             offset: momentDate.utcOffset(),
291             offsetString: momentDate.format('Z'),
292             dayOfWeek: momentDate.day(),
293             dayName: momentDate.format('dddd')
294         };
295     } catch (error) {
296         return {
297             error: error instanceof Error ? error.message : 'Unknown error',
298             originalDate: date,
299             userTimezone: timezone
300         };
301     }
302 }
303 }
304

```

## Ø=ÜÄ shared\services\tokens-operations.service.ts

---

```

1 import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2 import { getFirestore, doc, setDoc, deleteDoc } from 'firebase/firestore';
3 import { isPlatformBrowser } from '@angular/common';
4
5 /**
6  * Interface for link token data.
7  *
8  * @interface LinkToken
9  */
10 export interface LinkToken {
11     id: string;
12     [key: string]: any;
13 }
14
15 /**
16  * Service for managing link tokens in Firebase.
17  *
18  * This service provides operations for creating and deleting link tokens
19  * that are used for user authentication and account linking. Tokens are
20  * stored in the `tokens` collection.
21  *
22  * Features:
23  * - Create link token
24  * - Delete link token
25  *
26  * Token Structure:
27  * - Stored in: `tokens/{tokenId}`
28  * - Used for: User authentication, account linking
29  *

```



```

30 * Relations:
31 * - Used by AuthService for token management
32 * - Used for logout everywhere functionality (token revocation)
33 *
34 * @service
35 * @injectable
36 * @providedIn root
37 */
38 @Injectable({
39   providedIn: 'root'
40 })
41 export class TokensOperationsService {
42   private isBrowser: boolean;
43   private db: ReturnType<typeof getFirestore> | null = null;
44
45   constructor(@Inject(PLATFORM_ID) private platformId: Object) {
46     this.isBrowser = isPlatformBrowser(this.platformId);
47     if (this.isBrowser) {
48       const { firebaseApp } = require('../../firebase/firebase.init.ts');
49       this.db = getFirestore(firebaseApp);
50     }
51   }
52
53   /**
54    * Crear token de enlace
55    */
56   async createLinkToken(token: LinkToken): Promise<void> {
57     if (this.db) {
58       await setDoc(doc(this.db, 'tokens', token.id), token);
59     } else {
60       console.warn('Firestore not available in SSR');
61       return;
62     }
63   }
64
65   /**
66    * Eliminar token de enlace
67    */
68   async deleteLinkToken(tokenId: string): Promise<void> {
69     if (this.db) {
70       await deleteDoc(doc(this.db, 'tokens', tokenId));
71     } else {
72       console.warn('Firestore not available in SSR');
73       return;
74     }
75   }
76 }
77

```

## Ø=ÜÄ shared\services\tradelocker-api.service.ts

---

```

1 import { Injectable } from '@angular/core';
2 import { HttpClient, HttpHeaders } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4 import { map } from 'rxjs/operators';
5
6 /**
7  * Interface for TradeLocker API credentials.
8  *
9  * @interface TradeLockerCredentials
10 */
11 export interface TradeLockerCredentials {
12   email: string;
13   password: string;
14   server: string;
15 }

```

```

16
17 /**
18  * Interface for TradeLocker token response.
19  *
20  * @interface TradeLockerTokenResponse
21  */
22 export interface TradeLockerTokenResponse {
23     accessToken: string;
24     tokenType: string;
25     expiresIn: number;
26 }
27
28 /**
29  * Interface for TradeLocker account data.
30  *
31  * @interface TradeLockerAccount
32  */
33 export interface TradeLockerAccount {
34     accountId: string;
35     accountName: string;
36     balance: number;
37     currency: string;
38     server: string;
39 }
40
41 /**
42  * Service for interacting with the TradeLocker API.
43  *
44  * This service provides methods to authenticate with TradeLocker, fetch
45  * account balances, trading history, and instrument details. It handles
46  * JWT token management and API communication.
47  *
48  * Features:
49  * - JWT token authentication
50  * - Token refresh
51  * - Account validation
52  * - Account balance fetching
53  * - Trading history retrieval
54  * - Instrument details fetching
55  * - All instruments listing
56  * - User key generation
57  *
58  * API Endpoints:
59  * - Base URL: https://demo.tradelocker.com/backend-api
60  * - Auth: /auth/jwt/token, /auth/jwt/refresh
61  * - Trade: /trade/accounts/{accountId}/state, /trade/accounts/{accountId}/ordersHistory
62  * - Instruments: /trade/instruments/{tradableInstrumentId}, /trade/accounts/{accountId}/
63 instruments
64  * Relations:
65  * - Used by ReportService for fetching trading data
66  * - Used by CreateAccountPopupComponent for account validation
67  * - Used by TradingAccountsComponent for balance fetching
68  *
69  * @service
70  * @injectable
71  * @providedIn root
72  */
73 @Injectable({
74     providedIn: 'root'
75 })
76 export class TradeLockerApiService {
77     private readonly baseUrl = 'https://demo.tradelocker.com/backend-api';
78
79     constructor(private http: HttpClient) {}
80
81     /**
82      * Get JWT token from TradeLocker
83      */
84     getJWTToken(credentials: TradeLockerCredentials): Observable<TradeLockerTokenResponse> {
85         const tokenUrl = `${this.baseUrl}/auth/jwt/token`;

```

```

86
87     const headers = new HttpHeaders({
88         'Content-Type': 'application/json',
89     });
90
91     const body = {
92         email: credentials.email,
93         password: credentials.password,
94         server: credentials.server
95     };
96
97     return this.http.post<TradeLockerTokenResponse>(tokenUrl, body, { headers });
98 }
99
100 refreshToken(accessToken: string): Observable<any> {
101     const refreshUrl = `${this.baseUrl}/auth/jwt/refresh`;
102
103     const headers = new HttpHeaders({
104         'Content-Type': 'application/json',
105         'Authorization': `Bearer ${accessToken}`
106     });
107
108     return this.http.post<any>(refreshUrl, { headers });
109 }
110
111 /**
112  * Validate account credentials in TradeLocker
113  */
114 async validateAccount(credentials: TradeLockerCredentials): Promise<boolean> {
115     try {
116         const tokenResponse = await this.getJWTToken(credentials).toPromise();
117         return !!(tokenResponse && tokenResponse.accessToken);
118     } catch (error) {
119         console.error('Error validating account in TradeLocker:', error);
120         return false;
121     }
122 }
123
124 /**
125  * Get account balance from TradeLocker
126  */
127 getAccountBalance(accountId: string, userKey: string, accountNumber: number):
128 Observable<any> {
129     const balanceUrl = `${this.baseUrl}/trade/accounts/${accountId}/state`;
130
131     const headers = new HttpHeaders({
132         'Content-Type': 'application/json',
133         'Authorization': `Bearer ${userKey}`,
134         'accNum': accountNumber.toString()
135     });
136
137     return this.http.get(balanceUrl, { headers });
138 }
139
140 /**
141  * Get trading history from TradeLocker
142  */
143 getTradingHistory(userKey: string, accountId: string, accNum: number): Observable<any> {
144     const historyUrl = `${this.baseUrl}/trade/accounts/${accountId}/ordersHistory`;
145
146     const headers = new HttpHeaders({
147         'Content-Type': 'application/json',
148         'Authorization': `Bearer ${userKey}`,
149         'accNum': accNum.toString()
150     });
151
152     return this.http.get(historyUrl, { headers });
153 }
154
155 /**
156  * Get user key for API calls

```

```

156     */
157     getUserKey(email: string, password: string, server: string): Observable<string> {
158         const credentials: TradeLockerCredentials = { email, password, server };
159         return this.getJWTToken(credentials).pipe(
160             map(response => response.accessToken)
161         );
162     }
163
164     /**
165      * Get instrument details
166      */
167     getInstrumentDetails(accessToken: string, tradableInstrumentId: string, routeId: string,
168     accNum: number): Observable<any> {
169         const instrumentsUrl = `${this.baseUrl}/trade/instruments/${tradableInstrumentId}`;
170
171         const headers = new HttpHeaders({
172             'Content-Type': 'application/json',
173             'Authorization': `Bearer ${accessToken}`,
174             'accNum': accNum.toString()
175         });
176
177         const params = {
178             routeId: routeId
179         };
180
181         return this.http.get(instrumentsUrl, { headers, params });
182     }
183
184     /**
185      * Get all instruments for an account
186      */
187     getAllInstruments(accessToken: string, accountId: string, accNum: number): Observable<any>
188     {
189         const instrumentsUrl = `${this.baseUrl}/trade/accounts/${accountId}/instruments`;
190
191         const headers = new HttpHeaders({
192             'Content-Type': 'application/json',
193             'Authorization': `Bearer ${accessToken}`,
194             'accNum': accNum.toString()
195         });
196
197         return this.http.get(instrumentsUrl, { headers });
198     }
199

```

## shared\services\user-management.service.ts

```

1 import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2 import { getFirestore, collection, getDocs, doc, getDoc, updateDoc, deleteDoc, query, where,
3 orderBy } from '@angular/firestore';
4 import { User, UserStatus } from '../features/overview/models/overview';
5
6 /**
7  * Service for managing user data operations (administrative).
8  *
9  * This service provides administrative operations for user management,
10  * including fetching all users, filtering by status, and getting top users.
11  * It's designed for admin interfaces and user management dashboards.
12  *
13  * Features:
14  * - Get all users from Firebase
15  * - Get user by ID
16  * - Update user data
17  * - Delete user
18  * - Get users by status (active, banned, etc.)
19  * - Get top users (ordered by number of trades)

```

```

20  *
21  * Usage:
22  * Primarily used by admin components like UsersDetailsComponent for
23  * managing and viewing user data.
24  *
25  * Relations:
26  * - Used by UsersDetailsComponent for user management
27  * - Used by OverviewComponent for user statistics
28  *
29  * @service
30  * @injectable
31  * @providedIn root
32  */
33  @Injectable({
34    providedIn: 'root'
35  })
36  export class UserManagementService {
37    private isBrowser: boolean;
38    private db: ReturnType<typeof getFirestore> | null = null;
39
40    constructor(@Inject(PLATFORM_ID) private platformId: Object) {
41      this.isBrowser = isPlatformBrowser(this.platformId);
42      if (this.isBrowser) {
43        const { firebaseApp } = require('../../firebase/firebase.init.ts');
44        this.db = getFirestore(firebaseApp);
45      }
46    }
47
48    /**
49     * Get all users from Firebase
50     */
51    async getAllUsers(): Promise<User[]> {
52      if (!this.db) {
53        console.warn('Firestore not available in SSR');
54        return [];
55      }
56
57      try {
58        const snapshot = await getDocs(collection(this.db, 'users'));
59        const users: User[] = [];
60
61        snapshot.forEach((doc) => {
62          const userData = doc.data() as User;
63          (userData as any).id = doc.id;
64          users.push(userData);
65        });
66
67        return users;
68      } catch (error) {
69        console.error('Error getting all users:', error);
70        return [];
71      }
72    }
73
74    /**
75     * Get user by ID
76     */
77    async getUserById(userId: string): Promise<User | null> {
78      if (!this.db) {
79        console.warn('Firestore not available in SSR');
80        return null;
81      }
82
83      try {
84        const userDoc = await getDoc(doc(this.db, 'users', userId));
85        if (userDoc.exists()) {
86          const userData = userDoc.data() as User;
87          (userData as any).id = userDoc.id;
88          return userData;
89        }

```

```

90         return null;
91     } catch (error) {
92         console.error('Error getting user by ID:', error);
93         return null;
94     }
95 }
96
97 /**
98  * Update user data
99  */
100 async updateUser(userId: string, userData: Partial<User>): Promise<void> {
101     if (!this.db) {
102         console.warn('Firestore not available in SSR');
103         return;
104     }
105
106     try {
107         await updateDoc(doc(this.db, 'users', userId), userData);
108     } catch (error) {
109         console.error('Error updating user:', error);
110         throw error;
111     }
112 }
113
114 /**
115  * Delete user
116  */
117 async deleteUser(userId: string): Promise<void> {
118     if (!this.db) {
119         console.warn('Firestore not available in SSR');
120         return;
121     }
122
123     try {
124         await deleteDoc(doc(this.db, 'users', userId));
125     } catch (error) {
126         console.error('Error deleting user:', error);
127         throw error;
128     }
129 }
130
131 /**
132  * Get users by status
133  */
134 async getUsersByStatus(status: UserStatus): Promise<User[]> {
135     if (!this.db) {
136         console.warn('Firestore not available in SSR');
137         return [];
138     }
139
140     try {
141         const q = query(
142             collection(this.db, 'users'),
143             where('status', '==', status)
144         );
145
146         const snapshot = await getDocs(q);
147         const users: User[] = [];
148
149         snapshot.forEach((doc) => {
150             const userData = doc.data() as User;
151             (userData as any).id = doc.id;
152             users.push(userData);
153         });
154
155         return users;
156     } catch (error) {
157         console.error('Error getting users by status:', error);
158         return [];
159     }

```

```

160     }
161
162     /**
163      * Get top users (ordered by some criteria)
164      */
165     async getTopUsers(limitCount: number = 10): Promise<User[]> {
166         if (!this.db) {
167             console.warn('Firestore not available in SSR');
168             return [];
169         }
170
171         try {
172             const q = query(
173                 collection(this.db, 'users'),
174                 orderBy('number_trades', 'desc'),
175                 limit(limitCount)
176             );
177
178             const snapshot = await getDocs(q);
179             const users: User[] = [];
180
181             snapshot.forEach((doc) => {
182                 const userData = doc.data() as User;
183                 (userData as any).id = doc.id;
184                 users.push(userData);
185             });
186
187             return users;
188         } catch (error) {
189             console.error('Error getting top users:', error);
190             return [];
191         }
192     }
193 }
194

```

## Ø=ÜÄ shared\services\users-operations.service.ts

---

```

1  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
2  import { getFirestore, doc, setDoc, getDoc, collection, getDocs, deleteDoc } from 'firebase/
3  firestore';
4  import { isPlatformBrowser } from '@angular/common';
5  import { User } from '../../features/overview/models/overview';
6
7  /**
8   * Service for user data operations in Firebase.
9   *
10   * This service provides CRUD operations for user documents in Firestore.
11   * It handles user creation, retrieval, updates, and deletion. It's used
12   * throughout the application for user data management.
13   *
14   * Features:
15   * - Get user data by UID
16   * - Create new user
17   * - Get user by ID
18   * - Get user by email
19   * - Update user data
20   * - Get all users
21   * - Delete user
22   *
23   * User Data Structure:
24   * - Stored in: `users/{userId}`
25   * - Includes: profile data, trading statistics, subscription info
26   *
27   * Relations:
28   * - Used by AuthService for user operations
29   * - Used by ProfileDetailsComponent for profile updates
30   */

```

```

29  * - Used by various components for user data access
30  *
31  * @service
32  * @injectable
33  * @providedIn root
34  */
35  @Injectable({
36    providedIn: 'root'
37  })
38  export class UsersOperationsService {
39    private isBrowser: boolean;
40    private db: ReturnType<typeof getFirestore> | null = null;
41
42    constructor(@Inject(PLATFORM_ID) private platformId: Object) {
43      this.isBrowser = isPlatformBrowser(this.platformId);
44      if (this.isBrowser) {
45        const { firebaseApp } = require('../..../firebase/firebase.init.ts');
46        this.db = getFirestore(firebaseApp);
47      }
48    }
49
50    /**
51     * Obtener datos de un usuario por UID
52     */
53    async getUserData(uid: string): Promise<User> {
54      if (this.db) {
55        const userDoc = doc(this.db, 'users', uid);
56        return getDoc(userDoc).then((doc) => {
57          if (doc.exists()) {
58            return doc.data() as User;
59          } else {
60            throw new Error('User not found');
61          }
62        });
63      } else {
64        console.warn('Firestore not available in SSR');
65        return Promise.resolve({} as User);
66      }
67    }
68
69    /**
70     * Crear usuario
71     */
72    async createUser(user: User): Promise<void> {
73      if (!this.db) {
74        console.warn('Firestore not available in SSR');
75        return;
76      }
77      await setDoc(doc(this.db, 'users', user.id), user);
78    }
79
80    /**
81     * Obtener un usuario por su ID
82     */
83    async getUserById(userId: string): Promise<User | null> {
84      try {
85        if (!this.isBrowser || !this.db) {
86          return null;
87        }
88
89        const userDoc = await getDoc(doc(this.db, 'users', userId));
90        if (userDoc.exists()) {
91          return { id: userDoc.id, ...userDoc.data() } as User;
92        }
93        return null;
94      } catch (error) {
95        console.error('Error obteniendo usuario por ID:', error);
96        return null;
97      }
98    }

```



```

99
100 /**
101  * Buscar un usuario por su email
102  */
103 async getUserByEmail(email: string): Promise<User | null> {
104   try {
105     if (!this.isBrowser || !this.db) {
106       return null;
107     }
108
109     const usersSnapshot = await getDocs(collection(this.db, 'users'));
110
111     for (const doc of usersSnapshot.docs) {
112       const userData = doc.data() as User;
113       if (userData.email === email) {
114         return { ...userData, id: doc.id } as User;
115       }
116     }
117
118     return null;
119   } catch (error) {
120     console.error('Error buscando usuario por email:', error);
121     return null;
122   }
123 }
124
125 /**
126  * Actualizar un usuario existente
127  */
128 async updateUser(userId: string, userData: Partial<User>): Promise<void> {
129   try {
130     if (!this.isBrowser || !this.db) {
131       throw new Error('No se puede actualizar usuario en el servidor');
132     }
133
134     await setDoc(doc(this.db, 'users', userId), {
135       ...userData,
136       lastUpdated: new Date().getTime()
137     }, { merge: true });
138   } catch (error) {
139     console.error('Error actualizando usuario:', error);
140     throw error;
141   }
142 }
143
144 /**
145  * Obtener todos los usuarios
146  */
147 async getAllUsers(): Promise<User[]> {
148   if (!this.db) {
149     console.warn('Firestore not available in SSR');
150     return [];
151   }
152
153   try {
154     const snapshot = await getDocs(collection(this.db, 'users'));
155     const users: User[] = [];
156
157     snapshot.forEach((doc) => {
158       const data = doc.data() as User;
159       (data as any).id = doc.id;
160       users.push(data);
161     });
162
163     return users;
164   } catch (error) {
165     console.error('Error getting all users:', error);
166     return [];
167   }
168 }

```

```

169
170 /**
171  * Eliminar un usuario
172  */
173 async deleteUser(userId: string): Promise<void> {
174     try {
175         if (!this.isBrowser || !this.db) {
176             throw new Error('No se puede eliminar usuario en el servidor');
177         }
178
179         await deleteDoc(doc(this.db, 'users', userId));
180         console.log('Usuario eliminado exitosamente:', userId);
181     } catch (error) {
182         console.error('Error eliminando usuario:', error);
183         throw error;
184     }
185 }
186 }
187

```

## Ø=ÜÁ shared\sidebar-menu

### Ø=ÜÄ shared\sidebar-menu\sidebar.component.ts

---

```

1  import { Component, OnInit, OnDestroy } from '@angular/core';
2  import { Router, RouterLink, RouterLinkActive, NavigationEnd } from '@angular/router';
3  import { filter } from 'rxjs/operators';
4  import { AuthService } from '../../../features/auth/service/authService';
5  import { Store } from '@ngrx/store';
6  import { selectUser } from '../../../features/auth/store/user.selectios';
7  import { setUserData } from '../../../features/auth/store/user.actions';
8  import { User, UserStatus } from '../../../features/overview/models/overview';
9
10 /**
11  * Sidebar navigation component for the application.
12  *
13  * This component provides the main navigation sidebar with user information,
14  * menu items, and logout functionality. It dynamically adjusts its width based
15  * on the current route and user state, and manages CSS custom properties for
16  * layout consistency.
17  *
18  * Features:
19  * - User information display (name, initials, admin status, ban status)
20  * - Collapsible sidebar (minimized/expanded states)
21  * - Dynamic width management via CSS custom properties
22  * - Route-based visibility (hidden on login/signup pages)
23  * - Logout functionality
24  * - Dashboard section toggle
25  * - Responsive to route changes
26  *
27  * Width Management:
28  * - Expanded: 230px
29  * - Minimized: 80px
30  * - Hidden: 0px (on login/signup routes)
31  * - Uses CSS custom property: --sidebar-width
32  *
33  * Relations:
34  * - AuthService: Handles logout functionality
35  * - Store (NgRx): Gets current user data
36  * - Router: Monitors route changes for width adjustment
37  *
38  * @component
39  * @selector app-sidebar

```

```

40  * @standalone true
41  */
42  @Component({
43    selector: 'app-sidebar',
44    imports: [RouterLink, RouterLinkActive],
45    templateUrl: './sidebar.component.html',
46    styleUrls: ['./sidebar.component.scss'],
47    standalone: true,
48  })
49  export class Sidebar implements OnDestroy {
50    isDashboardOpen = true;
51    isSidebarMinimized = false;
52    userName: string = '';
53    lastName: string = '';
54    isAdmin: boolean = false;
55    userToken: string = '';
56    isBanned: boolean = false;
57    private hasInitializedWidth = false;
58
59    constructor(
60      private authService: AuthService,
61      private router: Router,
62      private store: Store
63    ) {
64      // Inicializar el ancho a 0 inmediatamente para evitar el flash
65      if (typeof document !== 'undefined') {
66        const root = document.documentElement;
67        root.style.setProperty('--sidebar-width', '0px');
68      }
69
70      this.store.select(selectUser).subscribe((user) => {
71        this.userName = user?.user?.firstName || '';
72        this.lastName = user?.user?.lastName || '';
73        this.isAdmin = user?.user?.isAdmin || false;
74        this.userToken = user?.user?.tokenId || '';
75        this.isBanned = user?.user?.status === UserStatus.BANNED;
76
77        // Solo actualizar el ancho del sidebar una vez que tengamos datos del usuario
78        if (this.userName && !this.hasInitializedWidth) {
79          this.hasInitializedWidth = true;
80          this.updateSidebarWidth(this.router.url);
81        } else if (!this.userName && this.hasInitializedWidth) {
82          // Si el usuario se desloguea, resetear
83          this.hasInitializedWidth = false;
84          if (typeof document !== 'undefined') {
85            const root = document.documentElement;
86            root.style.setProperty('--sidebar-width', '0px');
87          }
88        }
89      });
90
91      // Escuchar cambios de ruta para ajustar el sidebar
92      this.router.events
93        .pipe(filter(event => event instanceof NavigationEnd))
94        .subscribe((event: NavigationEnd) => {
95          // Solo actualizar si ya tenemos usuario
96          if (this.userName) {
97            this.updateSidebarWidth(event.url);
98          }
99        });
100    }
101
102    private updateSidebarWidth(url: string) {
103      if (typeof document === 'undefined') return;
104
105      const root = document.documentElement;
106      const routesWithoutSidebar = ['/login', '/signup'];
107      const hasSidebar = !routesWithoutSidebar.some(route => url.includes(route));
108
109      if (hasSidebar) {

```

```

110         // Hay sidebar, usar el ancho según el estado
111         if (this.isSidebarMinimized) {
112             root.style.setProperty('--sidebar-width', '80px');
113         } else {
114             root.style.setProperty('--sidebar-width', '230px');
115         }
116     } else {
117         // No hay sidebar, resetear a 0
118         root.style.setProperty('--sidebar-width', '0px');
119     }
120 }
121
122 ngOnDestroy() {
123     // Resetear la variable CSS cuando el componente se destruye (ej: al ir a login)
124     if (typeof document !== 'undefined') {
125         const root = document.documentElement;
126         root.style.setProperty('--sidebar-width', '0px');
127     }
128 }
129
130 onlyNameInitials() {
131     return this.userName.charAt(0) + this.lastName.charAt(1);
132 }
133
134 toggleDashboard() {
135     this.isDashboardOpen = !this.isDashboardOpen;
136 }
137
138 toggleSidebar() {
139     this.isSidebarMinimized = !this.isSidebarMinimized;
140     // Actualizar variable CSS usando la nueva lógica
141     this.updateSidebarWidth(this.router.url);
142 }
143
144 logout() {
145     this.authService
146         .logout()
147         .then(() => {
148             // Limpiar todo el localStorage
149             localStorage.clear();
150             this.store.dispatch(setUserData({ user: null }));
151             this.router.navigate(['/login']);
152         })
153         .catch((error) => {
154             alert('Logout failed. Please try again.');
```

## shared\utils

### shared\utils\number-formatter.service.ts

---

```

1 import { Injectable } from '@angular/core';
2
3 /**
4  * Service for formatting numbers, currency, and percentages.
5  *
6  * This service provides comprehensive number formatting utilities for displaying
7  * numeric values in various formats throughout the application. It handles
8  * currency formatting, percentage formatting, number formatting with separators,
9  * and input value formatting.
```

```

10  *
11  * Features:
12  * - Currency formatting (USD with $ symbol)
13  * - Percentage formatting (with % symbol)
14  * - Number formatting with thousand separators
15  * - Input value formatting (for form inputs)
16  * - Currency value parsing (from formatted strings)
17  * - Integer formatting
18  * - Null/undefined handling
19  * - Decimal place control
20  *
21  * Formatting Methods:
22  * - formatCurrency(): Formats as currency with $ symbol
23  * - formatPercentage(): Formats as percentage with % symbol
24  * - formatNumber(): Formats with thousand separators
25  * - formatCurrencyValue(): Formats currency without $ symbol
26  * - formatPercentageValue(): Formats percentage without % symbol
27  * - formatInteger(): Formats as integer (no decimals)
28  * - formatInputValue(): Formats during typing
29  * - parseCurrencyValue(): Parses formatted currency back to number
30  *
31  * Relations:
32  * - Used by CurrencyFormatPipe, NumberFormatPipe, PercentageFormatPipe
33  * - Used by form components for input formatting
34  *
35  * @service
36  * @injectable
37  * @providedIn root
38  */
39  @Injectable({
40    providedIn: 'root'
41  })
42  export class NumberFormatterService {
43
44    constructor() { }
45
46    /**
47     * Formats a number as currency with $ symbol and proper separators
48     * @param value - The number to format
49     * @param decimals - Number of decimal places (default: 2)
50     * @returns Formatted currency string
51     */
52    formatCurrency(value: number | string | null | undefined): string {
53      if (value === null || value === undefined || value === '') {
54        return '$0.00';
55      }
56
57      const numValue = typeof value === 'string' ? parseFloat(value) : value;
58
59      if (isNaN(numValue)) {
60        return '$0.00';
61      }
62
63      // Truncate to 2 decimal places
64      const truncated = Math.floor(numValue * 100) / 100;
65
66      return new Intl.NumberFormat('en-US', {
67        style: 'currency',
68        currency: 'USD',
69        minimumFractionDigits: 2,
70        maximumFractionDigits: 2
71      }).format(truncated);
72    }
73
74    /**
75     * Formats a number as percentage with % symbol
76     * @param value - The number to format
77     * @param decimals - Number of decimal places (default: 2)
78     * @returns Formatted percentage string
79     */

```

```

80 formatPercentage(value: number | string | null | undefined): string {
81     if (value === null || value === undefined || value === '') {
82         return '0.00%';
83     }
84
85     const numValue = typeof value === 'string' ? parseFloat(value) : value;
86
87     if (isNaN(numValue)) {
88         return '0.00%';
89     }
90
91     // Truncate to 2 decimal places
92     const truncated = Math.floor(numValue * 100) / 100;
93
94     return new Intl.NumberFormat('en-US', {
95         style: 'percent',
96         minimumFractionDigits: 2,
97         maximumFractionDigits: 2
98     }).format(truncated / 100);
99 }
100
101 /**
102  * Formats a number with proper separators and 2 decimal places
103  * @param value - The number to format
104  * @param decimals - Number of decimal places (default: 2)
105  * @returns Formatted number string
106  */
107 formatNumber(value: number | string | null | undefined, decimals: number = 2): string {
108     if (value === null || value === undefined || value === '') {
109         return '0.00';
110     }
111
112     const numValue = typeof value === 'string' ? parseFloat(value) : value;
113
114     if (isNaN(numValue)) {
115         return '0.00';
116     }
117
118     // Truncate to specified decimal places
119     const truncated = Math.floor(numValue * Math.pow(10, decimals)) / Math.pow(10, decimals);
120
121     return new Intl.NumberFormat('en-US', {
122         minimumFractionDigits: decimals,
123         maximumFractionDigits: decimals
124     }).format(truncated);
125 }
126
127 /**
128  * Formats a number as currency without the $ symbol (for display purposes)
129  * @param value - The number to format
130  * @returns Formatted number string with separators
131  */
132 formatCurrencyValue(value: number | string | null | undefined): string {
133     if (value === null || value === undefined || value === '') {
134         return '0.00';
135     }
136
137     const numValue = typeof value === 'string' ? parseFloat(value) : value;
138
139     if (isNaN(numValue)) {
140         return '0.00';
141     }
142
143     // Truncate to 2 decimal places
144     const truncated = Math.floor(numValue * 100) / 100;
145
146     return new Intl.NumberFormat('en-US', {
147         minimumFractionDigits: 2,
148         maximumFractionDigits: 2
149     }).format(truncated);

```

```

150 }
151
152 /**
153  * Formats a number as percentage without the % symbol (for display purposes)
154  * @param value - The number to format
155  * @returns Formatted percentage value string
156  */
157 formatPercentageValue(value: number | string | null | undefined): string {
158   if (value === null || value === undefined || value === '') {
159     return '0.00';
160   }
161
162   const numValue = typeof value === 'string' ? parseFloat(value) : value;
163
164   if (isNaN(numValue)) {
165     return '0.00';
166   }
167
168   // Truncate to 2 decimal places
169   const truncated = Math.floor(numValue * 100) / 100;
170
171   return new Intl.NumberFormat('en-US', {
172     minimumFractionDigits: 2,
173     maximumFractionDigits: 2
174   }).format(truncated);
175 }
176
177 /**
178  * Formats a number as an integer (no decimal places) with proper separators
179  * @param value - The number to format
180  * @returns Formatted integer string
181  */
182 formatInteger(value: number | string | null | undefined): string {
183   if (value === null || value === undefined || value === '') {
184     return '0';
185   }
186
187   const numValue = typeof value === 'string' ? parseFloat(value) : value;
188
189   if (isNaN(numValue)) {
190     return '0';
191   }
192
193   // Round to nearest integer
194   const rounded = Math.round(numValue);
195
196   return new Intl.NumberFormat('en-US', {
197     minimumFractionDigits: 0,
198     maximumFractionDigits: 0
199   }).format(rounded);
200 }
201
202 /**
203  * Formats input value during typing (removes non-numeric characters except decimal point)
204  * @param input - The input string to clean
205  * @returns Cleaned numeric string
206  */
207 cleanNumericInput(input: string): string {
208   return input.replace(/[^0-9.]/g, '');
209 }
210
211 /**
212  * Formats a number for input display with proper separators during typing
213  * @param input - The input string
214  * @returns Formatted string with separators
215  */
216 formatInputValue(input: string): string {
217   if (input === '') {
218     return '';
219   }

```

```

220
221 // Clean the input
222 let cleaned = this.cleanNumericInput(input);
223
224 // Avoid more than one decimal point
225 const parts = cleaned.split('.');
226 if (parts.length > 2) {
227     cleaned = parts[0] + '.' + parts[1];
228 }
229
230 // Convert to number
231 const value = parseFloat(cleaned);
232 if (isNaN(value)) {
233     return '';
234 }
235
236 // Format with separators and preserve decimal places
237 return value.toLocaleString('en-US', {
238     minimumFractionDigits: parts[1] ? parts[1].length : 0,
239     maximumFractionDigits: parts[1] ? parts[1].length : 2
240 });
241 }
242
243 /**
244  * Formats a number as currency for display (on blur)
245  * @param value - The number to format
246  * @returns Formatted currency string with $ symbol
247  */
248 formatCurrencyDisplay(value: number | string | null | undefined): string {
249     if (value === null || value === undefined || value === '') {
250         return '';
251     }
252
253     const numValue = typeof value === 'string' ? parseFloat(value) : value;
254
255     if (isNaN(numValue) || numValue <= 0) {
256         return '';
257     }
258
259     return new Intl.NumberFormat('en-US', {
260         style: 'currency',
261         currency: 'USD',
262         minimumFractionDigits: 2,
263         maximumFractionDigits: 2
264     }).format(numValue);
265 }
266
267 /**
268  * Parses a formatted currency string back to a number
269  * @param formattedValue - The formatted currency string
270  * @returns The numeric value
271  */
272 parseCurrencyValue(formattedValue: string): number {
273     if (!formattedValue) return 0;
274
275     const numericValue = parseFloat(formattedValue.replace(/^[^0-9.-]/g, ''));
276     return isNaN(numericValue) ? 0 : numericValue;
277 }
278 }
279

```