

# 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)

## Ø=ÜÁ shared\components

### Ø=ÜÁ 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  export { LoadingSpinnerComponent } from './loading-spinner/loading-spinner.component';
9  export { PlanBannerComponent } from './plan-banner/plan-banner.component';
10
```

## Ø=ÜÁ shared\components\birthday-input

### Ø=ÜÁ 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 onDateChange(): void {
136     if (this.dateValue.month && this.dateValue.day && this.dateValue.year) {
137         const dateString = `${this.dateValue.year}-${this.dateValue.month}-`  
 ${this.dateValue.day} dateString);
138         } else {
139             this.onChange('');
140         }
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   core
3 import { CommonModule } from '@angular/common';
4 import { FormsModule } from '@angular/forms';
5 import { TradeLocker ApiService } from '../../../../../services/tradelocker-api.service';
6 import { AuthService } from '../../../../../features/auth/service/authService';
7 import { AccountData } from '../../../../../features/auth/models/userModel';
8 import { Timestamp } from 'firebase/firestore';
9 import { NumberFormatterService } from '../../../../../utils/number-formatter.service';
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  * - TradeLocker ApiService: 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 tradeLocker ApiService: TradeLocker ApiService,
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
86    // Form data
87    newAccount = {
88      emailTradingAccount: '',
89      brokerPassword: '',
90      accountName: '',
91      broker: '',
92      server: '',
93      accountID: '',
94      initialBalance: 0,
95      accountNumber: 0,
96    };
97
98    // Properties for formatted balance input
99    initialBalanceInput: string = '';
100   initialBalanceDisplay: string = '';
101
102  // Confirmation modals
103  showCancelConfirm = false;
104  showSuccessModal = false;
105
106  onClose() {
107    this.close.emit();
108  }
109
110  onCancel() {
111    this.showCancelConfirm = true;
112  }
113
114  onConfirmCancel() {
115    this.showCancelConfirm = false;
116    this.cancel.emit();
117  }
118
119  onContinueEditing() {
120    this.showCancelConfirm = false;
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) {required fields.');
131  }
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
160 again.`);
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.format(this.newAccount.initialBalance).replace(',', '.');
280   } else {
281     this.initialBalanceInput = '';
282     this.initialBalanceDisplay = '';
283   }
284 }
285
286
287 private createAccountObject(): AccountData {
288   const timestamp = Date.now().toString(36);
289   const randomPart = Math.random().toString(36).substring(2, 8);
290   const uniqueId = `id_${timestamp}_${randomPart}`;
291
292   return {
293     id: uniqueId,
294     userId: this.userId,
295     emailTradingAccount: this.newAccount.emailTradingAccount,
296     brokerPassword: this.newAccount.brokerPassword,
297     broker: this.newAccount.broker,
298     server: this.newAccount.server,
299     accountName: this.newAccount.accountName,
300     accountId: this.newAccount.accountID,
301     accountNumber: this.newAccount.accountNumber,
302     initialBalance: this.newAccount.initialBalance,
303     createdAt: Timestamp.now(),
304     netPnl: 0,
305     profit: 0,
306     bestTrade: 0,
307   };
308 }
309
310 /**
311  * Validates if the account exists in TradeLocker
312  * Simply tries to get JWT token - if successful, credentials are valid
313  */
314 private async validateAccountInTradeLocker(): Promise<boolean> {
315   try {
316     return await this.tradeLocker ApiService.validateAccount({
317       email: this.newAccount.emailTradingAccount,
318       password: this.newAccount.brokerPassword,
319       server: this.newAccount.server
320     });
321   } catch (error) {
322     console.error('Error validating account in TradeLocker:', error);
323     return false;
324   }
325 }
326
327 /**
328  * Validates that broker + server + accountId combination is unique across all accounts
329  * Returns validation result with appropriate message
330  */
331 private async validateAccountUniqueness(): Promise<{isValid: boolean, message: string}> {
332   try {
333     // Check if broker + server + accountId combination already exists
334     const accountExists = await this.authService.checkAccountExists(
335       this.newAccount.broker,
336       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=ÜÀ shared\components\global-alert

### Ø=ÜÀ 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=ÜÁ 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

```

## Ø=ÜÁ shared\components\order-summary

### Ø=ÜÁ 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 }

```



## Ø=ÜÀ shared\components\password-input

### Ø=ÜÀ 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=ÜÀ shared\components\phone-input

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

```

```

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     showCountryDropdown: boolean = false;
66     touched: boolean = false;
67     searchTerm: string = '';
68     filteredCountries: CountryCode[] = [];
69     countries: CountryCode[] = [];
70     loading: boolean = true;
71
72   constructor(private countryService: CountryService) {}
73
74   ngOnInit(): void {
75     this.loadCountries();
76   }
77
78   private loadCountries(): void {
79     this.countryService.getCountry().subscribe({
80       next: (countries) => {
81         this.countries = countries;
82         this.filteredCountries = countries;
83         this.loading = false;
84
85         // Buscar Colombia por defecto, si no existe usar US
86         const colombia = countries.find(c => c.code === 'CO');
87         if (colombia) {
88           this.selectedCountry = colombia;
89         }
90       },
91       error: (error) => {
92         console.error('Error loading countries:', error);
93         this.loading = false;
94         // Mantener el país por defecto en caso de error
95       }
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

```

## Ø=ÜÀ shared\components\plan-banner

### Ø=ÜÀ 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

```

## Ø=ÜÀ shared\components\strategy-card

### Ø=ÜÀ 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   @Input() showCustomizeButton: boolean = true;
79   @Input() customizeButtonText: string = 'Customize strategy';
80
81   @Output() edit = new EventEmitter<string>();
82   @Output() favorite = new EventEmitter<string>();
83   @Output() moreOptions = new EventEmitter<string>();
84   @Output() customize = new EventEmitter<string>();
85   @Output() editStrategy = new EventEmitter<string>();
86   @Output() duplicate = new EventEmitter<string>();
87

```

```

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.id);
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 &amp; this.editingStrategyName.trim() ===
161             this.strategy.name) {
162             this.isSavingName = true;
163
164             try {
165                 // Actualizar en Firebase
166                 await this.updateStrategyNameInFirebase(newName);
167
168                 // Actualizar el nombre localmente
169                 this.strategy.name = newName;
170
171                 // Emitir evento para notificar al componente padre
172                 this.nameChanged.emit({
173                     id: this.strategy.id,
174                     newName: newName
175                 });
176             } catch (error) {
177                 // No actualizamos el nombre local si falla en Firebase
178                 console.error('Error updating the name of the strategy:', error);
179             } finally {
180                 this.isSavingName = false;
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 /**
286 * Calcula las reglas activas desde la colección configurations
287 */
288 private async calculateActiveRules(): Promise<void> {
289     if (!this.db || !this.strategy.configurationId) {
290         return;
291     }
292
293     try {
294         const configDoc = await getDoc(doc(this.db, 'configurations',
295         this.strategy.configurationId));
296         if (configDoc.exists()) {
297             const configData = configDoc.data();
298             let activeRulesCount = 0;

```

```

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

```

## Ø=ÜÄ 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

```

## Ø=ÜÁ shared\components\strategy-guide-modal

### Ø=ÜÄ 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(): number {
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=ÜÀ shared\components\subscription-processing

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

---

```

1  import { Component, Input, Output, EventEmitter, OnInit, OnDestroy, inject } from '@angular/core';
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'`;
169             return 'Your subscription has been activated successfully.';
170     }
171 }
172
173

```

## Ø=ÜÀ shared\components\text-input

### Ø=ÜÀ 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

```

## Ø=ÜÁ shared\context

### Ø=ÜÁ 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

```

## Ø=ÜÄ 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 { TradeLocker ApiService } 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 ApplicationContextState {
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  * - TradeLocker ApiService: 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 ApplicationContextService {
158       // Estado inicial
159       private initialState: ApplicationContextState = {
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<ApplicationContextState>(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: number } }>({ });
228
229  // Signals para datos de reportes
230  public reportData = signal<{
231    accountHistory: any[];
232    stats: any;
233    balanceData: any;
234    monthlyReports: any[];
235  }>({
236    accountHistory: [],
237    stats: null,
238    balanceData: null,
239    monthlyReports: []
240  });
241
242  // Signals para datos de overview
243  public overviewData = signal<{
244    allUsers: User[];
245    subscriptions: any[];
246    monthlyReports: any[];
247    allAccounts: AccountData[];
248    allStrategies: ConfigurationOverview[];
249  }>({
250    allUsers: [],
251    subscriptions: [],
252    monthlyReports: [],
253    allAccounts: [],
254    allStrategies: []
255  });
256
257  // Computed signals para datos derivados
258  public activeStrategies = computed(() =>
259    this.userStrategies().filter(strategy => strategy.status === true)
260  );
261
262  public totalAccounts = computed(() =>
263    this.userAccounts().length
264  );
265
266  public totalStrategies = computed(() =>
267    this.userStrategies().length
268  );
269
270  public canCreateAccount = computed(() => {
271    const plan = this.userPlan();
272    const currentCount = this.totalAccounts();
273    return plan ? currentCount < plan.maxAccounts : false;
274  });
275
276  public canCreateStrategy = computed(() => {
277    const plan = this.userPlan();
278    const currentCount = this.totalStrategies();
279    return plan ? currentCount < plan.maxStrategies : false;
280  });
281
282  public planLimitations = computed(() => ({
283    maxAccounts: this.userPlan()?.maxAccounts || 0,
284    currentAccounts: this.totalAccounts(),
285    maxStrategies: this.userPlan()?.maxStrategies || 0,
286    currentStrategies: this.totalStrategies(),
287    canCreateAccount: this.canCreateAccount(),
288    canCreateStrategy: this.canCreateStrategy()
289  }));
290
291  // 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: TradeLocker ApiService) {
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<AppState>): 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     {};
517     plans.forEach(plan => {
518         // Usar el nombre original del plan como clave
519         limits[plan.name] = {
520             tradingAccounts: plan.tradingAccounts,
521             strategies: plan.strategies
522         };
523     });
524     this.planLimits.set(limits);
525 }
526
527 getPlanByName(planName: string): Plan | undefined {
528     const plans = this.globalPlans();
529     return plans.find(plan => plan.name.toLowerCase() === planName.toLowerCase());
530 }
531
532 getPlanById(planId: string): Plan | undefined {
533     const plans = this.globalPlans();
534     return plans.find(plan => plan.id === planId);
535 }
536
537 getPlanLimits(planName: string): { tradingAccounts: number; strategies: number } | null {
538     const limits = this.planLimits();
539     // Buscar por nombre exacto primero
540     if (limits[planName]) {
541         return limits[planName];
542     }
543
544     // Si no se encuentra, buscar case-insensitive
545     const planKey = Object.keys(limits).find(key =>
546         key.toLowerCase() === planName.toLowerCase()
547     );
548
549     return planKey ? limits[planKey] : null;
550 }
551
552 // ===== MÉTODOS DE HISTORIAL DE PLUGINS =====
553
554 setPluginHistory(history: PluginHistoryData[]): void {
555     this.pluginHistory.set(history);
556 }
557
558 addPluginHistoryEntry(entry: PluginHistoryData): void {
559     const currentHistory = this.pluginHistory();
560     this.pluginHistory.set([entry, ...currentHistory]);
561 }
562
563 // ===== MÉTODOS DE CACHÉ DE API =====
564
565 setTradeLockerData(accountId: string, data: TradeLockerAccountData): void {
566     const currentState = this.stateSubject.value;
567     const newTradeLockerData = new Map(currentState.tradeLockerData);
568     newTradeLockerData.set(accountId, data);
569
570     this.updateState({
571         tradeLockerData: newTradeLockerData
572     });

```

```

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 ApplicationContextState['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 ApplicationContextState['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(): ApplicationContextState {
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 ApplicationContextState['isLoading'], maxAge: number = 5 * 60 * 1000); implementar lógica para verificar si los datos están obsoletos
700     // Esto se puede usar para decidir si hacer nuevas peticiones
701     return true; // Placeholder
702 }
703
704
705     // ===== MÉTODOS DE SUSCRIPCIÓN =====
706
707     subscribeToUserChanges(): Observable<User | null> {
708         return this.currentUser$;
709     }
710
711     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: number; strategies: string[]; members: string[] } }> {
733 }
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 currentDate = this.reportData();
744     this.reportData.set({ ...currentData, ...data });
745 }
746
747 updateReportStats(stats: any): void {
748     const currentDate = this.reportData();
749     this.reportData.set({ ...currentData, stats });
750 }
751
752 updateReportBalance(balanceData: any): void {
753     const currentDate = this.reportData();
754     this.reportData.set({ ...currentData, balanceData });
755 }
756
757 updateReportHistory(accountHistory: any[]): void {
758     const currentDate = 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 currentDate = this.overviewData();
772     this.overviewData.set({ ...currentData, ...data });
773 }
774
775 updateOverviewUsers(allUsers: User[]): void {
776     const currentDate = this.overviewData();
777     this.overviewData.set({ ...currentData, allUsers });
778 }
779
780 updateOverviewSubscriptions(subscriptions: any[]): void {
781     const currentDate = 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
918     return {
919         netPnl: Math.round((totalGains - totalLosses) * 100) / 100,
920         tradeWinPercent: Math.round(winPercent * 100) / 100,
921         profitFactor: Math.round(profitFactor * 100) / 100,
922         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       existingBalanceData: data.balanceData !== null && data.balanceData !== undefined
987         ? data.balanceData
988         : existingBalanceData,
989       lastUpdated: data.lastUpdated
990     };
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  * Cargar trading history desde localStorage por cuenta
999 */
1000 loadTradingHistoryFromLocalStorage(accountId: string): {
1001   accountHistory: GroupedTradeFinal[];
1002   stats: any;
1003   balanceData: any;
1004   lastUpdated: number;
1005 } | null {
1006   try {
1007     const key = `tradeSwitch_tradingHistory_${accountId}`;
1008     const data = localStorage.getItem(key);
1009     if (data) {
1010       return JSON.parse(data);
1011     }
1012   } catch (error) {
1013     console.error('Error loading trading history from localStorage:', error);
1014   }
1015   return null;
1016 }
1017 /**
1018  * Limpiar trading history de una cuenta específica
1019 */
1020 clearTradingHistoryForAccount(accountId: string): void {
1021   const currentState = this.stateSubject.value;
1022   const newTradingHistory = new Map(currentState.tradingHistoryByAccount);
1023   newTradingHistory.delete(accountId);
1024
1025   this.updateState({
1026     tradingHistoryByAccount: newTradingHistory
1027   });
1028
1029   // Limpiar también del localStorage
1030   try {
1031     const key = `tradeSwitch_tradingHistory_${accountId}`;
1032     localStorage.removeItem(key);
1033   } catch (error) {
1034     console.error('Error clearing trading history from localStorage:', error);
1035   }
1036 }
1037 /**
1038  * Cargar datos de reporte desde localStorage por accountID
1039 */
1040 loadReportDataFromLocalStorage(accountID: string): {
1041   tradingAccount: AccountData;
1042   accountHistory: GroupedTradeFinal[];
1043   stats: any;
1044   balanceData: any;
1045   lastUpdated: number;
1046 } | null {
1047   try {
1048     const key = `tradeSwitch_reportData_${accountID}`;
1049     const data = localStorage.getItem(key);
1050     if (data) {
1051       return JSON.parse(data);
1052     }
1053   } catch (error) {
1054     console.error('Error loading report data from localStorage:', error);
1055   }
1056   return null;
1057 }
1058
1059
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_${accountID}`;
1073
1074     // Cargar datos existentes para preservar balanceData si es null
1075     let existingBalanceData = null;
1076     try {
1077       const existingData = localStorage.getItem(key);
1078       if (existingData) {
1079         const parsed = JSON.parse(existingData);
1080         existingBalanceData = parsed.balanceData;
1081       }
1082     } catch (error) {
1083       // Si no hay datos existentes, continuar
1084     }
1085
1086     const dataToSave = {
1087       tradingAccount,
1088       accountHistory: reportData.accountHistory,
1089       stats: reportData.stats,
1090       // Solo sobrescribir balanceData si no es null, de lo contrario preservar el
1091       existingBalanceData: reportData.balanceData !== null && reportData.balanceData !== undefined
1092         ? reportData.balanceData
1093         : existingBalanceData,
1094       lastUpdated: reportData.lastUpdated
1095     };
1096
1097     localStorage.setItem(key, JSON.stringify(dataToSave));
1098   } catch (error) {
1099     console.error('Error saving report data to localStorage:', error);
1100   }
1101 }
1102
1103 /**
1104  * Limpiar datos de reporte de localStorage por accountID
1105 */
1106 clearReportDataFromLocalStorage(accountID: string): void {
1107   try {
1108     const key = `tradeSwitch_reportData_${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 { ApplicationContextService } 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

```

## Ø=ÜÄ 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

```

## Ø=ÜÄ 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 }
```

```

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

```

## Ø=ÜÁ shared\interfaces

### Ø=ÜÁ 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=ÜÁ 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.');
91         });
92     }
93 }
94

```

## Ø=ÜÁ shared\pipes

### Ø=ÜÁ 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

```

### Ø=ÜÁ 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

```

## Ø=ÜÄ 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=ÜÀ shared\pop-ups\alert-popup

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

```

## Ø=ÜÁ shared\pop-ups\confirm-pop-up

### Ø=ÜÁ 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  * >
30  * </app-confirm-popup>
31  *
32  * Relations:
33  * - Used by components that require user confirmation before actions
34  *
35  * @component
36  * @selector app-confirm-popup
37  * @standalone true
38  */
39 @Component({
40   selector: 'app-confirm-popup',
41   imports: [CommonModule],
42   templateUrl: './confirm-popup.component.html',
43   styleUrls: ['./confirm-popup.component.scss'],
44   standalone: true,
45 })

```

```

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=ÜÁ shared\pop-ups\edit-pop-up

### Ø=ÜÁ 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=ÜÄ 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=ÜÁ shared\pop-ups\loading-pop-up

### Ø=ÜÁ 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

```

## Ø=ÜÁ shared\pop-ups\stripe-loader-popup

### Ø=ÜÄ 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
```

## Ø=ÜÁ shared\services

### Ø=ÜÄ 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 !* use 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   const batch = this.db.batch();
81
82   // Delete user document
83   const userDocRef = this.db.doc(`users/${userId}`);
84   batch.delete(userDocRef);
85
86   // Delete user strategies
87   const strategiesRef = this.db.collection(`users/${userId}/strategies`);
88   batch.delete(strategiesRef);
89
90   // Delete monthly reports
91   const monthlyReportsRef = this.db.collection(`users/${userId}/monthly_reports`);
92   batch.delete(monthlyReportsRef);
93
94   // Delete plugin history
95   const pluginHistoryRef = this.db.collection(`users/${userId}/plugin_history`);
96   batch.delete(pluginHistoryRef);
97
98   // Delete tokens
99   const tokensRef = this.db.collection(`users/${userId}/tokens`);
100  batch.delete(tokensRef);
101
102  // Delete user subscriptions
103  const subscriptionsRef = this.db.collection(`users/${userId}/subscription`);
104  batch.delete(subscriptionsRef);
105
106  // Delete user document
107  const userDocRef = this.db.doc(`users/${userId}`);
108  batch.delete(userDocRef);
109
110  // Commit the batch
111  await batch.commit();
112
113  return true;
114}
```

```

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

```

```

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

```

```

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  import { AccountData } from '../../../../../features/auth/models/userModel';
4
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): Promise<boolean> {
105        if (!this.db) {
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    }

```

```

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 }

```

## Ø=ÜÄ 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     hideAlert(): void {
64       this.alertSubject.next({
65         visible: false,
66         config: null
67       });
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

```

## Ø=ÜÄ 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 { ApplicationContextService } 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 ApplicationContextService 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 ApplicationContextService 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  * - ApplicationContextService: 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 applicationContext: ApplicationContextService,
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.applicationContext.setUserPlan(null);
90         }
91       });
92     }
93   }
94 }

```

```

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('L Error cargando planes globales:', error);
132         }
133     }
134
135     private async updateUserPlanFromSubscription(subscription: Subscription | null): Promise<void> {
136         if (!subscription) {
137             this.appContext.setUserPlan(null);
138             return;
139         }
140
141         const status = subscription.status;
142
143         // Estado baneado: bloquear uso
144         if (status === UserStatus.BANNED) {
145             this.appContext.setUserPlan({
146                 planId: subscription.planId,
147                 planName: 'Banned',
148                 maxAccounts: 0,
149                 maxStrategies: 0,
150                 features: [],
151                 isActive: false,
152                 expiresAt: subscription.periodEnd ? (subscription.periodEnd as any).toMillis?.
153 () ?? undefinedtérminos y condiciones
154                 status: UserStatus.BANNED,
155                 price: '0'
156             } as any);
157             return;
158         }
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 // Firebase Auth primitives
200 getAuth() { return getAuth(); }
201 register(user: UserCredentials) { return createUserWithEmailAndPassword(getAuth(),
202   user.email, user.password); } return signInWithEmailAndPassword(getAuth(), user.email,
203   user.password);
204 signInWithGoogle() { const provider = new GoogleAuthProvider(); return
205   signInWithPopup(provider); } new OAuthProvider('apple.com'); return
206   signInWithPopup(getAuth(), provider); }
207
208 async sendPasswordReset(email: string): Promise<void> {
209   const { sendPasswordResetEmail } = await import('firebase/auth');
210   return sendPasswordResetEmail(getAuth(), email);
211 }
212
213 // Observabilidad de sesión
214 isAuthenticated(): Observable<boolean> {
215   return this.authStateSubject.asObservable().pipe(filter((state): state is boolean =>
216   state !== null));
217
218   getCurrentUser(): any {
219     try {
220       if (!this.isBrowser) return null;
221       return getAuth().currentUser;
222     } catch (error) {
223       console.error('Error obteniendo usuario actual:', error);
224       return null;
225     }
226   }
227
228   async getBearerTokenFirebase(userId: string): Promise<string> {
229     const token = await getAuth().currentUser?.getIdToken();
230     if (!token) throw new Error('Token not found');
231     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 getUserId(userId: string): Promise<User | null> {
258         return this.usersOperationsService.getUserId(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
267         this.accountsOperationsService.createLinkToken(token); }
268         this.accountsOperationsService.deleteLinkToken(token); }
269
270     // Accounts operations
271     async createAccount(account: AccountData) {
272         this.appContext.setLoading('accounts', true);
273         this.appContext.setError('accounts', null);
274         try {
275             await this.accountsOperationsService.createAccount(account);
276             this.appContext.addAccount(account);
277
278             // Actualizar conteos del usuario
279             if (account.userId) {
280                 await this.updateUserCounts(account.userId);
281             }
282
283             this.appContext.setLoading('accounts', false);
284         } catch (error) {
285             this.appContext.setLoading('accounts', false);
286             this.appContext.setError('accounts', 'Error al crear cuenta');
287             throw error;
288         }
289     }
290
291     async getUserAccounts(userId: string): Promise<AccountData[] | null> {
292         this.appContext.setLoading('accounts', true);
293         this.appContext.setError('accounts', null);
294         try {
295             const accounts = await this.accountsOperationsService.getUserAccounts(userId);
296             this.appContext.setUserAccounts(accounts || []);
297             this.appContext.setLoading('accounts', false);
298             return accounts;
299         } catch (error) {
300             this.appContext.setLoading('accounts', false);
301             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> checkEmailExists(string, currentUserId: string): Promise<boolean>
309     this.accountsOperationsService.deleteAccount(string, currentUserId: string, currentUserId:
310     string): Promise<boolean> deleteAccount(string, currentUserId: string, currentUserId: string);
311     this.accountsOperationsService.getStrategyViews(string, currentUserId: string): Promise<any>
312     this.accountsOperationsService.getAccount(string, currentUserId: string): Promise<AccountData> }
313
314     const userId = await this.accountsOperationsService.deleteAccount(accountId);
315
316     // Actualizar conteos del usuario después de eliminar la cuenta
317     if (userId) {
318       await this.updateUserCounts(userId);
319     }
320
321     // Verificar si un email de usuario ya está registrado
322     async getUserByEmail(email: string): Promise<User | null> {
323       try {
324         return await this.usersOperationsService.getUserByEmail(email);
325       } catch (error) {
326         console.error('Error checking if email exists:', error);
327         return null;
328       }
329     }
330
331     // Método para obtener datos del usuario para validaciones (cuentas y estrategias)
332     async getUserDataForValidation(userId: string): Promise<{
333       accounts: AccountData[];
334       strategies: any[];
335     }> {
336       try {
337         const [accounts, strategies] = await Promise.all([
338           this.accountsOperationsService.getUserAccounts(userId),
339           this.strategyOperationsService.getUserStrategyViews(userId)
340         ]);
341
342         return {
343           accounts: accounts || [],
344           strategies: strategies || []
345         };
346       } catch (error) {
347         console.error('Error getting user data for validation:', error);
348         return {
349           accounts: [],
350           strategies: []
351         };
352       }
353     }
354
355     /**
356      * Actualizar los conteos de trading_accounts y strategies del usuario
357      */
358     async updateUserCounts(userId: string): Promise<void> {
359       try {
360         const [tradingAccountsCount, strategiesCount] = await Promise.all([
361           this.accountsOperationsService.getAllLengthUserAccounts(userId),
362           this.strategyOperationsService.getAllLengthConfigurationsOverview(userId)
363         ]);
364
365         await this.updateUser(userId, {
366           trading_accounts: tradingAccountsCount,
367           strategies: strategiesCount
368         });
369       } catch (error) {
370         console.error('Error updating user counts:', error);
371       }
372     }

```

```
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=id,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.id && country.id.root && country.id.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.id.root, country.id.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 }

```

## Ø=ÜÄ shared\services\firebase-data.service.ts

---

```

1 import { getFirestore, collection, getDocs, doc, getDoc, setDoc, updateDoc, deleteDoc,
2 addDoc, query, where, orderBy, limit, startAt, endAt, getDatabase, firestore };
3 import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
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): Promise<any> {
160     if (!this.db) {
161       console.warn('Firestore not available in SSR');
162       return [];
163     }
164
165     try {
166       const q = query(
167         collection(this.db, collectionName),
168         where(field, operator, value)
169       );
170
171       const snapshot = await getDocs(q);
172       const documents: any[] = [];
173
174       snapshot.forEach((doc) => {
175         const data = doc.data();
176         (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}:`, error);
202   }
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:', error);
52     }
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

```

## Ø=ÜÄ 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  import { limitTo, orderBy, startAt, endAt } from 'firebase/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/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): Promise<MonthlyReport | null> {
160      const q = query(
161        collection(this.db, 'monthly_reports'),
162        where('userId', '==', userId),
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 * Delete monthly report
191 */
192 async deleteMonthlyReport(reportId: string): Promise<void> {
193     if (!this.db) {
194         console.warn('Firestore not available in SSR');
195         return;
196     }
197
198     try {
199         await deleteDoc(doc(this.db, 'monthly_reports', reportId));
200     } catch (error) {
201         console.error('Error deleting monthly report:', error);
202         throw error;
203     }
204 }
205
206 }
207

```

## Ø=ÜÄ 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   snapshotChanges } from '@angular/fire/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 {

```

```

95     const usersCol = collection(this.db, 'users');
96     let qRef: any = query(usersCol, orderBy('subscription_date', 'desc'), limit(pageSize));
97     if (startAfterDocId) {
98         const cursor = await getDoc(doc(this.db, 'users', startAfterDocId));
99         if (cursor.exists()) {
100             qRef = query(usersCol, orderBy('subscription_date', 'desc'), startAfter(cursor),
101 limit(pageSize));
102         }
103     const snapshot = await getDocs(qRef);
104     const lastDoc = snapshot.docs[snapshot.docs.length - 1];
105     return { docs: snapshot.docs, lastDocId: lastDoc?.id };
106 } catch (error) {
107     console.error('Error getting users page:', error);
108     return { docs: [], lastDocId: undefined };
109 }
110 }
111 /**
112 * Página de cuentas por usuario para la tabla (si se requiere desplegar cuentas)
113 */
114 async getUserAccountsPage(userId: string, pageSize: number, startAfterAccountId?: string) {
115     if (!this.db) {
116         console.warn('Firestore not available in SSR');
117         return { docs: [], lastDocId: undefined };
118     }
119     try {
120         const accountsCol = collection(this.db, 'accounts');
121         let qRef: any = query(
122             accountsCol,
123             where('userId', '==', userId),
124             orderBy('accountID', 'desc'),
125             limit(pageSize)
126         );
127         if (startAfterAccountId) {
128             const cursor = await getDoc(doc(this.db, 'accounts', startAfterAccountId));
129             if (cursor.exists()) {
130                 qRef = query(
131                     accountsCol,
132                     where('userId', '==', userId),
133                     orderBy('accountID', 'desc'),
134                     startAfter(cursor),
135                     limit(pageSize)
136                 );
137             }
138         }
139     }
140     const snapshot = await getDocs(qRef);
141     const lastDoc = snapshot.docs[snapshot.docs.length - 1];
142     return { docs: snapshot.docs, lastDocId: lastDoc?.id };
143 } catch (error) {
144     console.error('Error getting user accounts page:', error);
145     return { docs: [], lastDocId: undefined };
146 }
147 }
148 /**
149 * Get monthly reports data
150 */
151 async getMonthlyReportsData() {
152     if (!this.db) {
153         console.warn('Firestore not available in SSR');
154         return null;
155     }
156     try {
157         const snapshot = await getDocs(collection(this.db, 'monthly_reports'));
158         return snapshot;
159     } catch (error) {
160         console.error('Error getting monthly reports data:', error);
161         return null;
162     }
163 }
164

```

```

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

```

```

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

```

```

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 * UTILIDAD: Obtener el conteo total de planes
246 * @returns Promise con el número total de planes
247 */
248 async getPlansCount(): Promise<number> {
249     return getDocs(query(collection(db, 'plan')))
250         .then((snapshot) => snapshot.docs.length)
251         .catch((error) => {
252             return 0;
253         })
254     }
255 }
256 }

```

## Ø=ÜÄ 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 * Interface for plugin history data.
18 */
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 * Service for managing plugin activation history.
32 */
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 /**
39 * Features:
40 * - Get plugin usage history for a user
41 * - 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 * MÉTODO NUEVO: Determinar si el plugin está activo basándose en las fechas
109 * LÓGICA MEJORADA CON ZONA HORARIA:
110

```

```

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.timezoneService.convertToUTC(activeDates[activeDates.length - 1]);
140              this.timezoneService.convertToUTC(inactiveDates[inactiveDates.length - 1]);
141
142          // Si la última fecha de active es mayor que la de inactive, está activo
143          return lastActiveDate > lastInactiveDate;
144      }
145
146      // Si dateInactive tiene más elementos que dateActive, está inactivo
147      return false;
148  }
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              }
182          );
183      }
184  }

```

```

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

```

## Ø=ÜÄ 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>): Promise<void> {
83             if (!this.db) throw new Error('Firestore not available in SSR');
84             await updateDoc(doc(this.db, 'users', userId, 'reasons', reasonId), data as any);
85         }
86
87         async getOpenLatestReason(userId: string): Promise<BanReasonRecord | null> {
88             // Para evitar índices compuestos, ordenamos por dateBan y tomamos el más reciente
89             const colRef = this.reasonsCollectionPath(userId);
90             const qRef = query(colRef, orderBy('dateBan', 'desc'), limit(1));
91             const snapshot = await getDocs(qRef);
92             if (snapshot.empty) return null;
93             const docSnap = snapshot.docs[0];
94             const data = docSnap.data() as BanReasonRecord;
95             return { ...data, id: docSnap.id };
96         }
97     }
98
99
100

```

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

```

```

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:', error);
188    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 createdDate: Date;
199
200  // Handle different Firebase timestamp types
201  if (createdAt && typeof createdAt.toDate === 'function') {
202    // It's a Firebase Timestamp
203    createdDate = createdAt.toDate();
204  } else if (createdAt && createdAt.seconds) {
205    // It's an object with seconds
206    createdDate = new Date(createdAt.seconds * 1000);
207  } else if (createdAt instanceof Date) {
208    // Already a date
209    createdDate = createdAt;
210  } else if (typeof createdAt === 'string') {
211    // It's a date string
212    createdDate = 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(createdDate.getFullYear(), createdDate.getMonth(),
222  createdDate.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

```

## Ø=ÜÄ 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, addDoc } from '@angular/fire/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,
74             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         console.warn('Firestore not available in SSR');
116

```

```

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: StrategyState): Promise<void> {
148     console.warn('Firestore not available in SSR');
149     return;
150 }
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 any);
208     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, configurationId: string, shouldBeActive: boolean = false): Promise<string> {
219   if (!this.db) {
220     console.warn('Firestore not available in SSR');
221     throw new Error('Firestore not available');
222   }
223
224   try {
225     const now = new Date();
226     const overviewData: ConfigurationOverview = {
227       userId,
228       name,
229       status: shouldBeActive, // Activa solo si no hay otra activa
230       created_at: now,
231       updated_at: now,
232       days_active: 0,
233       configurationId,
234       dateActive: [now.toISOString()],
235     };
236
237     const docRef = await addDoc(collection(this.db, 'configuration-overview'), overviewData);
238     return docRef.id;
239   } catch (error) {
240     console.error('Error creating configuration overview:', error);
241     throw error;
242   }
243 }
244
245 /**
246 * Actualizar configuration por ID
247 */
248 async updateConfigurationById(configurationId: string, configuration: StrategyState): Promise<void> {
249   if (!this.db) {
250     console.warn('Firestore not available in SSR');
251     throw new Error('Firestore not available');
252   }
253
254   try {
255     const docRef = doc(this.db, 'configurations', configurationId);
256     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 * Obtener configuración por ID
264 */
265 async getConfigurationById(configurationId: string): Promise<StrategyState | null> {
266     if (!this.db) {
267         console.warn('Firestore not available in SSR');
268         return null;
269     }
270
271     try {
272         const docRef = doc(this.db, 'configurations', configurationId);
273         const docSnap = await getDoc(docRef);
274         if (docSnap.exists()) {
275             return docSnap.data() as StrategyState;
276         }
277         return null;
278     } catch (error) {
279         console.error('Error getting configuration by ID:', error);
280         return null;
281     }
282 }
283 }
284 /**
285 * Obtener configuración por configurationOverviewId (método legacy para compatibilidad)
286 */
287 async getConfigurationByOverviewId(overviewId: string): Promise<StrategyState | null> {
288     if (!this.db) {
289         console.warn('Firestore not available in SSR');
290         return null;
291     }
292
293     try {
294         // Buscar en configurations donde configurationOverviewId coincida
295         const q = query(
296             collection(this.db, 'configurations'),
297             where('configurationOverviewId', '==', overviewId),
298             limit(1)
299         );
300
301         const querySnapshot = await getDocs(q);
302         if (!querySnapshot.empty) {
303             const doc = querySnapshot.docs[0];
304             return doc.data() as StrategyState;
305         }
306
307         return null;
308     } catch (error) {
309         console.error('Error getting configuration by overview ID:', error);
310         return null;
311     }
312 }
313 }
314 /**
315 * Obtener todas las estrategias de un usuario
316 */
317 async getUserStrategyViews(userId: string): Promise<ConfigurationOverview[]> {
318     if (!this.db) {
319         console.warn('Firestore not available in SSR');
320         return [];
321     }
322
323     try {
324         // 1. Primero obtener todos los configuration-overview del usuario
325         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('L Error getting user strategies:', error);
359   return [];
360 }
361
362 /**
363 * Obtener configuración activa (método legacy para compatibilidad)
364 */
365 async getActiveConfiguration(userId: string): Promise<ConfigurationOverview | null> {
366   if (!this.db) {
367     console.warn('Firestore not available in SSR');
368     return null;
369   }
370
371   try {
372     // Buscar estrategia activa en configuration-overview
373     const q = query(
374       collection(this.db, 'configuration-overview'),
375       where('userId', '==', userId),
376       where('status', '==', true),
377       limit(1)
378     );
379
380     const querySnapshot = await getDocs(q);
381     if (!querySnapshot.empty) {
382       const doc = querySnapshot.docs[0];
383       const data = doc.data() as ConfigurationOverview;
384       (data as any).id = doc.id;
385       return data;
386     }
387
388     return null;
389   } catch (error) {
390     console.error('Error getting active configuration:', error);
391     return null;
392   }
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     console.warn('Firestore not available in SSR');
440     return;
441   }
442
443   try {
444     const strategyRef = doc(this.db, 'configuration-overview', strategyId);
445     const strategyDoc = await getDoc(strategyRef);
446
447     if (!strategyDoc.exists()) {
448       throw new Error('Strategy not found');
449     }
450
451     const currentDate = strategyDoc.data();
452     const updateData: any = {};
453
454     // Solo actualizar si se proporciona un valor válido
455     if (dateActive !== undefined && dateActive !== null) {
456       const currentDateActive = currentDate['dateActive'] || [];
457       const newDateActive = [...currentDateActive, dateActive.toISOString()];
458       updateData.dateActive = newDateActive;
459
460       // Si se está activando, cambiar status a true
461       updateData.status = true;
462     }
463
464     // Solo actualizar si se proporciona un valor válido
465     if (dateInactive !== undefined && dateInactive !== null) {
466       const currentDateInactive = currentDate['dateInactive'] || [];

```

```

467     const newDataInactive = [...currentDateInactive, dateInactive.toISOString()];
468     updateData.dateInactive = newDataInactive;
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 currentDate = 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 = currentDate['dateInactive'] || [];
553     const newDateInactive = [...currentDateInactive,
554       Timestamp.fromDate(strategy.dateInactive)];
555     updateData['dateInactive'] = newDateInactive;
556   }
557
558   await updateDoc(overviewDocRef, updateData);
559 }
560
561 // 3. Marcar configuration como deleted usando configurationId
562 if (strategy.configurationId) {
563   try {
564     const configDocRef = doc(this.db, 'configurations', strategy.configurationId);
565     await updateDoc(configDocRef, {
566       deleted: true,
567       deleted_at: Timestamp.now(),
568       updated_at: Timestamp.now()
569     });
570   } catch (error) {
571     console.warn('Configuration not found for soft deletion:', error);
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 están 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  * Generar ID único para configuration-overview
615  */
616 private generateOverviewId(): string {
617   return 'overview_' + Date.now() + '_' + Math.random().toString(36).substr(2, 9);
618 }
619 }
620 }
621

```

## Ø=ÜÄ shared\services\subscription-service.ts

---

```

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

```

```

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 {
133   try {
134     const paymentRef = doc(db, 'users', userId, 'subscription', paymentId);
135     const paymentSnap = await getDoc(paymentRef);
136
137     if (paymentSnap.exists()) {
138       return {
139         id: paymentSnap.id,
140         ...paymentSnap.data()
141       } as Subscription;
142     } else {
143       return null;
144     }
145   } catch (error) {
146     console.error('Error al obtener pago:', error);
147     throw error;
148   }
149 }
150
151 // TODO: IMPLEMENTAR ENDPOINT DE CREACIÓN DE PAGO - Reemplazar Firebase con API real
152 /**
153 * Crea un nuevo pago
154 * @param userId ID del usuario
155 * @param paymentData Datos del pago (sin id)
156 * @returns Promise con el ID del pago creado
157 */
158 async createSubscription(userId: string, paymentData: Omit<Subscription, 'id' | '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 * Actualiza un pago existente
177 * @param userId ID del usuario
178 * @param paymentId ID del pago
179 * @param updateData Datos a actualizar
180 * @returns Promise void
181 */
182 async updateSubscription(userId: string, paymentId: string, updateData: Partial<Omit<Subscription, 'id' | 'created_at' | 'updated_at'>>): Promise<void> {
183   const paymentRef = doc(db, 'users', userId, 'subscription', paymentId);
184   const updatePayload = {
185     ...updateData,
186     updated_at: Timestamp.now()
187   };
188 }

```

```

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

```

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

## Ø=ÜÄ 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('-05:00')) 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 }
```

```

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

```

```

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. El parámetro 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  * Obtener información de debug sobre zona horaria
273  * @param date Fecha a analizar
274  * @param userTimezone Zona horaria del usuario (opcional)
275  * @returns Información de debug
276 */
277 getTimezoneDebugInfo(date: Date | string, userTimezone?: string): any {
278   const timezone = userTimezone || this.getUserTimezone();
279
280   try {
281     const momentDate = moment.tz(date, timezone);
282     const utcDate = moment.utc(date);
283
284     return {
285       originalDate: date,
286       userTimezone: timezone,
287       localTime: momentDate.format('YYYY-MM-DD HH:mm:ss'),
288       utcTime: utcDate.format('YYYY-MM-DD HH:mm:ss'),
289       offset: momentDate.utcOffset(),
290       offsetString: momentDate.format('Z'),
291       dayOfWeek: momentDate.day(),
292       dayName: momentDate.format('dddd')
293     };
294   } catch (error) {
295     return {
296       error: error instanceof Error ? error.message : 'Unknown error',
297       originalDate: date,
298       userTimezone: timezone
299     };
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/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 TradeLocker ApiService {
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 ${accessToken}`
106    });
107
108    return this.http.post<any>(refreshUrl, { headers });
109  }
110
111  /**
112   * Validate account credentials in TradeLocker
113   */
114  async validateAccount(credentials: TradeLockerCredentials): Promise<boolean> {
115    try {
116      const tokenResponse = await this.getJWTToken(credentials).toPromise();
117      return !(tokenResponse && tokenResponse.accessToken);
118    } catch (error) {
119      console.error('Error validating account in TradeLocker:', error);
120      return false;
121    }
122  }
123
124  /**
125   * Get account balance from TradeLocker
126   */
127  getAccountBalance(accountId: string, userKey: string, accountNumber: number): Observable<any> {
128    const balanceUrl = `${this.baseUrl}/trade/accounts/${accountId}/state`;
129
130    const headers = new HttpHeaders({
131      'Content-Type': 'application/json',
132      'Authorization': `Bearer ${userKey}`,
133      'accNum': accountNumber.toString()
134    });
135
136    return this.http.get(balanceUrl, { headers });
137  }
138
139  /**
140   * Get trading history from TradeLocker
141   */
142  getTradingHistory(userKey: string, accountId: string, accNum: number): Observable<any> {
143    const historyUrl = `${this.baseUrl}/trade/accounts/${accountId}/ordersHistory`;
144
145    const headers = new HttpHeaders({
146      'Content-Type': 'application/json',
147      'Authorization': `Bearer ${userKey}`,
148      'accNum': accNum.toString()
149    });
150
151    return this.http.get(historyUrl, { headers });
152  }
153
154  /**
155   * 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    * Get instrument details
165    */
166   getInstrumentDetails(accessToken: string, tradableInstrumentId: string, routeId: string,
167   accNum: number): Observable<any> {
168     const instrumentsUrl = `${this.baseUrl}/trade/instruments/${tradableInstrumentId}`;
169
170     const headers = new HttpHeaders({
171       'Content-Type': 'application/json',
172       'Authorization': `Bearer ${accessToken}`,
173       'accNum': accNum.toString()
174     });
175
176     const params = {
177       routeId: routeId
178     };
179
180     return this.http.get(instrumentsUrl, { headers, params });
181   }
182
183 /**
184  * Get all instruments for an account
185 */
186 getAllInstruments(accessToken: string, accountId: string, accNum: number): Observable<any> {
187   const instrumentsUrl = `${this.baseUrl}/trade/accounts/${accountId}/instruments`;
188
189   const headers = new HttpHeaders({
190     'Content-Type': 'application/json',
191     'Authorization': `Bearer ${accessToken}`,
192     'accNum': accNum.toString()
193   });
194
195   return this.http.get(instrumentsUrl, { headers });
196 }
197
198 }
199

```

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

```

```

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=ÜÁ 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.');
155         });
156     }
157 }
158

```

## Ø=ÜÁ shared\utils

### Ø=ÜÁ 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

```