

TradeSwitch App

Features TypeScript Code

Generated: 11/25/2025, 6:38:23 PM

Total Files: 72

Table of Contents

1. features\account (1 files)
2. features\account\components\plan-management (1 files)
3. features\account\components\profile-details (1 files)
4. features\account\components\subscription-history (1 files)
5. features\account\mocks (1 files)
6. features\account\models (1 files)
7. features\add-account (1 files)
8. features\auth\login (1 files)
9. features\auth\models (2 files)

10. features\auth\service (1 files)
11. features\auth\signup (1 files)
12. features\auth\signup\components\plan-selection (1 files)
13. features\auth\store (3 files)
14. features\overview (1 files)
15. features\overview\components\top-list (1 files)
16. features\overview\components\tradeSwitch-table (1 files)
17. features\overview\models (1 files)
18. features\overview\services (1 files)
19. features\report (1 files)
20. features\report\components\calendar (1 files)
21. features\report\components\pnlGraph (1 files)
22. features\report\components\rule-short (1 files)
23. features\report\components\statCard (1 files)
24. features\report\components\trades-popup (1 files)
25. features\report\components\winLossChart (1 files)
26. features\report\models (1 files)
27. features\report\service (1 files)
28. features\report\store (3 files)
29. features\report\utils (2 files)
30. features\revenue (1 files)
31. features\revenue\components\orders-table (1 files)
32. features\revenue\components\revenue-table (1 files)
33. features\revenue\components\revenueGraph (1 files)
34. features\revenue\components\subscriptions-table (1 files)
35. features\revenue\mocks (1 files)
36. features\revenue\models (1 files)
37. features\strategy (1 files)
38. features\strategy\components\assets-allowed (1 files)
39. features\strategy\components\days-allowed (1 files)
40. features\strategy\components\hours-allowed (1 files)
41. features\strategy\components\max-daily-trades (1 files)
42. features\strategy\components\risk-per-trade (1 files)
43. features\strategy\components\risk-per-trade\models (1 files)
44. features\strategy\components\risk-reward (1 files)
45. features\strategy\edit-strategy (1 files)
46. features\strategy\edit-strategy\components\edit-assets-allowed (1 files)
47. features\strategy\edit-strategy\components\edit-days-allowed (1 files)
48. features\strategy\edit-strategy\components\edit-hours-allowed (1 files)
49. features\strategy\edit-strategy\components\edit-max-daily-trades (1 files)
50. features\strategy\edit-strategy\components\edit-risk-per-trade (1 files)
51. features\strategy\edit-strategy\components\edit-risk-per-trade\models (1 files)
52. features\strategy\edit-strategy\components\edit-risk-reward (1 files)
53. features\strategy\models (1 files)
54. features\strategy\service (1 files)
55. features\strategy\services (2 files)
56. features\strategy\store (3 files)

- 57. features\trading-accounts (1 files)
- 58. features\trading-accounts\components\accounts-table (1 files)
- 59. features\trading-accounts\components\show-confirmation (1 files)
- 60. features\users-details (1 files)
- 61. features\users-details\components\create-user-role-popup (1 files)
- 62. features\users-details\components\user-modal (1 files)
- 63. features\users-details\components\users-table (1 files)

Ø=ÜÄ features\account

Ø=ÜÄ features\account\account.component.ts

```
1 import { Component, OnInit } from '@angular/core';
2 import { User } from '../overview/models/overview';
3 import { Store } from '@ngrx/store';
4 import { SettingsService } from '../strategy/service/strategy.service';
5 import { ReportService } from '../report/service/report.service';
6 import { CommonModule } from '@angular/common';
7 import { PlanSettingsComponent } from './components/plan-management/plan-settings.component';
8 import { ProfileDetailsComponent } from './components/profile-details/profile-
9 details[subscriptionHistoryComponent } from './components/subscription-history/subscription-
10 history[details } from './models/account-settings';
11 import { MOCK_PLAN_DETAILS } from './mocks/account-mocks';
12 import { ApplicationContextService } from '../../../../../shared/context';
13 import { ActivatedRoute } from '@angular/router';
14
15 /**
16 * Main component of the user account module.
17 *
18 * This component acts as the main container that manages navigation between
19 * different sections of the user account:
20 * - Profile Details: Profile details and configuration
21 * - Plan Management: Plan and subscription management
22 *
23 * Related to:
24 * - ProfileDetailsComponent: Displays and allows editing of profile data
25 * - PlanSettingsComponent: Manages user subscription plans
26 * - ApplicationContextService: Gets current user data
27 * - ActivatedRoute: Reads URL parameters to select the correct tab
28 *
29 * @component
30 * @selector app-account
31 * @standalone true
32 */
33 @Component({
34   selector: 'app-account',
35   imports: [CommonModule, PlanSettingsComponent, ProfileDetailsComponent],
36   templateUrl: './account.component.html',
37   styleUrls: ['./account.component.scss'],
38   standalone: true,
39 })
40 export class AccountComponent implements OnInit {
41   /** Current user in the system */
42   user: User | null = null;
43
44   /** Index of the selected tab (0: Profile Details, 1: Plan Management) */
45   selectedIndex: number = 0;
46
47   /** Array of available tabs in the interface */
48   tabs: { label: string }[] = [
49     { label: 'Profile Details' },
50     { label: 'Plan Management' },
51     /*{ label: 'Subscription History' },*/
52   ];
53
54   /** Selected plan details (currently uses mock data) */
55   selectedPlanDetails: PlanDetails | null = null;
56
57   /**
58    * Constructor for AccountComponent
59    *
60    * @param store - NgRx Store to access global state
61    * @param strategySvc - Strategy service (injected but not currently used)
62    * @param reportSvc - Report service (injected but not currently used)
63    * @param applicationContext - Application context service to get user data
64    * @param route - ActivatedRoute to read URL parameters
65  }
```

```

65      */
66  constructor(
67    private store: Store,
68    private strategySvc: SettingsService,
69    private reportSvc: ReportService,
70    private appContext: ApplicationContextService,
71    private route: ActivatedRoute
72  ) {}
73
74 /**
75 * Initializes the component on load.
76 *
77 * Performs the following actions:
78 * 1. Loads plan details from mocks
79 * 2. Checks URL query parameters to select the correct tab
80 *     (if ?tab=plan exists, selects the Plan Management tab)
81 * 3. Subscribes to user changes from the application context
82 * 4. If no user is available, attempts to load it
83 *
84 * @memberof AccountComponent
85 */
86 ngOnInit(): void {
87   this.selectedPlanDetails = MOCK_PLAN_DETAILS;
88
89 // Verificar query parameters para seleccionar la pestaña correcta
90 this.route.queryParams.subscribe(params => {
91   if (params['tab'] === 'plan') {
92     this.selectedIndex = 1; // Plan Management tab
93   }
94 });
95
96 // Suscribirse a los datos del usuario desde el contexto
97 this.appContext.currentUser$.subscribe(user => {
98   this.user = user;
99 });
100
101 // Cargar datos del usuario si no están disponibles
102 if (!this.user) {
103   this.appContext.setLoading('user', true);
104   // Aquí se podría implementar la carga de datos del usuario si es necesario
105   this.appContext.setLoading('user', false);
106 }
107 }
108
109 /**
110 * Changes the selected tab in the interface.
111 *
112 * This method is executed when the user clicks on a different tab.
113 * Updates the selected index, which causes the corresponding component
114 * to be displayed in the template.
115 *
116 * @param index - Index of the tab to select (0: Profile Details, 1: Plan Management)
117 * @memberof AccountComponent
118 */
119 selectTypeData(index: number): void {
120   this.selectedIndex = index;
121 }
122 }
123

```

Ø=ÜÁ features\account\components\plan-management

Ø=ÜÄ features\account\components\plan-management\plan-settings.component.ts

```
1 import { Component, Input, OnInit, inject } from '@angular/core';
2
3 import { CommonModule } from '@angular/common';
4 import { Store } from '@ngrx/store';
5 import { SettingsService } from '../../../../../strategy/service/strategy.service';
6 import { ReportService } from '../../../../../report/service/report.service';
7 import { User } from '../../../../../overview/models/overview';
8 import { PlanCard, PlanDetails } from '../../../../../models/account-settings';
9 import { PLANS } from '../../../../../mocks/account-mocks';
10 import { Subscription, SubscriptionService } from '../../../../../shared/services/subscription-
11 Service';
12 import { PlanService } from '../../../../../shared/services/planService';
13 import { AuthService } from '../../../../../auth/service/authService';
14 import { Plan } from '../../../../../shared/services/planService';
15 import { selectUser } from '../../../../../auth/store/user.selectios';
16 import { SubscriptionProcessingComponent } from '../../../../../shared/components/subscription-
17 processing';
18 import { ApplicationContextService } from '../../../../../shared/context/context';
19 import { LoadingSpinnerComponent } from '../../../../../shared/components/loading-spinner/
20 loading';
21 stripe-loader-popup.component';
22 /**
23 * Component for managing user subscription plans.
24 *
25 * This component allows the user to:
26 * - View their current plan and renewal details
27 * - Compare and switch between different available plans
28 * - Validate downgrades before switching to a lower plan
29 * - Manage their subscription through the Stripe portal
30 * - Cancel their current plan
31 *
32 * Related to:
33 * - AccountComponent: Receives planDetails as Input
34 * - SubscriptionService: Gets and updates user subscriptions
35 * - PlanService: Gets information about available plans
36 * - AuthService: Gets authentication tokens for API calls
37 * - ApplicationContextService: Accesses global plans and user data
38 * - Stripe: Integration for checkout and subscription management portal
39 *
40 * Main flow:
41 * 1. On initialization, loads user plan and available plans
42 * 2. Builds plan cards from global context data
43 * 3. Calculates renewal dates and remaining days
44 * 4. Handles plan changes with downgrade validation
45 * 5. Integrates with Stripe for payments and subscription management
46 *
47 * @component
48 * @selector app-plan-settings
49 * @standalone true
50 */
51 @Component({
52   selector: 'app-plan-settings',
53   imports: [CommonModule, LoadingSpinnerComponent, StripeLoaderPopupComponent /StripeProcessingService],
54   styleUrl: './plan-settings.component.scss',
55   standalone: true,
56 })
57 export class PlanSettingsComponent implements OnInit {
58   /** Plan details received from parent component (AccountComponent) */
59   @Input() planDetails: PlanDetails | null = null;
60
61   /** Array of available plan cards to display in the interface */
62   plansData: PlanCard[] = [];
63
64
```

```

65  /** Current user plan obtained from the service */
66  userPlan: Plan | undefined = undefined;
67
68  /** Plan renewal date formatted as string */
69  renewalDate: string = '';
70
71  /** Remaining days until next renewal */
72  remainingDays: number = 0;
73
74  /** Flag to determine if user has free plan (shows N/A for renewal) */
75  isFreePlan: boolean = false;
76
77  // Estado de carga inicial
78  initialLoading: boolean = true;
79
80
81  user: User | null = null;
82  selectedIndex: number = 0;
83  tabs: { label: string }[] = [
84    { label: 'Profile Details' },
85    { label: 'Plan Management' },
86    { label: 'Billing Management' },
87  ];
88
89  // Estados para cancelar plan
90  showCancelPlanProcessing = false;
91
92  // Estados para validación de downgrade
93  showDowngradeValidation = false;
94  downgradeValidationData: {
95    targetPlan: string;
96    currentAccounts: number;
97    maxAccounts: number;
98    currentStrategies: number;
99    maxStrategies: number;
100   accountsToDelete: number;
101   strategiesToDelete: number;
102 } | null = null;
103
104 // Estados para pop-up de carga y error de redirección
105 showRedirectLoading = false;
106 showRedirectError = false;
107 redirectErrorMessage = '';
108 private windowCheckInterval: any = null;
109
110 // Inyectar servicios
111 private subscriptionService = inject(SubscriptionService);
112 private planService = inject(PlanService);
113 private authService = inject(AuthService);
114 private applicationContext = inject(AppContextService);
115
116 constructor(
117   private store: Store,
118   private strategySvc: SettingsService,
119   private reportSvc: ReportService
120 ) {}
121
122 /**
123  * Initializes the component on load.
124  *
125  * Performs the following actions in order:
126  * 1. Subscribes to changes in global plans from context
127  * 2. Attempts to build plan cards immediately
128  * 3. If no plans are loaded, loads them manually from PlanService
129  * 4. Loads current user plan from SubscriptionService
130  *
131  * @async
132  * @memberof PlanSettingsComponent
133  */
134 async ngOnInit(): Promise<void> {

```

```

135     this.initialLoading = true;
136
137     try {
138         // Suscribirse a cambios en los planes globales
139         this.appContext.subscribeToGlobalPlansChanges().subscribe(plans => {
140             if (plans.length > 0) {
141                 this.buildPlansData();
142             }
143         });
144
145         // También intentar construir inmediatamente por si ya están cargados
146         this.buildPlansData();
147
148         // Si no hay planes, intentar cargarlos manualmente
149         if (this.appContext.globalPlans().length === 0) {
150             await this.loadPlansManually();
151         }
152
153         await this.loadUserPlan();
154     } finally {
155         this.initialLoading = false;
156     }
157 }
158
159 /**
160 * Loads the current user plan from subscription.
161 *
162 * This method:
163 * 1. Gets the current user from the store
164 * 2. Finds the user's most recent subscription
165 * 3. Gets the plan associated with the subscription from PlanService
166 * 4. Calculates renewal date and remaining days
167 * 5. If there's no subscription or plan, sets default free plan
168 *
169 * Related to:
170 * - SubscriptionService.getUserLatestSubscription(): Gets user subscription
171 * - PlanService.getPlanById(): Gets plan details by ID
172 * - calculateRenewalDate(): Calculates renewal date
173 * - setDefaultFreePlan(): Sets free plan if there's no subscription
174 *
175 * @private
176 * @async
177 * @memberof PlanSettingsComponent
178 */
179 private async loadUserPlan(): Promise<void> {
180     try {
181         // Obtener el usuario actual
182         this.getUserData();
183         if (!this.user) {
184             console.error('L User not found');
185             return;
186         }
187
188         // Obtener la suscripción del usuario
189         const subscription = await
190         this.subscriptionService.getUserLatestSubscription(this.user.id);
191         // Buscar el plan por ID
192         const plan: Plan | undefined = await
193         this.planService.getPlanById(subscription.planId);
194         if (plan) {
195             this.userPlan = plan;
196             this.isFreePlan = plan.name.toLowerCase() === 'free';
197
198             // Usar periodEnd si existe, sino usar created_at
199             if (subscription.periodEnd) {
200                 this.calculateRenewalDate(subscription.periodEnd);
201             } else {
202                 this.calculateRenewalDate(subscription.created_at);
203             }
204         } else {

```

```

205         // Si no se encuentra el plan, usar plan gratuito por defecto
206         this.setDefaultFreePlan();
207     }
208     } else {
209         // Si no hay suscripción, usar plan gratuito por defecto
210         this.setDefaultFreePlan();
211     }
212   } catch (error) {
213     console.error('Error loading user plan:', error);
214     this.setDefaultFreePlan();
215   }
216 }
217
218 /**
219 * Builds the array of plan cards from global context plans.
220 *
221 * This method transforms plans obtained from ApplicationContextService into
222 * PlanCard objects used to display cards in the interface.
223 *
224 * For each plan:
225 * - Assigns price and period
226 * - Marks the second plan as "most popular"
227 * - Assigns icons and colors based on position
228 * - Builds the features array (trading accounts, strategies, etc.)
229 * - Defines the CTA button text
230 *
231 * Related to:
232 * - ApplicationContextService.orderedPlans(): Gets ordered plans
233 * - ApplicationContextService.getPlanLimits(): Gets limits for each plan
234 *
235 * @private
236 * @memberof PlanSettingsComponent
237 */
238 private buildPlansData(): void {
239   // Usar planes del contexto global
240   const orderedPlans = this.appContext.orderedPlans();
241
242   if (orderedPlans.length === 0) {
243     return;
244   }
245
246   // Construir plansData desde los planes ordenados del contexto
247   this.plansData = orderedPlans.map((plan, index) => ({
248     name: plan.name,
249     price: parseInt(plan.price) || 0,
250     period: '/month',
251     mostPopular: index === 1, // Marcar el segundo plan como más popular (Starter)
252     icon: index === 0 ? 'triangle' : index === 1 ? 'circle' : 'square',
253     color: index === 0 ? '#4b7ee8' : index === 1 ? '#4b7ee8' : '#d1ff81',
254     features: [
255       { label: 'Trading Accounts', value: plan.tradingAccounts.toString() },
256       { label: 'Strategies', value: plan.strategies.toString() },
257       { label: 'Consistency Rules', value: 'YES' },
258       { label: 'Trading Journal', value: 'YES' },
259       { label: 'Live Statistics', value: 'YES' }
260     ],
261     cta: `Get ${plan.name} Now`
262   }));
263 }
264
265 /**
266 * Loads plans manually from PlanService if they're not in context.
267 *
268 * This method runs as a fallback when plans are not available
269 * in the global context. Loads all plans from PlanService and
270 * sets them in the context for future use.
271 *
272 * Related to:
273 * - PlanService.getAllPlans(): Gets all available plans

```

```

275     * - ApplicationContextService.setGlobalPlans(): Sets plans in context
276     * - buildPlansData(): Builds cards after loading
277     *
278     * @private
279     * @async
280     * @memberof PlanSettingsComponent
281     */
282     private async loadPlansManually(): Promise<void> {
283         try {
284             const plans = await this.planService.getAllPlans();
285             this.appContext.setGlobalPlans(plans);
286             this.buildPlansData();
287         } catch (error) {
288             console.error('L Error cargando planes manualmente:', error);
289         }
290     }
291
292     /**
293      * Gets current user data from NgRx store.
294      *
295      * Subscribes to selectUser selector to get current user
296      * and update the component's user property.
297      *
298      * Related to:
299      * - Store.select(selectUser): NgRx selector to get user
300      *
301      * @private
302      * @memberof PlanSettingsComponent
303      */
304     private getUserData() {
305         this.store.select(selectUser).subscribe({
306             next: (user) => {
307                 this.user = user.user;
308             },
309             error: (err) => {
310                 console.error('Error fetching user data', err);
311             },
312         });
313     }
314
315     /**
316      * Sets the free plan as the user's default plan.
317      *
318      * This method runs when:
319      * - User doesn't have an active subscription
320      * - Plan associated with subscription cannot be found
321      * - An error occurs loading user plan
322      *
323      * Searches for "Free" plan in global context, or creates a default one
324      * if not available. Sets renewal date as "N/A"
325      * and remaining days to 0.
326      *
327      * Related to:
328      * - ApplicationContextService.getPlanByName(): Searches for Free plan in context
329      *
330      * @private
331      * @memberof PlanSettingsComponent
332      */
333     private setDefaultFreePlan(): void {
334         // Buscar el plan Free en los planes del contexto global
335         const freePlan = this.appContext.getPlanByName('Free');
336
337         if (freePlan) {
338             this.userPlan = freePlan;
339         } else {
340             // Fallback si no se encuentra el plan Free en el contexto
341             this.userPlan = {
342                 id: 'free',
343                 name: 'Free',
344                 price: '0',

```

```

345     strategies: 1,
346     tradingAccounts: 1,
347     createdAt: new Date(),
348     updatedAt: new Date()
349   );
350 }
351 this.isFreePlan = true;
352 this.renewalDate = 'N/A';
353 this.remainingDays = 0;
354 }
355 /**
356 * Calculates renewal date and remaining days until renewal.
357 *
358 * This method:
359 * 1. Checks if user has free plan (returns N/A if so)
360 * 2. Converts periodEnd to Date object (handles both Firebase Timestamp and Date)
361 * 3. Formats renewal date in readable format
362 * 4. Calculates remaining days from today until renewal date
363 * 5. If date has passed, sets days to 0
364 *
365 * Related to:
366 * - loadUserPlan(): Called after getting user subscription
367 *
368 * @private
369 * @param periodEnd - Subscription period end date (can be Firebase Timestamp or Date)
370 * @memberof PlanSettingsComponent
371 */
372 private calculateRenewalDate(periodEnd?: any): void {
373   // Si es plan Free, no calcular nada
374   if (this.isFreePlan) {
375     this.renewalDate = 'N/A';
376     this.remainingDays = 0;
377     return;
378   }
379
380   let renewalDate: Date;
381
382   if (periodEnd) {
383     // Usar periodEnd de la subscription
384     renewalDate = periodEnd.toDate() ? periodEnd.toDate() : new Date(periodEnd);
385   } else {
386     // Fallback: usar fecha actual
387     renewalDate = new Date();
388   }
389
390   // Formatear fecha de renovación
391   this.renewalDate = renewalDate.toLocaleDateString('en-US', {
392     year: 'numeric',
393     month: 'long',
394     day: 'numeric'
395   });
396
397   // Calcular días restantes desde hoy hasta la fecha de renovación
398   const today = new Date();
399   today.setHours(0, 0, 0, 0); // Reset horas para comparación precisa
400   renewalDate.setHours(0, 0, 0, 0);
401
402   const timeDiff = renewalDate.getTime() - today.getTime();
403   this.remainingDays = Math.ceil(timeDiff / (1000 * 3600 * 24));
404
405   // Si ya pasó la fecha de renovación, mostrar 0 días
406   if (this.remainingDays < 0) {
407     this.remainingDays = 0;
408   }
409 }
410
411 selectTypeData(index: number): void {
412   this.selectedIndex = index;
413 }
414

```

```

415
416  /**
417   * Gets capitalized current plan name.
418   *
419   * Helper to display plan name in readable format
420   * (first letter uppercase, rest lowercase).
421   *
422   * @returns Capitalized plan name or "Free Plan" if no plan
423   * @memberof PlanSettingsComponent
424   */
425   getCapitalizedPlanName(): string {
426     if (!this.userPlan?.name) return 'Free Plan';
427     const name = this.userPlan.name;
428     return name.charAt(0).toUpperCase() + name.slice(1).toLowerCase();
429   }
430
431  /**
432   * Capitalizes any plan name.
433   *
434   * Generic helper to format plan names
435   * (first letter uppercase, rest lowercase).
436   *
437   * @param planName - Plan name to capitalize
438   * @returns Capitalized plan name or empty string if not provided
439   * @memberof PlanSettingsComponent
440   */
441   capitalizePlanName(planName: string): string {
442     if (!planName) return '';
443     return planName.charAt(0).toUpperCase() + planName.slice(1).toLowerCase();
444   }
445
446  /**
447   * Checks if a plan is the user's current plan.
448   *
449   * Compares the provided plan name with the user's current
450   * plan name (case-insensitive comparison).
451   *
452   * @param planName - Name of plan to check
453   * @returns true if plan is current plan, false otherwise
454   * @memberof PlanSettingsComponent
455   */
456   isCurrentPlan(planName: string): boolean {
457     if (!this.userPlan) return false;
458     return this.userPlan.name.toLowerCase() === planName.toLowerCase();
459   }
460
461  /**
462   * Gets the number of trading accounts allowed for a plan.
463   *
464   * Searches for plan limits in global context and returns
465   * the number of allowed trading accounts.
466   *
467   * Related to:
468   * - ApplicationContextService.getPlanLimits(): Gets plan limits
469   *
470   * @param planName - Plan name
471   * @returns Number of trading accounts as string (default "1")
472   * @memberof PlanSettingsComponent
473   */
474   getTradingAccounts(planName: string): string {
475     // Usar datos del contexto global
476     const limits = this.appContext.getPlanLimits(planName);
477     return limits ? limits.tradingAccounts.toString() : '1';
478   }
479
480  /**
481   * Gets the number of strategies allowed for a plan.
482   *
483   * Searches for plan limits in global context and returns
484   * the number of allowed strategies.

```

```

485  *
486  * Related to:
487  * - ApplicationContextService.getPlanLimits(): Gets plan limits
488  *
489  * @param planName - Plan name
490  * @returns Number of strategies as string (default "1")
491  * @memberof PlanSettingsComponent
492  */
493 getStrategies(planName: string): string {
494     // Usar datos del contexto global
495     const limits = this.appContext.getPlanLimits(planName);
496     return limits ? limits.strategies.toString() : '1';
497 }
498 /**
499  * Gets button text based on plan status.
500  *
501  * Returns:
502  * - "Current plan" if plan is user's current plan
503  * - "Change plan" for all other cases
504  *
505  * @param planName - Plan name
506  * @returns Corresponding button text
507  * @memberof PlanSettingsComponent
508  */
509 getButtonText(planName: string): string {
510     // Verificar si el plan de la card es el plan actual del usuario
511     const currentPlanName = this.userPlan?.name.toLowerCase();
512     const cardPlanName = planName.toLowerCase();
513
514     // Si el plan de la card coincide con el plan actual del usuario
515     if (currentPlanName === cardPlanName) {
516         return 'Current plan';
517     }
518
519     // Para todos los demás casos
520     return 'Change plan';
521 }
522
523 /**
524  * Determines if a plan's button should be disabled.
525  *
526  * Button is disabled if:
527  * - Plan is user's current plan
528  *
529  * @param planName - Plan name
530  * @returns true if button should be disabled, false otherwise
531  * @memberof PlanSettingsComponent
532  */
533 isButtonDisabled(planName: string): boolean {
534     const isCurrentPlanFree = this.userPlan?.name.toLowerCase() === 'free';
535
536     // Solo deshabilitar el botón FREE cuando el usuario tiene plan FREE
537     if (this.userPlan?.name.toLowerCase() === planName.toLowerCase()) {
538         return true;
539     }
540
541     return false;
542 }
543
544 /**
545  * Handles plan change when user selects a new plan.
546  *
547  * This method is the main entry point for changing plans.
548  * Performs the following actions:
549  * 1. Checks if selected plan is current plan (does nothing if so)
550  * 2. Validates if it's a downgrade and checks if user has resources exceeding target plan
551  * 3. If downgrade and there are excess resources, shows validation modal
552  * 4. If current plan is FREE and target is also FREE, does nothing
553  * 5. If current plan is FREE, creates Stripe checkout session
554

```

```

555     * 6. If current plan is NOT FREE, opens Stripe portal for management
556     *
557     * Related to:
558     * - isCurrentPlan(): Checks if it's current plan
559     * - isDowngrade(): Determines if it's a downgrade
560     * - validateDowngrade(): Validates if downgrade is possible
561     * - createCheckoutSession(): Creates checkout session for paid plans
562     * - openStripePortal(): Opens Stripe portal for subscription management
563     *
564     * @async
565     * @param plan - Selected plan card
566     * @memberof PlanSettingsComponent
567     */
568     async onPlanChange(plan: PlanCard): Promise<void> {
569         try {
570             // Verificar si es el plan actual
571             if (this.isCurrentPlan(plan.name)) {
572                 return; // No hacer nada si es el plan actual
573             }
574
575             // Validar si es un downgrade y si el usuario tiene recursos que exceden el plan de
576             const isDowngrade = this.isDowngrade(plan.name);
577
578             if (isDowngrade) {
579                 const validationResult = await this.validateDowngrade(plan.name);
580
581                 if (!validationResult.canDowngrade) {
582                     this.showDowngradeValidationModal(validationResult);
583                     return;
584                 }
585             }
586
587             // Verificar si el plan actual es FREE
588             const isCurrentPlanFree = this.userPlan?.name.toLowerCase() === 'free';
589             const isTargetPlanFree = plan.name.toLowerCase() === 'free';
590
591             // Si el plan actual es FREE y el plan de destino también es FREE, no hacer nada
592             if (isCurrentPlanFree && isTargetPlanFree) {
593                 return; // No hacer nada si ambos son Free
594             }
595
596             // Si llegamos aquí, significa que puede hacer el cambio de plan
597             // Mostrar pop-up de carga solo para planes de pago
598             this.showRedirectLoading = true;
599
600             // Variable para controlar si hay error
601             let hasError = false;
602             let errorMessage = '';
603
604             try {
605                 if (isCurrentPlanFree) {
606                     // Si el plan actual es FREE y hace click en otro plan !' crear checkout session
607                     await this.createCheckoutSession(plan.name);
608                 } else {
609                     // Si el plan actual NO es FREE !' abrir portal de Stripe
610                     await this.openStripePortal();
611                 }
612             } catch (error) {
613                 // Marcar que hay error pero no mostrar pop-up aún
614                 hasError = true;
615                 errorMessage = 'Error redirecting to payment. Please try again.';
616                 console.error('Error during plan change:', error);
617             }
618
619             // Esperar mínimo 2 segundos antes de mostrar error o ocultar loader
620             setTimeout(() => {
621                 if (hasError) {
622                     // Si hay error, mostrar pop-up de error
623                     this.showRedirectLoading = false;
624                     this.showRedirectError = true;

```

```

625     this.redirectErrorMessage = errorMessage;
626
627     // Limpiar intervalo si existe
628     if (this.windowCheckInterval) {
629         clearInterval(this.windowCheckInterval);
630         this.windowCheckInterval = null;
631     }
632     }
633     // Si no hay error, el loader se ocultará automáticamente cuando se cierre la ventana
634     }, 2000);
635 } catch (error) {
636     console.error('Error processing plan change:', error);
637     // Eliminar el alert y manejar el error de forma más elegante
638     console.error('Error processing your request. Please try again.');
639 }
640 }
641 /**
642 * Creates a Stripe checkout session for a new plan.
643 *
644 * This method runs when user has FREE plan and wants to
645 * switch to a paid plan. Performs:
646 * 1. Gets complete plan from context to obtain planPriceId
647 * 2. Gets Firebase authentication token
648 * 3. Makes POST request to API to create checkout session
649 * 4. Redirects user to Stripe checkout URL
650 *
651 * Related to:
652 * - ApplicationContextService.getPlanByName(): Gets plan by name
653 * - AuthService.getBearerTokenFirebase(): Gets authentication token
654 * - API: https://api.tradeswitch.io/payments/create-checkout-session
655 *
656 * @private
657 * @async
658 * @param {string} planName - Selected plan name
659 * @throws {Error} if planPriceId is not found or session creation fails
660 * @memberof PlanSettingsComponent
661 */
662 private async createCheckoutSession(planName: string): Promise<void> {
663     try {
664         // Obtener el plan completo del contexto para obtener el priceId
665         const selectedPlan = this.appContext.getPlanByName(planName);
666
667         if (!selectedPlan || !selectedPlan.planPriceId) {
668             throw new Error('Plan price ID not found');
669         }
670
671         // Obtener el token de Firebase
672         const bearerTokenFirebase = await
673             this.authService.getBearerTokenFirebase(this.user?.id || '');
674
675         // Crear checkout session
676         const response = await fetch(`https://api.tradeswitch.io/payments/create-checkout-
677         session`{method: 'POST',
678             headers: {
679                 'Content-Type': 'application/json',
680                 'Authorization': `Bearer ${bearerTokenFirebase}`
681             },
682             body: JSON.stringify({
683                 priceId: selectedPlan.planPriceId,
684             })
685         });
686
687         if (!response.ok) {
688             const errorText = await response.text();
689             throw new Error(`Error creating checkout session: ${response.status}
690 ${response.statusText} - ${errorText}`);
691
692         const responseData = await response.json();
693         const checkoutUrl = responseData.body?.url || responseData.url;
694

```

```

695     if (!checkoutUrl) {
696         throw new Error('Checkout URL not found in response');
697     }
698
699     // Redirigir a la página de checkout
700     window.location.href = checkoutUrl;
701
702 } catch (error) {
703     console.error('Error creating checkout session:', error);
704     // No ocultar el loader aquí, dejar que el timeout de 2 segundos lo maneje
705     throw error;
706 }
707
708 /**
709 * Opens Stripe subscription management portal in a new window.
710 *
711 * This method runs when user has a paid plan and wants to
712 * manage their subscription. Performs:
713 * 1. Gets Firebase authentication token
714 * 2. Makes POST request to API to create portal session
715 * 3. Opens portal in a new window
716 * 4. Monitors if window closes to hide loader
717 * 5. Has a safety timeout of 8 seconds
718 *
719 * Related to:
720 * - AuthService.getBearerTokenFirebase(): Gets authentication token
721 * - API: https://api.tradeswitch.io/payments/create-portal-session
722 * - windowCheckInterval: Interval to check if window closed
723 *
724 * @private
725 * @async
726 * @throws Error if session creation fails or window cannot be opened
727 * @memberof PlanSettingsComponent
728 */
729
730 private async openStripePortal(): Promise<void> {
731     try {
732         const bearerTokenFirebase = await
733         this.authService.getBearerTokenFirebase(this.user?.id || '');
734         const response = await fetch('https://api.tradeswitch.io/payments/create-portal-
735         session', {
736             method: 'POST',
737             headers: {
738                 'Content-Type': 'application/json',
739                 'Authorization': `Bearer ${bearerTokenFirebase}`
740             },
741             body: JSON.stringify({
742                 userId: this.user?.id
743             })
744         });
745
746         if (!response.ok) {
747             throw new Error('Error creating portal session');
748         }
749
750         const responseData = await response.json();
751         const portalSessionUrl = responseData.body?.url || responseData.url;
752
753         if (!portalSessionUrl) {
754             throw new Error('Portal session URL not found in response');
755         }
756
757         // Abrir portal en nueva ventana
758         const newWindow = window.open(portalSessionUrl, '_blank');
759
760         // Verificar si la ventana se abrió correctamente
761         if (!newWindow || newWindow.closed || typeof newWindow.closed == 'undefined') {
762             throw new Error('Failed to open Stripe portal. Please check your pop-up blocker.');
763         }
764
765         // Verificar periódicamente si la ventana sigue abierta

```

```

765     this.windowCheckInterval = setInterval(() => {
766       if (newWindow.closed) {
767         // La ventana se cerró, ocultar loading
768         clearInterval(this.windowCheckInterval);
769         this.showRedirectLoading = false;
770       }
771     }, 500);
772 
773   // Timeout de seguridad: si después de 10 segundos no se ha cerrado la ventana,
774   // oculta el loading
775   setTimeout(() => {
776     if (this.windowCheckInterval) {
777       clearInterval(this.windowCheckInterval);
778     }
779     this.showRedirectLoading = false;
780   }, 8000);
781 
781 } catch (error) {
782   console.error('Error opening Stripe portal:', error);
783   // No ocultar el loader aquí, dejar que el timeout de 2 segundos lo maneje
784   throw error;
785 }
786 }
787 
788 /**
789 * Shows processing modal to cancel plan.
790 *
791 * This method runs when user clicks the cancel plan button.
792 * Shows a confirmation modal.
793 *
794 * @memberof PlanSettingsComponent
795 */
796 onCancelPlan(): void {
797   this.showCancelPlanProcessing = true;
798 }
799 
800 /**
801 * Confirms and executes user plan cancellation.
802 *
803 * This method:
804 * 1. Gets user's most recent subscription
805 * 2. Updates subscription with CANCELLED status and empty planId
806 * 3. Reloads user plan data
807 *
808 * Related to:
809 * - SubscriptionService.getUserLatestSubscription(): Gets subscription
810 * - SubscriptionService.updateSubscription(): Updates subscription
811 * - loadUserPlan(): Reloads plan after cancellation
812 *
813 * @async
814 * @memberof PlanSettingsComponent
815 */
816 async confirmCancelPlan(): Promise<void> {
817   if (!this.user) {
818     return;
819   }
820 
821   try {
822     // Obtener la suscripción actual del usuario
823     const subscriptions = await
824     this.subscriptionService.getUserLatestSubscription(this.user.id);
825     if (subscriptions) {
826       // Obtener la suscripción más reciente
827       const latestSubscription = subscriptions;
828 
829       // Actualizar la suscripción con status CANCELLED y planId vacío
830       await this.subscriptionService.updateSubscription(this.user.id,
831       latestSubscription.id, {
832         status: UserStatus.CANCELLED,
833         planId: ''
834       });

```

```

835         // Recargar los datos del usuario
836         await this.loadUserPlan();
837
838     } else {
839         console.error('No active subscription found to cancel.');
840     }
841
842 } catch (error) {
843     console.error('Error cancelling plan. Please try again.');
844 } finally {
845     this.showCancelPlanProcessing = false;
846 }
847 }
848
849 /**
850 * Cancels the plan cancellation process.
851 *
852 * Hides processing modal without performing any action.
853 *
854 * @memberof PlanSettingsComponent
855 */
856 cancelCancelPlan(): void {
857     this.showCancelPlanProcessing = false;
858 }
859
860 /**
861 * Opens Stripe portal to manage subscription.
862 *
863 * Similar to openStripePortal(), but runs from a specific
864 * "Manage Subscription" button. Opens portal in a new window.
865 *
866 * Related to:
867 * - AuthService.getBearerTokenFirebase(): Gets authentication token
868 * - API: https://api.tradeswitch.io/payments/create-portal-session
869 *
870 * @async
871 * @memberof PlanSettingsComponent
872 */
873 async onManageSubscription(): Promise<void> {
874     try {
875         const bearerTokenFirebase = await
876 this.authService.getBearerTokenFirebase(this.user?.id || '');
877         const response = await fetch('https://api.tradeswitch.io/payments/create-portal-
878 session' method: 'POST',
879         headers: {
880             'Content-Type': 'application/json',
881             'Authorization': `Bearer ${bearerTokenFirebase}`
882         },
883         body: JSON.stringify({
884             userId: this.user?.id
885         })
886     );
887
888     if (!response.ok) {
889         throw new Error('Error creating portal session');
890     }
891
892     const responseData = await response.json();
893     const portalSessionUrl = responseData.body?.url || responseData.url;
894
895     if (!portalSessionUrl) {
896         throw new Error('Portal session URL not found in response');
897     }
898
899     window.open(portalSessionUrl, '_blank');
900 } catch (error) {
901     console.error('Error opening Stripe portal:', error);
902     // Manejar el error de forma más elegante sin alert
903 }
904 }
```

```

905 /**
906  * Determines if a plan change is a downgrade (change to a lower plan).
907  *
908  * Compares current plan level with target plan level.
909  * Levels are: Free (1), Starter (2), Pro (3).
910  *
911  * Related to:
912  * - getPlanLevel(): Gets numeric level of a plan
913  *
914  * @private
915  * @param targetPlanName - Target plan name
916  * @returns true if it's a downgrade, false otherwise
917  * @memberof PlanSettingsComponent
918  */
919
920 private isDowngrade(targetPlanName: string): boolean {
921   if (!this.userPlan) return false;
922
923   const currentPlanLevel = this.getPlanLevel(this.userPlan.name);
924   const targetPlanLevel = this.getPlanLevel(targetPlanName);
925
926   return targetPlanLevel < currentPlanLevel;
927 }
928
929 /**
930  * Gets numeric level of a plan for comparison.
931  *
932  * Levels are:
933  * - Free: 1
934  * - Starter: 2
935  * - Pro: 3
936  *
937  * @private
938  * @param planName - Plan name
939  * @returns Numeric level of plan (default 1)
940  * @memberof PlanSettingsComponent
941  */
942 private getPlanLevel(planName: string): number {
943   const planLevels: { [key: string]: number } = {
944     'free': 1,
945     'starter': 2,
946     'pro': 3
947   };
948   return planLevels[planName.toLowerCase()] || 1;
949 }
950
951 /**
952  * Validates if user can downgrade to a specific plan.
953  *
954  * This method checks if user has resources (trading accounts
955  * or strategies) that exceed target plan limits. If so,
956  * downgrade is not allowed until user removes excess resources.
957  *
958  * Performs:
959  * 1. Gets target plan limits
960  * 2. Loads current user data (accounts and strategies)
961  * 3. Calculates how many resources must be deleted
962  * 4. Determines if downgrade is possible
963  *
964  * Related to:
965  * - getTradingAccounts(): Gets account limit of target plan
966  * - getStrategies(): Gets strategy limit of target plan
967  * - AuthService.getUserDataForValidation(): Gets current user data
968  *
969  * @private
970  * @async
971  * @param targetPlanName - Target plan name
972  * @returns Object with validation information (canDowngrade, resources to delete, etc.)
973  * @memberof PlanSettingsComponent
974  */

```

```

975 private async validateDowngrade(targetPlanName: string): Promise<{
976   canDowngrade: boolean;
977   targetPlan: string;
978   currentAccounts: number;
979   maxAccounts: number;
980   currentStrategies: number;
981   maxStrategies: number;
982   accountsToDelete: number;
983   strategiesToDelete: number;
984 }> {
985   if (!this.user?.id) {
986     throw new Error('User ID not available');
987   }
988
989   // Obtener límites del plan de destino usando la lógica existente
990   const targetMaxAccountsStr = this.getTradingAccounts(targetPlanName);
991   const targetMaxStrategiesStr = this.getStrategies(targetPlanName);
992
993   const targetMaxAccounts = parseInt(targetMaxAccountsStr);
994   const targetMaxStrategies = parseInt(targetMaxStrategiesStr);
995
996   // Cargar datos actuales del usuario directamente desde Firebase
997   const userData = await this.authService.getUserDataForValidation(this.user.id);
998   const currentAccounts = userData.accounts.length;
999   const currentStrategies = userData.strategies.length;
1000
1001   const accountsToDelete = Math.max(0, currentAccounts - targetMaxAccounts);
1002   const strategiesToDelete = Math.max(0, currentStrategies - targetMaxStrategies);
1003
1004   const canDowngrade = accountsToDelete === 0 && strategiesToDelete === 0;
1005
1006   return {
1007     canDowngrade,
1008     targetPlan: targetPlanName,
1009     currentAccounts,
1010     maxAccounts: targetMaxAccounts,
1011     currentStrategies,
1012     maxStrategies: targetMaxStrategies,
1013     accountsToDelete,
1014     strategiesToDelete
1015   };
1016 }
1017
1018 /**
1019  * Shows downgrade validation modal.
1020  *
1021  * Sets validation data and shows modal that informs
1022  * user about resources they must delete before downgrading.
1023  *
1024  * @private
1025  * @param validationData - Validation data obtained from validateDowngrade()
1026  * @memberof PlanSettingsComponent
1027  */
1028 private showDowngradeValidationModal(validationData: any): void {
1029   this.downgradeValidationData = validationData;
1030   this.showDowngradeValidation = true;
1031 }
1032
1033 /**
1034  * Closes downgrade validation modal.
1035  *
1036  * Hides modal and clears validation data.
1037  *
1038  * @memberof PlanSettingsComponent
1039  */
1040 closeDowngradeValidation(): void {
1041   this.showDowngradeValidation = false;
1042   this.downgradeValidationData = null;
1043 }
1044

```

```

1045     /**
1046      * Navigates to resource management pages.
1047      *
1048      * This method runs when user wants to delete resources
1049      * before downgrading. Currently only shows a console message.
1050      *
1051      * TODO: Implement real navigation to resource management pages.
1052      *
1053      * @memberof PlanSettingsComponent
1054      */
1055     goToManageResources(): void {
1056       this.showDowngradeValidation = false;
1057       this.downgradeValidationData = null;
1058
1059       // Navegar a las páginas de gestión de recursos
1060       // TODO: Implementar navegación a las páginas de gestión de recursos
1061       console.log('Please delete excess resources before downgrading your plan.');
1062     }
1063
1064     /**
1065      * Closes redirect error pop-up.
1066      *
1067      * Hides error pop-up, clears message and stops any
1068      * active window check interval.
1069      *
1070      * Related to:
1071      * - windowCheckInterval: Interval that checks if Stripe window closed
1072      *
1073      * @memberof PlanSettingsComponent
1074      */
1075     closeRedirectError(): void {
1076       this.showRedirectError = false;
1077       this.redirectErrorMessage = '';
1078
1079       // Limpiar intervalo si existe
1080       if (this.windowCheckInterval) {
1081         clearInterval(this.windowCheckInterval);
1082         this.windowCheckInterval = null;
1083       }
1084
1085       // Ocultar loading si está visible
1086       this.showRedirectLoading = false;
1087     }
1088   }

```

Ø=ÜÁ features\account\components\profile-details

Ø=ÜÁ features\account\components\profile-details\profile-details.component.ts

```

1 import { Component, OnInit, inject } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms';
4 import { Store } from '@ngrx/store';
5 import { User } from '../../../../../overview/models/overview';
6 import { selectUser } from '../../../../../auth/store/user.selectios';
7 import { AuthService } from '../../../../../auth/service/authService';
8 import { updatePassword, EmailAuthProvider, reauthenticateWithCredential, deleteUser } from
9 import { AccountDeletionService } from '../../../../../shared/services/account-
10 deletion.service';
11 import { ApplicationContextService } from '../../../../../shared/context';
12 import { PasswordInputComponent } from '../../../../../shared/components/password-input/
13 password-input.component';
14 /**

```

```

15 * Component for managing user profile details.
16 *
17 * This component allows the user to:
18 * - View and edit personal information (name, last name, email, phone, birthday)
19 * - Change password with validation and reauthentication
20 * - Delete account completely (Firebase data and authentication)
21 *
22 * Related to:
23 * - AccountComponent: Displayed in "Profile Details" tab
24 * - AuthService: Updates user data and gets current user
25 * - ApplicationContextService: Gets and updates user data in context
26 * - AccountDeletionService: Deletes all user data from Firebase
27 * - Store (NgRx): Updates user state in store
28 * - Firebase Auth: Reauthentication and password change
29 *
30 * Main flow:
31 * 1. On initialization, subscribes to user data from context
32 * 2. Populates form with user data
33 * 3. Allows updating profile (synchronizes with Firebase, context and store)
34 * 4. Allows changing password (requires reauthentication)
35 * 5. Allows deleting account (deletes Firebase data and authentication)
36 *
37 * @component
38 * @selector app-profile-details
39 * @standalone true
40 */
41 @Component({
42   selector: 'app-profile-details',
43   imports: [CommonModule, ReactiveFormsModule, PasswordInputComponent],
44   templateUrl: './profile-details.component.html',
45   styleUrls: ['./profile-details.component.scss'],
46   standalone: true,
47 })
48 export class ProfileDetailsComponent implements OnInit {
49   user: User | null = null;
50   profileForm: FormGroup;
51   passwordForm: FormGroup;
52   isLoading = false;
53   showPasswordForm = false;
54   passwordChangeMessage = '';
55   passwordChangeError = '';
56   showDeleteModal = false;
57   isDeletingAccount = false;
58   deleteAccountError = '';
59
60   // Inyectar servicios
61   private authService = inject(AuthService);
62   private fb = inject(FormBuilder);
63   private accountDeletionService = inject(AccountDeletionService);
64   private router = inject(Router);
65   private applicationContext = inject(ApplicationContextService);
66
67   constructor(private store: Store) {
68     this.profileForm = this.fb.group({
69       firstName: ['', [Validators.required, Validators.minLength(2)]],
70       lastName: ['', [Validators.required, Validators.minLength(2)]],
71       email: ['', [Validators.required, Validators.email]],
72       phoneNumber: '',
73       birthday: ['', [Validators.required]],
74     });
75
76     this.passwordForm = this.fb.group({
77       currentPassword: ['', [Validators.required]],
78       newPassword: ['', [Validators.required]], // Las validaciones específicas las maneja
79       componentPassword: ['', [Validators.required]],
80     }, { validators: this.passwordMatchValidator });
81   }
82
83   ngOnInit(): void {
84     this.subscribeToContextData();

```

```

85     }
86
87     /**
88      * Subscribes to user data from application context.
89      *
90      * When user data is received, populates the form
91      * with current information.
92      *
93      * Related to:
94      * - ApplicationContextService.currentUser$: Observable of current user
95      * - populateForm(): Populates form with user data
96      *
97      * @private
98      * @memberof ProfileDetailsComponent
99      */
100    private subscribeToContextData(): void {
101      // Suscribirse a los datos del usuario desde el contexto
102      this.appContext.currentUser$.subscribe({
103        next: (user) => {
104          this.user = user;
105          if (this.user) {
106            this.populateForm();
107          }
108        },
109        error: (err) => {
110          console.error('Error fetching user data from context', err);
111        },
112      });
113    }
114
115    /**
116     * Populates profile form with current user data.
117     *
118     * Updates all form fields with user values
119     * (firstName, lastName, email, phoneNumber, birthday).
120     *
121     * @private
122     * @memberof ProfileDetailsComponent
123     */
124    private populateForm(): void {
125      if (this.user) {
126        this.profileForm.patchValue({
127          firstName: this.user.firstName || '',
128          lastName: this.user.lastName || '',
129          email: this.user.email || '',
130          phoneNumber: this.user.phoneNumber || '',
131          birthday: this.user.birthday || '',
132        });
133      }
134    }
135
136    /**
137     * Custom validator to verify that passwords match.
138     *
139     * Compares newPassword and confirmPassword fields of the form.
140     * If they don't match, sets an error on confirmPassword field.
141     *
142     * @private
143     * @param form - FormGroup of password form
144     * @returns null if passwords match, error object if they don't match
145     * @memberof ProfileDetailsComponent
146     */
147    private passwordMatchValidator(form: FormGroup) {
148      const newPassword = form.get('newPassword');
149      const confirmPassword = form.get('confirmPassword');
150
151      if (newPassword && confirmPassword && newPassword.value !== confirmPassword.value) {
152        confirmPassword.setErrors({ passwordMismatch: true });
153        return { passwordMismatch: true };
154      }

```

```

155     return null;
156 }
158 /**
159 * Updates user profile with form data.
160 *
161 * This method synchronizes changes in three places:
162 * 1. Firebase: Updates data in database
163 * 2. ApplicationContextService: Updates user in context (source of truth)
164 * 3. Store (NgRx): Updates user state in store
165 *
166 * Only updates firstName, lastName and birthday (email cannot be changed here).
167 *
168 * Related to:
169 * - AuthService.updateUser(): Updates data in Firebase
170 * - ApplicationContextService.updateUserData(): Updates context
171 * - Store.dispatch(): Updates NgRx store
172 *
173 * @async
174 * @memberof ProfileDetailsComponent
175 */
176
177 async onUpdateProfile(): Promise<void> {
178   if (this.profileForm.valid && this.user) {
179     this.isLoading = true;
180     try {
181       const updatedData = {
182         firstName: this.profileForm.value.firstName,
183         lastName: this.profileForm.value.lastName,
184         birthday: this.profileForm.value.birthday,
185       };
186
187       // 1. Actualizar en Firebase
188       await this.authService.updateUser(this.user.id, updatedData);
189
190       // 2. Actualizar el usuario en el contexto (fuente de verdad)
191       this.appContext.updateUserData(updatedData);
192
193       // 3. Actualizar el usuario en el store
194       this.store.dispatch({
195         type: '[User] Update User',
196         user: {
197           ...this.user,
198           ...updatedData
199         }
200       });
201
202       console.log('' Profile updated successfully');
203     } catch (error) {
204       console.error(''L Error updating profile: '', error);
205     } finally {
206       this.isLoading = false;
207     }
208   }
209 }
210
211 /**
212 * Changes user password.
213 *
214 * This method requires reauthentication for security before changing password.
215 * Performs:
216 * 1. Verifies user is authenticated
217 * 2. Reauthenticates user with current password
218 * 3. Updates password in Firebase Auth
219 * 4. Resets form and hides password form
220 *
221 * Handles specific errors:
222 * - auth/wrong-password: Current password incorrect
223 * - auth/weak-password: New password too weak
224 * - auth/requires-recent-login: Requires signing in again

```

```

225  *
226  * Related to:
227  * - Firebase Auth: reauthenticateWithCredential, updatePassword
228  * - AuthService.getCurrentUser(): Gets current user from Firebase
229  *
230  * @async
231  * @memberof ProfileDetailsComponent
232  */
233  async onChangePassword(): Promise<void> {
234    if (this.passwordForm.valid && this.user) {
235      this.isLoading = true;
236      this.passwordChangeMessage = '';
237      this.passwordChangeError = '';
238
239      try {
240        const currentUser = this.authService.getCurrentUser();
241        if (!currentUser) {
242          throw new Error('User not authenticated');
243        }
244
245        const currentPassword = this.passwordForm.value.currentPassword;
246        const newPassword = this.passwordForm.value.newPassword;
247
248        // Reautenticar al usuario antes de cambiar la contraseña
249        const credential = EmailAuthProvider.credential(
250          this.user.email,
251          currentPassword
252        );
253
254        await reauthenticateWithCredential(currentUser, credential);
255
256        // Cambiar la contraseña
257        await updatePassword(currentUser, newPassword);
258
259        this.passwordChangeMessage = 'Password updated successfully';
260        this.passwordForm.reset();
261        this.showPasswordForm = false;
262
263      } catch (error: any) {
264        console.error('L Error changing password:', error);
265
266        if (error.code === 'auth/wrong-password') {
267          this.passwordChangeError = 'Current password is incorrect';
268        } else if (error.code === 'auth/weak-password') {
269          this.passwordChangeError = 'New password is too weak';
270        } else if (error.code === 'auth/requires-recent-login') {
271          this.passwordChangeError = 'For security, please sign in again';
272        } else {
273          this.passwordChangeError = 'Error changing password. Please try again';
274        }
275      } finally {
276        this.isLoading = false;
277      }
278    }
279  }
280
281 /**
282  * Shows or hides password change form.
283  *
284  * When hidden, resets form and clears success and error messages.
285  *
286  * @memberof ProfileDetailsComponent
287  */
288 togglePasswordForm(): void {
289  this.showPasswordForm = !this.showPasswordForm;
290  this.passwordChangeMessage = '';
291  this.passwordChangeError = '';
292  this.passwordForm.reset();
293}
294

```

```

295  /**
296   * Gets error message for a profile form field.
297   *
298   * Returns specific error messages based on error type:
299   * - required: Field required
300   * - email: Invalid email
301   * - minlength: Minimum length not reached
302   *
303   * Only shows errors if field has been touched.
304   *
305   * @param fieldName - Form field name
306   * @returns Error message or empty string if no error
307   * @memberof ProfileDetailsComponent
308   */
309  getFieldNameError(fieldName: string): string {
310    const field = this.profileForm.get(fieldName);
311    if (field?.errors && field.touched) {
312      if (field.errors['required']) {
313        return `${fieldName} is required`;
314      }
315      if (field.errors['email']) {
316        return 'Invalid email';
317      }
318      if (field.errors['minlength']) {
319        return `${fieldName} must have at least ${field.errors['minlength'].requiredLength} characters`;
320      }
321    }
322    return '';
323  }
324
325 /**
326  * Gets error message for a password form field.
327  *
328  * Returns specific error messages based on error type:
329  * - required: Field required
330  * - minlength: Password must have at least 6 characters
331  * - passwordMismatch: Passwords do not match
332  *
333  * Only shows errors if field has been touched.
334  *
335  * @param fieldName - Form field name
336  * @returns Error message or empty string if no error
337  * @memberof ProfileDetailsComponent
338  */
339  getPasswordFieldError(fieldName: string): string {
340    const field = this.passwordForm.get(fieldName);
341    if (field?.errors && field.touched) {
342      if (field.errors['required']) {
343        return `${fieldName} is required`;
344      }
345      if (field.errors['minlength']) {
346        return 'Password must have at least 6 characters';
347      }
348      if (field.errors['passwordMismatch']) {
349        return 'Passwords do not match';
350      }
351    }
352    return '';
353  }
354
355 /**
356  * Shows confirmation modal to delete account.
357  *
358  * This method opens the modal that requests confirmation before
359  * proceeding with account deletion.
360  *
361  * @memberof ProfileDetailsComponent
362  */
363  showDeleteAccountModal(): void {
364    this.showDeleteModal = true;

```

```

365     this.deleteAccountError = '';
366 }
367
368 /**
369 * Cancels account deletion process.
370 *
371 * Hides confirmation modal without performing any action.
372 *
373 * @memberof ProfileDetailsComponent
374 */
375 cancelDeleteAccount(): void {
376     this.showDeleteModal = false;
377     this.deleteAccountError = '';
378 }
379
380 /**
381 * Deletes user account and all associated data.
382 *
383 * This method performs complete deletion in the following order:
384 * 1. Deletes all user data from Firebase (AccountDeletionService)
385 * 2. Deletes user from Firebase Authentication
386 * 3. Clears local NgRx store
387 * 4. Redirects user to login page
388 *
389 * Handles specific errors:
390 * - auth/requires-recent-login: Requires signing in again for security
391 * - auth/too-many-requests: Too many attempts, try again later
392 *
393 * Related to:
394 * - AccountDeletionService.deleteUserData(): Deletes Firebase data
395 * - Firebase Auth deleteUser(): Deletes user from authentication
396 * - Store.dispatch(): Clears user state
397 * - Router.navigate(): Redirects to login
398 *
399 * @async
400 * @memberof ProfileDetailsComponent
401 */
402 async confirmDeleteAccount(): Promise<void> {
403     if (!this.user) {
404         this.deleteAccountError = 'User not found';
405         return;
406     }
407
408     this.isDeletingAccount = true;
409     this.deleteAccountError = '';
410
411     try {
412
413         // 1. Delete all Firebase data
414         const firebaseDataDeleted: boolean = await
415         this.accountDeletionService.deleteUserData(this.user.id);
416         if (!firebaseDataDeleted) {
417             throw new Error('Error deleting Firebase data');
418         }
419
420         // 2. Delete user from Firebase Auth
421         const currentUser = this.authService.getCurrentUser();
422         if (currentUser) {
423             await deleteUser(currentUser);
424         }
425
426         // 3. Clear local store
427         this.store.dispatch({
428             type: '[User] Clear User'
429         });
430
431         // 4. Redirect to login
432         this.router.navigate(['/login']);
433
434     } catch (error: any) {

```

```

435     console.error('L Error deleting account:', error);
436
437     if (error.code === 'auth/requires-recent-login') {
438         this.deleteAccountError = 'For security, you need to sign in again before deleting
439         your account'if (error.code === 'auth/too-many-requests') {
440             this.deleteAccountError = 'Too many attempts. Please try again later';
441         } else {
442             this.deleteAccountError = 'Error deleting account. Please try again';
443         }
444     } finally {
445         this.isDeletingAccount = false;
446     }
447 }
448 }
449

```

Ø=ÜÁ features\account\components\subscription-history

Ø=ÜÁ features\account\components\subscription-history\subscription-history.component.ts

```

1 import { Component, OnInit, inject } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { FormsModule } from '@angular/forms';
4 import { Store } from '@ngrx/store';
5 import { User } from '../../../../../overview/models/overview';
6 import { selectUser } from '../../../../../auth/store/user.selectios';
7 import { SubscriptionService } from '../../../../../shared/services/subscription-service';
8 import { PlanService } from '../../../../../shared/services/planService';
9 import { Subscription } from '../../../../../shared/services/subscription-service';
10 import { Plan } from '../../../../../shared/services/planService';
11 import { UserStatus } from '../../../../../overview/models/overview';
12
13 /**
14 * Component to display user subscription history.
15 *
16 * This component allows the user to:
17 * - View subscription history
18 * - Filter subscriptions by plan, date and search term
19 * - Paginate results
20 * - View details of each subscription (plan, status, date, amount)
21 *
22 * Related to:
23 * - AccountComponent: Displayed in "Subscription History" tab (currently commented)
24 * - SubscriptionService: Gets user subscriptions
25 * - PlanService: Gets plan information to display names
26 * - Store (NgRx): Gets current user
27 *
28 * Features:
29 * - Search by subscription ID
30 * - Filters by plan and date range
31 * - Result pagination
32 * - Date and currency formatting
33 * - Mapping subscription statuses to CSS classes and readable text
34 *
35 * NOTE: This component is currently commented in AccountComponent template.
36 *
37 * @component
38 * @selector app-subscription-history
39 * @standalone true
40 */
41 @Component({
42   selector: 'app-subscription-history',
43   imports: [CommonModule, FormsModule],

```

```

44     templateUrl: './subscription-history.component.html',
45     styleUrls: ['./subscription-history.component.scss'],
46     standalone: true,
47   })
48   export class SubscriptionHistoryComponent implements OnInit {
49     user: User | null = null;
50     subscriptions: Subscription[] = [];
51     filteredSubscriptions: Subscription[] = [];
52     plans: Plan[] = [];
53     plansMap: { [key: string]: Plan } = {};
54
55     // Pagination
56     currentPage = 1;
57     itemsPerPage = 10;
58     totalPages = 0;
59
60     // Filters
61     searchTerm = '';
62     showFilter = false;
63     selectedPlan = '';
64     dateFrom = '';
65     dateTo = '';
66
67     planNames: string[] = [];
68
69     // Loading state
70     isLoading = false;
71
72     // Inyectar servicios
73     private subscriptionService = inject(SubscriptionService);
74     private planService = inject(PlanService);
75
76     constructor(private store: Store) {}
77
78     async ngOnInit(): Promise<void> {
79       this.getUserData();
80       await this.loadPlans();
81     }
82
83     /**
84      * Gets current user data from NgRx store.
85      *
86      * Subscribes to selectUser selector and when user is obtained,
87      * loads user subscriptions.
88      *
89      * Related to:
90      * - Store.select(selectUser): NgRx selector to get user
91      * - loadPayments(): Loads subscriptions when user exists
92      *
93      * @private
94      * @memberof SubscriptionHistoryComponent
95      */
96     private getUserData(): void {
97       this.store.select(selectUser).subscribe({
98         next: (userData) => {
99           this.user = userData.user;
100          if (this.user) {
101            this.loadPayments();
102          }
103        },
104        error: (err) => {
105          console.error('Error fetching user data', err);
106        },
107      });
108    }
109
110   /**
111    * Loads all available plans from PlanService.
112    *
113    * This method:

```

```

114     * 1. Gets all plans from PlanService
115     * 2. Creates an array of plan names for filters
116     * 3. Creates a plan map (by ID) for quick lookup
117     *
118     * Related to:
119     * - PlanService.getAllPlans(): Gets all plans
120     * - getPlanName(): Uses plansMap to get plan names
121     *
122     * @private
123     * @async
124     * @memberof SubscriptionHistoryComponent
125     */
126     private async loadPlans(): Promise<void> {
127         try {
128             this.plans = await this.planService.getAllPlans();
129             this.planNames = this.plans.map(plan => plan.name);
130
131             // Crear mapa de planes para búsqueda rápida por ID
132             this.plansMap = {};
133             this.plans.forEach(plan => {
134                 this.plansMap[plan.id] = plan;
135             });
136         } catch (error) {
137             console.error('Error loading plans:', error);
138         }
139     }
140
141     /**
142      * Loads user subscriptions from SubscriptionService.
143      *
144      * Gets user's most recent subscription and displays it
145      * in the list. Currently only loads one subscription (the most recent).
146      *
147      * Related to:
148      * - SubscriptionService.getUserLatestSubscription(): Gets most recent subscription
149      * - calculatePagination(): Calculates pagination after loading
150      *
151      * @private
152      * @async
153      * @memberof SubscriptionHistoryComponent
154      */
155     private async loadPayments(): Promise<void> {
156         if (!this.user) return;
157
158         this.isLoading = true;
159         try {
160             const latest = await this.subscriptionService.getUserLatestSubscription(this.user.id);
161             this.subscriptions = latest ? [latest] : [];
162             this.filteredSubscriptions = [...this.subscriptions];
163             this.calculatePagination();
164         } catch (error) {
165             console.error('Error loading subscriptions:', error);
166         } finally {
167             this.isLoading = false;
168         }
169     }
170
171     /**
172      * Calculates total number of pages based on filtered results.
173      *
174      * If current page is greater than total pages,
175      * resets to page 1.
176      *
177      * @private
178      * @memberof SubscriptionHistoryComponent
179      */
180     private calculatePagination(): void {
181         this.totalPages = Math.ceil(this.filteredSubscriptions.length / this.itemsPerPage);
182         if (this.currentPage > this.totalPages) {
183             this.currentPage = 1;

```

```

184     }
185   }
186
187   /**
188    * Getter that returns paginated subscriptions for current page.
189    *
190    * Calculates range of subscriptions to display based on:
191    * - currentPage: Current page
192    * - itemsPerPage: Number of items per page
193    *
194    * @returns Array of subscriptions for current page
195    * @memberof SubscriptionHistoryComponent
196    */
197   get paginatedSubscriptions(): Subscription[] {
198     const startIndex = (this.currentPage - 1) * this.itemsPerPage;
199     const endIndex = startIndex + this.itemsPerPage;
200     return this.filteredSubscriptions.slice(startIndex, endIndex);
201   }
202
203   /**
204    * Executes when search term changes.
205    *
206    * Applies current filters (including new search term).
207    *
208    * @memberof SubscriptionHistoryComponent
209    */
210   onSearchChange(): void {
211     this.applyFilters();
212   }
213
214   /**
215    * Shows or hides filter panel.
216    *
217    * @memberof SubscriptionHistoryComponent
218    */
219   openFilter(): void {
220     this.showFilter = !this.showFilter;
221   }
222
223   /**
224    * Applies all active filters to subscriptions.
225    *
226    * This method filters subscriptions according to:
227    * - Search term: Searches in subscription ID
228    * - Date from: Filters subscriptions from a specific date
229    * - Date to: Filters subscriptions until a specific date
230    *
231    * After filtering, recalculates pagination and resets to page 1.
232    *
233    * @memberof SubscriptionHistoryComponent
234    */
235   applyFilters(): void {
236     let filtered = [...this.subscriptions];
237
238     // Search filter
239     if (this.searchTerm) {
240       filtered = filtered.filter(subscription =>
241         subscription.id?.toLowerCase().includes(this.searchTerm.toLowerCase())
242       );
243     }
244
245     // Date filters
246     if (this.dateFrom) {
247       const fromDate = new Date(this.dateFrom);
248       filtered = filtered.filter(subscription => {
249         const subscriptionDate = subscription.created_at.toDate();
250         return subscriptionDate >= fromDate;
251       });
252     }
253

```

```

254     if (this.dateTo) {
255         const toDate = new Date(this.dateTo);
256         toDate.setHours(23, 59, 59, 999); // End of day
257         filtered = filtered.filter(subscription => {
258             const subscriptionDate = subscription.created_at.toDate();
259             return subscriptionDate <= toDate;
260         });
261     }
262
263     this.filteredSubscriptions = filtered;
264     this.calculatePagination();
265     this.currentPage = 1;
266 }
267
268 /**
269 * Clears all active filters.
270 *
271 * Resets all filter fields (search, plan, dates)
272 * and reapplies filters (showing all results).
273 *
274 * @memberof SubscriptionHistoryComponent
275 */
276 clearFilters(): void {
277     this.searchTerm = '';
278     this.selectedPlan = '';
279     this.dateFrom = '';
280     this.dateTo = '';
281     this.applyFilters();
282 }
283
284 /**
285 * Navigates to previous page.
286 *
287 * Only navigates if there's a previous page (currentPage > 1).
288 *
289 * @memberof SubscriptionHistoryComponent
290 */
291 prevPage(): void {
292     if (this.currentPage > 1) {
293         this.currentPage--;
294     }
295 }
296
297 /**
298 * Navigates to next page.
299 *
300 * Only navigates if there's a next page (currentPage < totalPages).
301 *
302 * @memberof SubscriptionHistoryComponent
303 */
304 nextPage(): void {
305     if (this.currentPage < this.totalPages) {
306         this.currentPage++;
307     }
308 }
309
310 /**
311 * Navigates directly to a specific page.
312 *
313 * Validates that page is within valid range (1 to totalPages).
314 *
315 * @param page - Page number to navigate to
316 * @memberof SubscriptionHistoryComponent
317 */
318 goToPage(page: number): void {
319     if (page >= 1 && page <= this.totalPages) {
320         this.currentPage = page;
321     }
322 }
323

```

```

324  /**
325   * Gets plan name by its ID.
326   *
327   * Searches for plan in plan map (plansMap) and returns its name.
328   * If not found, returns "Unknown Plan".
329   *
330   * Related to:
331   * - plansMap: Plan map created in loadPlans()
332   *
333   * @param planId - Plan ID
334   * @returns Plan name or "Unknown Plan" if not found
335   * @memberof SubscriptionHistoryComponent
336   */
337  getPlanName(planId: string): string {
338    const plan = this.plansMap[planId];
339    return plan ? plan.name : 'Unknown Plan';
340  }
341
342
343
344 /**
345  * Converts subscription status to readable text for display.
346  *
347  * Maps UserStatus states to more friendly text:
348  * - PURCHASED !' "Completed"
349  * - PENDING !' "Pending"
350  * - PROCESSING !' "Processing"
351  * - CANCELLED !' "Cancelled"
352  * - EXPIRED !' "Expired"
353  * - BANNED !' "Banned"
354  * - ADMIN !' "Admin"
355  * - CREATED !' "Created"
356  *
357  * @param status - Subscription status (UserStatus)
358  * @returns Readable status text
359  * @memberof SubscriptionHistoryComponent
360  */
361  getStatusDisplay(status: string): string {
362    const statusMap: { [key: string]: string } = {
363      [UserStatus.PURCHASED]: 'Completed',
364      [UserStatus.PENDING]: 'Pending',
365      [UserStatus.PROCESSING]: 'Processing',
366      [UserStatus.CANCELLED]: 'Cancelled',
367      [UserStatus.EXPIRED]: 'Expired',
368      [UserStatus.BANNED]: 'Banned',
369      [UserStatus.ADMIN]: 'Admin',
370      [UserStatus.CREATED]: 'Created'
371    };
372    return statusMap[status] || status;
373  }
374
375 /**
376  * Gets CSS class corresponding to subscription status.
377  *
378  * Maps UserStatus states to CSS classes for styling:
379  * - PURCHASED !' "completed"
380  * - PENDING !' "pending"
381  * - PROCESSING !' "processing"
382  * - CANCELLED !' "cancelled"
383  * - EXPIRED !' "expired"
384  * - BANNED !' "banned"
385  * - ADMIN !' "admin"
386  * - CREATED !' "created"
387  *
388  * @param status - Subscription status (UserStatus)
389  * @returns CSS class name
390  * @memberof SubscriptionHistoryComponent
391  */
392  getStatusClass(status: string) {
393    const statusClassMap: { [key: string]: string } = {

```

```

394     [UserStatus.PURCHASED]: 'completed',
395     [UserStatus.PENDING]: 'pending',
396     [UserStatus.PROCESSING]: 'processing',
397     [UserStatus.CANCELLED]: 'cancelled',
398     [UserStatus.EXPIRED]: 'expired',
399     [UserStatus.BANNED]: 'banned',
400     [UserStatus.ADMIN]: 'admin',
401     [UserStatus.CREATED]: 'created'
402   };
403   return statusClassMap[status] || 'unknown';
404 }
405
406 /**
407 * Formats a date for display in the interface.
408 *
409 * Handles both Firebase Timestamp objects and Date objects.
410 * Formats date in readable format (e.g.: "Jan 15, 2024, 10:30 AM").
411 *
412 * @param date - Date to format (can be Firebase Timestamp or Date)
413 * @returns Formatted date or "N/A" if no date
414 * @memberof SubscriptionHistoryComponent
415 */
416 formatDate(date: any): string {
417   if (!date) return 'N/A';
418   const dateObj = date.toDate ? date.toDate() : new Date(date);
419   return dateObj.toLocaleDateString('en-US', {
420     year: 'numeric',
421     month: 'short',
422     day: 'numeric',
423     hour: '2-digit',
424     minute: '2-digit'
425   });
426 }
427
428 /**
429 * Formats an amount as currency according to provided currency code.
430 *
431 * Uses Intl.NumberFormat to format amount with corresponding
432 * currency symbol.
433 *
434 * @param amount - Amount to format
435 * @param currency - Currency code (e.g.: "usd", "eur")
436 * @returns Amount formatted as currency (e.g.: "$99.00")
437 * @memberof SubscriptionHistoryComponent
438 */
439 formatCurrency(amount: number, currency: string): string {
440   return new Intl.NumberFormat('en-US', {
441     style: 'currency',
442     currency: currency.toUpperCase()
443   }).format(amount);
444 }
445 }
446

```

Ø=ÜÁ features\account\mocks

Ø=ÜÁ features\account\mocks\account-mocks.ts

```

1 import { PlanCard, PlanDetails } from '../models/account-settings';
2
3 /**
4  * Mock data for user plan details.
5 *

```

```

6  * This object simulates the current user's plan information.
7  * Currently used in AccountComponent to initialize plan data.
8  *
9  * NOTE: In production, this data should be obtained from the subscription service.
10 *
11 * @constant MOCK_PLAN_DETAILS
12 * @type {PlanDetails}
13 * @see PlanDetails
14 */
15 export const MOCK_PLAN_DETAILS: PlanDetails = {
16   currentPlan: 'Pro Plan',
17   renewalDate: '2025-08-27',
18   remainingUntilRenewal: '16 days',
19   price: 250.0,
20   activationFee: null,
21   billingCycle: 'Monthly',
22 };
23
24 /**
25 * Array of available plans for subscription.
26 *
27 * This array contains the definition of all plans displayed
28 * in the plan selection interface. Each plan includes:
29 * - Price and period information
30 * - Features and limits (trading accounts, strategies, etc.)
31 * - Visual information (icons, colors)
32 * - Action button text (CTA)
33 *
34 * Used in:
35 * - PlanSettingsComponent: As initial data before loading from service
36 *
37 * NOTE: In production, this data should be obtained from PlanService.
38 *
39 * @constant PLANS
40 * @type {PlanCard[]}
41 * @see PlanCard
42 */
43 export const PLANS: PlanCard[] = [
44   {
45     name: 'Free',
46     price: 0,
47     period: '/month',
48     icon: 'circle',
49     color: '#4b7ee8',
50     features: [
51       { label: 'Trading Accounts', value: '1' },
52       { label: 'Consistency Rules', value: 'YES' },
53       { label: 'Trading Journal', value: 'YES' },
54       { label: 'Live Statistics', value: 'YES' },
55     ],
56     cta: 'Change Plan',
57   },
58   {
59     name: 'Starter',
60     price: 35,
61     period: '/month',
62     icon: 'circle',
63     color: '#4b7ee8',
64     features: [
65       { label: 'Trading Accounts', value: '2' },
66       { label: 'Consistency Rules', value: 'YES' },
67       { label: 'Trading Journal', value: 'YES' },
68       { label: 'Live Statistics', value: 'YES' },
69     ],
70     cta: 'Change Plan',
71   },
72   {
73     name: 'Pro',
74     price: 99,
75     period: '/month',

```

```

76     mostPopular: true,
77     icon: 'square',
78     color: '#d1ff81',
79     features: [
80       { label: 'Trading Accounts', value: '6' },
81       { label: 'Consistency Rules', value: 'YES' },
82       { label: 'Trading Journal', value: 'YES' },
83       { label: 'Live Statistics', value: 'YES' },
84     ],
85     cta: 'Get Starter Now',
86   },
87 ];
88

```

Ø=ÜÁ features\account\models

Ø=ÜÁ features\account\models\account-settings.ts

```

1  /**
2  * Interface that represents the details of the user's current plan.
3  *
4  * This interface contains all information related to the user's active
5  * subscription plan, including billing and renewal information.
6  *
7  * Used in:
8  * - AccountComponent: To pass plan data to child component
9  * - PlanSettingsComponent: To display current plan information
10 *
11 * @interface PlanDetails
12 */
13 export interface PlanDetails {
14   currentPlan: string;
15   renewalDate: string;
16   remainingUntilRenewal: string;
17   price: number;
18   activationFee: string | null;
19   billingCycle: string;
20 }
21
22 /**
23 * Interface that represents a plan card in the user interface.
24 *
25 * This interface defines the data structure needed to display a plan
26 * in the plan comparison and selection interface. Includes visual
27 * information (icons, colors) and functional information (price, features, CTA).
28 *
29 * Used in:
30 * - PlanSettingsComponent: To build and display available plan cards
31 * - account-mocks.ts: To define mock data for plans
32 *
33 * @interface PlanCard
34 */
35 export interface PlanCard {
36   name: string;
37   price: number;
38   period: string;
39   mostPopular?: boolean;
40   icon: string;
41   color: string;
42   features: [
43     { label: string;
44       value: string;
45   }[];

```

```
46   cta: string;
47 }
48
```

Ø=ÜÀ features\add-account

Ø=ÜÀ features\add-account\add-account.component.ts

```
1 import { CommonModule } from '@angular/common';
2 import { Component } from '@angular/core';
3 import {
4   FormBuilder,
5   FormGroup,
6   ReactiveFormsModule,
7   Validators,
8 } from '@angular/forms';
9 import {
10   PhoneInputComponent,
11   TextInputComponent,
12 } from '../../../../../shared/components';
13 import { Router, RouterLink } from '@angular/router';
14 import { PasswordInputComponent } from '../../../../../shared/components/password-input/password-
15 import { AuthService } from '../auth/service/authService';
16 import { Store } from '@ngrx/store';
17 import { selectUser } from '../auth/store/user.selectios';
18 import { AccountData } from '../auth/models/userModel';
19 import { Timestamp } from 'firebase/firestore';
20 import { first } from 'rxjs';
21
22 /**
23 * Component for adding a new trading account.
24 *
25 * This component provides a form interface for users to register
26 * a new trading account with their broker. It collects account
27 * information including email, password, broker details, and account
28 * identification data.
29 *
30 * Related to:
31 * - AuthService: Creates the account in Firebase
32 * - Store (NgRx): Gets current user data
33 * - Router: Navigates to trading accounts page after creation
34 *
35 * @component
36 * @selector app-add-account
37 * @standalone true
38 */
39 @Component({
40   selector: 'app-add-account',
41   standalone: true,
42   imports: [
43     CommonModule,
44     ReactiveFormsModule,
45     TextInputComponent,
46     PasswordInputComponent,
47     RouterLink,
48   ],
49   templateUrl: './add-account.component.html',
50   styleUrls: ['./add-account.component.scss'],
51 })
52 export class AddAccountComponent {
53   /** Form group containing all trading account input fields */
54   accountForm: FormGroup;
```

```

56  /**
57  * Constructor for AddAccountComponent.
58  *
59  * Initializes the reactive form with all required fields and validators:
60  * - emailTradingAccount: Required email validation
61  * - brokerPassword: Required, minimum 6 characters
62  * - broker: Required broker name
63  * - server: Required server name
64  * - accountName: Required account name
65  * - accountID: Required account ID
66  * - accountNumber: Required, numeric pattern only
67  *
68  * @param fb - FormBuilder for creating reactive forms
69  * @param authService - Service for authentication and account operations
70  * @param router - Router for navigation
71  * @param store - NgRx Store for accessing user state
72  */
73 constructor(
74     private fb: FormBuilder,
75     private authService: AuthService,
76     private router: Router,
77     private store: Store
78 ) {
79     this.accountForm = this.fb.group({
80         emailTradingAccount: ['', [Validators.required, Validators.email]],
81         brokerPassword: ['', [Validators.required, Validators.minLength(6)]],
82         broker: ['', [Validators.required]],
83         server: ['', [Validators.required]],
84         accountName: ['', [Validators.required]],
85         accountID: ['', [Validators.required]],
86         accountNumber: [
87             '',
88             [Validators.required, Validators.pattern('^[0-9]*$')],
89         ],
90     });
91 }
92 /**
93 * Handles form submission when user clicks the submit button.
94 *
95 * Validates the form and either:
96 * - Processes registration if form is valid
97 * - Marks all form fields as touched to show validation errors if invalid
98 *
99 * @memberof AddAccountComponent
100 */
101 onSubmit

```

```

126  private processRegistration(): void {
127    this.store
128      .select(selectUser)
129      .pipe(first())
130      .subscribe((user) => {
131        const userId = user?.user?.id || '';
132        this.authService.createAccount(this.createAccountObject(userId));
133        this.router.navigate(['/trading-accounts']);
134      });
135    }
136  }
137 /**
138  * Marks all form controls as touched to trigger validation error display.
139  *
140  * This method iterates through all form controls and marks them as touched,
141  * which causes Angular to display validation error messages for invalid fields.
142  *
143  * @private
144  * @memberof AddAccountComponent
145  */
146 private markFormGroupTouched(): void {
147   Object.keys(this.accountForm.controls).forEach((key) => {
148     const control = this.accountForm.get(key);
149     control?.markAsTouched();
150   });
151 }
152 /**
153  * Creates an AccountData object from form values.
154  *
155  * Generates a unique ID for the account using timestamp and random string,
156  * then constructs an AccountData object with all form values and metadata.
157  *
158  * The unique ID is generated using:
159  * - Current timestamp in base36 format
160  * - Random string (6 characters)
161  * - Format: `id_{timestamp}_{random}`
162  *
163  * @private
164  * @param id - User ID to associate with the account
165  * @returns AccountData object ready to be saved to Firebase
166  * @memberof AddAccountComponent
167  */
168 private createAccountObject(id: string): AccountData {
169   const timestamp = Date.now().toString(36);
170   const randomPart = Math.random().toString(36).substring(2, 8);
171   const uniqueId = `id_${timestamp}_${randomPart}`;
172   return {
173     id: uniqueId,
174     userId: id,
175     emailTradingAccount: this.accountForm.value.emailTradingAccount,
176     brokerPassword: this.accountForm.value.brokerPassword,
177     broker: this.accountForm.value.broker,
178     server: this.accountForm.value.server,
179     accountName: this.accountForm.value.accountName,
180     accountID: this.accountForm.value.accountID,
181     accountNumber: Number(this.accountForm.value.accountNumber),
182     createdAt: Timestamp.now(),
183   };
184 }
185 }
186 }
187

```

Ø=ÜÁ features\auth\login

Ø=ÜÄ features\auth\login\login.ts

```
1 import { Component } from '@angular/core';
2 import { NgIf } from '@angular/common';
3 import {
4   FormBuilder,
5   FormGroup,
6   ReactiveFormsModule,
7   Validators,
8   AbstractControl,
9   ValidationErrors,
10 } from '@angular/forms';
11 import { PasswordInputComponent } from '../../../../../shared/components/password-input/password-
12 import { TextInputComponent } from '../../../../../shared/components';
13 import { AuthService } from '../../../../../shared/services/auth.service';
14 import { Router, RouterLink } from '@angular/router';
15 import { Store } from '@ngrx/store';
16 import { setUserData } from '../store/user.actions';
17 import { User } from '../../../../../overview/models/overview';
18 import { UserCredentials } from '../models/userModel';
19 import { ApplicationContextService } from '../../../../../shared/context';
20 import { AlertService } from '../../../../../shared/services/alert.service';
21 import { ForgotPasswordPopupComponent } from '../../../../../shared/pop-ups/forgot-password/forgot-
22 password.component';
23 @Component({
24   selector: 'app-login',
25   standalone: true,
26   imports: [
27     ReactiveFormsModule,
28     PasswordInputComponent,
29     TextInputComponent,
30     RouterLink,
31     ForgotPasswordPopupComponent,
32   ],
33   templateUrl: './login.html',
34   styleUrls: ['./login.scss'],
35 })
36 export class Login {
37   loginForm: FormGroup;
38   showPassword = false;
39   forgotVisible = false;
40
41   constructor(
42     private fb: FormBuilder,
43     private authService: AuthService,
44     private store: Store,
45     private router: Router,
46     private applicationContext: ApplicationContextService,
47     private alertService: AlertService
48   ) {
49     this.loginForm = this.fb.group({
50       loginEmail: ['', [Validators.required, Validators.email, this.emailValidator]],
51       password: ['', [Validators.required]],
52       rememberMe: [false],
53     });
54   }
55
56   onSubmit(): void {
57     // Validar campos antes de proceder
58     this.validateLoginFields();
59
60     if (this.loginForm.valid) {
```

```

62     const userCredentials = this.createUserCredentialsObject();
63
64     // Establecer estado de carga
65     this.appContext.setLoading('user', true);
66     this.appContext.setError('user', null);
67
68     this.authService
69       .login(userCredentials)
70       .then((response: any) => {
71       this.authService
72         .getUserData(response.user.uid)
73         .then((userData: User) => {
74           // Actualizar contexto con datos del usuario
75           this.appContext.setCurrentUser(userData);
76
77           // Mantener compatibilidad con NgRx
78           this.store.dispatch(setUserData({ user: userData }));
79
80           // Limpiar estado de carga
81           this.appContext.setLoading('user', false);
82
83           // Navegar según el tipo de usuario
84           if (userData.isAdmin) {
85             this.router.navigate(['/overview']);
86           } else {
87             this.router.navigate(['/strategy']);
88           }
89         })
90       .catch((error: any) => {
91         this.appContext.setLoading('user', false);
92         this.appContext.setError('user', 'Error al obtener datos del usuario');
93         this.handleLoginError(error);
94       });
95     })
96     .catch((error: any) => {
97       this.appContext.setLoading('user', false);
98       this.appContext.setError('user', 'Error de autenticación');
99       this.handleLoginError(error);
100    });
101  }
102}
103
104 openForgot(): void {
105   this.forgotVisible = true;
106 }
107
108 closeForgot(): void {
109   this.forgotVisible = false;
110 }
111
112 private createUserCredentialsObject(): UserCredentials {
113   return {
114     email: this.loginForm.value.loginEmail,
115     password: this.loginForm.value.password,
116   };
117 }
118
119 togglePasswordVisibility(): void {
120   this.showPassword = !this.showPassword;
121 }
122
123 signInWithGoogle(): void {
124   // TODO: Implement Google sign-in
125   console.log('Google sign-in');
126 }
127
128 signInWithApple(): void {
129   // TODO: Implement Apple sign-in
130   console.log('Apple sign-in');
131 }

```

```

132 // Validador personalizado para email
133 private emailValidator(control: AbstractControl): ValidationErrors | null {
134   if (!control.value) return null;
135
136   const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
137
138   if (!emailRegex.test(control.value)) {
139     return { invalidEmailFormat: true };
140   }
141
142   return null;
143 }
144
145 // Validar campos del formulario
146 private validateLoginFields(): void {
147   const errors: string[] = [];
148
149   const emailControl = this.loginForm.get('loginEmail');
150   const passwordControl = this.loginForm.get('password');
151
152   // Validar email
153   if (emailControl?.errors?.['required']) {
154     errors.push('Email is required');
155   } else if (emailControl?.errors?.['email']) {
156     errors.push('Invalid email format');
157   } else if (emailControl?.errors?.['invalidEmailFormat']) {
158     errors.push('Invalid email format. Must contain @ and a valid domain');
159   }
160
161   // Validar contraseña
162   if (passwordControl?.errors?.['required']) {
163     errors.push('Password is required');
164   }
165
166   // Mostrar alerta si hay errores
167   if (errors.length > 0) {
168     this.alertService.showError('Validation errors:\n\n' + errors.join('\n'), 'Validation
169     Error');
170   }
171
172   // Manejar errores de login
173   private handleLoginError(error: any): void {
174     console.error('Login error:', error);
175
176     let errorMessage = 'Login failed. ';
177
178     // Verificar tipo de error específico
179     if (error.code === 'auth/user-not-found') {
180       errorMessage += 'No account found with this email.';
181     } else if (error.code === 'auth/wrong-password') {
182       errorMessage += 'Incorrect password.';
183     } else if (error.code === 'auth/invalid-email') {
184       errorMessage += 'Invalid email format.';
185     } else if (error.code === 'auth/user-disabled') {
186       errorMessage += 'This account has been disabled.';
187     } else if (error.code === 'auth/too-many-requests') {
188       errorMessage += 'Too many failed attempts. Please try again later.';
189     } else {
190       // Error genérico - verificar si es problema de credenciales
191       errorMessage += 'Email or password is incorrect. Please check your credentials and try
192       again.';
193
194       this.alertService.showError(errorMessage, 'Login Error');
195     }
196   }
197 }
198

```

Ø=ÜÁ features\auth\models

Ø=ÜÄ features\auth\models\linkModels.ts

```
1
2
3 interface LinkToken {
4   id: string;
5   userId: string;
6 }
```

Ø=ÜÄ features\auth\models\userModel.ts

```
1 import { Timestamp } from 'firebase/firestore';
2
3 export interface UserCredentials {
4   email: string;
5   password: string;
6 }
7
8 export interface AccountData {
9   id: string;
10  userId: string;
11  emailTradingAccount: string;
12  brokerPassword: string;
13  broker: string;
14  server: string;
15  accountName: string;
16  accountID: string;
17  accountNumber: number;
18  initialBalance?: number;
19  balance?: number;
20  netPnl?: number;
21  profit?: number;
22  bestTrade?: number;
23  createdAt: Timestamp;
24 }
25
```

Ø=ÜÁ features\auth\service

Ø=ÜÄ features\auth\service\authService.ts

```
1 // Deliberadamente mantener esta ruta para compatibilidad.
2 // Reexporta el nuevo servicio centralizado en shared/services.
3 export { AuthService } from '../../../../../shared/services/auth.service';
4
```

Ø=ÜÁ features\auth\signup

Ø=ÜÄ features\auth\signup\signup.ts

```
1 import { Component, OnInit } from '@angular/core';
2 import { Router, RouterLink } from '@angular/router';
3 import { CommonModule } from '@angular/common';
4 import {
5   FormBuilder,
6   FormGroup,
7   Validators,
8   ReactiveFormsModule,
9   AbstractControl,
10  ValidationErrors,
11 } from '@angular/forms';
12 import { PhoneInputComponent } from '../../../../../shared/components/phone-input/phone-
13 import { BirthdayInputComponent } from '../../../../../shared/components/birthday-input/birthday-
14 import { TextInputComponent } from '../../../../../shared/components/text-input/text-
15 import { AuthService } from '../../../../../shared/services/auth.service';
16 import { PasswordInputComponent } from '../../../../../shared/components/password-input/password-
17 import { UserStatus } from '../../../../../overview/models/overview';
18 import { UserCredentials } from '../models/userModel';
19 import { Plan, PlanService } from '../../../../../shared/services/planService';
20 import { PlanSelectionComponent, PlanCard } from './components/plan-selection/plan-
21 import { SubscriptionService } from '../../../../../shared/services/subscription-
22 import { setUserData } from '../store/user.actions';
23 import { Store } from '@ngrx/store';
24 import { ApplicationContextService } from '../../../../../shared/context';
25 import { StripeLoaderPopupComponent } from '../../../../../shared/pop-ups/stripe-loader-popup/
26 import { AlertService } from '../../../../../shared/services/alert.service';
27
28 @Component({
29   selector: 'app-signup',
30   standalone: true,
31   imports: [
32     CommonModule,
33     ReactiveFormsModule,
34     PhoneInputComponent,
35     BirthdayInputComponent,
36     TextInputComponent,
37     PasswordInputComponent,
38     RouterLink,
39     PlanSelectionComponent,
40     StripeLoaderPopupComponent,
41   ],
42   templateUrl: './signup.html',
43   styleUrls: ['./signup.scss'],
44 })
45 export class SignupComponent implements OnInit {
46   signupForm: FormGroup;
47   accountForm: FormGroup;
48   currentStep = 1;
49   isAdminSignup: boolean = false;
50   showPlanSelection = false;
51   userData: any = null;
52   selectedPlan: PlanCard | null = null;
53   currentUserID: string = '';
54
55   // Estados para loader y error de Stripe
56   showStripeLoader = false;
57   showStripeError = false;
58   stripeErrorMessage = '';
59
60   constructor(
61     private fb: FormBuilder,
```

```

62     private authService: AuthService,
63     private router: Router,
64     private planService: PlanService,
65     private subscriptionService: SubscriptionService,
66     private store: Store,
67     private applicationContext: ApplicationContextService,
68     private alertService: AlertService
69   ) {
70     this.signupForm = this.fb.group({
71       firstName: ['', [Validators.required, Validators.minLength(2)]],
72       lastName: ['', [Validators.required, Validators.minLength(2)]],
73       phoneNumber: ['', [Validators.required, this.phoneValidator]],
74       birthday: ['', [Validators.required, this.ageValidator]],
75       email: ['', [Validators.required, Validators.email, this.emailValidator]],
76       password: ['', [Validators.required, Validators.minLength(6)]],
77     });
78
79     this.accountForm = this.fb.group({
80       emailTradingAccount: ['', [Validators.required, Validators.email]],
81       brokerPassword: ['', [Validators.required, Validators.minLength(6)]],
82       server: ['', [Validators.required]],
83       accountName: ['', [Validators.required]],
84       accountId: ['', [Validators.required]],
85       accountNumber: [
86         '',
87         [Validators.required, Validators.pattern('^[0-9]*$')],
88       ],
89     });
90   }
91
92   ngOnInit(): void {
93     const currentUrl = this.router.url;
94     if (currentUrl === '/admin-signup') {
95       this.isAdminSignup = true;
96     }
97   }
98
99   onChange(): void {
100    console.log('Form changed:', this.signupForm.value);
101  }
102  async onSubmit(): Promise<void> {
103    if (this.signupForm.valid) {
104      try {
105        // Establecer estado de carga
106        this.applicationContext.setLoading('user', true);
107        this.applicationContext.setError('user', null);
108
109        // Crear credenciales del usuario
110        const userCredentials = this.createUserCredentialsObject();
111
112        // Verificar que el email no esté ya registrado
113        const existingUser = await this.authService.getUserByEmail(userCredentials.email);
114
115        if (existingUser) {
116          this.alertService.showError('This email is already registered. Please use a
117 different email');
118          return;
119        }
120
121        // Crear el usuario en Firebase Auth
122        const userResponse = await this.authService.register(userCredentials);
123        const userId = userResponse.user.uid;
124
125        // Crear el token y objeto usuario
126        const token = this.createTokenObject(userId);
127        const user: User = await this.createUserObject(userId, token.id);
128
129        // Configurar como admin si corresponde
130        if (this.isAdminSignup) {
131          user.isAdmin = true;

```

```

132         user.status = UserStatus.ADMIN;
133     } else {
134         user.status = UserStatus.ACTIVE;
135     }
136
137     // Crear usuario y token en Firestore
138     await this.authService.createUser(user as User);
139     await this.authService.createLinkToken(token);
140
141     // Crear suscripción gratuita activa por defecto
142     const freeSubscriptionData: Omit<Subscription, 'id' | 'created_at' | 'updated_at'> =
143         {
144             planId: "Cb1B0tpxdE6AP6eMZDo0",
145             status: UserStatus.ACTIVE,
146             userId: userId,
147         };
148
149     await this.subscriptionService.createSubscription(userId, freeSubscriptionData);
150     // Iniciar sesión automáticamente
151     const loginResponse = await this.authService.login(userCredentials);
152     const userData = await this.authService.getUserData(loginResponse.user.uid);
153
154     // Actualizar contexto con datos completos del usuario
155     this.appContext.setCurrentUser(userData);
156
157     // Mantener compatibilidad con NgRx
158     this.store.dispatch(setUserData({ user: userData }));
159
160     // Guardar userId para usar en la selección de plan
161     this.currentUserId = userId;
162
163     // Limpiar estado de carga
164     this.appContext.setLoading('user', false);
165
166     // Guardar datos del usuario para mostrar en la selección de planes
167     this.userData = this.signupForm.value;
168
169     // Si es admin, ir directo al dashboard
170     if (userData.isAdmin) {
171         this.router.navigate(['/overview']);
172     } else {
173         // Usuario normal: mostrar selección de planes
174         this.showPlanSelection = true;
175     }
176
177 } catch (error: any) {
178     this.appContext.setLoading('user', false);
179     this.appContext.setError('user', 'Error during registration');
180
181     const errorMessage = error.message || 'An error occurred during registration. Please
try again.';
182     this.alertService.showError(errorMessage, 'Registration Error');
183
184     this.handleRegistrationError(error);
185 }
186     this.showValidationErrors();
187 }
188
189
190 private markFormGroupTouched(): void {
191     Object.keys(this.signupForm.controls).forEach((key) => {
192         const control = this.signupForm.get(key);
193         control?.markAsTouched();
194     });
195 }
196
197 signInWithGoogle(): void {
198     console.log('Sign in with Google');
199 }
200
201 signInWithApple(): void {

```

```

202     console.log('Sign in with Apple');
203 }
204
205 private createUserCredentialsObject(): UserCredentials {
206     return {
207         email: this.signupForm.value.email,
208         password: this.signupForm.value.password,
209     };
210 }
211
212 private async createUserObject(id: string, tokenId: string): Promise<User> {
213
214     return {
215         id: id,
216         email: this.signupForm.value.email,
217         tokenId: tokenId,
218         firstName: this.signupForm.value.firstName,
219         lastName: this.signupForm.value.lastName,
220         phoneNumber: this.signupForm.value.phoneNumber,
221         birthday: this.signupForm.value.birthday,
222         best_trade: 0,
223         netPnl: 0,
224         number_trades: 0,
225         profit: 0,
226         status: UserStatus.CREATED,
227         strategy_followed: 0,
228         subscription_date: new Date().getTime(),
229         lastUpdated: new Date().getTime(),
230         total_spend: 0,
231         isAdmin: false,
232         trading_accounts: 0,
233         strategies: 0,
234     };
235 }
236
237 private createTokenObject(userId: string): LinkToken {
238     return {
239         id: this.signupForm.value.email.split('@')[0] + userId.substring(0, 4),
240         userId: userId,
241     };
242 }
243
244 // Validadores personalizados
245 private phoneValidator(control: AbstractControl): ValidationErrors | null {
246     if (!control.value) return null;
247
248     const phoneRegex = /^[+]?[1-9][\d]{0,15}$/;
249     const cleanPhone = control.value.replace(/[\s\-\(\)]/g, '');
250
251     if (!phoneRegex.test(cleanPhone)) {
252         return { invalidPhone: true };
253     }
254
255     if (cleanPhone.length < 10 || cleanPhone.length > 15) {
256         return { invalidPhoneLength: true };
257     }
258
259     return null;
260 }
261
262 private ageValidator(control: AbstractControl): ValidationErrors | null {
263     if (!control.value) return null;
264
265     const today = new Date();
266     const birthDate = new Date(control.value);
267     let age = today.getFullYear() - birthDate.getFullYear();
268     const monthDiff = today.getMonth() - birthDate.getMonth();
269
270     if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) {
271         age--;

```

```

272     }
273
274     if (age < 18) {
275         return { underage: true };
276     }
277
278     return null;
279 }
280
281 private emailValidator(control: AbstractControl): ValidationErrors | null {
282     if (!control.value) return null;
283
284     const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
285
286     if (!emailRegex.test(control.value)) {
287         return { invalidEmailFormat: true };
288     }
289
290     return null;
291 }
292
293 private showValidationErrors(): void {
294     const errors: string[] = [];
295
296     // Verificar errores específicos de cada campo
297     const firstNameControl = this.signupForm.get('firstName');
298     const lastNameControl = this.signupForm.get('lastName');
299     const emailControl = this.signupForm.get('email');
300     const phoneControl = this.signupForm.get('phoneNumber');
301     const birthdayControl = this.signupForm.get('birthday');
302     const passwordControl = this.signupForm.get('password');
303
304     if (firstNameControl?.errors?.['required']) {
305         errors.push('First name is required');
306     } else if (firstNameControl?.errors?.['minlength']) {
307         errors.push('First name must be at least 2 characters');
308     }
309
310     if (lastNameControl?.errors?.['required']) {
311         errors.push('Last name is required');
312     } else if (lastNameControl?.errors?.['minlength']) {
313         errors.push('Last name must be at least 2 characters');
314     }
315
316     if (emailControl?.errors?.['required']) {
317         errors.push('Email is required');
318     } else if (emailControl?.errors?.['email']) {
319         errors.push('Invalid email format');
320     } else if (emailControl?.errors?.['invalidEmailFormat']) {
321         errors.push('Invalid email format. Must contain @ and a valid domain');
322     }
323
324     if (phoneControl?.errors?.['required']) {
325         errors.push('Phone number is required');
326     } else if (phoneControl?.errors?.['invalidPhone']) {
327         errors.push('Invalid phone number format');
328     } else if (phoneControl?.errors?.['invalidPhoneLength']) {
329         errors.push('Phone number must be between 10 and 15 digits');
330     }
331
332     if (birthdayControl?.errors?.['required']) {
333         errors.push('Birthday is required');
334     } else if (birthdayControl?.errors?.['underage']) {
335         errors.push('You must be 18 years or older to register');
336     }
337
338     if (passwordControl?.errors?.['required']) {
339         errors.push('Password is required');
340     } else if (passwordControl?.errors?.['minlength']) {
341         errors.push('Password must be at least 6 characters');

```

```

342     }
343
344     // Mostrar alerta con todos los errores
345     if (errors.length > 0) {
346       this.alertService.showError('Validation errors:\n\n' + errors.join('\n'), 'Validation
347 Error');
348
349     this.markFormGroupTouched();
350   }
351
352   async onPlanSelected(plan: PlanCard): Promise<void> {
353     this.selectedPlan = plan;
354
355     // Si selecciona el plan Free, redirigir al dashboard (sin loader ni pop-ups de Stripe)
356     if (plan.name.toLowerCase() === 'free') {
357       this.router.navigate(['/strategy']);
358       return;
359     }
360
361     // Solo para planes de pago: mostrar loader y manejar errores de Stripe
362     try {
363       this.showStripeLoader = true;
364
365       // Variable para controlar si hay error
366       let hasError = false;
367       let errorMessage = '';
368
369       try {
370         await this.createCheckoutSession(plan.name);
371       } catch (error) {
372         // Marcar que hay error pero no mostrar pop-up aún
373         hasError = true;
374         errorMessage = 'Error redirecting to payment. Please try again.';
375         console.error('Error during checkout session creation:', error);
376       }
377
378       // Esperar mínimo 2 segundos antes de mostrar error o ocultar loader
379       setTimeout(() => {
380         if (hasError) {
381           // Si hay error, mostrar pop-up de error
382           this.showStripeLoader = false;
383           this.showStripeError = true;
384           this.stripeErrorMessage = errorMessage;
385         } else {
386           // Si no hay error, el loader se ocultará automáticamente por la redirección
387           this.showStripeLoader = false;
388         }
389       }, 2000);
390
391     } catch (error: any) {
392       console.error('L Error in plan selection:', error);
393       this.showStripeLoader = false;
394       this.showStripeError = true;
395       this.stripeErrorMessage = 'Error processing your plan selection. Please try again.';
396     }
397   }
398
399   private async createCheckoutSession(planName: string): Promise<void> {
400     try {
401       // Obtener el plan completo desde el servicio
402       const plans = await this.planService.searchPlansByName(planName);
403       const selectedPlan = plans && plans.length > 0 ? plans[0] : null;
404
405       if (!selectedPlan || !selectedPlan.planPriceId) {
406         throw new Error('Plan price ID not found');
407       }
408
409       // Obtener el token de Firebase
410       const bearerTokenFirebase = await
411     this.authService.getBearerTokenFirebase(this.currentUser);

```

```

412     // Crear checkout session
413     const response = await fetch('https://api.tradeswitch.io/payments/create-checkout-
414 session' method: 'POST',
415     headers: {
416         'Content-Type': 'application/json',
417         'Authorization': `Bearer ${bearerTokenFirebase}`
418     },
419     body: JSON.stringify({
420         priceId: selectedPlan.planPriceId,
421     })
422 });
423
424 if (!response.ok) {
425     const errorText = await response.text();
426     throw new Error(`Error creating checkout session: ${response.status}
427 ${response.statusText} - ${errorText}`);
428
429     const responseData = await response.json();
430     const checkoutUrl = responseData.body?.url || responseData.url;
431
432 if (!checkoutUrl) {
433     throw new Error('Checkout URL not found in response');
434 }
435
436 // Redirigir a la página de checkout de Stripe
437 window.location.href = checkoutUrl;
438
439 } catch (error) {
440     console.error('L Error creating checkout session:', error);
441     throw error;
442 }
443 }
444
445 onGoBackToSignup(): void {
446     this.showPlanSelection = false;
447 }
448
449 // Método para cerrar el pop-up de error de Stripe
450 closeStripeError(): void {
451     this.showStripeError = false;
452     this.stripeErrorMessage = '';
453     this.showPlanSelection = true; // Volver a mostrar la selección de planes
454 }
455
456 private handleRegistrationError(error: any): void {
457     console.log('Registration error:', error);
458     // Los errores comunes serán manejados por Firebase
459 }
460 }
461

```

Ø=ÜÁ features\auth\signup\components\plan-selection

Ø=ÜÄ features\auth\signup\components\plan-selection\plan-selection.component.ts

```

1 import { Component, EventEmitter, Input, Output } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { Router } from '@angular/router';
4
5 export interface PlanCard {
6     name: string;
7     price: number;
8     period: string;

```

```

9   mostPopular?: boolean;
10  icon: 'triangle' | 'circle' | 'square';
11  color: string;
12  features: {
13    label: string;
14    value: string | number;
15  }[];
16  cta: string;
17 }
18
19 @Component({
20   selector: 'app-plan-selection',
21   standalone: true,
22   imports: [CommonModule],
23   templateUrl: './plan-selection.component.html',
24   styleUrls: ['./plan-selection.component.scss'
25 })
26 export class PlanSelectionComponent {
27   @Input() userData: any = null;
28   @Output() planSelected = new EventEmitter<PlanCard>();
29   @Output() goBack = new EventEmitter<void>();
30
31   plansData: PlanCard[] = [
32     {
33       name: 'Free',
34       price: 0,
35       period: '/month',
36       icon: 'triangle',
37       color: '#4b7ee8',
38       features: [
39         { label: 'Trading Accounts', value: 1 },
40         { label: 'Strategies', value: 1 },
41         { label: 'Consistency Rules', value: 'YES' },
42         { label: 'Trading Journal', value: 'YES' },
43         { label: 'Live Statistics', value: 'YES' }
44       ],
45       cta: 'Get Free Now'
46     },
47     {
48       name: 'Starter',
49       price: 35,
50       period: '/month',
51       icon: 'circle',
52       color: '#4b7ee8',
53       features: [
54         { label: 'Trading Accounts', value: 2 },
55         { label: 'Strategies', value: 3 },
56         { label: 'Consistency Rules', value: 'YES' },
57         { label: 'Trading Journal', value: 'YES' },
58         { label: 'Live Statistics', value: 'YES' }
59       ],
60       cta: 'Get Starter Now'
61     },
62     {
63       name: 'Pro',
64       price: 99,
65       period: '/month',
66       mostPopular: true,
67       icon: 'square',
68       color: '#d1ff81',
69       features: [
70         { label: 'Trading Accounts', value: 6 },
71         { label: 'Strategies', value: 8 },
72         { label: 'Consistency Rules', value: 'YES' },
73         { label: 'Trading Journal', value: 'YES' },
74         { label: 'Live Statistics', value: 'YES' }
75       ],
76       cta: 'Get Pro Now'
77     }
78 ];

```

```

79     constructor(private router: Router) {}
80
81     onPlanSelect(plan: PlanCard): void {
82         this.planSelected.emit(plan);
83     }
84
85     onGoBack(): void {
86         this.goBack.emit();
87     }
88
89     isNumeric(value: string | number): boolean {
90         if (typeof value === 'number') {
91             return true;
92         }
93         if (typeof value === 'string') {
94             return !isNaN(Number(value)) && !isNaN(parseFloat(value));
95         }
96         return false;
97     }
98 }
99
100

```

Ø=ÜÄ features\auth\store

Ø=ÜÄ features\auth\store\user.actions.ts

```

1 import { createAction, props } from '@ngrx/store';
2 import { User } from '../../../../../overview/models/overview';
3
4 export const setUserData = createAction(
5     '[Auth] Set user data',
6     props<{ user: User | null }>()
7 );
8

```

Ø=ÜÄ features\auth\store\user.reducer.ts

```

1 import { createReducer, on } from '@ngrx/store';
2 import { setUserData } from './user.actions';
3 import { User } from '../../../../../overview/models/overview';
4
5 export interface UserState {
6     user: User | null;
7 }
8
9 export const initialState: UserState = {
10     user: null,
11 };
12
13 export const userReducer = createReducer(
14     initialState,
15     on(setUserData, (state, { user }) => ({ ...state, user }))
16 );
17

```

Ø=ÜÄ features\auth\store\user.selectios.ts

```
1 import { createFeatureSelector, createSelector } from '@ngrx/store';
2 import { UserState } from './user.reducer';
3 import e from 'express';
4
5 export const selectUser = createFeatureSelector<UserState>('user');
6
7 export const getUserId = createSelector(selectUser, (state) => state.user?.id);
8
9 export const getUserEmail = createSelector(
10   selectUser,
11   (state) => state.user?.email
12 );
13
14 export const getUserName = createSelector(
15   selectUser,
16   (state) => state.user?.firstName
17 );
18
19 export const getUserLastName = createSelector(
20   selectUser,
21   (state) => state.user?.lastName
22 );
23
24 export const getUserPhoneNumber = createSelector(
25   selectUser,
26   (state) => state.user?.phoneNumber
27 );
28
29 export const getUserBirthday = createSelector(
30   selectUser,
31   (state) => state.user?.birthday
32 );
33
34 export const getUserIdToken = createSelector(
35   selectUser,
36   (state) => state.user?.tokenId
37 );
38
39 export const getIsAdmin = createSelector(
40   selectUser,
41   (state) => state.user?.isAdmin
42 );
43
```

Ø=ÜÄ features\overview

Ø=ÜÄ features\overview\overview.component.ts

```
1 import { Store } from '@ngrx/store';
2 import { CommonModule } from '@angular/common';
3 import { Component } from '@angular/core';
4 import { statCardComponent } from '../report/components/statCard/stat_card.component';
5 import { OverviewService } from './services/overview.service';
6 import { overviewSubscriptionData, User } from './models/overview';
7 import { LoadingPopupComponent } from '../../../../../shared/pop-ups/loading-pop-up/loading-
8 paper-component';
9 import { TradeSwitchTableComponent } from './components/tradeSwitch-table/
10 tradeSwitchTablecomponent'; from './components/top-list/top-list.component';
11 import { RouterLink } from '@angular/router';
```

```

12 import { ApplicationContextService } from '../../../../../shared/context';
13 import { PlanService } from '../../../../../shared/services/planService';
14 import { SubscriptionService } from '../../../../../shared/services/subscription-service';
15
16 /**
17  * Main overview component for displaying dashboard statistics and user data.
18  *
19  * This component provides a comprehensive dashboard view that includes:
20  * - User statistics and metrics
21  * - Revenue calculations based on subscriptions
22  * - Top 10 users by profit
23  * - Subscription data overview
24  * - CSV export functionality with date range selection
25  *
26  * Related to:
27  * - OverviewService: Fetches user and subscription data
28  * - ApplicationContextService: Manages global overview data state
29  * - PlanService: Gets plan information for revenue calculation
30  * - SubscriptionService: Gets user subscription data
31  * - TradeSwitchTableComponent: Displays user table with filtering
32  * - TopListComponent: Displays top users list
33  *
34  * @component
35  * @selector app-overview
36  * @standalone true
37 */
38 @Component({
39   selector: 'app-overview',
40   imports: [
41     CommonModule,
42     statCardComponent,
43     LoadingPopupComponent,
44     FormsModule,
45     TradeSwitchTableComponent,
46     TopListComponent,
47     RouterLink,
48   ],
49   templateUrl: './overview.component.html',
50   styleUrls: ['./overview.component.scss'],
51   standalone: true,
52 })
53 export class Overview {
54   /** Top 10 users sorted by profit */
55   topUsers: User[] = [];
56
57 /**
58  * Constructor for Overview component.
59  *
60  * @param store - NgRx Store (injected but not currently used)
61  * @param overviewSvc - Service for fetching overview data
62  * @param appContext - Application context service for global state management
63  * @param planService - Service for fetching plan information
64  * @param subscriptionService - Service for fetching subscription data
65  */
66   constructor(
67     private store: Store,
68     private overviewSvc: OverviewService,
69     private appContext: ApplicationContextService,
70     private planService: PlanService,
71     private subscriptionService: SubscriptionService
72   ) {}
73
74   loading = false;
75   subscriptionsData: overviewSubscriptionData | null = null;
76   usersData: User[] = [];
77   newUsers = 0;
78   newUsersGrowthPercentage = 0;
79   calculatedRevenue = 0;
80   paidSubscriptions = 0;
81

```

```

82 // Loading granular por sección
83 private loadingStates = {
84   users: false,
85   cards: false,
86   revenue: false,
87   subscriptions: false,
88 };
89
90 // Export modal state
91 showExportModal = false;
92 exportStartDate: string = '';
93 exportEndDate: string = '';
94 exportError: string = '';
95 showDateDropdown = false;
96 // Calendar state (single-picker for range)
97 calYear = 0;
98 calMonth = 0; // 0-11
99 weeks: { date: Date; inMonth: boolean }[][] = [];
100
101 /**
102  * Builds a calendar grid for the export date picker.
103  *
104  * Creates a 6-week grid (42 days) starting from the Sunday of the week
105  * containing the first day of the specified month. Each day is marked
106  * with whether it belongs to the current month.
107  *
108  * @private
109  * @param year - Year for the calendar
110  * @param month - Month for the calendar (0-11, where 0 is January)
111  * @memberof Overview
112  */
113 private buildCalendar(year: number, month: number) {
114   // Start from Sunday of the week containing the 1st of the month
115   const firstOfMonth = new Date(year, month, 1);
116   const start = new Date(firstOfMonth);
117   const day = start.getDay(); // 0 Sun .. 6 Sat
118   start.setDate(start.getDate() - day);
119
120   const grid: { date: Date; inMonth: boolean }[] = [];
121   for (let i = 0; i < 42; i++) {
122     const d = new Date(start);
123     d.setDate(start.getDate() + i);
124     grid.push({ date: d, inMonth: d.getMonth() === month });
125   }
126   // chunk into weeks
127   this.weeks = [];
128   for (let i = 0; i < 6; i++) {
129     this.weeks.push(grid.slice(i * 7, i * 7 + 7));
130   }
131 }
132
133 /**
134  * Gets the formatted month and year label for the calendar.
135  *
136  * @returns Formatted string like "January 2024"
137  * @memberof Overview
138  */
139 get monthLabel(): string {
140   const m = new Date(this.calYear, this.calMonth, 1);
141   return m.toLocaleString(undefined, { month: 'long', year: 'numeric' });
142 }
143
144 /**
145  * Opens the export modal and initializes the calendar to current month.
146  *
147  * Resets export error and builds the calendar for the current date.
148  *
149  * @memberof Overview
150  */
151 openExportModal() {

```

```

152     this.showExportModal = true;
153     this.exportError = '';
154     const today = new Date();
155     this.calYear = today.getFullYear();
156     this.calMonth = today.getMonth();
157     this.buildCalendar(this.calYear, this.calMonth);
158 }
159 /**
160 * Initializes the component on load.
161 *
162 * Subscribes to context data and loads all configuration data
163 * including users, revenue, and subscription information.
164 *
165 * @memberof Overview
166 */
167 ngOnInit(): void {
168     this.subscribeToContextData();
169     this.loadConfig();
170 }
171 /**
172 * Subscribes to overview data from the application context.
173 *
174 * Listens to changes in overview data (users and subscriptions)
175 * and updates local component state when data changes.
176 *
177 * Related to:
178 * - ApplicationContextService.overviewData$: Observable of overview data
179 *
180 * @private
181 * @memberof Overview
182 */
183 private subscribeToContextData() {
184     // Suscribirse a los datos de overview desde el contexto
185     this.appContext.overviewData$.subscribe(data => {
186         this.userData = data.allUsers;
187         // Convertir subscriptions a overviewSubscriptionData si es necesario
188         this.subscriptionsData = data.subscriptions as any;
189     });
190     // No usamos el loading global del contexto aquí; control fino local
191 }
192 /**
193 * Loads all configuration data for the overview dashboard.
194 *
195 * Performs the following operations in sequence:
196 * 1. Resets all loading states
197 * 2. Loads user data
198 * 3. Calculates revenue
199 * 4. Loads subscription overview data
200 * 5. Checks if all data is loaded to hide loading indicator
201 *
202 * Related to:
203 * - getUsersData(): Fetches and processes user data
204 * - calculateRevenue(): Calculates total revenue from subscriptions
205 * - getOverviewSubscriptionData(): Fetches subscription statistics
206 * - checkAllLoaded(): Verifies all data is loaded
207 *
208 * @async
209 * @memberof Overview
210 */
211 async loadConfig() {
212     this.loading = true;
213     this.loadingStates = { users: false, cards: false, revenue: false, subscriptions:
214     false };
215     await this.getUsersData();
216     await this.calculateRevenue();
217     this.getOverviewSubscriptionData();
218     this.checkAllLoaded();
219 }
220
221

```

```

222     }
223
224     /**
225      * Fetches user data from Firebase and processes it.
226      *
227      * Retrieves all users, filters out admin users, calculates new users
228      * for today, and filters top 10 users by profit. Updates loading
229      * states when complete.
230      *
231      * Related to:
232      * - OverviewService.getUsersData(): Fetches users from Firebase
233      * - calculateNewUsers(): Calculates new users registered today
234      * - filterTop10Users(): Filters and sorts top users by profit
235      *
236      * @async
237      * @memberof Overview
238      */
239     async getUsersData() {
240       return this.overviewSvc
241         .getUsersData()
242         .then(docSnap) => {
243           if (docSnap && !docSnap.empty && docSnap.docs.length > 0) {
244             this.usersData = docSnap.docs
245               .map((doc) => doc.data() as User)
246               .filter((user) => !user.isAdmin);
247
248             // Calcular nuevos usuarios basándose en la fecha actual
249             this.calculateNewUsers();
250             this.filterTop10Users();
251             this.loadingStates.users = true;
252             this.loadingStates.cards = true; // top users y métricas listas
253             this.checkAllLoaded();
254           } else {
255             this.loadingStates.users = true;
256             this.loadingStates.cards = true;
257             this.checkAllLoaded();
258             console.warn('No config');
259           }
260         })
261         .catch((err) => {
262           this.loadingStates.users = true;
263           this.loadingStates.cards = true;
264           this.checkAllLoaded();
265         });
266     }
267
268     /**
269      * Fetches overview subscription data from Firebase.
270      *
271      * Retrieves subscription statistics including monthly revenue
272      * and user counts. Updates loading state when complete.
273      *
274      * Related to:
275      * - OverviewService.getOverviewSubscriptionData(): Fetches subscription data
276      *
277      * @memberof Overview
278      */
279     getOverviewSubscriptionData() {
280       this.overviewSvc
281         .getOverviewSubscriptionData()
282         .then(docSnap) => {
283           if (docSnap && !docSnap.empty && docSnap.docs.length > 0) {
284             const data = docSnap.docs[0].data() as overviewSubscriptionData;
285             this.subscriptionsData = data;
286             this.loadingStates.subscriptions = true;
287             this.checkAllLoaded();
288           } else {
289             console.warn('No config');
290             this.loadingStates.subscriptions = true;
291             this.checkAllLoaded();

```

```

292         }
293     })
294     .catch((err) => {
295       this.loadingStates.subscriptions = true;
296       this.checkAllLoaded();
297
298       console.error('Error to get the config', err);
299     });
300   }
301
302 /**
303  * Calculates the number of new users registered today.
304  *
305  * Filters users by subscription_date to find users registered
306  * between start and end of current day. Then calculates growth
307  * percentage.
308  *
309  * Related to:
310  * - calculateGrowthPercentage(): Calculates percentage of new users
311  *
312  * @memberof Overview
313  */
314 calculateNewUsers() {
315   const today = new Date();
316   const startOfDay = new Date(today.getFullYear(), today.getMonth(), today.getDate());
317   const endOfDay = new Date(today.getFullYear(), today.getMonth(), today.getDate() + 1);
318
319   // Convertir las fechas a timestamps para comparar con subscription_date
320   const startOfDayTimestamp = startOfDay.getTime();
321   const endOfDayTimestamp = endOfDay.getTime();
322
323   // Filtrar usuarios que se registraron hoy
324   this.newUsers = this.userData.filter(user =>
325     user.subscription_date >= startOfDayTimestamp &&
326     user.subscription_date < endOfDayTimestamp
327   ).length;
328
329   // Calcular el porcentaje de crecimiento
330   this.calculateGrowthPercentage();
331 }
332
333 /**
334  * Calculates the growth percentage of new users.
335  *
336  * Computes the percentage of new users relative to total users.
337  * Rounds to 1 decimal place. Returns 0 if there are no users.
338  *
339  * Formula: (newUsers / totalUsers) * 100
340  *
341  * @memberof Overview
342  */
343 calculateGrowthPercentage() {
344   const totalUsers = this.userData.length;
345
346   if (totalUsers === 0) {
347     this.newUsersGrowthPercentage = 0;
348     return;
349   }
350
351   // Calcular el porcentaje de nuevos usuarios respecto al total
352   // (nuevos usuarios / total usuarios) * 100
353   this.newUsersGrowthPercentage = (this.newUsers / totalUsers) * 100;
354
355   // Redondear a 1 decimal
356   this.newUsersGrowthPercentage = Math.round(this.newUsersGrowthPercentage * 10) / 10;
357 }
358
359 /**
360  * Calculates total revenue from user subscriptions.
361  *

```

```

362     * This method:
363     * 1. Loads all available plans
364     * 2. Counts users per plan by checking their latest subscription
365     * 3. Calculates revenue for each plan (userCount * planPrice)
366     * 4. Sums total revenue across all plans
367     * 5. Counts users with paid subscriptions (plans with price > 0)
368     *
369     * Related to:
370     * - PlanService.getAllPlans(): Gets all subscription plans
371     * - SubscriptionService.getUserLatestSubscription(): Gets user's current plan
372     *
373     * @async
374     * @memberof Overview
375     */
376     async calculateRevenue() {
377         try {
378             // Cargar todos los planes
379             const plans = await this.planService.getAllPlans();
380
381             if (plans.length === 0) {
382                 this.calculatedRevenue = 0;
383                 this.loadingStates.revenue = true;
384                 this.checkAllLoaded();
385                 return;
386             }
387
388             // Mapa para contar usuarios por plan
389             const planUserCountMap: { [planId: string]: number } = {};
390
391             // Inicializar contadores para cada plan
392             plans.forEach(plan => {
393                 planUserCountMap[plan.id] = 0;
394             });
395
396             // Recorrer todos los usuarios y obtener sus subscriptions
397             for (const user of this.userData) {
398                 try {
399                     const subscription = await
this.subscriptionService.getUserLatestSubscription(user.id);
400                     if (subscription && subscription.planId) {
401                         if (planUserCountMap.hasOwnProperty(subscription.planId)) {
402                             planUserCountMap[subscription.planId]++;
403                         }
404                     }
405                 } catch (error) {
406                     console.error(`Error obteniendo subscription para usuario ${user.id}:`, error);
407                 }
408             }
409
410             // Calcular el revenue total y contar usuarios con suscripciones pagas
411             let totalRevenue = 0;
412             let paidUsersCount = 0;
413
414             for (const plan of plans) {
415                 const userCount = planUserCountMap[plan.id] || 0;
416                 const planPrice = parseFloat(plan.price) || 0;
417                 const revenueForPlan = userCount * planPrice;
418                 totalRevenue += revenueForPlan;
419
420                 // Si el plan tiene precio > 0, contar esos usuarios como suscripciones pagas
421                 if (planPrice > 0 && userCount > 0) {
422                     paidUsersCount += userCount;
423                 }
424             }
425
426             this.calculatedRevenue = totalRevenue;
427             this.paidSubscriptions = paidUsersCount;
428             this.loadingStates.revenue = true;
429             this.checkAllLoaded();
430         } catch (error) {
431

```

```

432     console.error('Error calculando revenue:', error);
433     this.calculatedRevenue = 0;
434     this.loadingStates.revenue = true;
435     this.checkAllLoaded();
436   }
437 }
438 /**
439 * Filters and sorts users to get top 10 by profit.
440 *
441 * Filters users with profit > 0, sorts them in descending order
442 * by profit, and takes the first 10 users.
443 *
444 * @memberof Overview
445 */
446 filterTop10Users() {
447   this.topUsers = this.userData
448     .filter((user) => user.profit > 0)
449     .sort((a, b) => b.profit - a.profit)
450     .slice(0, 10);
451 }
452 }
453 /**
454 * Checks if all data sections have finished loading.
455 *
456 * Verifies that users, cards, revenue, and subscriptions data
457 * are all loaded. If all are loaded, hides the main loading indicator.
458 *
459 * @private
460 * @memberof Overview
461 */
462 private checkAllLoaded() {
463   const allLoaded = this.loadingStates.users && this.loadingStates.cards &&
464   this.loadingStates.revenue && this.loadingStates.subscriptions;
465   this.loading = false;
466 }
467 }
468 }
469 // ===== Export Data by Date =====
470 /**
471 * Closes the export modal and resets all export-related state.
472 *
473 * Clears export dates, errors, and hides the date dropdown.
474 *
475 * @memberof Overview
476 */
477 closeExportModal() {
478   this.showExportModal = false;
479   this.exportStartDate = '';
480   this.exportEndDate = '';
481   this.exportError = '';
482   this.showDateDropdown = false;
483 }
484 }
485 /**
486 * Handles date change in export modal.
487 *
488 * Resets end date if start date is not selected yet (UX requirement).
489 * Clears any export errors.
490 *
491 * @memberof Overview
492 */
493 onExportDateChange() {
494   // No hard validation for end-only; UX requires start first
495   if (!this.exportStartDate && this.exportEndDate) {
496     // reset end if start not picked yet
497     this.exportEndDate = '';
498   }
499   this.exportError = '';
500 }
501

```

```

502     }
503
504     /**
505      * Navigates to the previous month in the calendar.
506      *
507      * Updates calendar year and month, then rebuilds the calendar grid.
508      *
509      * @memberof Overview
510      */
511     prevMonth() {
512       const d = new Date(this.calYear, this.calMonth - 1, 1);
513       this.calYear = d.getFullYear();
514       this.calMonth = d.getMonth();
515       this.buildCalendar(this.calYear, this.calMonth);
516     }
517
518     /**
519      * Navigates to the next month in the calendar.
520      *
521      * Updates calendar year and month, then rebuilds the calendar grid.
522      *
523      * @memberof Overview
524      */
525     nextMonth() {
526       const d = new Date(this.calYear, this.calMonth + 1, 1);
527       this.calYear = d.getFullYear();
528       this.calMonth = d.getMonth();
529       this.buildCalendar(this.calYear, this.calMonth);
530     }
531
532     /**
533      * Handles day selection in the calendar for date range picker.
534      *
535      * Implements a two-click date range selection:
536      * - First click: Sets start date
537      * - Second click: Sets end date (if after start) or moves start (if before)
538      * - Third click: Starts new selection
539      *
540      * @param day - Selected date from calendar
541      * @memberof Overview
542      */
543     onDayPick(day: Date) {
544       const iso = (d: Date) => this.formatLocalDate(d);
545       if (!this.exportStartDate) {
546         this.exportStartDate = iso(day);
547         this.exportEndDate = '';
548         return;
549       }
550       const start = this.parseLocalDate(this.exportStartDate);
551       if (!this.exportEndDate) {
552         if (day < start) {
553           // If clicked before start, move start to that day
554           this.exportStartDate = iso(day);
555         } else {
556           this.exportEndDate = iso(day);
557         }
558         return;
559       }
560       // If both set, start a new selection
561       this.exportStartDate = iso(day);
562       this.exportEndDate = '';
563     }
564
565     /**
566      * Checks if a calendar day is within the selected date range.
567      *
568      * Returns true if the day falls between start and end dates
569      * (inclusive). Handles cases where only start date is selected.
570      *
571      * @param day - Date to check

```

```

572     * @returns true if day is in selected range, false otherwise
573     * @memberof Overview
574     */
575     isSelected(day: Date): boolean {
576         if (!this.exportStartDate && !this.exportEndDate) return false;
577         const time = new Date(day.getFullYear(), day.getMonth(), day.getDate()).getTime();
578         const start = this.exportStartDate
579             ? this.parseLocalDate(this.exportStartDate).setHours(0, 0, 0, 0)
580             : Number.NaN;
581         const end = this.exportEndDate
582             ? this.parseLocalDate(this.exportEndDate).setHours(23, 59, 59, 999)
583             : start;
584         return time >= start && time <= end;
585     }
586
587     /**
588      * Exports user data to CSV file with optional date filtering.
589      *
590      * This method:
591      * 1. Determines date range from selected dates (or exports all if no dates)
592      * 2. Filters users by subscription_date within the range
593      * 3. Generates CSV content with headers and user data
594      * 4. Creates a downloadable blob and triggers download
595      * 5. Closes the export modal
596      *
597      * CSV includes: User ID, Name, Email, Status, Strategies, Trading Accounts,
598      * Strategy Followed %, Net PnL, Profit, Best Trade, Subscription Date
599      *
600      * Related to:
601      * - parseLocalDate(): Parses date strings
602      * - escapeCsv(): Escapes special characters in CSV values
603      * - closeExportModal(): Closes modal after export
604      *
605      * @memberof Overview
606      */
607     exportDataAsCSV() {
608         // Determine range
609         let startTs: number | null = null;
610         let endTs: number | null = null;
611
612         if (this.exportStartDate) {
613             const start = this.parseLocalDate(this.exportStartDate);
614             // start of day
615             startTs = new Date(start.getFullYear(), start.getMonth(), start.getDate()).getTime();
616             if (this.exportEndDate) {
617                 const end = this.parseLocalDate(this.exportEndDate);
618                 // end of day (inclusive)
619                 endTs = new Date(end.getFullYear(), end.getMonth(), end.getDate(), 23, 59, 59,
620 999).getTime();
621                 // If only start provided, range is only that day
622                 endTs = new Date(start.getFullYear(), start.getMonth(), start.getDate(), 23, 59, 59,
623 999).getTime();
624             }
625
626             // Filter users by subscription_date (fallback to lastUpdated if needed)
627             const filtered = this.userData.filter(u => {
628                 const ts = (u.subscription_date ?? u.lastUpdated) || 0;
629                 if (startTs !== null && endTs !== null) {
630                     return ts >= startTs && ts <= endTs;
631                 }
632                 return true; // no dates -> export all
633             });
634
635             const rows: string[] = [];
636             // Header
637             rows.push([
638                 'User ID', 'First Name', 'Last Name', 'Email', 'Status', 'Strategies', 'Trading
639                 Account', 'Strategy Followed', 'Net PnL', 'Profit', 'Best Trade', 'Subscription Date'
640             ]);
641             // Data

```

```

642  for (const u of filtered) {
643    const subDate = u.subscription_date ? new Date(u.subscription_date).toISOString() : '';
644    rows.push([
645      `${u.id ?? ''}`,
646      this.escapeCsv(u.firstName ?? ''),
647      this.escapeCsv(u.lastName ?? ''),
648      this.escapeCsv(u.email ?? ''),
649      `${u.status ?? ''}`,
650      `${u.strategies ?? 0}`,
651      `${u.trading_accounts ?? 0}`,
652      `${u.strategy_followed ?? 0}`,
653      `${u.netPnl ?? 0}`,
654      `${u.profit ?? 0}`,
655      `${u.best_trade ?? 0}`,
656      subDate,
657    ].join(',')));
658  }
659
660  const csvContent = rows.join('\n');
661  const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8;' });
662  const url = URL.createObjectURL(blob);
663  const link = document.createElement('a');
664  link.href = url;
665  const filename = this.exportStartDate || this.exportEndDate ?
`export${Date.now()}${ownloadExportFileName}`;
666  link.click();
667  URL.revokeObjectURL(url);
668
669  this.closeExportModal();
670 }
671
672 /**
673  * Escapes special characters in CSV values.
674  *
675  * Wraps value in quotes if it contains comma, quote, or newline.
676  * Doubles any quotes within the value.
677  *
678  * @private
679  * @param value - String value to escape
680  * @returns Escaped CSV value
681  * @memberof Overview
682  */
683 private escapeCsv(value: string): string {
684   if (value.includes(',') || value.includes('"') || value.includes('\n')) {
685     return '"' + value.replace(/\"/g, '\"\"') + '"';
686   }
687   return value;
688 }
689
690 /**
691  * Formats a Date object to YYYY-MM-DD string format.
692  *
693  * @private
694  * @param d - Date to format
695  * @returns Formatted date string (YYYY-MM-DD)
696  * @memberof Overview
697  */
698 private formatLocalDate(d: Date): string {
699   const y = d.getFullYear();
700   const m = String(d.getMonth() + 1).padStart(2, '0');
701   const day = String(d.getDate()).padStart(2, '0');
702   return `${y}-${m}-${day}`;
703 }
704
705 /**
706  * Parses a YYYY-MM-DD string to a Date object.
707  *
708  * @private
709  * @param s - Date string in YYYY-MM-DD format
710  * @returns Parsed Date object
711

```

```

712     * @memberof Overview
713     */
714     private parseLocalDate(s: string): Date {
715         // Expecting YYYY-MM-DD
716         const [y, m, d] = s.split('-').map(Number);
717         return new Date(y, (m || 1) - 1, d || 1);
718     }
719 }
720

```

Ø=ÜÁ features\overview\components\top-list

Ø=ÜÄ features\overview\components\top-list\top-list.component.ts

```

1  import { CommonModule } from '@angular/common';
2  import { Component, Input } from '@angular/core';
3  import { User, UserStatus } from '../../../../../models/overview';
4
5  /**
6   * Component for displaying a single user in the top users list.
7   *
8   * This component displays user information in a card format,
9   * showing user initials, name, and profit. It's used in the
10  * overview dashboard to show the top 10 users by profit.
11  *
12  * Related to:
13  * - OverviewComponent: Passes user data as Input
14  *
15  * @component
16  * @selector app-top-list
17  * @standalone true
18  */
19 @Component({
20   selector: 'app-top-list',
21   templateUrl: './top-list.component.html',
22   styleUrls: ['./top-list.component.scss'],
23   standalone: true,
24   imports: [CommonModule],
25 })
26 export class TopListComponent {
27   @Input() user: User = {
28     best_trade: 0,
29     birthday: new Date(),
30     firstName: '',
31     id: '',
32     lastName: '',
33     netPnl: 0,
34     number_trades: 0,
35     phoneNumber: '',
36     profit: 0,
37     status: UserStatus.CREATED,
38     strategy_followed: 0,
39     subscription_date: 0,
40     tokenId: '',
41     email: '',
42     total_spend: 0,
43     isAdmin: false,
44     lastUpdated: 0,
45     trading_accounts: 0,
46     strategies: 0,
47   };
48   constructor() {}

```

```

50
51  /**
52   * Gets user initials from first and last name.
53   *
54   * Takes the first character of firstName and lastName,
55   * converts them to uppercase, and concatenates them.
56   *
57   * @param user - User object containing firstName and lastName
58   * @returns Two-letter initials string (e.g., "JD" for John Doe)
59   * @memberof TopListComponent
60   */
61 onlyNameInitials(user: User) {
62   return (
63     user.firstName.charAt(0).toUpperCase() +
64     user.lastName.charAt(0).toUpperCase()
65   );
66 }
67
68 /**
69  * Formats profit value for display.
70  *
71  * Formats profit with appropriate currency symbol and scaling:
72  * - $0 for zero
73  * - $X.XX for values less than 1000
74  * - $X.XK for values 1000 or greater (e.g., $5.2K for 5200)
75  *
76  * @param profit - Profit value to format
77  * @returns Formatted profit string
78  * @memberof TopListComponent
79  */
80 formatProfit(profit: number): string {
81   if (profit === 0) return '$0';
82   if (Math.abs(profit) < 1000) {
83     return `$$${profit.toFixed(2)}`;
84   }
85   const k = profit / 1000;
86   return `$$${k.toFixed(1)}K`;
87 }
88 }
89

```

Ø=ÜÁ features\overview\components\tradeSwitch-table

Ø=ÜÄ features\overview\components\tradeSwitch-table\tradeSwitchTable.component.ts

```

1 import { Component, Input, Injectable, HostListener, ElementRef, ViewChild, OnInit,
2   CommonModule } from '@angular/common';
3 import { User, UserStatus } from '../../../../../models/overview';
4 import { FormsModule } from '@angular/forms';
5 import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';
6 import { AuthService } from '../../../../../features/auth/service/authService';
7 import { AccountData } from '../../../../../features/auth/models/userModel';
8
9 /**
10  * Component for displaying a comprehensive user table with filtering and pagination.
11  *
12  * This component provides a detailed table view of all users with the following features:
13  * - Search by user name
14  * - Filter by status, strategy followed percentage, number of strategies, and trading
15  * accounts
16  * - Expandable rows showing user's trading accounts
17  * - Pagination for large datasets
18  * - CSV-friendly data display
19  *

```

```

19 * Related to:
20 * - OverviewComponent: Receives users array as Input
21 * - AuthService: Fetches user trading accounts
22 * - NumberFormatterService: Formats currency values
23 *
24 * @component
25 * @selector app-trade-switch-table
26 * @standalone true
27 */
28 @Component({
29   selector: 'app-trade-switch-table',
30   standalone: true,
31   imports: [CommonModule, FormsModule],
32   templateUrl: './tradeSwitchTable.component.html',
33   styleUrls: ['./tradeSwitchTable.component.scss'],
34 })
35 @Injectable()
36 export class TradeSwitchTableComponent implements OnInit, OnDestroy {
37   @Input() users: User[] = [];
38   @ViewChild('filterModal', { static: false }) filterModal!: ElementRef;
39   @ViewChild('filterButton', { static: false }) filterButton!: ElementRef;
40
41   // Mapa de userId -> AccountData[]
42   userAccountsMap: Map<string, AccountData[]> = new Map();
43   // Set de userIds expandidos
44   expandedUsers: Set<string> = new Set();
45   // Flag para saber si las cuentas están cargadas
46   accountsLoaded = false;
47
48   // Valores iniciales (usados en el formulario)
49   initialStatus: UserStatus | string = '';
50   initialMinStrat: number = 0;
51   initialMaxStrat: number = 100;
52   initialStrategies: number = 0; // 0-8, si es 0 no filtra
53   initialTradingAccounts: number = 0; // 0-8, si es 0 no filtra
54
55   // Valores aplicados (usados para filtrar)
56   appliedStatus: UserStatus | string = '';
57   appliedMinStrat: number = 0;
58   appliedMaxStrat: number = 100;
59   appliedStrategies: number = 0; // 0-8, si es 0 no filtra
60   appliedTradingAccounts: number = 0; // 0-8, si es 0 no filtra
61
62   showFilter = false;
63   currentPage: number = 1;
64   itemsPerPage: number = 10;
65
66   private numberFormatter = new NumberFormatterService();
67
68   constructor(private authService: AuthService) {}
69
70   ngOnInit() {
71     this.loadAllAccounts();
72   }
73
74   ngOnDestroy() {
75     // Cleanup si es necesario
76   }
77
78   /**
79    * Loads all trading accounts and groups them by userId.
80    *
81    * Fetches all accounts from Firebase and creates a map
82    * where each userId maps to an array of their accounts.
83    *
84    * Related to:
85    * - AuthService.getAllAccounts(): Fetches all accounts from Firebase
86    *
87    * @async
88    * @memberof TradeSwitchTableComponent

```

```

89     */
90     async loadAllAccounts() {
91         try {
92             const allAccounts = await this.authService.getAllAccounts();
93             if (allAccounts) {
94                 // Agrupar cuentas por userId
95                 const accountsMap = new Map<string, AccountData[]>();
96                 allAccounts.forEach(account => {
97                     if (!accountsMap.has(account.userId)) {
98                         accountsMap.set(account.userId, []);
99                     }
100                    accountsMap.get(account.userId)! .push(account);
101                });
102                this.userAccountsMap = accountsMap;
103            }
104            this.accountsLoaded = true;
105        } catch (error) {
106            console.error('Error loading accounts:', error);
107            this.accountsLoaded = true;
108        }
109    }
110
111 /**
112 * Gets trading accounts for a specific user.
113 *
114 * @param userId - User ID to get accounts for
115 * @returns Array of AccountData for the user, or empty array if none
116 * @memberof TradeSwitchTableComponent
117 */
118 getUserAccounts(userId: string): AccountData[] {
119     return this.userAccountsMap.get(userId) || [];
120 }
121
122 /**
123 * Checks if a user has any trading accounts.
124 *
125 * @param userId - User ID to check
126 * @returns true if user has accounts, false otherwise
127 * @memberof TradeSwitchTableComponent
128 */
129 hasAccounts(userId: string): boolean {
130     const accounts = this.userAccountsMap.get(userId);
131     return accounts ? accounts.length > 0 : false;
132 }
133
134 /**
135 * Toggles the expansion state of a user row.
136 *
137 * Adds or removes the userId from the expandedUsers set
138 * to show/hide the user's trading accounts.
139 *
140 * @param userId - User ID to toggle expansion for
141 * @memberof TradeSwitchTableComponent
142 */
143 toggleExpand(userId: string) {
144     if (this.expandedUsers.has(userId)) {
145         this.expandedUsers.delete(userId);
146     } else {
147         this.expandedUsers.add(userId);
148     }
149 }
150
151 /**
152 * Checks if a user row is currently expanded.
153 *
154 * @param userId - User ID to check
155 * @returns true if user row is expanded, false otherwise
156 * @memberof TradeSwitchTableComponent
157 */
158 isExpanded(userId: string): boolean {

```

```

159     return this.expandedUsers.has(userId);
160 }
161
162 private _searchTerm = '';
163
164 /**
165  * Gets the current search term.
166  *
167  * @returns Current search term string
168  * @memberof TradeSwitchTableComponent
169  */
170 get searchTerm(): string {
171     return this._searchTerm;
172 }
173
174 /**
175  * Sets the search term and resets to first page.
176  *
177  * When search term changes, automatically navigates to page 1
178  * to show filtered results from the beginning.
179  *
180  * @param val - New search term value
181  * @memberof TradeSwitchTableComponent
182  */
183 set searchTerm(val: string) {
184     this._searchTerm = val;
185     this.goToPage(1);
186 }
187
188 /**
189  * Host listener for clicks outside the filter modal.
190  *
191  * Closes the filter modal when user clicks outside of it
192  * or the filter button. Prevents closing when clicking inside.
193  *
194  * @param event - Mouse click event
195  * @memberof TradeSwitchTableComponent
196  */
197 @HostListener('document:click', ['$event'])
198 onClickOutside(event: MouseEvent) {
199     if (this.showFilter &&
200         this.filterModal?.nativeElement &&
201         this.filterButton?.nativeElement &&
202         !this.filterModal.nativeElement.contains(event.target) &&
203         !this.filterButton.nativeElement.contains(event.target)) {
204         this.closeFilter();
205     }
206 }
207
208 /**
209  * Getter that returns filtered users based on search term and applied filters.
210  *
211  * Filters users by:
212  * - Search term: Matches against first name and last name
213  * - Status: Matches user's display status
214  * - Strategy followed: Range between min and max percentage
215  * - Strategies: Exact number match (0 means no filter)
216  * - Trading accounts: Exact number match (0 means no filter)
217  *
218  * @returns Array of filtered User objects
219  * @memberof TradeSwitchTableComponent
220  */
221 get filteredUsers(): User[] {
222     const lower = this._searchTerm.trim().toLowerCase();
223
224     return this.users.filter((user) => {
225         const matchesSearch = `${user.firstName.split(' ')[0]} ${
226             user.lastName.split(' ')[0]
227         }`  

228             .toLowerCase()

```

```

229     .includes(lower);
230
231     const userDisplayStatus = this.statusClass(user);
232     const matchesStatus =
233         !this.appliedStatus || this.appliedStatus === '' || userDisplayStatus ===
234     this.appliedStatus;
235         let matchesMinStrat = user.strategy_followed !== undefined && user.strategy_followed
236     >= (this.appliedMinStrat ?? 0) && user.strategy_followed !== undefined && user.strategy_followed
237     <= (this.appliedMaxStrat ?? 100);
238         if (user.strategy_followed === undefined) {
239             matchesMinStrat = false;
240             matchesMaxStrat = false;
241         }
242
243         // Si appliedStrategies es 0, no se filtra; si es > 0, se filtra por ese número exacto
244         const matchesStrategies = this.appliedStrategies === 0 || (user.strategies ?? 0) ===
245     this.appliedStrategies;
246         // Si appliedTradingAccounts es 0, no se filtra; si es > 0, se filtra por ese número
247         const matchesTradingAccounts = this.appliedTradingAccounts === 0 ||
248     (user.trading_accounts ?? 0) === this.appliedTradingAccounts;
249         return (
250             matchesSearch &&
251             matchesStatus &&
252             matchesMinStrat &&
253             matchesMaxStrat &&
254             matchesStrategies &&
255             matchesTradingAccounts
256         );
257     });
258 }
259
260 /**
261 * Getter that returns paginated users for current page.
262 *
263 * Calculates the slice of filtered users to display based on
264 * currentPage and itemsPerPage.
265 *
266 * @returns Array of users for current page
267 * @memberof TradeSwitchTableComponent
268 */
269 get paginatedUsers(): User[] {
270     const start = (this.currentPage - 1) * this.itemsPerPage;
271     const end = start + this.itemsPerPage;
272     return this.filteredUsers.slice(start, end);
273 }
274
275 /**
276 * Getter that calculates total number of pages.
277 *
278 * Based on filtered users count and items per page.
279 *
280 * @returns Total number of pages
281 * @memberof TradeSwitchTableComponent
282 */
283 get totalPages(): number {
284     return Math.ceil(this.filteredUsers.length / this.itemsPerPage);
285 }
286
287 /**
288 * Determines the display status class for a user.
289 *
290 * Status logic:
291 * - "banned" if user status is banned
292 * - "created" if all user metrics are zero (new/inactive user)
293 * - "active" if user has any activity (non-zero metrics)
294 *
295 * @param user - User object to determine status for
296 * @returns Status class string ('banned', 'created', or 'active')
297 * @memberof TradeSwitchTableComponent
298 */

```

```

299  statusClass(user: User): string {
300    // Si el status es banned, retornar banned
301    if (String(user.status) === 'banned') {
302      return 'banned';
303    }
304
305    // Verificar si todos los valores están en 0
306    const allValuesZero =
307      (user.trading_accounts ?? 0) === 0 &&
308      (user.strategies ?? 0) === 0 &&
309      (user.strategy_followed ?? 0) === 0 &&
310      (user.netPnl ?? 0) === 0 &&
311      (user.profit ?? 0) === 0 &&
312      (user.number_trades ?? 0) === 0 &&
313      (user.total_spend ?? 0) === 0;
314
315    // Si todos los valores están en 0, retornar created
316    if (allValuesZero) {
317      return 'created';
318    }
319
320    // Si no todos están en 0, retornar active
321    return 'active';
322  }
323
324  /**
325   * Gets display status string with capitalized first letter.
326   *
327   * @param user - User object to get status for
328   * @returns Capitalized status string (e.g., "Active", "Created", "Banned")
329   * @memberof TradeSwitchTableComponent
330   */
331  getDisplayStatus(user: User): string {
332    const status = this.statusClass(user);
333    return status.charAt(0).toUpperCase() + status.slice(1);
334  }
335
336  /**
337   * Determines CSS class for return value display.
338   *
339   * Returns 'green' for positive or zero values, 'red' for negative values.
340   *
341   * @param returnValue - Numeric value to determine class for
342   * @returns CSS class string ('green' or 'red')
343   * @memberof TradeSwitchTableComponent
344   */
345  returnClass(returnValue: number) {
346    return returnValue >= 0 ? 'green' : 'red';
347  }
348
349  /**
350   * Toggles the filter panel visibility.
351   *
352   * @memberof TradeSwitchTableComponent
353   */
354  openFilter() {
355    this.showFilter = !this.showFilter;
356  }
357
358  /**
359   * Closes the filter panel.
360   *
361   * @memberof TradeSwitchTableComponent
362   */
363  closeFilter() {
364    this.showFilter = false;
365  }
366
367  /**
368   * Applies filter values from initial to applied state.

```

```

369     *
370     * Copies all initial filter values to applied values,
371     * closes the filter panel, and applies the filters.
372     *
373     * Related to:
374     * - applyFilters(): Applies the filters and resets pagination
375     *
376     * @memberof TradeSwitchTableComponent
377     */
378   apply() {
379     // Aplicar los valores iniciales a los aplicados
380     this.appliedStatus = this.initialStatus;
381     this.appliedMinStrat = this.initialMinStrat;
382     this.appliedMaxStrat = this.initialMaxStrat;
383     this.appliedStrategies = this.initialStrategies;
384     this.appliedTradingAccounts = this.initialTradingAccounts;
385
386     this.showFilter = false;
387     this.applyFilters();
388   }
389
390 /**
391  * Applies filters and resets to first page.
392  *
393  * Navigates to page 1 when filters are applied to show
394  * filtered results from the beginning.
395  *
396  * @memberof TradeSwitchTableComponent
397  */
398   applyFilters() {
399     this.goToPage(1);
400   }
401
402 /**
403  * Resets all filters to default values.
404  *
405  * Clears both initial and applied filter values,
406  * closes the filter panel, and applies the reset filters.
407  *
408  * @memberof TradeSwitchTableComponent
409  */
410   resetFilters() {
411     this.initialStatus = '';
412     this.initialMinStrat = 0;
413     this.initialMaxStrat = 100;
414     this.initialStrategies = 0;
415     this.initialTradingAccounts = 0;
416
417     this.appliedStatus = '';
418     this.appliedMinStrat = 0;
419     this.appliedMaxStrat = 100;
420     this.appliedStrategies = 0;
421     this.appliedTradingAccounts = 0;
422
423     this.showFilter = false; // Cerrar el filter
424     this.applyFilters();
425   }
426
427 /**
428  * Getter that returns the strategy followed percentage range as a formatted string.
429  *
430  * Ensures min is not greater than max by using Math.min/max.
431  * Used for real-time display in the filter UI.
432  *
433  * @returns Formatted range string (e.g., "0% - 100%")
434  * @memberof TradeSwitchTableComponent
435  */
436   get stratFollowedRange(): string {
437     // Asegurar que min no sea mayor que max
438     const min = Math.min(this.initialMinStrat, this.initialMaxStrat);

```

```

439     const max = Math.max(this.initialMinStrat, this.initialMaxStrat);
440     return `${min}% - ${max}%`;
441 }
442
443 /**
444 * Ensures minStrat is not greater than maxStrat.
445 *
446 * If minStrat exceeds maxStrat, swaps the values.
447 * Called when minStrat input changes.
448 *
449 * @memberof TradeSwitchTableComponent
450 */
451 onMinStratChange() {
452     if (this.initialMinStrat > this.initialMaxStrat) {
453         const temp = this.initialMinStrat;
454         this.initialMinStrat = this.initialMaxStrat;
455         this.initialMaxStrat = temp;
456     }
457 }
458
459 /**
460 * Ensures maxStrat is not less than minStrat.
461 *
462 * If maxStrat is less than minStrat, swaps the values.
463 * Called when maxStrat input changes.
464 *
465 * @memberof TradeSwitchTableComponent
466 */
467 onMaxStratChange() {
468     if (this.initialMaxStrat < this.initialMinStrat) {
469         const temp = this.initialMaxStrat;
470         this.initialMaxStrat = this.initialMinStrat;
471         this.initialMinStrat = temp;
472     }
473 }
474
475 /**
476 * Gets user initials from first and last name.
477 *
478 * Takes first character of firstName and lastName and concatenates them.
479 *
480 * @param user - User object containing firstName and lastName
481 * @returns Two-letter initials string
482 * @memberof TradeSwitchTableComponent
483 */
484 onlyNameInitials(user: User) {
485     return user.firstName.charAt(0) + user.lastName.charAt(0);
486 }
487
488 /**
489 * Navigates to a specific page.
490 *
491 * Validates page number is within valid range (1 to totalPages)
492 * and updates currentPage.
493 *
494 * @param page - Page number to navigate to
495 * @memberof TradeSwitchTableComponent
496 */
497 goToPage(page: number) {
498     if (page < 1) page = 1;
499     if (page > this.totalPages) page = this.totalPages;
500     this.currentPage = page;
501 }
502
503 /**
504 * Navigates to the previous page.
505 *
506 * @memberof TradeSwitchTableComponent
507 */
508 prevPage() {

```

```

509     this.goToPage(this.currentPage - 1);
510 }
511
512 /**
513 * Navigates to the next page.
514 *
515 * @memberof TradeSwitchTableComponent
516 */
517 nextPage() {
518     this.goToPage(this.currentPage + 1);
519 }
520
521 /**
522 * Formats a numeric value as currency.
523 *
524 * Uses NumberFormatterService to format the value with
525 * appropriate currency symbol and formatting.
526 *
527 * Related to:
528 * - NumberFormatterService.formatCurrency(): Formats currency values
529 *
530 * @param value - Numeric value to format (can be null or undefined)
531 * @returns Formatted currency string
532 * @memberof TradeSwitchTableComponent
533 */
534 formatCurrency(value: number | null | undefined): string {
535     return this.numberFormatter.formatCurrency(value);
536 }
537
538

```

Ø=ÜÁ features\overview\models

Ø=ÜÁ features\overview\models\overview.ts

```

1 /**
2 * Enumeration of possible user statuses in the system.
3 *
4 * Represents the different states a user account can have,
5 * from creation to active use, cancellation, or administrative actions.
6 *
7 * @enum UserStatus
8 */
9 export enum UserStatus {
10     ADMIN = 'admin',
11     CREATED = 'created',
12     PURCHASED = 'purchased',
13     PENDING = 'pending',
14     ACTIVE = 'active',
15     PROCESSING = 'processing',
16     CANCELLED = 'cancelled',
17     EXPIRED = 'expired',
18     BANNED = 'banned',
19 }
20
21 /**
22 * Interface representing subscription overview statistics.
23 *
24 * Contains aggregated data about subscriptions for a specific month,
25 * including revenue and user count.
26 *
27 * Used in:
28 * - OverviewComponent: Displays subscription statistics

```

```

29 * - OverviewService: Fetches subscription data from Firebase
30 *
31 * @interface overviewSubscriptionData
32 */
33 export interface overviewSubscriptionData {
34   month: string;
35   revenue: number;
36   users: number;
37 }
38
39 /**
40 * Interface representing a user in the system.
41 *
42 * Contains comprehensive user information including personal data,
43 * trading statistics, account information, and subscription details.
44 *
45 * Used throughout the overview module for displaying user data,
46 * calculating statistics, and filtering users.
47 *
48 * @interface User
49 */
50 export interface User {
51   id: any;
52   best_trade: number;
53   birthday: Date;
54   email: string;
55   firstName: string;
56   lastName: string;
57   netPnl: number;
58   number_trades: number;
59   phoneNumber: string;
60   profit: number;
61   status: UserStatus;
62   strategy_followed: number;
63   subscription_date: number;
64   lastUpdated: number;
65   tokenId: string;
66   total_spend: number;
67   trading_accounts: number;
68   strategies: number;
69   isAdmin: boolean;
70 }
71

```

Ø=ÜÁ features\overview\services

Ø=ÜÁ features\overview\services\overview.service.ts

```

1 import { Injectable } from '@angular/core';
2 import { OverviewDataService } from '../../../../../shared/services/overview-data.service';
3 import { AppContextService } from '../../../../../shared/context';
4
5 /**
6 * Service for fetching and managing overview data.
7 *
8 * This service acts as a bridge between the OverviewComponent and
9 * the data layer, handling loading states and error management
10 * through the AppContextService.
11 *
12 * Related to:
13 * - OverviewDataService: Fetches raw data from Firebase
14 * - AppContextService: Manages global loading states and data updates
15 *

```

```

16   * @injectable
17   * @providedIn root
18   */
19 @Injectable({ providedIn: 'root' })
20 export class OverviewService {
21   /**
22    * Constructor for OverviewService.
23    *
24    * @param overviewDataService - Service for fetching overview data from Firebase
25    * @param applicationContext - Application context service for state management
26    */
27   constructor(
28     private overviewDataService: OverviewDataService,
29     private applicationContext: ApplicationContextService
30   ) {}
31
32   /**
33    * Fetches overview subscription data from Firebase.
34    *
35    * Manages loading state and error handling through ApplicationContextService.
36    * Updates the global overview subscriptions data when successful.
37    *
38    * Related to:
39    * - OverviewDataService.getOverviewSubscriptionData(): Fetches data from Firebase
40    * - ApplicationContextService.setLoading(): Manages loading state
41    * - ApplicationContextService.updateOverviewSubscriptions(): Updates global state
42    * - ApplicationContextService.setError(): Manages error state
43    *
44    * @async
45    * @returns Promise resolving to subscription data document snapshot
46    * @throws Error if data fetch fails
47    */
48   async getOverviewSubscriptionData() {
49     this.applicationContext.setLoading('overview', true);
50     this.applicationContext.setError('overview', null);
51
52     try {
53       const subscriptions = await this.overviewDataService.getOverviewSubscriptionData();
54       this.applicationContext.updateOverviewSubscriptions(subscriptions?.docs?.map(doc =>
55         doc.data()));
56       this.applicationContext.setLoading('overview', false);
57       return subscriptions;
58     } catch (error) {
59       this.applicationContext.setLoading('overview', false);
60       this.applicationContext.setError('overview', 'Error al obtener datos de suscripciones');
61       throw error;
62     }
63   }
64   /**
65    * Fetches user data from Firebase.
66    *
67    * Manages loading state and error handling through ApplicationContextService.
68    * Updates the global overview users data when successful.
69    *
70    * Related to:
71    * - OverviewDataService.getUsersData(): Fetches data from Firebase
72    * - ApplicationContextService.setLoading(): Manages loading state
73    * - ApplicationContextService.updateOverviewUsers(): Updates global state
74    * - ApplicationContextService.setError(): Manages error state
75    *
76    * @async
77    * @returns Promise resolving to users data document snapshot
78    * @throws Error if data fetch fails
79    */
80   async getUsersData() {
81     this.applicationContext.setLoading('overview', true);
82     this.applicationContext.setError('overview', null);
83
84     try {
85       const users = await this.overviewDataService.getUsersData();

```

```

86     const usersData = users?.docs?.map(doc => doc.data() as any) || [];
87     this.appContext.updateOverviewUsers(usersData);
88     this.appContext.setLoading('overview', false);
89     return users;
90   } catch (error) {
91     this.appContext.setLoading('overview', false);
92     this.appContext.setError('overview', 'Error al obtener datos de usuarios');
93     throw error;
94   }
95 }
96
97 }
```

Ø=ÜÁ features\report

Ø=ÜÁ features\report\report.component.ts

```

1 import { Component, Inject, OnInit, PLATFORM_ID } from '@angular/core';
2 import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3 import { HttpClient, HttpHeaders } from '@angular/common/http';
4 import { CommonModule, isPlatformBrowser } from '@angular/common';
5 import { FormsModule, ReactiveFormsModule } from '@angular/forms';
6 import { Store } from '@ngrx/store';
7 import { AppContextService } from '../../../../../shared/context';
8 import {
9   getUserKey,
10  setAvgWnL,
11  setGroupedTrades,
12  setNetPnL,
13  setProfitFactor,
14  setTotalTrades,
15  setTradeWin,
16  setUserKey,
17 } from './store/report.actions';
18 import { selectGroupedTrades, selectReport } from './store/report.selectors';
19 import { interval, last, map, Subscription } from 'rxjs';
20 import { ReportService } from './service/report.service';
21 import {
22   displayConfigData,
23   GroupedTrade,
24   GroupedTradeFinal,
25   MonthlyReport,
26   PluginHistoryRecord,
27   StatConfig,
28 } from './models/report.model';
29 import {
30   calculateAvgWinLossTrades,
31   calculateNetPnl,
32   calculateProfitFactor,
33   calculateTotalTrades,
34   calculateTradeWinPercent,
35 } from './utils/normalization-utils';
36 import { statCardComponent } from './components/statCard/stat_card.component';
37 import { PnlGraphComponent } from './components/pnlGraph/pnlGraph.component';
38 import { CalendarComponent } from './components/calendar/calendar.component';
39 import { SettingsService } from '../strategy/service/strategy.service';
40 import { resetConfig } from '../strategy/store/strategy.actions';
41 import { ConfigurationOverview, RuleType, StrategyState } from '../strategy/models';
42 import { winLossChartComponent } from './components/winLossChart/win-loss-chart.component';
43 import moment from 'moment-timezone';
44 import { Router } from '@angular/router';
45 import { User } from '../overview/models/overview';
46 import { selectUser } from '../auth/store/user.selectios';
```

```

47 import { AuthService } from '../auth/service/authService';
48 import { getBestTrade, getTotalSpend } from './utils/firebase-data-utils';
49 import { Timestamp } from 'firebase/firestore';
50 import { initialStrategyState } from '../strategy/store/strategy.reducer';
51 import { AccountData } from '../auth/models/userModel';
52 import { PlanLimitationsGuard } from '../../guards/plan-limitations.guard';
53 import { PlanLimitationModalData } from '../../shared/interfaces/plan-limitation-
54 modal-interface';
55 import { PlanLimitationModalComponent } from '../../shared/components/plan-limitation-modal/
56 planlimitationmodalcomponent';'....../shared/components/strategy-card/strategy-
57 cardinterface';'....../shared/components/loading-spinner/loading-
58 spinner{compartmantHistoryService, PluginHistory } from '../../shared/services/plugin-
59 history{stimesonheService } from '../../shared/services/timezone.service';
60 /**
61 * Main component for displaying trading reports and analytics.
62 *
63 * This component is the central hub for displaying comprehensive trading data including:
64 * - Trading statistics (Net PnL, Win Rate, Profit Factor, etc.)
65 * - PnL charts with monthly/yearly views
66 * - Calendar view of trades with strategy compliance
67 * - Win/Loss ratio visualization
68 * - Account balance information
69 * - Strategy configuration display
70 *
71 * Key Features:
72 * - Fetches trading history from TradeLocker API
73 * - Processes and groups trades by position
74 * - Calculates trading statistics
75 * - Manages multiple trading accounts
76 * - Caches data in localStorage for performance
77 * - Updates monthly reports in Firebase
78 * - Handles plan limitations and access control
79 *
80 * Data Flow:
81 * 1. Component initializes and loads saved data from localStorage
82 * 2. Subscribes to ApplicationContextService for user, accounts, and strategies
83 * 3. Fetches fresh data from API for current account
84 * 4. Processes trades and calculates statistics
85 * 5. Updates NgRx store and ApplicationContextService
86 * 6. Displays data in child components (charts, calendar, stats)
87 *
88 * Relations:
89 * - ReportService: Fetches trading data from API
90 * - ApplicationContextService: Global state management
91 * - Store (NgRx): Local state for report data
92 * - AuthService: User authentication and account management
93 * - SettingsService: Strategy configuration
94 * - CalendarComponent: Calendar view of trades
95 * - PnlGraphComponent: PnL chart visualization
96 * - WinLossChartComponent: Win/loss ratio chart
97 * - statCardComponent: Individual statistic cards
98 *
99 * @component
100 * @selector app-report
101 * @standalone true
102 */
103 @Component({
104   selector: 'app-report',
105   templateUrl: './report.component.html',
106   styleUrls: ['./report.component.scss'],
107   standalone: true,
108   imports: [
109     CommonModule,
110     FormsModule,
111     ReactiveFormsModule,
112     statCardComponent,
113     PnlGraphComponent,
114     CalendarComponent,
115     WinLossChartComponent,
116     PlanLimitationModalComponent,

```

```

117     LoadingSpinnerComponent,
118   ],
119 }
120 export class ReportComponent implements OnInit {
121   accessToken: string | null = null;
122   accountDetails: any = null;
123   accountsData: AccountData[] = [];
124   accountHistory: GroupedTradeFinal[] = [];
125   errorMessage: string | null = null;
126   stats?: StatConfig;
127   userKey!: string;
128   config!: displayConfigData[];
129   loading = false;
130   fromDate = '';
131   toDate = '';
132   user: User | null = null;
133   requestYear: number = 0;
134   private updateSubscription?: Subscription;
135   private loadingTimeout?: any;
136   strategies: ConfigurationOverview[] = [];
137
138   // Account management
139   currentAccount: AccountData | null = null;
140   showAccountDropdown = false;
141   showReloadButton = false;
142
143   // Balance data from API
144   balanceData: any = null;
145
146   // Loading state tracking for complete data loading
147   private loadingStates = {
148     userData: false,
149     accounts: false,
150     strategies: false,
151     userKey: false,
152     historyData: false,
153     balanceData: false,
154     config: false
155   };
156
157   // Flag para rastrear si hay peticiones en curso
158   private hasPendingRequests = false;
159
160   // Flag para rastrear si las estadísticas están completamente procesadas
161   private statsProcessed = false;
162
163   // Flag para rastrear si las gráficas están completamente renderizadas
164   private chartsRendered = false;
165
166   // Local storage keys
167   private readonly STORAGE_KEYS = {
168     REPORT_DATA: 'tradeSwitch_reportData',
169     ACCOUNTS_DATA: 'tradeSwitch_accountsData',
170     CURRENT_ACCOUNT: 'tradeSwitch_currentAccount',
171     USER_DATA: 'tradeSwitch_userData'
172   };
173
174   // Plan limitation modal
175   planLimitationModal: PlanLimitationModalData = {
176     showModal: false,
177     modalType: 'blocked',
178     title: '',
179     message: '',
180     primaryButtonText: '',
181     onPrimaryAction: () => {}
182   };
183
184   constructor(
185     @Inject(PLATFORM_ID) private platformId: any,
186     private store: Store,

```

```

187     private reportService: ReportService,
188     private userService: AuthService,
189     private strategySvc: SettingsService,
190     private router: Router,
191     private planLimitationsGuard: PlanLimitationsGuard,
192     private appContext: ApplicationContextService,
193     private pluginHistoryService: PluginHistoryService,
194     private timezoneService: TimezoneService
195   ) {}
196
197   ngAfterViewInit() {
198     if (isPlatformBrowser(this.platformId)) {
199       const container = document.querySelector('.stats-card-container');
200       if (container) {
201         container.addEventListener('wheel', function (e) {
202           if ((e as any).deltaY !== 0) {
203             e.preventDefault();
204             container.scrollLeft += (e as any).deltaY;
205           }
206         });
207       }
208     }
209   }
210
211   ngOnInit() {
212     // SIEMPRE iniciar loading al entrar a la ventana
213     this.startLoading();
214
215     // Cargar datos básicos
216     this.loadSavedData();
217
218     // Suscribirse a los datos del contexto
219     this.subscribeToContextData();
220
221     // Obtener datos frescos
222     this.getUserData();
223     this.initializeStrategies();
224     this.listenGroupedTrades();
225     this.fetchUserRules();
226     this.checkUserAccess();
227
228     // Cargar datos de todas las cuentas
229     this.loadAllAccountsData();
230   }
231
232   private subscribeToContextData() {
233     // Suscribirse a los datos del usuario
234     this.appContext.currentUser$.subscribe(user => {
235       this.user = user;
236     });
237
238     // Suscribirse a las cuentas del usuario - SIEMPRE cargar la primera
239     this.appContext.userAccounts$.subscribe(accounts => {
240       // PRIMERO: Verificar si faltan datos antes de salir
241       if (accounts.length > 0) {
242         const currentAccountInList = accounts[0];
243
244         // Si tenemos la misma cuenta pero nos faltan datos, cargarlos
245         if (this.currentAccount &&
246             this.currentAccount.accountID === currentAccountInList.accountID &&
247             (!this.balanceData || !this.stats || this.accountHistory.length === 0)) {
248           this.startInternalLoading();
249           this.loadSavedReportData(this.currentAccount.accountID);
250           return; // Salir después de cargar
251         }
252       }
253
254       // Evitar bucles infinitos - solo procesar si hay cambios reales
255       if (JSON.stringify(this.accountsData) === JSON.stringify(accounts)) {
256         return;

```

```

257     }
258
259     this.accountsData = accounts;
260     if (accounts.length > 0) {
261         // Solo procesar si la cuenta actual cambió o si no hay cuenta actual
262         const newAccount = accounts[0];
263         const accountChanged = !this.currentAccount ||
264             this.currentAccount.accountID !== newAccount.accountID;
265
266         if (accountChanged) {
267             this.currentAccount = newAccount;
268
269             // Verificar si es una cuenta nueva (recién registrada)
270             const isNew = this.isNewAccount(this.currentAccount);
271
272             if (isNew) {
273                 // Cuenta nueva - hacer peticiones a la API
274                 this.startInternalLoading();
275                 this.fetchUserKey(this.currentAccount);
276             } else {
277                 // Cuenta existente - SIEMPRE mostrar loading
278                 this.startInternalLoading();
279                 this.loadSavedReportData(this.currentAccount.accountID);
280             }
281         }
282     } else {
283         // Si no hay cuentas, limpiar datos y parar loading
284         this.currentAccount = null;
285         this.accountHistory = [];
286         this.stats = undefined;
287         this.balanceData = null;
288         this.stopInternalLoading();
289     }
290 });
291
292 // Suscribirse a las estrategias del usuario
293 this.appContext.userStrategies$.subscribe(strategies => {
294     this.strategies = strategies;
295 });
296
297
298 private startLoading() {
299     // Loading general solo para cuentas
300     this.loading = true;
301
302     // Timeout de seguridad para cuentas
303     if (this.loadingTimeout) {
304         clearTimeout(this.loadingTimeout);
305     }
306
307     this.loadingTimeout = setTimeout(() => {
308         this.loading = false;
309     }, 2000); // 10 segundos máximo para cuentas
310 }
311
312 private startInternalLoading() {
313     // Loading interno para datos de reporte
314     this.hasPendingRequests = true;
315     this.statsProcessed = false;
316     this.chartsRendered = false;
317
318     // Reset account-related loading states
319     this.loadingStates = {
320         userData: this.loadingStates.userData,
321         accounts: this.loadingStates.accounts,
322         strategies: this.loadingStates.strategies,
323         userKey: false,
324         historyData: false,
325         balanceData: false,
326         config: this.loadingStates.config

```

```

327     };
328
329     // Limpiar datos temporales para evitar mostrar valores 0
330     this.accountHistory = [];
331     this.stats = undefined;
332     // NO limpiar balanceData aquí - se mantendrá del localStorage
333
334     // Limpiar el store para evitar mostrar datos anteriores
335     this.store.dispatch(setGroupedTrades({ groupedTrades: [] }));
336     this.store.dispatch(setNetPnL({ netPnL: 0 }));
337     this.store.dispatch(setTradeWin({ tradeWin: 0 }));
338     this.store.dispatch(setProfitFactor({ profitFactor: 0 }));
339     this.store.dispatch(setAvgWnL({ avgWnL: 0 }));
340     this.store.dispatch(setTotalTrades({ totalTrades: 0 }));
341
342     // Aumentar el tiempo de loading para evitar parpadeos
343     setTimeout(() => {
344       this.checkIfAllDataLoaded();
345     }, 800); // Esperar 1 segundo antes de verificar
346   }
347
348   private stopLoading() {
349     // Solo para loading interno
350     this.hasPendingRequests = false;
351     this.statsProcessed = true; // Marcar que las estadísticas están procesadas
352     this.chartsRendered = true; // Marcar que las gráficas están renderizadas
353   }
354
355   private stopInternalLoading() {
356     // Parar loading interno
357     this.hasPendingRequests = false;
358     this.statsProcessed = true;
359     this.chartsRendered = true;
360   }
361
362 /**
363  * Verificar si una cuenta es nueva (recién registrada)
364  * Una cuenta es nueva si no tiene datos guardados en localStorage
365  */
366 private isNewAccount(account: AccountData): boolean {
367   if (!account || !isPlatformBrowser(this.platformId)) {
368     return false;
369   }
370
371   try {
372     // Verificar si existe datos guardados para esta cuenta
373     const savedData = this.appContext.loadReportDataFromLocalStorage(account.accountID);
374
375     // Si no hay datos guardados, es una cuenta nueva
376     return !savedData || !savedData.accountHistory || !savedData.stats;
377   } catch (error) {
378     console.error('Error verificando si la cuenta es nueva:', error);
379     // En caso de error, asumir que es nueva
380     return true;
381   }
382 }
383
384 private setLoadingState(key: keyof typeof this.loadingStates, value: boolean) {
385   this.loadingStates[key] = value;
386   this.checkIfAllDataLoaded();
387 }
388
389 private checkIfAllDataLoaded() {
390   // Verificar si todos los datos críticos están cargados
391   const criticalDataLoaded =
392     this.loadingStates.userData &&
393     this.loadingStates.accounts &&
394     this.loadingStates.strategies &&
395     this.loadingStates.config;
396

```

```

397 // Si hay cuenta actual, verificar que los datos necesarios estén cargados
398 // Balance e historial son independientes - no requieren ambos
399 const accountDataLoaded = !this.currentAccount ||
400     (this.loadingStates.userKey &&
401      (this.loadingStates.historyData || this.loadingStates.balanceData));
402
403 // Verificar que los datos estén realmente disponibles para mostrar en la UI
404 const uiDataReady = this.isUIDataReady();
405
406 // Verificar que no haya peticiones pendientes
407 const noPendingRequests = !this.hasPendingRequests;
408
409 // Verificar que las estadísticas estén completamente procesadas
410 const statsReady = this.statsProcessed || !this.currentAccount;
411
412 // Verificar que las gráficas estén completamente renderizadas
413 const chartsReady = this.chartsRendered || !this.currentAccount;
414
415 // Verificación adicional: asegurar que los datos estén realmente en localStorage
416 const dataInLocalStorage = !this.currentAccount || this.isDataInLocalStorage();
417
418 // Evitar bucles infinitos - solo procesar si no está ya procesado
419 const shouldProcess = !this.statsProcessed || !this.chartsRendered;
420
421 if (criticalDataLoaded && accountDataLoaded && uiDataReady && noPendingRequests &&
422 statsReady && chartsReady && dataInLocalStorage && shouldProcess) {
423     this.loadSavedReportData(this.currentAccount?.accountId || '');
424 }
425
426
427 private isDataInLocalStorage(): boolean {
428     if (!this.currentAccount || !isPlatformBrowser(this.platformId)) {
429         return true;
430     }
431
432     try {
433         const reportDataKey = `${this.STORAGE_KEYS.REPORT_DATA}
434 _${this.currentAccount?.accountId}`.localStorage.getItem(reportDataKey);
435         if (savedReportData) {
436             const data = JSON.parse(savedReportData);
437             return data.accountHistory && data.stats && data.balanceData !== null;
438         }
439     } catch (error) {
440         console.error('Error checking localStorage data:', error);
441     }
442
443     return false;
444 }
445
446 private isUIDataReady(): boolean {
447     // Verificar que todos los datos necesarios para la UI estén disponibles
448
449     // Si no hay cuenta, no necesitamos datos de trading
450     if (!this.currentAccount) {
451         return true;
452     }
453
454     // Verificar que tenemos datos de balance (puede ser 0, pero debe estar cargado)
455     const hasBalanceData = this.balanceData !== null && this.balanceData !== undefined;
456
457     // Verificar que tenemos datos de trading history (puede ser array vacío, pero debe
458     // estar cargado)
459     const hasHistoryData = Array.isArray(this.accountHistory);
460
461     // Verificar que tenemos estadísticas completamente calculadas
462     const hasStats = this.stats !== null &&
463         this.stats !== undefined &&
464         typeof this.stats.netPnl === 'number' &&
465         typeof this.stats.tradeWinPercent === 'number' &&
466         typeof this.stats.profitFactor === 'number' &&
467         typeof this.stats.totalTrades === 'number' &&
```

```

467         typeof this.stats.avgWinLossTrades === 'number';
468
469 // Verificar que tenemos estrategias (puede ser array vacío, pero debe estar cargado)
470 const hasStrategies = Array.isArray(this.strategies);
471
472 // Para cuentas nuevas, no verificar datos reales - solo que estén cargados
473 const hasProcessedData = this.statsProcessed && this.chartsRendered;
474
475 // Balance e historial son independientes - al menos uno debe estar disponible
476 const hasAccountData = hasBalanceData || hasHistoryData;
477
478 return hasAccountData && hasStats && hasStrategies && hasProcessedData;
479 }
480
481 private hasRealChartData(): boolean {
482     // Si no hay datos de trading history, las gráficas mostrarán valores por defecto
483     if (!this.accountHistory || this.accountHistory.length === 0) {
484         return true; // Es válido mostrar gráficas vacías si no hay datos
485     }
486
487     // Verificar que los datos de trading history tengan valores reales
488     const hasRealTradingData = this.accountHistory.some(trade =>
489         trade.pnl !== undefined &&
490         trade.pnl !== null &&
491         trade.lastModified !== undefined &&
492         trade.lastModified !== null
493     );
494
495     // Verificar que las estadísticas no sean solo valores por defecto
496     const hasRealStats = this.stats ? (
497         this.stats.netPnl !== 0 ||
498         this.stats.totalTrades > 0 ||
499         this.stats.tradeWinPercent !== 0 ||
500         this.stats.profitFactor !== 0
501     ) : false;
502
503     return hasRealTradingData && hasRealStats;
504 }
505
506 private loadSavedReportData(accountID: string) {
507     if (!isPlatformBrowser(this.platformId) || !accountID) {
508         // Si no hay accountID, parar loading interno
509         this.stopInternalLoading();
510         return;
511     }
512
513     try {
514         const savedData = this.appContext.loadReportDataFromLocalStorage(accountID);
515         if (savedData && savedData.accountHistory && savedData.stats) {
516             // Simular tiempo de loading para mostrar el spinner
517             setTimeout(() => {
518                 this.accountHistory = savedData.accountHistory;
519                 this.stats = savedData.stats;
520                 this.balanceData = savedData.balanceData;
521
522                 // Actualizar el store
523                 const groupedTrades = Array.isArray(savedData.accountHistory) ?
524                     savedData.accountHistory.map((trade: any) => ({
525                         ...trade,
526                         pnl: trade.pnl ?? 0,
527                         isWon: trade.isWon ?? false,
528                         isOpen: trade.isOpen ?? false
529                     })) : [];
530                 this.store.dispatch(setGroupedTrades({ groupedTrades }));
531
532                 // Marcar como cargados
533                 this.setLoadingState('userKey', true);
534                 this.setLoadingState('historyData', true);
535                 // Marcar balance como cargado si existe (incluso si es null, pero está cargado)
536                 this.setLoadingState('balanceData', true);

```

```

537         this.hasPendingRequests = false;
538     }, 800); // Esperar 2 segundos para mostrar loading
539 } else {
540     // No hay datos guardados, parar loading interno
541     this.stopInternalLoading();
542 }
543 } catch (error) {
544     console.error('Error cargando datos de reporte guardados:', error);
545     // En caso de error, parar loading interno
546     this.stopInternalLoading();
547 }
548 }
549
550 // Métodos para persistencia local
551 private loadSavedData() {
552     if (!isPlatformBrowser(this.platformId)) return;
553
554     try {
555         // Solo cargar datos básicos para inicialización
556         // Los datos de reporte se cargarán desde el contexto
557
558         // Cargar cuentas guardadas (solo para inicialización)
559         const savedAccountsData = localStorage.getItem(this.STORAGE_KEYS.ACCOUNTS_DATA);
560         if (savedAccountsData) {
561             this.accountsData = JSON.parse(savedAccountsData);
562         }
563
564         // Cargar cuenta actual (solo para inicialización)
565         const savedCurrentAccount = localStorage.getItem(this.STORAGE_KEYS.CURRENT_ACCOUNT);
566         if (savedCurrentAccount) {
567             this.currentAccount = JSON.parse(savedCurrentAccount);
568         }
569
570         // Cargar datos de usuario (solo para inicialización)
571         const savedUserData = localStorage.getItem(this.STORAGE_KEYS.USER_DATA);
572         if (savedUserData) {
573             this.user = JSON.parse(savedUserData);
574         }
575
576         if (this.currentAccount) {
577             const savedAccountHistory =
578             localStorage.getItem(`Balance_${this.currentAccount.accountID}`);
579             if (savedAccountHistory) {
580                 this.balanceData = JSON.parse(savedAccountHistory);
581             }
582         }
583     } catch (error) {
584         console.error('Error cargando datos guardados:', error);
585         this.clearSavedData();
586     }
587 }
588
589 private saveDataToStorage() {
590     if (!isPlatformBrowser(this.platformId)) return;
591
592     try {
593         // Guardar datos de reporte usando el contexto
594         if (this.accountHistory.length > 0 && this.stats && this.currentAccount) {
595             const reportData = {
596                 accountHistory: this.accountHistory,
597                 stats: this.stats,
598                 balanceData: this.balanceData,
599                 lastUpdated: Date.now()
600             };
601             this.appContext.saveReportDataToLocalStorage(
602                 this.currentAccount.accountID,
603                 this.currentAccount,
604                 reportData
605             );
606         }
607     }

```

```

607
608     // Guardar cuentas
609     if (this.accountsData.length > 0) {
610         localStorage.setItem(this.STORAGE_KEYS.ACCOUNTS_DATA,
611         JSON.stringify(this.accountsData));
612
613     // Guardar cuenta actual
614     if (this.currentAccount) {
615         localStorage.setItem(this.STORAGE_KEYS.CURRENT_ACCOUNT,
616         JSON.stringify(this.currentAccount));
617
618     // Guardar datos de usuario
619     if (this.user) {
620         localStorage.setItem(this.STORAGE_KEYS.USER_DATA, JSON.stringify(this.user));
621     }
622     } catch (error) {
623         console.error('Error guardando datos:', error);
624     }
625 }
626
627
628 private clearSavedData() {
629     if (!isPlatformBrowser(this.platformId)) return;
630
631     try {
632         // Limpiar datos de reporte para la cuenta actual usando el contexto
633         if (this.currentAccount) {
634             this.appContext.clearReportDataFromLocalStorage(this.currentAccount.accountID);
635         }
636         localStorage.removeItem(this.STORAGE_KEYS.ACCOUNTS_DATA);
637         localStorage.removeItem(this.STORAGE_KEYS.CURRENT_ACCOUNT);
638         localStorage.removeItem(this.STORAGE_KEYS.USER_DATA);
639     } catch (error) {
640         console.error('Error limpiando datos guardados:', error);
641     }
642 }
643
644 private async initializeStrategies(): Promise<void> {
645     if (this.user?.id) {
646         try {
647             this.strategies = await this.strategySvc.getUserStrategyViews(this.user.id);
648             // Marcar estrategias como cargadas
649             this.setLoadingState('strategies', true);
650             // Verificar si todos los datos están listos después de cargar las estrategias
651             this.checkIfAllDataLoaded();
652         } catch (error) {
653             console.error('Error loading strategies:', error);
654             // Marcar estrategias como cargadas incluso en error
655             this.setLoadingState('strategies', true);
656             // Verificar si todos los datos están listos incluso en caso de error
657             this.checkIfAllDataLoaded();
658         }
659     } else {
660         // Si no hay usuario, marcar como cargado
661         this.setLoadingState('strategies', true);
662         // Verificar si todos los datos están listos
663         this.checkIfAllDataLoaded();
664     }
665 }
666
667 ngOnDestroy() {
668     this.updateSubscription?.unsubscribe();
669     if (this.loadingTimeout) {
670         clearTimeout(this.loadingTimeout);
671     }
672 }
673
674 getUserData() {
675     this.store.select(selectUser).subscribe({
676         next: (user) => {

```

```

677     this.user = user.user;
678     if (this.user) {
679         // Marcar datos de usuario como cargados
680         this.setLoadingState('userData', true);
681
682         // Guardar datos de usuario en localStorage
683         this.saveDataToStorage();
684
685         // Reutilizar cuentas del contexto (ya cargadas en login)
686         this.useAccountsFromContext();
687     },
688     error: (err) => {
689         console.error('Error fetching user data', err);
690         this.setLoadingState('userData', true); // Marcar como cargado incluso en error
691     },
692 });
693 );
694 }
695
696 useAccountsFromContext() {
697     // Reutilizar cuentas del contexto (ya cargadas en login)
698     const contextAccounts = this.appContext.userAccounts();
699     if (contextAccounts && contextAccounts.length > 0) {
700         // Evitar bucles infinitos - solo procesar si hay cambios reales
701         if (JSON.stringify(this.accountsData) === JSON.stringify(contextAccounts)) {
702             return;
703         }
704
705         this.accountsData = contextAccounts;
706         this.currentAccount = this.accountsData[0]; // Siempre la primera
707
708         // Guardar cuentas en localStorage
709         this.saveDataToStorage();
710
711         // Marcar cuentas como cargadas
712         this.setLoadingState('accounts', true);
713
714         // SIEMPRE iniciar loading interno para mostrar loading
715         this.startInternalLoading();
716
717         // Cargar datos de la primera cuenta
718         this.loadSavedReportData(this.currentAccount.accountID);
719         // Lanzar carga completa de todas las cuentas y actualizar métricas por cuenta
720         this.loadAllAccountsData();
721     } else {
722         // Si no hay cuentas en el contexto, iniciar loading y cargarlas
723         this.startLoading();
724         this.fetchUserAccounts();
725     }
726 }
727
728
729 fetchUserAccounts() {
730     this.userService.getUserAccounts(this.user?.id).then((accounts) => {
731         if (!accounts || accounts.length === 0) {
732             // No hay cuentas - usuario nuevo
733             this.accountsData = [];
734             this.currentAccount = null;
735             this.accountHistory = [];
736             this.stats = undefined;
737             this.balanceData = null;
738
739             // Marcar cuentas como cargadas
740             this.setLoadingState('accounts', true);
741
742             // Parar loading general
743             this.loading = false;
744
745             // Parar loading interno
746             this.stopInternalLoading();

```

```

747     } else {
748         this.accountsData = accounts;
749         this.currentAccount = accounts[0]; // Siempre la primera
750
751         // Guardar cuentas en localStorage
752         this.saveDataToStorage();
753
754         // Marcar cuentas como cargadas
755         this.setLoadingState('accounts', true);
756
757         // Parar loading general
758         this.loading = false;
759
760         // Iniciar loading interno y cargar datos de la primera cuenta
761         this.startInternalLoading();
762         this.loadSavedReportData(this.currentAccount.accountID);
763         // Lanzar carga completa de todas las cuentas y actualizar métricas por cuenta
764         this.loadAllAccountsData();
765     }
766 }).catch((error) => {
767     console.error('Error fetching user accounts:', error);
768     // En caso de error, limpiar todo
769     this.accountsData = [];
770     this.currentAccount = null;
771     this.accountHistory = [];
772     this.stats = undefined;
773     this.balanceData = null;
774
775     this.setLoadingState('accounts', true);
776     this.loading = false; // Parar loading general
777     this.stopInternalLoading();
778 });
779 }
780
781 fetchUserRules() {
782     this.strategySvc
783         .getStrategyConfig(this.user?.id)
784         .then((data) => {
785             if (data) {
786                 this.store.dispatch(resetConfig({ config: data }));
787                 this.config = this.prepareConfigDisplayData(data);
788             } else {
789                 this.store.dispatch(resetConfig({ config: initialStrategyState }));
790                 this.config = this.prepareConfigDisplayData(initialStrategyState);
791             }
792             // Marcar configuración como cargada
793             this.setLoadingState('config', true);
794             // Verificar si todos los datos están listos después de cargar la configuración
795             this.checkIfAllDataLoaded();
796         })
797         .catch((err) => {
798             console.error('Error to get the config', err);
799             this.store.dispatch(resetConfig({ config: initialStrategyState }));
800             this.config = this.prepareConfigDisplayData(initialStrategyState);
801             // Marcar configuración como cargada incluso en error
802             this.setLoadingState('config', true);
803             // Verificar si todos los datos están listos incluso en caso de error
804             this.checkIfAllDataLoaded();
805         });
806     }
807
808     onStrategyPercentageChange(percentage: number) {
809         if (this.user) {
810             this.updateFirebaseUserData(percentage);
811         }
812     }
813
814     updateFirebaseUserData(percentage: number) {
815         if (this.user) {
816             const updatedUser = {

```

```

817     ...this.user,
818     // TODO: revisar uso anterior de best_trade, ahora pertenece a AccountData
819     // TODO: revisar uso anterior de netPnl, ahora pertenece a AccountData
820     lastUpdated: new Date().getTime() as unknown as Timestamp,
821     // TODO: revisar uso anterior de number_trades, ahora pertenece a AccountData
822     strategy_followed: percentage,
823     // TODO: revisar uso anterior de profit, ahora pertenece a AccountData
824     // TODO: revisar uso anterior de total_spend, ahora pertenece a AccountData
825   };
826
827   const actualYear = new Date().getFullYear();
828
829   const requestYear = this.requestYear;
830   if (actualYear === requestYear) {
831     this.userService.createUser(updatedUser as unknown as User);
832   }
833
834   const monthlyReport = {
835     id: this.user?.id,
836     // TODO: revisar uso anterior de best_trade, ahora pertenece a AccountData
837     // TODO: revisar uso anterior de netPnl, ahora pertenece a AccountData
838     // TODO: revisar uso anterior de number_trades, ahora pertenece a AccountData
839     // TODO: revisar uso anterior de profit, ahora pertenece a AccountData
840     strategy_followed: percentage,
841     // TODO: revisar uso anterior de total_spend, ahora pertenece a AccountData
842     month: new Date().getMonth() + 1,
843     year: new Date().getFullYear(),
844   };
845
846   this.reportService.updateMonthlyReport(
847     monthlyReport as unknown as MonthlyReport
848   );
849 }
850
851
852 listenGroupedTrades() {
853   this.store.select(selectGroupedTrades).subscribe({
854     next: (groupedTrades) => {
855       this.accountHistory = groupedTrades;
856       // Calcular estadísticas inmediatamente cuando se reciben los datos
857       this.updateReportStats(this.store, groupedTrades);
858       // NO verificar aquí - se verificará desde updateReportStats
859     },
860   });
861 }
862
863
864 fetchUserKey(account: AccountData) {
865   // Usar el servicio que ya actualiza el contexto automáticamente
866   this.reportService
867     .getUserKey(
868       account.emailTradingAccount,
869       account.brokerPassword,
870       account.server
871     )
872     .subscribe({
873       next: (key: string) => {
874         this.userKey = key;
875         // Marcar userKey como cargado
876         this.setLoadingState('userKey', true);
877
878         const now = new Date();
879         const currentYear = now.getUTCFullYear();
880         this.fromDate = Date.UTC(currentYear, 0, 1, 0, 0, 0).toString();
881         this.toDate = Date.UTC(
882           currentYear,
883           11,
884           31,
885           23,
886           59,

```

```

887         59,
888         999
889     ).toString();
890     this.requestYear = currentYear;
891
892     this.fetchHistoryData(
893         key,
894         account.accountID,
895         account.accountNumber
896     );
897
898     this.store.dispatch(setUserKey({ userKey: key }));
899 },
900 error: (err) => {
901     console.error('Error fetching user key:', err);
902     this.store.dispatch(setUserKey({ userKey: '' }));
903     // Marcar userKey como cargado incluso en error
904     this.setLoadingState('userKey', true);
905     // Marcar que no hay peticiones pendientes
906     this.hasPendingRequests = false;
907 },
908 });
909 }
910
911 fetchHistoryData(
912     key: string,
913     accountId: string,
914     accNum: number
915 ) {
916     // Solo consultar balance si no existe ya
917     if (this.balanceData === null || this.balanceData === undefined) {
918         this.reportService.getBalanceData(accountId, key, accNum).subscribe({
919             next: (balanceData) => {
920                 this.balanceData = balanceData;
921                 this.setLoadingState('balanceData', true);
922                 // Verificar si todos los datos están listos después de cargar el balance
923                 this.checkIfAllDataLoaded();
924             },
925             error: (err) => {
926                 console.error('Error fetching balance data:', err);
927                 this.setLoadingState('balanceData', true);
928                 // Verificar si todos los datos están listos incluso en caso de error
929                 this.checkIfAllDataLoaded();
930             },
931         });
932     } else {
933         // Si ya existe balanceData, marcar como cargado
934         this.setLoadingState('balanceData', true);
935         this.checkIfAllDataLoaded();
936     }
937
938     // Solo hacer petición al trading history (la principal)
939     this.reportService
940         .getHistoryData(accountId, key, accNum)
941         .subscribe({
942             next: (groupedTrades: GroupedTradeFinal[]) => {
943                 // Reemplazar en lugar de acumular para evitar duplicados
944                 this.store.dispatch(
945                     setGroupedTrades({
946                         groupedTrades: groupedTrades,
947                     })
948                 );
949
950                 // Calcular estadísticas inmediatamente después de recibir los datos
951                 this.updateReportStats(this.store, groupedTrades);
952
953                 // Guardar datos en el contexto y localStorage por cuenta DESPUÉS de calcular stats
954                 if (this.currentAccount) {
955                     // Esperar a que las estadísticas estén calculadas
956                     // Guardar en el contexto

```

```

957     const contextData = {
958       accountHistory: groupedTrades,
959       stats: this.stats,
960       balanceData: this.balanceData,
961       lastUpdated: Date.now()
962     };
963     this.appContext.setTradingHistoryForAccount(this.currentAccount!.id,
964   contextData);
965     // Guardar datos en localStorage después de recibir respuesta exitosa
966     this.saveDataToStorage();
967
968     // Marcar history data como cargado DESPUÉS de guardar todo
969     this.setLoadingState('historyData', true);
970     // Marcar que no hay peticiones pendientes
971     this.hasPendingRequests = false;
972   } else {
973     // Si no hay cuenta actual, marcar como cargado inmediatamente
974     this.setLoadingState('historyData', true);
975     this.hasPendingRequests = false;
976   }
977 },
978 error: (err) => {
979   console.error('Error fetching history data:', err);
980   this.store.dispatch(setGroupedTrades({ groupedTrades: [] }));
981   // Marcar history data como cargado incluso en error
982   this.setLoadingState('historyData', true);
983   // Marcar que no hay peticiones pendientes
984   this.hasPendingRequests = false;
985 },
986 );
987 }
988
989 updateReportStats(store: Store, groupedTrades: GroupedTradeFinal[]) {
990   // Normalizar todos los trades - asignar valores por defecto cuando falten
991   const normalizedTrades = groupedTrades.map(trade => {
992     ...trade,
993     pnl: trade.pnl ?? 0, // Si no hay PnL, usar 0
994     entryPrice: trade.avgPrice ?? 0, // Usar avgPrice como entryPrice
995     exitPrice: trade.avgPrice ?? 0, // Usar avgPrice como exitPrice
996     buy_price: trade.side === 'buy' ? trade.price : '0', // Precio de compra si es buy
997     sell_price: trade.side === 'sell' ? trade.price : '0', // Precio de venta si es sell
998     quantity: Number(trade.qty) ?? 0 // Usar qty como quantity
999   ));
1000
1001   // Usar las funciones de utilidad existentes
1002   this.stats = {
1003     netPnl: calculateNetPnl(normalizedTrades),
1004     tradeWinPercent: calculateTradeWinPercent(normalizedTrades),
1005     profitFactor: calculateProfitFactor(normalizedTrades),
1006     avgWinLossTrades: calculateAvgWinLossTrades(normalizedTrades),
1007     totalTrades: calculateTotalTrades(normalizedTrades),
1008     activePositions: groupedTrades.filter(trade => trade.isOpen === true).length
1009   };
1010
1011   // Actualizar el store con las estadísticas calculadas
1012   store.dispatch(setNetPnl({ netPnl: this.stats?.netPnl || 0 }));
1013   store.dispatch(setTradeWin({ tradeWin: this.stats?.tradeWinPercent || 0 }));
1014   store.dispatch(setProfitFactor({ profitFactor: this.stats?.profitFactor || 0 }));
1015   store.dispatch(setAvgWnL({ avgWnL: this.stats?.avgWinLossTrades || 0 }));
1016   store.dispatch(setTotalTrades({ totalTrades: this.stats?.totalTrades || 0 }));
1017
1018   // Guardar stats actualizados en localStorage
1019   this.saveDataToStorage();
1020
1021   // Marcar que las estadísticas están procesadas
1022   this.statsProcessed = true;
1023   this.chartsRendered = true;
1024
1025   // Esperar un poco antes de verificar para evitar recargas múltiples
1026   setTimeout(() => {

```

```

1027     this.checkIfAllDataLoaded();
1028 }, 800);
1029 }
1030
1031 prepareConfigDisplayData(strategyState: StrategyState) {
1032   return this.transformStrategyStateToDisplayData(strategyState);
1033 }
1034
1035 transformStrategyStateToDisplayData(
1036   strategyState: StrategyState
1037 ): displayConfigData[] {
1038   const newConfig: displayConfigData[] = [];
1039
1040   Object.entries(strategyState).forEach(([key, value]) => {
1041     if (value.type) {
1042       let title = '';
1043
1044       if (value.type === RuleType.MAX_DAILY_TRADES) {
1045         title = 'Max Daily Trades';
1046       } else if (value.type === RuleType.RISK_REWARD_RATIO) {
1047         title = `${value.riskRewardRatio} Risk Reward`;
1048       } else if (value.type === RuleType.MAX_RISK_PER_TRADE) {
1049         title = `Limit my Risk per Trade: ${value.maxRiskPerTrade}`;
1050       } else if (value.type === RuleType.DAYS_ALLOWED) {
1051         const abbreviationsMap: { [key: string]: string } = {
1052           Monday: 'Mon',
1053           Tuesday: 'Tues',
1054           Wednesday: 'Wed',
1055           Thursday: 'Thurs',
1056           Friday: 'Fri',
1057           Saturday: 'Sat',
1058           Sunday: 'Sun',
1059         };
1060
1061         const formattedText = value.tradingDays
1062           .map((day: string) => abbreviationsMap[day] ?? day)
1063           .join(', ');
1064
1065         title = `Only allow trades on: ${formattedText}.`;
1066       } else if (value.type === RuleType.ASSETS_ALLOWED) {
1067         const formattedText = value.assetsAllowed.join(', ');
1068         title = `Only allow trades on the following assets: ${formattedText}`;
1069       } else if (value.type === RuleType.TRADING_HOURS) {
1070         const abbreviation = moment.tz(value.timezone).zoneAbbr();
1071         title = `Only allow trades between: ${value.tradingOpenTime} -
1072 ${value.tradingCloseTime} ${abbreviation}`;
1073
1074         newConfig.push({
1075           title,
1076           type: value.type,
1077           isActive: value.isActive,
1078         });
1079       }
1080     });
1081
1082     return newConfig.sort((a, b) => {
1083       if (a.isActive && !b.isActive) return -1;
1084       if (!a.isActive && b.isActive) return 1;
1085       return 0;
1086     });
1087   }
1088
1089   onYearChange($event: string) {
1090     this.startLoading();
1091     this.fromDate = Date.UTC(Number($event), 0, 1, 0, 0, 0).toString();
1092     this.toDate = Date.UTC(Number($event), 11, 31, 23, 59, 59, 999).toString();
1093     this.requestYear = Number($event);
1094
1095     // Reset account-related loading states
1096     this.setLoadingState('userKey', false);

```

```

1097     this.setLoadingState('historyData', false);
1098     this.setLoadingState('balanceData', false);
1099
1100     if (this.user) {
1101         this.fetchUserAccounts();
1102     }
1103 }
1104
1105 // Account management methods
1106 getCurrentAccountName(): string {
1107     return this.currentAccount?.accountName || 'No Account Selected';
1108 }
1109
1110 getCurrentAccountServer(): string {
1111     return this.currentAccount?.server || 'No Server';
1112 }
1113
1114 async getCurrentAccountPlan(): Promise<string> {
1115     return await this.getAccountPlan(this.currentAccount);
1116 }
1117
1118 async getAccountPlan(account: AccountData | null): Promise<string> {
1119     if (!account || !this.user?.id) return 'Free';
1120
1121     // Use the guard to get the real plan information
1122     return await this.getUserPlanName();
1123 }
1124
1125 private async getUserPlanName(): Promise<string> {
1126     if (!this.user?.id) return 'Free';
1127
1128     try {
1129         const limitations = await this.planLimitationsGuard.checkUserLimitations(this.user.id);
1130         return limitations.planName;
1131     } catch (error) {
1132         console.error('Error getting user plan name:', error);
1133         return 'Free';
1134     }
1135 }
1136
1137 toggleAccountDropdown() {
1138     this.showAccountDropdown = !this.showAccountDropdown;
1139 }
1140
1141 selectAccount(account: AccountData) {
1142     this.currentAccount = account;
1143     this.showAccountDropdown = false;
1144
1145     // Guardar cuenta seleccionada en localStorage
1146     this.saveDataToStorage();
1147
1148     // SIEMPRE iniciar loading interno para mostrar loading
1149     this.startInternalLoading();
1150
1151     // Reset account-related loading states
1152     this.setLoadingState('userKey', false);
1153     this.setLoadingState('historyData', false);
1154     this.setLoadingState('balanceData', false);
1155
1156     // Limpiar datos anteriores
1157     this.store.dispatch(setGroupedTrades({ groupedTrades: [] }));
1158
1159     // Verificar si existe localStorage para esta cuenta
1160     const savedData = this.appContext.loadReportDataFromLocalStorage(account.accountID);
1161
1162     if (savedData && savedData.accountHistory && savedData.stats) {
1163         // Si existe en localStorage, cargar directamente con loading
1164         this.loadSavedReportData(account.accountID);
1165     } else {
1166         // Si no existe, hacer peticiones a la API

```

```

1167      this.fetchUserKey(account);
1168    }
1169  }
1170
1171  goToEditStrategy() {
1172    this.router.navigate(['/edit-strategy']);
1173  }
1174
1175  async exportAllData() {
1176    const csvData = await this.generateAllReportsCSV();
1177    this.downloadCSV(csvData, `my-reports-${new Date().toISOString().split('T')[0]}.csv`);
1178  }
1179
1180  async generateAllReportsCSV(): Promise<string> {
1181    const headers = [
1182      'Date',
1183      'Account Name',
1184      'Plan',
1185      'Net P&L',
1186      'Trades Count',
1187      'Win Percentage',
1188      'Strategy Followed',
1189      'Profit Factor',
1190      'Avg Win/Loss Trades'
1191    ];
1192    const rows = [headers.join(',')];
1193
1194    // Add summary data
1195    const currentPlan = await this.getCurrentAccountPlan();
1196    const summaryRow = [
1197      new Date().toISOString().split('T')[0],
1198      this.getCurrentAccountName(),
1199      currentPlan,
1200      this.stats?.netPnl?.toFixed(2) || '0',
1201      this.stats?.totalTrades?.toString() || '0',
1202      `${this.stats?.tradeWinPercent?.toFixed(1) || '0'}%`,
1203      'Yes',
1204      this.stats?.profitFactor?.toFixed(2) || '0',
1205      this.stats?.avgWinLossTrades?.toFixed(2) || '0'
1206    ];
1207    rows.push(summaryRow.join(','));
1208
1209    // Add detailed trade data
1210    this.accountHistory.forEach(trade => {
1211      const tradeDate = new Date(Number(trade.lastModified)).toISOString().split('T')[0];
1212      const tradeRow = [
1213        tradeDate,
1214        this.getCurrentAccountName(),
1215        currentPlan, // Use the same plan for all trades
1216        (trade.pnl || 0).toFixed(2), // P&L calculado correctamente
1217        '1',
1218        trade.pnl && trade.pnl > 0 ? '100' : '0', // Win percentage basado en P&L real
1219        'Yes',
1220        '1.00',
1221        (trade.pnl || 0).toFixed(2) // P&L neto (mismo que P&L ya que no hay fees en el
1222        cálculo)
1223        rows.push(tradeRow.join(','));
1224      });
1225
1226      return rows.join('\n');
1227    }
1228
1229    downloadCSV(csvData: string, filename: string) {
1230      const blob = new Blob([csvData], { type: 'text/csv;charset=utf-8;' });
1231      const link = document.createElement('a');
1232
1233      if (link.download !== undefined) {
1234        const url = URL.createObjectURL(blob);
1235        link.setAttribute('href', url);
1236        link.setAttribute('download', filename);

```

```

1237     link.style.visibility = 'hidden';
1238     document.body.appendChild(link);
1239     link.click();
1240     document.body.removeChild(link);
1241   }
1242 }
1243
1244 // Navegar a trading accounts
1245 navigateToTradingAccounts() {
1246   this.router.navigate(['/trading-accounts']);
1247 }
1248
1249 // Método para recargar cuentas
1250 refreshAccounts() {
1251   if (this.user) {
1252     this.fetchUserAccounts();
1253   }
1254 }
1255
1256 // Método para recargar datos manualmente
1257 reloadData() {
1258   this.showReloadButton = false;
1259   this.startLoading();
1260
1261   // Limpiar datos anteriores y localStorage
1262   this.store.dispatch(setGroupedTrades({ groupedTrades: [] }));
1263   this.accountHistory = [];
1264   this.stats = undefined;
1265   this.balanceData = null;
1266   this.statsProcessed = false;
1267   this.chartsRendered = false;
1268   this.clearSavedData();
1269
1270   // Reset all loading states
1271   this.setLoadingState('userData', false);
1272   this.setLoadingState('accounts', false);
1273   this.setLoadingState('strategies', false);
1274   this.setLoadingState('userKey', false);
1275   this.setLoadingState('historyData', false);
1276   this.setLoadingState('balanceData', false);
1277   this.setLoadingState('config', false);
1278
1279   // Reiniciar el proceso de carga
1280   if (this.user) {
1281     this.useAccountsFromContext();
1282   } else {
1283     this.getUserData();
1284   }
1285 }
1286
1287 // Check user access and show blocking modal if needed
1288 async checkUserAccess() {
1289   if (!this.user?.id) return;
1290
1291   try {
1292     const accessCheck = await
1293     this.planLimitationsGuard.checkReportAccessWithModal(this.user.id);
1294     // Only show blocking modal if user has trading accounts (not first-time user with
1295     plan) if (!accessCheck.canAccess && accessCheck.modalData && this.accountsData.length > 0) {
1296       this.planLimitationModal = accessCheck.modalData;
1297     }
1298   } catch (error) {
1299     console.error('Error checking user access:', error);
1300   }
1301 }
1302
1303 // Plan limitation modal methods
1304 onClosePlanLimitationModal() {
1305   this.planLimitationModal.showModal = false;
1306 }

```

```

1307 // ===== MÉTODOS DE CARGA DE TODAS LAS CUENTAS =====
1308
1309 /**
1310 * Cargar datos de todas las cuentas del usuario
1311 */
1312 async loadAllAccountsData() {
1313     if (!this.user?.id || this.accountsData.length === 0) {
1314         return;
1315     }
1316
1317     try {
1318         // Cargar datos de todas las cuentas en paralelo
1319         const loadPromises = this.accountsData.map(account =>
1320             this.loadAccountData(account)
1321         );
1322
1323         await Promise.all(loadPromises);
1324
1325         // Calcular métricas globales del usuario
1326         await this.calculateAndUpdateUserMetrics();
1327
1328         // Calcular strategy_followed basado en todas las cuentas
1329         await this.calculateAndUpdateStrategyFollowed();
1330
1331         // NUEVO: Calcular y actualizar métricas por cuenta (netPnl, profit factor y bestTrade)
1332         await this.calculateAndUpdateAccountsMetrics();
1333
1334     } catch (error) {
1335         console.error('Error loading all accounts data:', error);
1336     }
1337 }
1338
1339 /**
1340 * Cargar datos de una cuenta específica
1341 */
1342 private async loadAccountData(account: AccountData): Promise<void> {
1343     try {
1344         // Obtener userKey
1345         const userKey = await this.reportService.getUserKey(
1346             account.emailTradingAccount,
1347             account.brokerPassword,
1348             account.server
1349         ).toPromise();
1350
1351         if (!userKey) {
1352             console.warn(`No se pudo obtener userKey para cuenta ${account.accountID}`);
1353             return;
1354         }
1355
1356         // Obtener trading history
1357         const tradingHistory = await this.reportService.getHistoryData(
1358             account.accountID,
1359             userKey,
1360             account.accountNumber
1361         ).toPromise();
1362
1363         // Obtener balance data
1364         const balanceData = await this.reportService.getBalanceData(
1365             account.accountID,
1366             userKey,
1367             account.accountNumber
1368         ).toPromise();
1369
1370         // Guardar en localStorage
1371         this.saveAccountDataToLocalStorage(account.accountID, {
1372             accountHistory: tradingHistory || [],
1373             balanceData: balanceData,
1374             lastUpdated: Date.now()
1375         });
1376

```

```

1377     } catch (error) {
1378       console.error(`Error loading data for account ${account.accountID}:`, error);
1379     }
1380   }
1382 
1383 /**
1384 * Calcular y actualizar métricas globales del usuario
1385 */
1386 private async calculateAndUpdateUserMetrics(): Promise<void> {
1387   if (!this.user?.id) return;
1388 
1389   try {
1390     // Recopilar todos los trades de todas las cuentas
1391     const allTrades: any[] = [];
1392     let globalProfitFactor = 0;
1393 
1394     for (const account of this.accountsData) {
1395       const accountData = this.loadAccountDataFromLocalStorage(account.accountID);
1396       if (accountData && accountData.accountHistory) {
1397         allTrades.push(...accountData.accountHistory);
1398       }
1399     }
1400 
1401     if (allTrades.length > 0) {
1402       // Calcular profit_factor global (usando la misma lógica que en la ventana)
1403       globalProfitFactor = this.calculateProfitFactor(allTrades);
1404 
1405       // Calcular best_trade global (mejor trade de todas las cuentas)
1406       const bestTrade = this.calculateGlobalBestTrade(allTrades);
1407 
1408       // Actualizar usuario en Firebase con las métricas globales
1409       const updatedUser = {
1410         ...this.user,
1411         profit: globalProfitFactor, // profit = profit_factor
1412         best_trade: bestTrade, // best_trade = mejor trade de todas las cuentas
1413         lastUpdated: new Date().getTime() as unknown as Timestamp
1414       };
1415 
1416       await this.userService.createUser(updatedUser as unknown as User);
1417     }
1418   } catch (error) {
1419     console.error('Error calculating user metrics:', error);
1420   }
1421 }
1422 
1423 /**
1424 * Calcular el mejor trade global de todas las cuentas
1425 */
1426 private calculateGlobalBestTrade(allTrades: any[]): number {
1427   if (!allTrades || allTrades.length === 0) return 0;
1428 
1429   // Normalizar trades
1430   const normalizedTrades = allTrades.map(trade => {
1431     ...trade,
1432     pnl: trade.pnl ?? 0
1433   });
1434 
1435   // Encontrar el trade con mayor ganancia o pérdida absoluta
1436   const bestTrade = normalizedTrades.reduce((best, trade) => {
1437     const currentAbs = Math.abs(trade.pnl);
1438     const bestAbs = Math.abs(best);
1439     return currentAbs > bestAbs ? trade.pnl : best;
1440   }, 0);
1441 
1442   return Math.round(bestTrade * 100) / 100;
1443 }
1444 
1445 /**

```

```

1447     * Calcular porcentaje de ganancia
1448     */
1449     private calculateWinPercent(trades: any[]): number {
1450         if (trades.length === 0) return 0;
1451         const winningTrades = trades.filter(trade => trade.pnl > 0).length;
1452         return Math.round((winningTrades / trades.length) * 100 * 100) / 100;
1453     }
1454
1455     /**
1456     * Calcular profit factor
1457     */
1458     private calculateProfitFactor(trades: any[]): number {
1459         const totalGains = trades
1460             .filter(t => t.pnl > 0)
1461             .reduce((sum, t) => sum + t.pnl, 0);
1462
1463         const totalLosses = Math.abs(trades
1464             .filter(t => t.pnl < 0)
1465             .reduce((sum, t) => sum + t.pnl, 0));
1466
1467         if (totalLosses === 0) {
1468             return totalGains > 0 ? 999.99 : 0;
1469         }
1470
1471         return Math.round((totalGains / totalLosses) * 100) / 100;
1472     }
1473
1474     /**
1475     * Guardar datos de cuenta en localStorage
1476     */
1477     private saveAccountDataToLocalStorage(accountID: string, data: any): void {
1478         try {
1479             const key = `tradeSwitch_reportData_${accountID}`;
1480             localStorage.setItem(key, JSON.stringify(data));
1481         } catch (error) {
1482             console.error('Error saving account data to localStorage:', error);
1483         }
1484     }
1485
1486
1487     /**
1488     * Calcular y actualizar strategy_followed en el usuario
1489     * NUEVA LÓGICA: Verifica cada trade individual y si el plugin estuvo activo en el momento
1490     * exacto del trade (con hora)
1491     private async calculateAndUpdateStrategyFollowed(): Promise<void> {
1492         if (!this.user?.id) return;
1493
1494         try {
1495             // 1. Obtener plugin history del usuario primero
1496             const pluginHistoryArray = await
1497             this.pluginHistoryService.getPluginUsageHistory(this.user.id);
1498             if (pluginHistoryArray.length === 0) {
1499                 // No hay plugin history, asumir 0%
1500                 const updatedUser = {
1501                     ...this.user,
1502                     strategy_followed: 0,
1503                     lastUpdated: new Date().getTime() as unknown as Timestamp
1504                 };
1505                 await this.userService.createUser(updatedUser as unknown as User);
1506                 return;
1507             }
1508
1509             const pluginHistory = pluginHistoryArray[0];
1510
1511             // 2. Contar trades con plugin activo vs total de trades (validando hora exacta)
1512             let totalTrades = 0;
1513             let tradesWithActivePlugin = 0;
1514
1515             for (const account of this.accountsData) {
1516                 const accountData = this.loadAccountDataFromLocalStorage(account.accountID);

```

```

1517     if (accountData && accountData.accountHistory) {
1518         for (const trade of accountData.accountHistory) {
1519             if (trade.createdAt) {
1520                 totalTrades++;
1521                 // Convertir fecha del trade a UTC (con hora completa)
1522                 const tradeDate =
1523                     this.timezoneService.convertTradeDateToUTC(trade.createdAt);
1524                 // Verificar si el plugin estaba activo en el momento exacto de este trade
1525                 if (this.wasPluginActiveAtTime(tradeDate, pluginHistory)) {
1526                     tradesWithActivePlugin++;
1527                 }
1528             }
1529         }
1530     }
1531 }
1532
1533 if (totalTrades === 0) {
1534     // No hay trades, porcentaje es 0
1535     const updatedUser = {
1536         ...this.user,
1537         strategy_followed: 0,
1538         lastUpdated: new Date().getTime() as unknown as Timestamp
1539     };
1540     await this.userService.createUser(updatedUser as unknown as User);
1541     return;
1542 }
1543
1544 // 3. Calcular porcentaje: (trades con plugin activo / total trades) * 100
1545 const strategyFollowedPercent = totalTrades > 0
1546     ? Math.round((tradesWithActivePlugin / totalTrades) * 100 * 10) / 10
1547     : 0;
1548
1549 // 4. Actualizar usuario en Firebase
1550 const updatedUser = {
1551     ...this.user,
1552     strategy_followed: strategyFollowedPercent,
1553     lastUpdated: new Date().getTime() as unknown as Timestamp
1554 };
1555
1556 await this.userService.createUser(updatedUser as unknown as User);
1557
1558 } catch (error) {
1559     console.error('Error calculating strategy_followed:', error);
1560 }
1561 }
1562
1563 /**
1564 * Verificar si el plugin estaba activo en un momento específico (con hora exacta)
1565 * @param tradeDate Fecha y hora del trade en UTC
1566 * @param pluginHistory Plugin history con dateActive y dateInactive
1567 */
1568 private wasPluginActiveAtTime(tradeDate: Date, pluginHistory: PluginHistory): boolean {
1569     if (!pluginHistory.dateActive || !pluginHistory.dateInactive) {
1570         return false;
1571     }
1572
1573     const dateActive = pluginHistory.dateActive;
1574     const dateInactive = pluginHistory.dateInactive;
1575
1576     // Obtener timestamp del trade en UTC
1577     const tradeTimestamp = tradeDate.getTime();
1578
1579     // Si dateActive tiene más elementos que dateInactive, está activo desde la última fecha
1580     if (dateActive.length > dateInactive.length) {
1581         const lastActiveDate = this.timezoneService.convertToUTC(dateActive[dateActive.length - 1]);
1582         const lastActiveTimestamp = lastActiveDate.getTime();
1583
1584         // El trade debe ser >= a la última fecha/hora activa
1585         return tradeTimestamp >= lastActiveTimestamp;
1586     }

```

```

1587
1588    // Si tienen la misma cantidad, hay pares de activación/desactivación
1589    // Verificar si el timestamp del trade está dentro de algún rango activo [dateActive[i],
1590    dateInactive[i]]# 0; i < dateActive.length; i++) {
1591        const activeDate = this.timezoneService.convertToUTC(dateActive[i]);
1592        const inactiveDate = this.timezoneService.convertToUTC(dateInactive[i]);
1593
1594        const activeTimestamp = activeDate.getTime();
1595        const inactiveTimestamp = inactiveDate.getTime();
1596
1597        // El trade debe estar >= activeTimestamp y < inactiveTimestamp (rango [active,
1598        inactive])#tradeTimestamp >= activeTimestamp && tradeTimestamp < inactiveTimestamp) {
1599            return true;
1600        }
1601    }
1602
1603    return false;
1604}
1605
1606 /**
1607 * Cargar datos de cuenta desde localStorage
1608 */
1609 private loadAccountDataFromLocalStorage(accountID: string): any {
1610     try {
1611         const key = `tradeSwitch_reportData_${accountID}`;
1612         const data = localStorage.getItem(key);
1613         return data ? JSON.parse(data) : null;
1614     } catch (error) {
1615         console.error('Error loading account data from localStorage:', error);
1616         return null;
1617     }
1618 }
1619
1620 // ===== MÉTODOS DE REFRESH =====
1621
1622 /**
1623 * Refrescar datos de la cuenta actual
1624 */
1625 refreshCurrentAccountData() {
1626     if (!this.currentAccount) {
1627         console.warn('No hay cuenta seleccionada para refrescar');
1628         return;
1629     }
1630
1631     // Iniciar loading interno
1632     this.startInternalLoading();
1633
1634     // Reset account-related loading states
1635     this.setLoadingState('userKey', false);
1636     this.setLoadingState('historyData', false);
1637     this.setLoadingState('balanceData', false);
1638
1639     // Limpiar datos anteriores COMPLETAMENTE
1640     this.store.dispatch(setGroupedTrades({ groupedTrades: [] }));
1641     this.accountHistory = [];
1642     this.stats = undefined;
1643     this.balanceData = null;
1644     this.statsProcessed = false;
1645     this.chartsRendered = false;
1646
1647     // Limpiar datos guardados de la cuenta actual del contexto
1648     this.appContext.clearTradingHistoryForAccount(this.currentAccount.accountID);
1649
1650     // Cargar datos frescos de la cuenta actual
1651     this.loadAccountData(this.currentAccount).then(() => {
1652         // Después de cargar, recalcular métricas globales del usuario
1653         this.calculateAndUpdateUserMetrics();
1654         // Recalcular strategy_followed
1655         this.calculateAndUpdateStrategyFollowed();
1656         // Recalcular métricas de la cuenta activa

```

```

1657      this.calculateAndUpdateAccountsMetrics([this.currentAccount!]);
1658    });
1659  }
1660
1661 /**
1662 * Calcular y actualizar métricas por cuenta (AccountData)
1663 * - netPnl: suma de pnl
1664 * - profit: profit factor de la cuenta
1665 * - bestTrade: mayor |pnl|
1666 * Si no hay users/cuentas cargadas, no hace nada
1667 */
1668 private async calculateAndUpdateAccountsMetrics(targetAccounts?: AccountData[]): Promise
1669   targetAccounts = targetAccounts || this.accountsData;
1670   if (!accounts || accounts.length === 0) return;
1671
1672   try {
1673     for (const account of accounts) {
1674       const local = this.loadAccountDataFromLocalStorage(account.accountID);
1675       const trades = Array.isArray(local?.accountHistory) ? local.accountHistory : [];
1676
1677       if (trades.length === 0) {
1678         await this.userService.updateAccount(account.id, {
1679           ...account,
1680           netPnl: 0,
1681           profit: 0,
1682           bestTrade: 0,
1683         } as AccountData);
1684         continue;
1685       }
1686
1687       // Normalizar PnL
1688       const normalized = trades.map((t: any) => ({ ...t, pnl: t.pnl ?? 0 }));
1689       const netPnl = calculateNetPnl(normalized);
1690       const profitFactor = this.calculateProfitFactor(normalized);
1691       const bestTrade = this.calculateGlobalBestTrade(normalized); // reutilizamos el
1692       cálculo de mejor trade
1693       await this.userService.updateAccount(account.id, {
1694         ...account,
1695         netPnl: Math.round(netPnl * 100) / 100,
1696         profit: Math.round(profitFactor * 100) / 100,
1697         bestTrade: Math.round(bestTrade * 100) / 100,
1698       } as AccountData);
1699     }
1700   } catch (error) {
1701     console.error('Error updating account metrics:', error);
1702   }
1703 }
1704 }

```

Ø=ÜÁ features\report\components\calendar

Ø=ÜÁ features\report\components\calendar\calendar.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import {
3   Component,
4   Input,
5   Output,
6   EventEmitter,
7   SimpleChanges,
8 } from '@angular/core';
9 import {
10   CalendarDay,

```

```

11     GroupedTradeFinal,
12     PluginHistoryRecord,
13   } from '../../../../../models/report.model';
14   import { ReportService } from '../../../../../service/report.service';
15   import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';
16   import { TradesPopupComponent } from '../trades-popup/trades-popup.component';
17   import { ConfigurationOverview } from '../../../../../strategy/models/strategy.model';
18   import { PluginHistoryService, PluginHistory } from '../../../../../shared/services/plugin-
19   history/appcontextservice' from '../../../../../shared/context';
20   import { TradeLocker ApiService } from '../../../../../shared/services/tradelocker-api.service';
21   import { TimezoneService } from '../../../../../shared/services/timezone.service';
22
23   @Component({
24     selector: 'app-calendar',
25     templateUrl: './calendar.component.html',
26     styleUrls: ['./calendar.component.scss'],
27     standalone: true,
28     imports: [CommonModule, TradesPopupComponent],
29   })
30   export class CalendarComponent {
31     @Input() groupedTrades!: GroupedTradeFinal[];
32     @Output() strategyFollowedPercentageChange = new EventEmitter<number>();
33     @Input() strategies!: ConfigurationOverview[];
34     @Input() userId!: string; // Necesario para obtener el plugin history
35
36     calendar: CalendarDay[][][] = [];
37     currentDate!: Date;
38     selectedMonth!: Date;
39
40     // Popup properties
41     showTradesPopup = false;
42     selectedDay: CalendarDay | null = null;
43
44     // Plugin history properties
45     pluginHistory: PluginHistory | null = null;
46
47     constructor(
48       private reportSvc: ReportService,
49       private pluginHistoryService: PluginHistoryService,
50       private appContext: AppContextService,
51       private tradeLocker ApiService: TradeLocker ApiService,
52       private timezoneService: TimezoneService
53     ) {}
54     private numberFormatter = new NumberFormatterService();
55
56     ngOnChanges(changes: SimpleChanges) {
57       this.currentDate = new Date();
58       this.selectedMonth = new Date(this.currentDate.getFullYear(),
59       this.currentDate.getMonth());
59
60       // Obtener userId desde el contexto de autenticación
61       this.loadUserIdAndInitialize();
62
63       // Procesar trades para obtener nombres de instrumentos
64       this.processTradesForCalendar();
65
66       this.generateCalendar(this.selectedMonth);
67
68       if (this.strategies && this.strategies.length > 0) {
69         this.getPercentageStrategyFollowedLast30Days();
70       }
71     }
72
73     emitStrategyFollowedPercentage(value: number): void {
74       this.strategyFollowedPercentageChange.emit(value);
75     }
76
77     /**
78      * Obtener userId desde el contexto de autenticación y inicializar
79      */
80     private async loadUserIdAndInitialize() {

```

```

81    try {
82        // Obtener el usuario actual desde el contexto
83        const currentUser = this.appContext.currentUser();
84
85        if (currentUser && currentUser.id) {
86            this.userId = currentUser.id;
87
88            // Cargar plugin history con el userId obtenido
89            await this.loadPluginHistory();
90        }
91    } catch (error) {
92        console.error('Error obteniendo userId desde contexto:', error);
93    }
94}
95
96 /**
97 * Convertir Timestamp de Firestore o ISO 8601 string a Date
98 */
99 private convertFirestoreTimestamp(timestamp: any): Date {
100    // Si es string ISO 8601 (formato prioritario)
101    if (typeof timestamp === 'string') {
102        return new Date(timestamp);
103    }
104    // Fallback: Firestore Timestamp (legacy)
105    if (timestamp && typeof timestamp === 'object' && 'seconds' in timestamp) {
106        return new Date(timestamp.seconds * 1000 + (timestamp.nanoseconds || 0) / 1000000);
107    }
108    // Fallback final
109    return new Date(timestamp);
110}
111
112 /**
113 * MEJORA: Convertir fecha a UTC considerando zona horaria del usuario
114 * MÉTODO ESPECÍFICO: Para fechas de trades del servidor
115 */
116 private convertToUTCWithTimezone(date: Date | string | number): Date {
117    try {
118        // Usar el método específico para fechas de trades
119        return this.timezoneService.convertTradeDateToUTC(date);
120    } catch (error) {
121        console.error('Error convirtiendo fecha a UTC:', error);
122        // Fallback: conversión básica
123        return new Date(date);
124    }
125}
126
127 /**
128 * Procesar trades para obtener nombres de instrumentos
129 */
130 private async processTradesForCalendar() {
131    if (!this.groupedTrades || this.groupedTrades.length === 0) {
132        return;
133    }
134
135    // Verificar si los trades ya tienen nombres de instrumentos correctos
136    const firstTrade = this.groupedTrades[0];
137    const needsProcessing = !firstTrade.instrument ||
138        firstTrade.instrument === firstTrade.tradeableInstrumentId ||
139        firstTrade.instrument === '' ||
140        firstTrade.instrument === 'Cargando...';
141
142    if (!needsProcessing) {
143        return; // Ya están procesados
144    }
145
146    try {
147        // Obtener instrumentos únicos (optimización: solo una petición por combinación única)
148        const uniqueInstruments = new Map<string, { tradeableInstrumentId: string, routeId: string }>();
149        this.groupedTrades.forEach(trade => {
150

```

```

151 if (trade.tradableInstrumentId && trade.routeId) {
152   const key = `${trade.tradableInstrumentId}-${trade.routeId}`;
153   if (!uniqueInstruments.has(key)) {
154     uniqueInstruments.set(key, {
155       tradableInstrumentId: trade.tradableInstrumentId,
156       routeId: trade.routeId
157     });
158   }
159 }
160 );
161
162 // Establecer "Cargando..." como valor inicial para todos los trades
163 this.groupedTrades.forEach(trade => {
164   if (trade.tradableInstrumentId && trade.routeId) {
165     trade.instrument = 'Cargando...';
166   }
167 });
168
169 // Obtener detalles de instrumentos (una sola petición por combinación única)
170 const instrumentDetailsMap = new Map<string, { lotSize: number, name: string }>();
171
172 for (const [key, instrument] of uniqueInstruments) {
173   try {
174     // Obtener el token real del usuario desde el contexto
175     const userKey = await this.getUserToken();
176     if (!userKey) {
177       throw new Error('Token de usuario no disponible');
178     }
179
180     const instrumentDetails = await this.reportSvc.getInstrumentDetails(
181       userKey, // Token real del usuario
182       instrument.tradableInstrumentId,
183       instrument.routeId,
184       1
185     ).toPromise();
186
187     if (instrumentDetails) {
188       instrumentDetailsMap.set(key, {
189         lotSize: instrumentDetails.lotSize || 1,
190         name: instrumentDetails.name || instrument.tradableInstrumentId
191       });
192     } else {
193       instrumentDetailsMap.set(key, {
194         lotSize: 1,
195         name: instrument.tradableInstrumentId
196       });
197     }
198   } catch (error) {
199     console.warn(`Error obteniendo detalles del instrumento ${key}:`, error);
200     instrumentDetailsMap.set(key, {
201       lotSize: 1,
202       name: instrument.tradableInstrumentId
203     });
204   }
205 }
206
207 // Actualizar trades con nombres de instrumentos (aplicar a todos los trades con la
208 misma combinación)
209 this.groupedTrades.forEach(trade => {
210   if (trade.tradableInstrumentId && trade.routeId) {
211     const key = `${trade.tradableInstrumentId}-${trade.routeId}`;
212     const instrumentDetails = instrumentDetailsMap.get(key);
213
214     if (instrumentDetails) {
215       trade.instrument = instrumentDetails.name;
216     } else {
217       trade.instrument = trade.tradableInstrumentId; // Fallback al ID
218     }
219   }
220 });

```

```

221     } catch (error) {
222       console.error('Error procesando trades para calendario:', error);
223       // En caso de error, establecer fallback
224       this.groupedTrades.forEach(trade => {
225         if (trade.tradableInstrumentId && trade.routeId) {
226           trade.instrument = trade.tradableInstrumentId;
227         }
228       });
229     }
230   }
231 }
232 /**
233 * Obtener el token del usuario desde el store
234 */
235 private async getUserToken(): Promise<string | null> {
236   try {
237     // Obtener la cuenta actual del contexto
238     const accounts = this.appContext.userAccounts();
239     if (!accounts || accounts.length === 0) {
240       return null;
241     }
242   }
243
244   const currentAccount = accounts[0]; // Tomar la primera cuenta
245
246   // Usar el TradeLocker ApiService para obtener el token
247   const userKey = await this.tradeLocker ApiService.getUserKey(
248     currentAccount.emailTradingAccount,
249     currentAccount.brokerPassword,
250     currentAccount.server
251   ).toPromise();
252
253   if (userKey) {
254     return userKey;
255   }
256
257   return null;
258 } catch (error) {
259   console.error('Error obteniendo token del usuario:', error);
260   return null;
261 }
262 }
263 /**
264 * Cargar plugin history para el usuario
265 */
266 async loadPluginHistory() {
267   try {
268     const pluginHistoryArray = await
269     this.pluginHistoryService.getPluginUsageHistory(this.userId);
270     this.pluginHistory = pluginHistoryArray[0];
271   } else {
272     this.pluginHistory = null;
273   }
274 } catch (error) {
275   console.error('Error loading plugin history:', error);
276   this.pluginHistory = null;
277 }
278 }
279 }
280 /**
281 * Determinar qué estrategia se siguió en una fecha específica
282 * NUEVA LÓGICA: Asociar trades con estrategias basándose en fechas exactas
283 * @param tradeDate - Fecha y hora exacta del trade a validar
284 * @returns nombre de la estrategia seguida o null si no se siguió ninguna
285 */
286 getStrategyFollowedOnDate(tradeDate: Date): string | null {
287   // PASO 1: Verificar si el plugin estaba activo en la fecha/hora exacta del trade
288   const pluginActiveRange = this.getPluginActiveRange(tradeDate);
289   if (!pluginActiveRange) {

```

```

291     return null; // Plugin no estaba activo
292 }
293
294 // PASO 2: Buscar estrategias que incluyan la fecha/hora exacta del trade
295 const activeStrategy = this.getActiveStrategyAtTime(tradeDate);
296 if (!activeStrategy) {
297     return null; // No había estrategia activa en ese momento
298 }
299
300 return activeStrategy;
301 }
302
303 /**
304 * PASO 1: Determinar si el plugin estaba activo en la fecha/hora exacta del trade
305 * MEJORA: Usar conversión UTC para comparaciones precisas
306 * @param tradeDate - Fecha y hora exacta del trade
307 * @returns rango activo del plugin o null si no estaba activo
308 */
309 private getPluginActiveRange(tradeDate: Date): { start: Date, end: Date } | null {
310     if (!this.pluginHistory || !this.pluginHistory.dateActive || !
311         this.pluginHistory.dateInactive) {
312     }
313
314     const dateActive = this.pluginHistory.dateActive;
315     const dateInactive = this.pluginHistory.dateInactive;
316     const now = new Date();
317
318     // MEJORA: Convertir fecha del trade a UTC para comparación precisa
319     const tradeDateUTC = this.convertToUTCWithTimezone(tradeDate);
320
321
322     // Crear rangos de actividad del plugin
323     const activeRanges: { start: Date, end: Date }[] = [];
324
325     // Si dateActive tiene más elementos que dateInactive, está activo hasta ahora
326     if (dateActive.length > dateInactive.length) {
327         // Crear rangos para todos los pares completos
328         for (let i = 0; i < dateInactive.length; i++) {
329             activeRanges.push({
330                 start: this.convertToUTCWithTimezone(dateActive[i]),
331                 end: this.convertToUTCWithTimezone(dateInactive[i])
332             });
333         }
334         // El último rango activo va desde la última fecha de active hasta ahora
335         activeRanges.push({
336             start: this.convertToUTCWithTimezone(dateActive[dateActive.length - 1]),
337             end: this.convertToUTCWithTimezone(now)
338         });
339     } else {
340         // Si tienen la misma cantidad, crear rangos de fechas
341         for (let i = 0; i < dateActive.length; i++) {
342             activeRanges.push({
343                 start: this.convertToUTCWithTimezone(dateActive[i]),
344                 end: this.convertToUTCWithTimezone(dateInactive[i])
345             });
346         }
347     }
348
349     // Verificar si el plugin estaba activo en la fecha/hora exacta del trade
350     for (const range of activeRanges) {
351         if (tradeDateUTC >= range.start && tradeDateUTC <= range.end) {
352             return range; // Plugin estaba activo en este rango
353         }
354     }
355
356     return null; // Plugin no estaba activo
357 }
358
359 /**
360 * PASO 2: Buscar la estrategia activa en la fecha/hora exacta del trade

```

```

361     * MEJORA: Usar conversión UTC para comparaciones precisas
362     * @param tradeDate - Fecha y hora exacta del trade
363     * @returns nombre de la estrategia activa o null si no había ninguna
364     */
365     private getActiveStrategyAtTime(tradeDate: Date): string | null {
366         if (!this.strategies || this.strategies.length === 0) {
367             return null;
368         }
369
370         // MEJORA: Convertir fecha del trade a UTC para comparación precisa
371         const tradeDateUTC = this.convertToUTCWithTimezone(tradeDate);
372
373
374         // Buscar estrategias activas en la fecha/hora exacta del trade
375         for (const strategy of this.strategies) {
376             // IMPORTANTE: NO filtrar estrategias eliminadas aquí
377             // Las estrategias eliminadas (soft delete) SÍ deben considerarse
378             // porque en el momento del trade existían y podrían haber sido seguidas
379
380             if (this.isStrategyActive(strategy, tradeDateUTC)) {
381                 return strategy.name || 'Unknown Strategy';
382             }
383         }
384
385         return null; // No había estrategia activa en ese momento
386     }
387
388     /**
389      * Verificar si una estrategia específica estaba activa en la fecha/hora exacta
390      * MEJORA: Usar conversión UTC para comparaciones precisas
391      * @param strategy - Estrategia a verificar
392      * @param tradeDate - Fecha y hora exacta del trade (ya en UTC)
393      * @returns true si la estrategia estaba activa, false si no
394      */
395     private isStrategyActiveAtTime(strategy: ConfigurationOverview, tradeDate: Date): boolean {
396         // Si la estrategia no tiene fechas de activación, no estaba activa
397         if (!strategy.dateActive || !strategy.dateInactive) {
398             return false;
399         }
400
401         const strategyActive = strategy.dateActive;
402         const strategyInactive = strategy.dateInactive;
403         const now = new Date();
404
405         // Crear rangos de actividad de la estrategia
406         const strategyRanges: { start: Date, end: Date }[] = [];
407
408         // Si strategyActive tiene más elementos que strategyInactive, está activa hasta ahora
409         if (strategyActive.length > strategyInactive.length) {
410             // Crear rangos para todos los pares completos
411             for (let i = 0; i < strategyInactive.length; i++) {
412                 strategyRanges.push({
413                     start:
414                     this.convertToUTCWithTimezone(this.convertFirestoreTimestamp(strategyActive[i])),
415                     end: this.convertToUTCWithTimezone(this.convertFirestoreTimestamp(strategyInactive[i]))
416                 })
417                 // El último rango activo va desde la última fecha de active hasta ahora
418                 strategyRanges.push({
419                     start: this.convertToUTCWithTimezone(this.convertFirestoreTimestamp(strategyActive[st
420                     rategyActive.length - 1].convertToUTCWithTimezone(now)
421                 }));
422             } else {
423                 // Si tienen la misma cantidad, crear rangos de fechas
424                 for (let i = 0; i < strategyActive.length; i++) {
425                     strategyRanges.push({
426                         start:
427                         this.convertToUTCWithTimezone(this.convertFirestoreTimestamp(strategyActive[i])),
428                         end: this.convertToUTCWithTimezone(this.convertFirestoreTimestamp(strategyInactive[i]))
429                     })
430                 }
431             }
432         }
433     }

```

```

431     // Verificar si la estrategia estaba activa en la fecha/hora exacta del trade
432     for (const range of strategyRanges) {
433         if (tradeDate >= range.start && tradeDate <= range.end) {
434             return true; // Estrategia estaba activa en este rango
435         }
436     }
437     return false; // Estrategia no estaba activa
438   }
439
440 /**
441 * Determinar si se siguió la estrategia basándose en los rangos de fechas del plugin
442 * @param tradeDate - Fecha del trade a validar
443 * @returns true si se siguió la estrategia, false si no
444 */
445 didFollowStrategy(tradeDate: Date): boolean {
446   return this.getStrategyFollowedOnDate(tradeDate) !== null;
447 }
448
449 /**
450 * Obtener información detallada sobre la estrategia seguida en un trade específico
451 * @param tradeDate - Fecha y hora exacta del trade
452 * @returns objeto con información detallada sobre la estrategia seguida
453 */
454 getTradeStrategyInfo(tradeDate: Date): {
455   followedStrategy: boolean;
456   strategyName: string | null;
457   pluginActive: boolean;
458   pluginActiveRange: { start: Date, end: Date } | null;
459   strategyActiveRange: { start: Date, end: Date } | null;
460 } {
461   const pluginActiveRange = this.getPluginActiveRange(tradeDate);
462   const strategyName = this.getActiveStrategyAtTime(tradeDate);
463
464   // Obtener el rango activo de la estrategia si existe
465   let strategyActiveRange: { start: Date, end: Date } | null = null;
466   if (strategyName && this.strategies) {
467     const strategy = this.strategies.find(s => s.name === strategyName);
468     if (strategy && strategy.dateActive && strategy.dateInactive) {
469       strategyActiveRange = this.getStrategyActiveRange(strategy, tradeDate);
470     }
471   }
472
473   return {
474     followedStrategy: strategyName !== null,
475     strategyName,
476     pluginActive: pluginActiveRange !== null,
477     pluginActiveRange,
478     strategyActiveRange
479   };
480 }
481
482 /**
483 * Obtener el rango activo de una estrategia específica en una fecha
484 * @param strategy - Estrategia a verificar
485 * @param tradeDate - Fecha del trade
486 * @returns rango activo de la estrategia o null si no estaba activa
487 */
488 private getStrategyActiveRange(strategy: ConfigurationOverview, tradeDate: Date): { start:
489 Date, end: Date } | null {
490   if (!strategy.dateActive || !strategy.dateInactive) {
491     return null;
492   }
493
494   const strategyActive = strategy.dateActive;
495   const strategyInactive = strategy.dateInactive;
496   const now = new Date();
497
498   // Crear rangos de actividad de la estrategia
499   const strategyRanges: { start: Date, end: Date }[] = [];
500

```

```

501
502 // Si strategyActive tiene más elementos que strategyInactive, está activa hasta ahora
503 if (strategyActive.length > strategyInactive.length) {
504     // Crear rangos para todos los pares completos
505     for (let i = 0; i < strategyInactive.length; i++) {
506         strategyRanges.push({
507             start: this.convertFirestoreTimestamp(strategyActive[i]),
508             end: this.convertFirestoreTimestamp(strategyInactive[i])
509         });
510     }
511     // El último rango activo va desde la última fecha de active hasta ahora
512     strategyRanges.push({
513         start: this.convertFirestoreTimestamp(strategyActive[strategyActive.length - 1]),
514         end: now
515     });
516 } else {
517     // Si tienen la misma cantidad, crear rangos de fechas
518     for (let i = 0; i < strategyActive.length; i++) {
519         strategyRanges.push({
520             start: this.convertFirestoreTimestamp(strategyActive[i]),
521             end: this.convertFirestoreTimestamp(strategyInactive[i])
522         });
523     }
524 }
525
526 // Buscar el rango que contiene la fecha del trade
527 for (const range of strategyRanges) {
528     if (tradeDate >= range.start && tradeDate <= range.end) {
529         return range;
530     }
531 }
532
533 return null;
534 }
535
536 generateCalendar(targetMonth: Date) {
537     const tradesByDay: { [date: string]: GroupedTradeFinal[] } = {};
538
539     // Primero, filtrar trades válidos (con positionId válido) y deduplicar
540     const validTrades = this.groupedTrades.filter(trade =>
541         trade.positionId &&
542         trade.positionId !== 'null' &&
543         trade.positionId !== '' &&
544         trade.positionId !== null
545     );
546
547     // Deduplicar por positionId
548     const uniqueTrades = validTrades.filter((trade, index, self) =>
549         index === self.findIndex(t => t.positionId === trade.positionId)
550     );
551
552     // Agrupar trades únicos por día usando la zona horaria del dispositivo
553     uniqueTrades.forEach((trade) => {
554         // MEJORA: Usar conversión correcta de fecha de trade
555         const tradeDate = this.convertTOUTCWithTimezone(Number(trade.lastModified));
556
557         // Usar la zona horaria local del dispositivo
558         const key = `${tradeDate.getFullYear()}-${tradeDate.getMonth()}-${tradeDate.getDate()}`
559     );
560     // Solo incluir trades que estén en el mes seleccionado
561     const tradeYear = tradeDate.getFullYear();
562     const tradeMonth = tradeDate.getMonth();
563     const targetYear = targetMonth.getFullYear();
564     const targetMonthIndex = targetMonth.getMonth();
565
566     if (tradeYear === targetYear && tradeMonth === targetMonthIndex) {
567         if (!tradesByDay[key]) tradesByDay[key] = [];
568         tradesByDay[key].push(trade);
569     }
570 });

```

```

571
572 // Generar calendario del mes objetivo
573 const year = targetMonth.getFullYear();
574 const month = targetMonth.getMonth();
575 const firstDay = new Date(year, month, 1);
576 const lastDay = new Date(year, month + 1, 0);
577
578 // Calcular inicio y fin de la semana
579 let startDay = new Date(firstDay);
580 startDay.setDate(firstDay.getDate() - firstDay.getDay());
581 let endDay = new Date(lastDay);
582 endDay.setDate(lastDay.getDate() + (6 - lastDay.getDay()));
583
584 const days: CalendarDay[] = [];
585 let currentDay = new Date(startDay);
586
587 while (currentDay <= endDay) {
588     const key = `${currentDay.getFullYear()}-${currentDay.getMonth()}-
589 ${currentDay.getDate()}`;
590     const tradesByDay = tradesByDay[key] || [];
591     const pnlTotal = trades.reduce((acc, t) => acc + (t.pnl ?? 0), 0);
592
593     const wins = trades.filter((t) => (t.pnl ?? 0) > 0).length;
594     const losses = trades.filter((t) => (t.pnl ?? 0) < 0).length;
595     const tradesCount = trades.length;
596     const tradeWinPercent = tradesCount > 0 ? Math.round((wins / tradesCount) * 1000) /
597 10 : 0;
598
599     // Determinar si se siguió la estrategia basándose en los rangos de fechas del plugin
600     // Para cada día, verificar si ALGÚN trade siguió la estrategia
601     let followedStrategy = false;
602     let strategyName: string | null = null;
603
604     if (tradesCount > 0) {
605         // Verificar cada trade individualmente usando su fecha/hora exacta
606         for (const trade of trades) {
607             // MEJORA: Usar conversión correcta de fecha de trade
608             const tradeDate = this.convertToUTCWithTimezone(Number(trade.lastModified));
609             const tradeStrategyInfo = this.getTradeStrategyInfo(tradeDate);
610
611             if (tradeStrategyInfo.followedStrategy) {
612                 followedStrategy = true;
613                 strategyName = tradeStrategyInfo.strategyName;
614                 break; // Si al menos un trade siguió la estrategia, el día cuenta
615             }
616         }
617
618         days.push({
619             date: new Date(currentDay),
620             trades: trades as GroupedTradeFinal[],
621             pnlTotal,
622             tradesCount: trades.length,
623             followedStrategy: followedStrategy,
624             tradeWinPercent: Math.round(tradeWinPercent),
625             strategyName: strategyName,
626             isCurrentMonth: currentDay.getMonth() === month && currentDay.getFullYear() === year,
627         });
628
629         currentDay.setDate(currentDay.getDate() + 1);
630     }
631
632     // Organizar en semanas
633     this.calendar = [];
634     for (let i = 0; i < days.length; i += 7) {
635         this.calendar.push(days.slice(i, i + 7));
636     }
637 }
638
639 getDateNDaysAgo(daysAgo: number): Date {
640     const date = new Date();

```

```

641     date.setDate(date.getDate() - daysAgo);
642     return date;
643   }
644
645   filterDaysInRange(
646     days: CalendarDay[],
647     fromDate: Date,
648     toDate: Date
649   ): CalendarDay[] {
650     return days.filter((day) => day.date >= fromDate && day.date <= toDate);
651   }
652
653   countStrategyFollowedDays(days: CalendarDay[]): number {
654     return days.filter((day) => day.followedStrategy && day.tradesCount > 0)
655       .length;
656   }
657
658   calculateStrategyFollowedPercentage(
659     days: CalendarDay[],
660     periodDays: number
661   ): number {
662     if (days.length === 0) return 0;
663
664     const fromDate = this.getDateNDaysAgo(periodDays - 1);
665     const toDate = new Date();
666
667     const daysInRange = this.filterDaysInRange(days, fromDate, toDate);
668     const count = this.countStrategyFollowedDays(daysInRange);
669
670     const percentage = (count / periodDays) * 100;
671
672     return Math.round(percentage * 10) / 10;
673   }
674
675   getPercentageStrategyFollowedLast30Days() {
676     const percentage = this.calculateStrategyFollowedPercentage(
677       this.calendar.flat(),
678       30
679     );
680     this.emitStrategyFollowedPercentage(percentage);
681   }
682
683   getCurrentMonthYear(): string {
684     const options: Intl.DateTimeFormatOptions = { month: 'short' };
685     const month = this.selectedMonth.toLocaleString('en-US', options);
686     const year = this.selectedMonth.getFullYear();
687     return `${month}, ${year}`;
688   }
689
690   // Navigation methods
691   canNavigateLeft(): boolean {
692     if (!this.groupedTrades || this.groupedTrades.length === 0) return false;
693
694     const earliestTradeDate = this.getEarliestTradeDate();
695     const firstDayOfSelectedMonth = new Date(this.selectedMonth.getFullYear(),
696       this.selectedMonth.getMonth(), 1);
697     return earliestTradeDate < firstDayOfSelectedMonth;
698   }
699
700   canNavigateRight(): boolean {
701     if (!this.groupedTrades || this.groupedTrades.length === 0) return false;
702
703     const latestTradeDate = this.getLatestTradeDate();
704     const lastDayOfSelectedMonth = new Date(this.selectedMonth.getFullYear(),
705       this.selectedMonth.getMonth() + 1, 0);
706     return latestTradeDate > lastDayOfSelectedMonth;
707   }
708
709   private getEarliestTradeDate(): Date {
710     if (!this.groupedTrades || this.groupedTrades.length === 0) return new Date();

```

```

711     const dates = this.groupedTrades.map(trade => new Date(Number(trade.lastModified)));
712     return new Date(Math.min(...dates.map(d => d.getTime())));
713   }
714
715   private getLatestTradeDate(): Date {
716     if (!this.groupedTrades || this.groupedTrades.length === 0) return new Date();
717
718     const dates = this.groupedTrades.map(trade => new Date(Number(trade.lastModified)));
719     return new Date(Math.max(...dates.map(d => d.getTime())));
720   }
721
722   navigateToPreviousMonth(): void {
723     if (this.canNavigateLeft()) {
724       this.selectedMonth = new Date(this.selectedMonth.getFullYear(),
725         this.selectedMonth.getMonth() - 1);
726       this.generateCalendar(this.selectedMonth);
727     }
728   }
729
730   navigateToNextMonth(): void {
731     if (this.canNavigateRight()) {
732       this.selectedMonth = new Date(this.selectedMonth.getFullYear(),
733         this.selectedMonth.getMonth() + 1);
734     }
735   }
736
737   navigateToCurrentMonth(): void {
738     this.selectedMonth = new Date(this.currentDate.getFullYear(),
739       this.currentDate.getMonth());
740   }
741
742   // Export functionality
743   exportData() {
744     const csvData = this.generateCSVData();
745     this.downloadCSV(csvData, `trading-data-${this.currentMonthYear.replace(' ', '-')}.toLowerCase().csv`);
746   }
747
748   generateCSVData(): string {
749     const headers = ['Date', 'PnL Total', 'Trades Count', 'Win Percentage', 'Strategy
750     Followed', `Row${this.strategyNames!}`.join(',')];
751
752     this.calendar.flat().forEach(day => {
753       const date = day.date.toISOString().split('T')[0];
754       const pnlTotal = day.pnlTotal.toFixed(2);
755       const tradesCount = day.tradesCount;
756       const winPercentage = day.tradeWinPercent;
757       const strategyFollowed = day.followedStrategy ? 'Yes' : 'No';
758       const strategyName = day.strategyName || 'None';
759
760       rows.push([date, pnlTotal, tradesCount, winPercentage, strategyFollowed,
761       strategyName].join(','));
762
763     return rows.join('\n');
764   }
765
766   downloadCSV(csvData: string, filename: string) {
767     const blob = new Blob([csvData], { type: 'text/csv;charset=utf-8;' });
768     const link = document.createElement('a');
769
770     if (link.download !== undefined) {
771       const url = URL.createObjectURL(blob);
772       link.setAttribute('href', url);
773       link.setAttribute('download', filename);
774       link.style.visibility = 'hidden';
775       document.body.appendChild(link);
776       link.click();
777       document.body.removeChild(link);
778     }
779   }
780

```

```

781 // Weekly summary methods
782 getWeekTotal(week: CalendarDay[]): number {
783   return week.reduce((total, day) => total + day.pnlTotal, 0);
784 }
785
786 getWeekActiveDays(week: CalendarDay[]): number {
787   return week.filter(day => day.tradesCount > 0).length;
788 }
789
790 // Popup methods
791 onDayClick(day: CalendarDay) {
792   if (day.tradesCount > 0) {
793     this.selectedDay = day;
794     this.showTradesPopup = true;
795   }
796 }
797
798 onClosePopup() {
799   this.showTradesPopup = false;
800   this.selectedDay = null;
801 }
802
803 formatCurrency(value: number): string {
804   return this.numberFormatter.formatCurrency(value);
805 }
806
807 formatPercentage(value: number): string {
808   return this.numberFormatter.formatPercentage(value);
809 }
810
811 /**
812 * Obtener resumen de estrategias seguidas en un período
813 * @param days - Array de días del calendario
814 * @returns objeto con estadísticas de estrategias
815 */
816 getStrategySummary(days: CalendarDay[]): {
817   totalDays: number;
818   strategyDays: number;
819   strategiesUsed: { [strategyName: string]: number };
820   strategyPercentage: number;
821 } {
822   const totalDays = days.length;
823   let strategyDays = 0;
824   const strategiesUsed: { [strategyName: string]: number } = {};
825
826   days.forEach(day => {
827     if (day.followedStrategy && day.strategyName) {
828       strategyDays++;
829       if (strategiesUsed[day.strategyName]) {
830         strategiesUsed[day.strategyName]++;
831       } else {
832         strategiesUsed[day.strategyName] = 1;
833       }
834     }
835   });
836
837   const strategyPercentage = totalDays > 0 ? Math.round((strategyDays / totalDays) * 100 * 10) / 10 : 0;
838   return {
839     totalDays,
840     strategyDays,
841     strategiesUsed,
842     strategyPercentage
843   };
844 }
845
846
847 /**
848 * Obtener resumen de estrategias para los últimos N días
849 * @param days - Número de días a analizar
850 * @returns resumen de estrategias

```

```

851     */
852     getStrategySummaryLastNDays(days: number): {
853       totalDays: number;
854       strategyDays: number;
855       strategiesUsed: { [strategyName: string]: number };
856       strategyPercentage: number;
857     } {
858       const fromDate = this.getDateNDaysAgo(days - 1);
859       const toDate = new Date();
860       const daysInRange = this.filterDaysInRange(this.calendar.flat(), fromDate, toDate);
861
862       return this.getStrategySummary(daysInRange);
863     }
864
865     /**
866      * Obtener análisis detallado de trades que siguieron estrategias
867      * @param days - Array de días del calendario
868      * @returns análisis detallado de trades y estrategias
869      */
870     getDetailedTradeAnalysis(days: CalendarDay[]): {
871       totalTrades: number;
872       tradesWithStrategy: number;
873       tradesWithoutStrategy: number;
874       strategyCompliance: number;
875       tradesByStrategy: { [strategyName: string]: number };
876       tradesWithoutPlugin: number;
877       tradesWithPluginButNoStrategy: number;
878     } {
879       let totalTrades = 0;
880       let tradesWithStrategy = 0;
881       let tradesWithoutStrategy = 0;
882       let tradesWithoutPlugin = 0;
883       let tradesWithPluginButNoStrategy = 0;
884       const tradesByStrategy: { [strategyName: string]: number } = {};
885
886       days.forEach(day => {
887         if (day.trades && day.trades.length > 0) {
888           day.trades.forEach(trade => {
889             totalTrades++;
890             // MEJORA: Usar conversión correcta de fecha de trade
891             const tradeDate = this.convertToUTCWithTimezone(Number(trade.lastModified));
892             const tradeStrategyInfo = this.getTradeStrategyInfo(tradeDate);
893
894             if (!tradeStrategyInfo.pluginActive) {
895               tradesWithoutPlugin++;
896             } else if (tradeStrategyInfo.followedStrategy) {
897               tradesWithStrategy++;
898               const strategyName = tradeStrategyInfo.strategyName || 'Unknown';
899               tradesByStrategy[strategyName] = (tradesByStrategy[strategyName] || 0) + 1;
900             } else {
901               tradesWithPluginButNoStrategy++;
902             }
903           });
904         }
905       });
906
907       tradesWithoutStrategy = tradesWithoutPlugin + tradesWithPluginButNoStrategy;
908       const strategyCompliance = totalTrades > 0 ? Math.round((tradesWithStrategy / totalTrades) * 100 * 10) / 10 : 0;
909
910       return {
911         totalTrades,
912         tradesWithStrategy,
913         tradesWithoutStrategy,
914         strategyCompliance,
915         tradesByStrategy,
916         tradesWithoutPlugin,
917         tradesWithPluginButNoStrategy
918       };
919     }
920

```

```

921  /**
922   * Obtener análisis detallado de trades para los últimos N días
923   * @param days - Número de días a analizar
924   * @returns análisis detallado de trades
925   */
926  getDetailedTradeAnalysisLastNDays(days: number): {
927    totalTrades: number;
928    tradesWithStrategy: number;
929    tradesWithoutStrategy: number;
930    strategyCompliance: number;
931    tradesByStrategy: { [strategyName: string]: number };
932    tradesWithoutPlugin: number;
933    tradesWithPluginButNoStrategy: number;
934  } {
935    const fromDate = this.getDateNDaysAgo(days - 1);
936    const toDate = new Date();
937    const daysInRange = this.filterDaysInRange(this.calendar.flat(), fromDate, toDate);
938
939    return this.getDetailedTradeAnalysis(daysInRange);
940  }
941
942  /**
943   * Verificar estado actual del plugin
944   * @returns true si el plugin está activo ahora, false si no
945   */
946  isPluginCurrentlyActive(): boolean {
947    if (!this.pluginHistory) {
948      return false;
949    }
950
951    return this.pluginHistoryService.isPluginActiveByDates(this.pluginHistory);
952  }
953
954  /**
955   * Obtener información detallada del estado del plugin
956   * @returns información completa del estado del plugin
957   */
958  getPluginStatusInfo(): {
959    isActive: boolean;
960    lastActiveDate: string | null;
961    lastInactiveDate: string | null;
962    activeRanges: { start: string, end: string }[];
963  } {
964    if (!this.pluginHistory) {
965      return {
966        isActive: false,
967        lastActiveDate: null,
968        lastInactiveDate: null,
969        activeRanges: []
970      };
971    }
972
973    const isActive = this.pluginHistoryService.isPluginActiveByDates(this.pluginHistory);
974    const lastActiveDate = this.pluginHistory.dateActive?.[this.pluginHistory.dateActive.length - 1] || null;
975    const dateInactive = this.pluginHistory.dateInactive?.[this.pluginHistory.dateInactive.length - 1] || null;
976
977    // Crear rangos activos para mostrar
978    const activeRanges: { start: string, end: string }[] = [];
979    if (this.pluginHistory.dateActive && this.pluginHistory.dateInactive) {
980      const dateActive = this.pluginHistory.dateActive;
981      const dateInactive = this.pluginHistory.dateInactive;
982
983      for (let i = 0; i < Math.min(dateActive.length, dateInactive.length); i++) {
984        activeRanges.push({
985          start: new Date(dateActive[i]).toISOString(),
986          end: new Date(dateInactive[i]).toISOString()
987        });
988      }
989    }
990  }

```

```

991     return {
992       isActive,
993       lastActiveDate,
994       lastInactiveDate,
995       activeRanges
996     };
997   }
998 }
999 }
1000

```

Ø=ÜÁ features\report\components\pnlGraph

Ø=ÜÄ features\report\components\pnlGraph\pnlGraph.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import {
3   Component,
4   Inject,
5   Input,
6   OnInit,
7   OnChanges,
8   Output,
9   PLATFORM_ID,
10  EventEmitter,
11  SimpleChanges,
12  HostListener,
13 } from '@angular/core';
14 import { GroupedTrade, GroupedTradeFinal } from '../../../../../models/report.model';
15 import { NgApexchartsModule } from 'ng-apexcharts';
16 import { getMonthlyPnL } from '../../../../../utils/normalization-utils';
17 import { FormsModule } from '@angular/forms';
18 import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';
19
20 /**
21 * Component for displaying PnL (Profit and Loss) chart.
22 *
23 * This component displays an area chart showing cumulative PnL over time.
24 * It supports filtering by year or custom date range, and displays monthly
25 * or dynamic date-based aggregations depending on the selected range.
26 *
27 * Features:
28 * - Area chart visualization using ApexCharts
29 * - Year-based filtering (past 2 years, current year, next 5 years)
30 * - Custom date range filtering
31 * - Monthly aggregation for year view
32 * - Dynamic aggregation (daily/weekly/monthly) for date ranges
33 * - Total profit calculation for filtered data
34 * - Interactive tooltips with percentage changes
35 *
36 * Relations:
37 * - NgApexchartsModule: Chart rendering
38 * - NumberFormatterService: Value formatting
39 *
40 * @component
41 * @selector app-PnL-Graph
42 * @standalone true
43 */
44 @Component({
45   selector: 'app-PnL-Graph',
46   templateUrl: './pnlGraph.component.html',
47   styleUrls: ['./pnlGraph.component.scss'],
48   standalone: true,

```

```

49     imports: [CommonModule, NgApexchartsModule, FormsModule],
50   })
51   export class PnlGraphComponent implements OnInit, OnChanges {
52     @Input() values!: GroupedTradeFinal[];
53     @Output() onYearChange = new EventEmitter<string>();
54
55     public chartOptions: any;
56     private numberFormatter = new NumberFormatterService();
57
58     year!: string;
59     dateRanges: { label: string; value: string }[] = [];
60
61     // Date filter properties
62     showDateFilter = false;
63     selectedStartDate: string = '';
64     selectedEndDate: string = '';
65     filteredData: GroupedTradeFinal[] = [];
66     originalData: GroupedTradeFinal[] = [];
67
68     constructor(@Inject(PLATFORM_ID) private platformId: any) {}
69
70     ngOnInit() {
71       this.year = new Date().getFullYear().toString();
72       this.generateYearRangesPast(3);
73       this.initializeData();
74     }
75
76     ngOnChanges(changes: SimpleChanges) {
77       if (changes['values'] && this.values) {
78         this.initializeData();
79       }
80     }
81
82     private initializeData() {
83       if (this.values && this.values.length > 0) {
84         this.originalData = [...this.values];
85         this.filteredData = [...this.values];
86         this.chartOptions = this.getChartOptions(this.filteredData);
87       } else {
88         // Mostrar gráfica vacía cuando no hay datos
89         this.originalData = [];
90         this.filteredData = [];
91         this.chartOptions = this.getEmptyChartOptions();
92       }
93     }
94
95     generateYearRangesPast(yearsBack: number) {
96       const now = new Date();
97       const currentYear = now.getFullYear();
98       this.dateRanges = [];
99
100      // Generate 2 years before current and 5 years after
101      for (let i = 2; i >= 0; i--) {
102        const year = currentYear - i;
103        this.dateRanges.push({
104          label: `Jan ${year} - Dec ${year}`,
105          value: `${year}`,
106        });
107      }
108
109      for (let i = 1; i <= 5; i++) {
110        const year = currentYear + i;
111        this.dateRanges.push({
112          label: `Jan ${year} - Dec ${year}`,
113          value: `${year}`,
114        });
115      }
116    }
117
118    getChartOptions(trades: GroupedTradeFinal[]): any {

```

```

119     const yearValue = this.year;
120
121     // Si hay filtro de fechas activo, no aplicar filtro de año
122     let filteredTrades = trades;
123     if (!this.selectedStartDate && !this.selectedEndDate) {
124         filteredTrades = this.applyYearFilter(trades);
125     }
126
127     // Use monthly chart by default
128     const chartConfig = this.getMonthlyChartConfig(filteredTrades, yearValue);
129
130     return {
131         chart: {
132             type: 'area',
133             height: 350,
134             toolbar: { show: false },
135             foreColor: '#fff',
136             fontFamily: 'Inter, Arial, sans-serif',
137             background: 'transparent',
138         },
139         series: [
140             {
141                 name: 'PnL',
142                 data: chartConfig.data,
143             },
144             ],
145             xaxis: {
146                 categories: chartConfig.categories,
147                 labels: {
148                     style: { colors: '#d8d8d8' },
149                 },
150                 axisBorder: { show: false },
151                 axisTicks: { show: false },
152             },
153             yaxis: {
154                 labels: {
155                     style: { colors: '#d8d8d8' },
156                 },
157             },
158             grid: {
159                 borderColor: '#333',
160                 strokeDashArray: 4,
161                 xaxis: {
162                     lines: {
163                         show: true,
164                     },
165                 },
166             },
167             dataLabels: { enabled: false },
168             stroke: {
169                 curve: 'straight',
170                 width: 1,
171                 colors: ['#EAF2F8'],
172             },
173             fill: {
174                 type: 'gradient',
175                 gradient: {
176                     shade: 'dark',
177                     type: 'vertical',
178                     gradientToColors: ['#3967D7'],
179                     opacityFrom: 0.4,
180                     opacityTo: 0,
181                 },
182             },
183             tooltip: {
184                 theme: 'dark',
185                 x: { show: true },
186                 custom: ({ series, seriesIndex, dataPointIndex, w }: any) => {
187                     const value = series[seriesIndex][dataPointIndex];
188                     const category = w.globals.categoryLabels[dataPointIndex];

```

```

189
190     const prevValue =
191         dataPointIndex > 0 ? series[seriesIndex][dataPointIndex - 1] : null;
192     let percentDiff: number | null = 0;
193     let direction = null;
194     let cardClass = '';
195     let validatorClass = '';
196
197     if (prevValue !== null && prevValue !== 0) {
198         percentDiff = ((value - prevValue) / Math.abs(prevValue)) * 100;
199         direction = percentDiff > 0 ? 'up' : 'down';
200         if (direction === 'up') {
201             cardClass = 'positive-container';
202             validatorClass = 'positive-validator';
203         } else {
204             cardClass = 'negative-container';
205             validatorClass = 'negative-validator';
206         }
207     } else {
208         percentDiff = null;
209         direction = null;
210     }
211
212     const formattedValue = this.getFormatedValue(value);
213     const formattedPercent = this.numberFormatter.formatPercentageValue(percentDiff);
214
215     return `<div class=" ${cardClass} regularText color-background d-flex flex-col
216 toolTip-containr items-center">$color-text-gray">${category}, ${yearValue}</p>
217 <div class="d-flex text-container items-center ">
218     <p class= "subtitle">
219         ${formattedValue}
220     </p>
221     ${
222         percentDiff != null
223         ? `<span class="smallText py-4 px-6 d-flex justify-center items-center
224 ${validatorClass}"${percentDiff}>${percentDiff}<span class="${
225             ? 'icon-status-arrow-up'
226             : 'icon-status-arrow'
227         } ml-3"></span></span>` :
228         ''
229     }
230 </div>
231
232
233     </div>`;
234 },
235     position: function (data: any, opts: any) {
236         return {
237             left: data.point.x,
238             top: data.point.y - 160,
239         };
240     },
241 },
242 );
243 }
244
245 applyYearFilter(trades: GroupedTradeFinal[]): GroupedTradeFinal[] {
246     if (!trades || trades.length === 0) return [];
247
248     let filteredTrades = [...trades];
249
250     // Filter by year
251     const yearValue = parseInt(this.year);
252     filteredTrades = filteredTrades.filter(trade => {
253         const tradeDate = new Date(Number(trade.lastModified));
254         return tradeDate.getFullYear() === yearValue;
255     });
256
257     return filteredTrades;
258 }

```

```

259
260     applyDateRangeFilter(trades: GroupedTradeFinal[], startDate: string, endDate: string):
261     GroupedTradeFinal[] & !endDate) {
262         return trades;
263     }
264
265     const start = startDate ? new Date(startDate) : new Date(0);
266     const end = endDate ? new Date(endDate) : new Date();
267
268     // Si solo hay fecha de inicio, usar fin del día
269     if (startDate && !endDate) {
270         end.setHours(23, 59, 59, 999);
271     }
272     // Si solo hay fecha de fin, usar inicio del día
273     if (!startDate && endDate) {
274         start.setHours(0, 0, 0, 0);
275     }
276
277     return trades.filter(trade => {
278         const tradeDate = new Date(Number(trade.lastModified));
279         return tradeDate >= start && tradeDate <= end;
280     });
281 }
282
283
284     getMonthlyChartConfig(trades: GroupedTradeFinal[], yearValue: string) {
285         const monthlyMap: { [label: string]: number } = {};
286
287         // Si hay filtro de fechas activo, generar categorías dinámicas
288         if (this.selectedStartDate || this.selectedEndDate) {
289             return this.getDateRangeChartConfig(trades);
290         }
291
292         trades.forEach(trade => {
293             const date = new Date(Number(trade.lastModified));
294             const tradeYear = date.getFullYear();
295             const yearValueNum = parseInt(yearValue);
296
297             // Only process trades from the selected year
298             if (tradeYear === yearValueNum) {
299                 const label = this.capitalizeFirstLetter(
300                     date.toLocaleString('en', { month: 'short' })
301                 );
302                 const sum = (monthlyMap[label] ?? 0) + (trade.pnl ?? 0);
303                 monthlyMap[label] = sum < 1 ? Math.round(sum * 100) / 100 : Math.round(sum);
304             }
305         });
306     }
307
308     const monthOrder = [
309         'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
310         'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec',
311     ];
312
313     const data = monthOrder.map(m => monthlyMap[m] ?? 0);
314     const categories = monthOrder;
315     return { data, categories };
316 }
317
318     getDateRangeChartConfig(trades: GroupedTradeFinal[]) {
319         const dateMap: { [key: string]: number } = {};
320
321         // Determinar el rango de fechas
322         let startDate: Date;
323         let endDate: Date;
324
325         if (this.selectedStartDate && this.selectedEndDate) {
326             startDate = new Date(this.selectedStartDate);
327             endDate = new Date(this.selectedEndDate);
328         } else if (this.selectedStartDate) {

```

```

329     startDate = new Date(this.selectedStartDate);
330     endDate = new Date();
331 } else if (this.selectedEndDate) {
332     startDate = new Date(0);
333     endDate = new Date(this.selectedEndDate);
334 } else {
335     // Fallback a año completo
336     const year = new Date().getFullYear();
337     startDate = new Date(year, 0, 1);
338     endDate = new Date(year, 11, 31);
339 }
340
341 // Procesar trades en el rango
342 trades.forEach(trade => {
343     const tradeDate = new Date(Number(trade.lastModified));
344     if (tradeDate >= startDate && tradeDate <= endDate) {
345         const label = this.formatDateForChart(tradeDate);
346         const sum = (dateMap[label] ?? 0) + (trade.pnl ?? 0);
347         dateMap[label] = sum < 1 ? Math.round(sum * 100) / 100 : Math.round(sum);
348     }
349 });
350
351 // Generar categorías dinámicas basadas en el rango
352 const categories = this.generateDateRangeCategories(startDate, endDate);
353 const data = categories.map(cat => dateMap[cat] ?? 0);
354
355 return { data, categories };
356 }
357
358 formatDateForChart(date: Date): string {
359     const month = date.toLocaleString('en', { month: 'short' });
360     const day = date.getDate();
361     return `${month} ${day}`;
362 }
363
364 generateDateRangeCategories(startDate: Date, endDate: Date): string[] {
365     const categories: string[] = [];
366     const current = new Date(startDate);
367
368     // Si el rango es menor a 30 días, mostrar por días
369     const diffTime = endDate.getTime() - startDate.getTime();
370     const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));
371
372     if (diffDays <= 30) {
373         // Mostrar por días
374         while (current <= endDate) {
375             categories.push(this.formatDateForChart(new Date(current)));
376             current.setDate(current.getDate() + 1);
377         }
378     } else if (diffDays <= 90) {
379         // Mostrar por semanas
380         while (current <= endDate) {
381             const weekEnd = new Date(current);
382             weekEnd.setDate(weekEnd.getDate() + 6);
383             if (weekEnd > endDate) weekEnd.setTime(endDate.getTime());
384
385             const startStr = this.formatDateForChart(new Date(current));
386             const endStr = this.formatDateForChart(weekEnd);
387             categories.push(` ${startStr} - ${endStr}`);
388
389             current.setDate(current.getDate() + 7);
390         }
391     } else {
392         // Mostrar por meses
393         while (current <= endDate) {
394             const month = current.toLocaleString('en', { month: 'short' });
395             const year = current.getFullYear();
396             categories.push(` ${month} ${year}`);
397
398             current.setMonth(current.getMonth() + 1);

```

```

399     }
400   }
401 
402   return categories;
403 }
404 
405   capitalizeFirstLetter(str: string): string {
406     if (!str) return '';
407     return str.charAt(0).toUpperCase() + str.slice(1);
408   }
409 
410   onYearSelected(year: string) {
411     this.year = year;
412     this.onYearChange.emit(year);
413     this.updateChart();
414   }
415 
416   getTotalProfit(): number {
417     // Apply the same filters as the chart
418     let filteredTrades = this.filteredData;
419     if (!this.selectedStartDate && !this.selectedEndDate) {
420       filteredTrades = this.applyYearFilter(this.filteredData);
421     }
422 
423     const totalProfit = filteredTrades.reduce(
424       (acc, trade) => acc + (trade.pnl ?? 0),
425       0
426     );
427     const result = Math.round(totalProfit * 100) / 100;
428     return result;
429   }
430 
431   // Date filter methods
432   toggleDateFilter() {
433     this.showDateFilter = !this.showDateFilter;
434   }
435 
436   closeDateFilter() {
437     this.showDateFilter = false;
438   }
439 
440   @HostListener('document:click', ['$event'])
441   onDocumentClick(event: Event) {
442     const target = event.target as HTMLElement;
443     const filterSection = target.closest('.filter-section');
444     if (!filterSection && this.showDateFilter) {
445       this.closeDateFilter();
446     }
447   }
448 
449   onStartDateSelected(date: string) {
450     this.selectedStartDate = date;
451   }
452 
453   onEndDateSelected(date: string) {
454     this.selectedEndDate = date;
455   }
456 
457   applyDateFilter() {
458     if (this.selectedStartDate || this.selectedEndDate) {
459       this.filteredData = this.applyDateRangeFilter(this.originalData,
460       this.selectedStartDate, this.selectedEndDate);
461       this.updateYearFromDateRange();
462     } else {
463       this.filteredData = [...this.originalData];
464     }
465     this.updateChart();
466     this.closeDateFilter();
467   }
468

```

```

469 updateYearFromDateRange() {
470   if (this.selectedStartDate) {
471     const startDate = new Date(this.selectedStartDate);
472     this.year = startDate.getFullYear().toString();
473   } else if (this.selectedEndDate) {
474     const endDate = new Date(this.selectedEndDate);
475     this.year = endDate.getFullYear().toString();
476   }
477 }
478
479 clearDateFilter() {
480   this.selectedStartDate = '';
481   this.selectedEndDate = '';
482   this.filteredData = [...this.originalData];
483   this.updateChart();
484 }
485
486 updateChart() {
487   this.chartOptions = this.getChartOptions(this.filteredData);
488 }
489
490 getAvailableYears(): string[] {
491   if (!this.values || this.values.length === 0) return [];
492
493   const years = new Set<number>();
494   this.values.forEach(trade => {
495     const date = new Date(Number(trade.lastModified));
496     years.add(date.getFullYear());
497   });
498
499   return Array.from(years).sort((a, b) => b - a).map(year => year.toString());
500 }
501
502 getEmptyChartOptions(): any {
503   const currentYear = new Date().getFullYear();
504   const months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
505   'Nov', 'Dec'];
506   return {
507     chart: {
508       type: 'area',
509       height: 350,
510       toolbar: { show: false },
511       foreColor: '#ffff',
512       fontFamily: 'Inter, Arial, sans-serif',
513       background: 'transparent',
514     },
515     series: [
516       {
517         name: 'PnL',
518         data: new Array(12).fill(0), // Array de 12 ceros para los 12 meses
519       },
520     ],
521     xaxis: {
522       categories: months,
523       labels: {
524         style: { colors: '#d8d8d8' },
525       },
526       axisBorder: { show: false },
527       axisTicks: { show: false },
528     },
529     yaxis: {
530       labels: {
531         style: { colors: '#d8d8d8' },
532       },
533     },
534     grid: {
535       borderColor: '#333',
536       strokeDashArray: 4,
537       xaxis: {
538         lines: {

```

```

539         show: true,
540     },
541   },
542 },
543 dataLabels: { enabled: false },
544 stroke: {
545   curve: 'straight',
546   width: 1,
547   colors: ['#EAF2F8'],
548 },
549 fill: {
550   type: 'gradient',
551   gradient: {
552     shade: 'dark',
553     type: 'vertical',
554     gradientToColors: ['#3967D7'],
555     opacityFrom: 0.4,
556     opacityTo: 0,
557   },
558 },
559 tooltip: {
560   theme: 'dark',
561   x: { show: true },
562   custom: function ({ series, seriesIndex, dataPointIndex, w }: any) {
563     const month = months[dataPointIndex];
564     return `
565       <div style="padding: 8px 12px; background: #lalala; border: 1px solid #333;
566 border-radius:<span style="color: #d8d8d8; font-size: 12px;">${month} ${currentYear}</span></div>
567         <div style="color: #fff; font-size: 14px; font-weight: 600;">PnL: $0.00</div>
568       </div>
569     `;
570   },
571 },
572 noData: {
573   text: 'No data available',
574   align: 'center',
575   verticalAlign: 'middle',
576   style: {
577     color: '#d8d8d8',
578     fontSize: '14px',
579     fontFamily: 'Inter, Arial, sans-serif'
580   }
581 },
582 };
583 }
584 }
585 getFormatedValue(value: number): string {
586   return this.numberFormatter.formatCurrencyValue(value);
587 }
588 }
589 }
590 }
591

```

Ø=ÜÁ features\report\components\rule-short

Ø=ÜÁ features\report\components\rule-short\rule-short.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { Component, Input } from '@angular/core';
3
4 /**
5  * Component for displaying a short rule indicator.

```

```

6  *
7  * This component displays a rule title and an active/inactive status indicator.
8  * It is used in the report view to show which trading rules are currently active.
9  *
10 * @component
11 * @selector app-rule-short
12 * @standalone true
13 */
14 @Component({
15   selector: 'app-rule-short',
16   templateUrl: './rule-short.component.html',
17   styleUrls: ['./rule-short.component.scss'],
18   standalone: true,
19   imports: [CommonModule],
20 })
21 export class RuleShortComponent {
22   @Input() title!: string;
23   @Input() isActive?: any;
24
25   constructor() {}
26 }
27

```

Ø=ÜÁ features\report\components\statCard

Ø=ÜÁ features\report\components\statCard\stat_card.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { Component, Input, Injectable } from '@angular/core';
3 import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';
4
5 /**
6  * Component for displaying a statistical card with formatted values.
7  *
8  * This component displays a title and value with automatic formatting based on the format
9  * type by auto-detecting the format from the title. It also applies color coding for certain
10 metrics.
11
12 * Format types:
13 * - 'currency': Formats as currency (e.g., $1,234.56)
14 * - 'percentage': Formats as percentage (e.g., 45.5%)
15 * - 'number': Formats as number or integer based on title
16
17 * Color coding:
18 * - Net P&L: Green for positive, red for negative
19 * - Profit Factor: Green if >= 1.0, red if < 1.0
20 * - Trade Win %: Green if >= 50%, red if < 50%
21 * - Avg Win/Loss: Green if >= 1.0, red if < 1.0
22
23 * @component
24 * @selector app-stat-card
25 * @standalone true
26 */
27 @Component({
28   selector: 'app-stat-card',
29   templateUrl: './stat_card.component.html',
30   styleUrls: ['./stat_card.component.scss'],
31   standalone: true,
32   imports: [CommonModule],
33 })
34 @Injectable()
35 export class statCardComponent {
36   @Input() title!: string;
37   @Input() value?: string | number;

```

```

37     @Input() formatType?: 'currency' | 'percentage' | 'number';
38
39     private numberFormatter = new NumberFormatterService();
40
41     /**
42      * Gets the formatted value based on format type or auto-detection.
43      *
44      * If formatType is specified, uses that format. Otherwise, auto-detects format
45      * from the title (e.g., titles containing "P&L" or "profit" use currency format).
46      *
47      * Related to:
48      * - NumberFormatterService: Handles actual formatting
49      *
50      * @returns Formatted value string
51      * @memberof statCardComponent
52      */
53     getFormattedValue(): string {
54         if (this.value === null || this.value === undefined) {
55             return '0';
56         }
57
58         switch (this.formatType) {
59             case 'currency':
60                 return this.numberFormatter.formatCurrency(this.value);
61             case 'percentage':
62                 return this.numberFormatter.formatPercentage(this.value);
63             case 'number':
64                 // Check if this is a count (like total trades, active positions) that should be an
65                 integer if (this.title.toLowerCase().includes('total') &&
66                         this.title.toLowerCase().includes('trade')) {
67                     return this.numberFormatter.formatInteger(this.value);
68                 }
69                 if (this.title.toLowerCase().includes('active') &&
70                     this.title.toLowerCase().includes('position')) {
71                     return this.numberFormatter.formatInteger(this.value);
72                 }
73                 if (this.title.toLowerCase().includes('users')) {
74                     return this.numberFormatter.formatInteger(this.value);
75                 }
76                 if (this.title.toLowerCase().includes('subscriptions')) {
77                     return this.numberFormatter.formatInteger(this.value);
78                 }
79                 return this.numberFormatter.formatNumber(this.value);
80         default:
81             // Auto-detect format based on title
82             if (this.title.toLowerCase().includes('p&l') ||
83                 this.title.toLowerCase().includes('revenue') ||
84                 this.title.toLowerCase().includes('sales') ||
85                 this.title.toLowerCase().includes('profit') ||
86                 this.title.toLowerCase().includes('balance')) {
87                 return this.numberFormatter.formatCurrency(this.value);
88             } else if (this.title.toLowerCase().includes('%') ||
89                         this.title.toLowerCase().includes('percent') ||
90                         this.title.toLowerCase().includes('win rate')) {
91                 return this.numberFormatter.formatPercentage(this.value);
92             } else if (this.title.toLowerCase().includes('total') &&
93                         this.title.toLowerCase().includes('trade')) {
94                 return this.numberFormatter.formatInteger(this.value);
95             } else if (this.title.toLowerCase().includes('active') &&
96                         this.title.toLowerCase().includes('position')) {
97                 return this.numberFormatter.formatInteger(this.value);
98             } else if (this.title.toLowerCase().includes('users')) {
99                 return this.numberFormatter.formatInteger(this.value);
100            } else if (this.title.toLowerCase().includes('subscriptions')) {
101                return this.numberFormatter.formatInteger(this.value);
102            } else {
103                return this.numberFormatter.formatNumber(this.value);
104            }
105        }
106    }

```

```

107 /**
108  * Gets the CSS color class for the value based on the metric type and value.
109 *
110 * Applies color coding for specific metrics:
111 * - Net P&L: Green (positive) or red (negative)
112 * - Profit Factor: Green (>= 1.0) or red (< 1.0)
113 * - Trade Win %: Green (>= 50%) or red (< 50%)
114 * - Avg Win/Loss: Green (>= 1.0) or red (< 1.0)
115 * - Other metrics: Default background color
116 *
117 *
118 * @returns CSS class name for value color
119 * @memberof statCardComponent
120 */
121 getValueColorClass(): string {
122   if (this.value === null || this.value === undefined) {
123     return 'color-background';
124   }
125
126   const numericValue = Number(this.value);
127
128   // Solo aplicar colores a métricas específicas
129
130   // Para Net P&L: rojo si es negativo, verde si es positivo
131   if (this.title.toLowerCase().includes('p&l')) {
132     return numericValue < 0 ? 'color-error' : 'color-success';
133   }
134
135   // Para Profit Factor: rojo si es menor a 1.0, verde si es mayor a 1.0
136   if (this.title.toLowerCase().includes('profit') &&
137       this.title.toLowerCase().includes('factor')) {
138     return numericValue < 1.0 ? 'color-error' : 'color-success';
139   }
140
141   // Para Trade Win %: rojo si es menor a 50%, verde si es mayor a 50%
142   if (this.title.toLowerCase().includes('win') &&
143       this.title.toLowerCase().includes('%')) {
144     return numericValue < 50 ? 'color-error' : 'color-success';
145   }
146
147   // Para Avg Win/Loss: rojo si es menor a 1.0, verde si es mayor a 1.0
148   if (this.title.toLowerCase().includes('avg') &&
149       this.title.toLowerCase().includes('win')) {
150     return numericValue < 1.0 ? 'color-error' : 'color-success';
151   }
152
153   // Para Balance, Total trades, Active positions: siempre blanco
154   return 'color-background';
155 }
156 }
157

```

Ø=ÜÁ features\report\components\trades-popup

Ø=ÜÁ features\report\components\trades-popup\trades-popup.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { Component, Input, Output, EventEmitter } from '@angular/core';
3 import { CalendarDay } from '../../../../../models/report.model';
4 import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';
5 import { GroupedTradeFinal } from '../../../../../models/report.model';
6 import { ConfigurationOverview } from '../../../../../strategy/models/strategy.model';
7

```

```

8  /**
9   * Interface representing a trade detail for display in the popup.
10  *
11  * @interface TradeDetail
12  */
13  export interface TradeDetail {
14    openTime: string;
15    ticker: string;
16    side: 'Long' | 'Short';
17    netPnl: number;
18    followedStrategy: boolean;
19    strategyName: string;
20  }
21
22 /**
23  * Component for displaying trades in a popup modal.
24  *
25  * This component displays detailed information about trades for a selected day,
26  * including trade time, ticker, side (Long/Short), PnL, and strategy compliance.
27  *
28  * Features:
29  * - Displays all trades for a selected calendar day
30  * - Shows trade details: time, ticker, side, PnL
31  * - Indicates if trades followed a strategy
32  * - Color-coded tickers and PnL values
33  * - Formatted currency and percentage values
34  *
35  * Relations:
36  * - CalendarComponent: Receives selected day data
37  * - NumberFormatterService: Value formatting
38  *
39  * @component
40  * @selector app-trades-popup
41  * @standalone true
42  */
43 @Component({
44   selector: 'app-trades-popup',
45   standalone: true,
46   imports: [CommonModule],
47   templateUrl: './trades-popup.component.html',
48   styleUrls: ['./trades-popup.component.scss']
49 })
50 export class TradesPopupComponent {
51   @Input() visible: boolean = false;
52   @Input() selectedDay: CalendarDay | null = null;
53   @Input() strategies: ConfigurationOverview[] = [];
54   @Output() close = new EventEmitter<void>();
55
56   trades: TradeDetail[] = [];
57   netPnl: number = 0;
58   netRoi: number = 0;
59   private numberFormatter = new NumberFormatterService();
60   selectedDate: string = '';
61
62   // Expose Math to template
63   Math = Math;
64
65   ngOnChanges() {
66     if (this.selectedDay && this.visible) {
67       this.loadTradesData();
68     }
69   }
70
71   loadTradesData() {
72     if (!this.selectedDay) return;
73
74     this.selectedDate = this.formatDate(this.selectedDay.date);
75     this.netPnl = this.selectedDay.pnlTotal;
76
77     console.log(this.selectedDay.trades);

```

```

78
79 // Convertir trades del día a formato de detalle
80 this.trades = this.selectedDay.trades.map((trade, index) => ({
81   openTime: this.formatTime(new Date(Number(trade.lastModified))),
82   ticker: trade.instrument ?? 'N/A',
83   side: this.determineSide(trade),
84   netPnl: trade.pnl ?? 0,
85   followedStrategy: this.selectedDay?.followedStrategy ?? false,
86   strategyName: this.getStrategyNameForTrade(trade)
87 }));
88
89 // Ordenar por tiempo (más reciente primero)
90 this.trades.sort((a, b) => b.openTime.localeCompare(a.openTime));
91 }
92
93 formatDate(date: Date): string {
94   const options: Intl.DateTimeFormatOptions = {
95     weekday: 'long',
96     year: 'numeric',
97     month: 'long',
98     day: 'numeric'
99   };
100  return date.toLocaleDateString('en-US', options);
101 }
102
103 formatTime(date: Date): string {
104   return date.toLocaleTimeString('en-US', {
105     hour12: false,
106     hour: '2-digit',
107     minute: '2-digit',
108     second: '2-digit'
109   });
110 }
111
112 determineSide(trade: GroupedTradeFinal): 'Long' | 'Short' {
113   // Usar el campo side real del trade para determinar Long/Short
114   if (trade.side === 'buy') {
115     return 'Long';
116   } else if (trade.side === 'sell') {
117     return 'Short';
118   }
119   // Fallback basado en PnL si no hay side
120   return (trade.pnl ?? 0) >= 0 ? 'Long' : 'Short';
121 }
122
123 getStrategyNameForTrade(trade: GroupedTradeFinal): string {
124   // Si no se siguió estrategia, no mostrar nombre
125   if (!this.selectedDay?.followedStrategy) {
126     return '';
127   }
128
129   if (!this.strategies || this.strategies.length === 0) {
130     return '-';
131   }
132
133   const tradeDate = new Date(Number(trade.lastModified));
134   const today = new Date();
135   today.setHours(23, 59, 59, 999); // 11:59 PM del día actual
136
137   // Buscar la estrategia que estaba activa en la fecha del trade
138   for (const strategy of this.strategies) {
139     // IMPORTANTE: NO filtrar estrategias eliminadas aquí
140     // Las estrategias eliminadas (soft delete) SÍ deben considerarse
141     // porque en el momento del trade existían y podrían haber sido seguidas
142
143     if (strategy.dateActive && strategy.dateActive.length > 0) {
144       // Revisar cada período de activación de esta estrategia
145       for (let i = 0; i < strategy.dateActive.length; i++) {
146         const activeDate = new Date(strategy.dateActive[i]);
147         let inactiveDate: Date;

```

```

148     // Si hay fecha de desactivación correspondiente, usarla
149     if (strategy.dateInactive && strategy.dateInactive.length > i) {
150         inactiveDate = new Date(strategy.dateInactive[i]);
151     } else {
152         // No hay fecha de desactivación, verificar si está activa actualmente
153         // Si dateActive tiene más elementos que dateInactive, está activa
154         const isCurrentlyActive = strategy.dateActive.length >
155             (strategy.dateInactive.length)0;
156         inactiveDate = today;
157     } else {
158         continue; // Esta activación ya fue desactivada
159     }
160 }
161
162     // Verificar si el trade está dentro de este rango de actividad
163     if (tradeDate >= activeDate && tradeDate <= inactiveDate) {
164         return strategy.name;
165     }
166 }
167 }
168 }
169 }
170
171     return '-';
172 }
173
174 getTickerColor(ticker: string): string {
175     // Asignar colores a diferentes tickers con transparencia
176     const colors: { [key: string]: string } = {
177         'YM': 'rgba(139, 92, 246, 0.8)',
178         'MYM': 'rgba(139, 92, 246, 0.6)',
179         'ES': 'rgba(59, 130, 246, 0.8)',
180         'NQ': 'rgba(16, 185, 129, 0.8)',
181         'RTY': 'rgba(245, 158, 11, 0.8)'
182     };
183     return colors[ticker] || 'rgba(107, 114, 128, 0.8)';
184 }
185
186 getPnlColor(pnl: number): string {
187     return pnl >= 0 ? '#10B981' : '#EF4444';
188 }
189
190 getStrategyIcon(followed: boolean): string {
191     return followed ? 'icon-check-box' : 'icon-uncheck-box';
192 }
193
194 getStrategyColor(followed: boolean): string {
195     return followed ? '#10B981' : '#EF4444';
196 }
197
198 onClose() {
199     this.close.emit();
200 }
201
202 onSortByTime() {
203     // Implementar lógica de ordenamiento
204     this.trades.sort((a, b) => b.openTime.localeCompare(a.openTime));
205 }
206
207 onFilter() {
208     // Implementar lógica de filtrado
209     console.log('Filter clicked');
210 }
211
212 formatCurrency(value: number): string {
213     return this.numberFormatter.formatCurrency(value);
214 }
215
216 formatPercentage(value: number): string {
217     return this.numberFormatter.formatPercentage(value);

```

```
218     }
219   }
220
```

Ø=ÜÁ features\report\components\winLossChart

Ø=ÜÁ features\report\components\winLossChart\win-loss-chart.component.ts

```
1 import { CommonModule } from '@angular/common';
2 import { Component, Input, OnInit, OnChanges, SimpleChanges, OnDestroy } from '@angular/
3   caper
4 import { NgApexchartsModule } from 'ng-apexcharts';
5 import { GroupedTradeFinal } from '../../../../../models/report.model';
6 import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';
7
8 /**
9  * Component for displaying win/loss ratio as a donut chart.
10 *
11 * This component displays a donut chart showing the percentage and monetary value
12 * of winning vs losing trades. The chart is responsive and adjusts donut size
13 * based on screen width.
14 *
15 * Features:
16 * - Donut chart visualization using ApexCharts
17 * - Win/loss percentage calculation
18 * - Win/loss monetary value calculation
19 * - Responsive design with adaptive donut size
20 * - Custom tooltips showing monetary values and percentages
21 * - Handles empty data state
22 *
23 * Relations:
24 * - NgApexchartsModule: Chart rendering
25 * - NumberFormatterService: Value formatting
26 *
27 * @component
28 * @selector app-win-loss-chart
29 * @standalone true
30 */
31 @Component({
32   selector: 'app-win-loss-chart',
33   templateUrl: './win-loss-chart.component.html',
34   styleUrls: ['./win-loss-chart.component.scss'],
35   standalone: true,
36   imports: [CommonModule, NgApexchartsModule],
37 })
38 export class WinLossChartComponent implements OnInit, OnChanges, OnDestroy {
39   @Input() values!: GroupedTradeFinal[];
40
41   public chartOptions: any;
42   private numberFormatter = new NumberFormatterService();
43   public winLossData: {
44     winValue: number;
45     lossValue: number;
46     winPercentage: number;
47     lossPercentage: number;
48   } = {
49     winValue: 0,
50     lossValue: 0,
51     winPercentage: 0,
52     lossPercentage: 0,
53   };
54   private resizeTimeout?: any;
```

```

56  private getDonutSize(): string {
57    if (typeof window !== 'undefined') {
58      if (window.innerWidth <= 480) {
59        return '85%'; // Más grueso en pantallas muy pequeñas para mejor visibilidad
60      } else if (window.innerWidth <= 768) {
61        return '80%'; // Moderadamente grueso en tablets
62      } else if (window.innerWidth <= 1024) {
63        return '75%'; // Tamaño medio en pantallas medianas
64      }
65    }
66    return '80%'; // Tamaño normal en desktop
67  }
68
69  private getChartSize(): number {
70    // El tamaño ahora se controla completamente por CSS
71    // Retornamos un valor que será ignorado por ApexCharts
72    return 100;
73  }
74
75  ngOnInit() {
76    this.winLossData = this.calculateWinLossData();
77    this.chartOptions = this.getChartOptions();
78
79    // Listener para redimensionar el gráfico con debounce
80    if (typeof window !== 'undefined') {
81      window.addEventListener('resize', () => {
82        if (this.resizeTimeout) {
83          clearTimeout(this.resizeTimeout);
84        }
85        this.resizeTimeout = setTimeout(() => {
86          this.chartOptions = this.getChartOptions();
87        }, 150);
88      });
89    }
90  }
91
92  ngOnChanges(changes: SimpleChanges) {
93    if (changes['values']) {
94      // Forzar recálculo y actualización del gráfico
95      this.winLossData = this.calculateWinLossData();
96      this.chartOptions = this.getChartOptions();
97
98      // Forzar re-renderizado del gráfico después de un pequeño delay
99      // Esto es especialmente importante cuando se cargan datos desde localStorage
100     setTimeout(() => {
101       this.chartOptions = this.getChartOptions();
102     }, 100);
103
104    // Segundo intento de actualización para asegurar que se renderice correctamente
105    setTimeout(() => {
106      this.chartOptions = this.getChartOptions();
107    }, 300);
108  }
109}
110
111  ngOnDestroy() {
112    if (this.resizeTimeout) {
113      clearTimeout(this.resizeTimeout);
114    }
115  }
116
117  getChartOptions(): any {
118    // Si no hay datos, mostrar círculo gris sin texto interno
119    if (!this.values || this.values.length === 0) {
120      return {
121        chart: {
122          type: 'donut',
123          height: '100%',
124          width: '100%',
125          toolbar: { show: false },

```

```

126         foreColor: '#fff',
127         fontFamily: 'Inter, Arial, sans-serif',
128         background: 'transparent',
129     },
130     series: [1], // Un solo valor para crear el círculo
131     labels: ['No Data'],
132     colors: ['#6B7280'], // Color gris
133     dataLabels: {
134         enabled: false
135     },
136     plotOptions: {
137         pie: {
138             donut: {
139                 size: this.getDonutSize(),
140                 labels: {
141                     show: false // No mostrar texto dentro del círculo
142                 }
143             }
144         },
145         legend: {
146             show: false
147         },
148         tooltip: {
149             enabled: false
150         },
151         stroke: {
152             show: false
153         }
154     };
155 }
156
157
158 return {
159     chart: {
160         type: 'donut',
161         height: '100%',
162         width: '100%',
163         toolbar: { show: false },
164         foreColor: '#fff',
165         fontFamily: 'Inter, Arial, sans-serif',
166         background: 'transparent',
167     },
168     series: [this.winLossData.winPercentage, this.winLossData.lossPercentage],
169     labels: ['Win', 'Loss'],
170     colors: ['#9BF526', '#EC221F'],
171     dataLabels: {
172         enabled: false
173     },
174     plotOptions: {
175         pie: {
176             donut: {
177                 size: this.getDonutSize(),
178                 labels: {
179                     show: false
180                 }
181             }
182         }
183     },
184     legend: {
185         show: false
186     },
187     tooltip: {
188         enabled: true,
189         custom: ({ series, seriesIndex, dataPointIndex, w }: any) => {
190             const percentage = w.globals.seriesPercent[seriesIndex];
191             const isWin = seriesIndex === 0; // Ahora el índice 0 es Win
192             const color = isWin ? '#468506' : '#EC221F';
193
194             // Usar los valores monetarios reales en lugar del porcentaje
195             const moneyValue = isWin ? this.winLossData.winValue : this.winLossData.lossValue;

```

```

196     const formattedValue = this.numberFormatter.formatCurrency(moneyValue);
197     const formattedPercentage = this.numberFormatter.formatPercentageValue(percentage);
198
199     return `

200     <div class="custom-tooltip" style="

201         background: rgba(255, 255, 255, 0.95);

202         border: 1px solid #e0e0e0;

203         border-radius: 8px;

204         padding: 12px;

205         box-shadow: 0 4px 12px rgba(0, 0, 0, 0.2);

206         color: #333;

207         font-family: Inter, Arial, sans-serif;

208         min-width: 80px;

209         text-align: center;

210         font-weight: 600;

211     ">

212     <div style="

213         font-size: 16px;

214         font-weight: 700;

215         color: #333;

216         margin-bottom: 4px;

217         ">${formattedValue}</div>

218     <div style="

219         font-size: 12px;

220         font-weight: 500;

221         color: ${color};

222         ">${formattedPercentage}%</div>

223     </div>

224     `;

225     }

226     },

227     stroke: {

228         show: false

229     }

230 };

231

232 calculateWinLossData() {

233     if (!this.values || this.values.length === 0) {

234         return { winValue: 0, lossValue: 0, winPercentage: 0, lossPercentage: 0 };

235     }

236

237     let totalWinTrades = 0;

238     let totalLossTrades = 0;

239     let winAmount = 0;

240     let lossAmount = 0;

241

242     this.values.forEach(trade => {

243         const pnl = trade.pnl ?? 0;

244         if (pnl > 0) {

245             totalWinTrades++;

246             winAmount += pnl;

247         } else if (pnl < 0) {

248             totalLossTrades++;

249             lossAmount += Math.abs(pnl);

250         }

251     });

252

253     // Calcular solo winValue (plata real ganada) y winPercentage

254     const winValue = winAmount; // Plata real ganada

255     const totalTrades = totalWinTrades + totalLossTrades;

256     const winPercentage = totalTrades > 0 ? (totalWinTrades / totalTrades) * 100 : 0;

257

258     // Calcular lossValue y lossPercentage como lo que falta para completar

259     const lossValue = lossAmount; // Plata real perdida

260     const lossPercentage = 100 - winPercentage; // Lo que falta para llegar al 100%

261

262     const result = {

263         winValue: Math.round(winValue * 100) / 100,

264         lossValue: Math.round(lossValue * 100) / 100,

```

```

266     winPercentage: Math.round(winPercentage * 10) / 10,
267     lossPercentage: Math.round(lossPercentage * 10) / 10
268   };
269
270   return result;
271 }
272
273 getWinLossData() {
274   return this.calculateWinLossData();
275 }
276
277 formatCurrency(value: number): string {
278   return this.numberFormatter.formatCurrency(value);
279 }
280
281 formatPercentage(value: number): string {
282   return this.numberFormatter.formatPercentage(value);
283 }
284 }
285

```

Ø=ÜÁ features\report\models

Ø=ÜÄ features\report\models\report.model.ts

```

1  import { RuleType } from '../../../../../strategy/models/strategy.model';
2
3
4 /**
5  * Interface representing a historical trade record from the trading API.
6  *
7  * This interface maps the raw array data structure returned by the TradeLocker API
8  * into a structured object with named properties for easier access and manipulation.
9  *
10 * @interface historyTrade
11 */
12 export interface historyTrade {
13   id: string;
14   tradableInstrumentId: string;
15   routeId: string;
16   qty: string;
17   side: string;
18   type: string;
19   status: string;
20   filledQty: string;
21   avgPrice: string;
22   price: string;
23   stopPrice: string;
24   validity: string;
25   expireDate: string;
26   createdDate: string;
27   lastModified: string;
28   isOpen: string;
29   positionId: string;
30   stopLoss: string;
31   stopLossType: string;
32   takeProfit: string;
33   takeProfitType: string;
34   strategyId: string;
35 }
36
37 /**
38  * Interface representing a grouped trade with position information.

```

```

39  *
40  * This interface is used for intermediate processing of trades before final grouping.
41  * It contains position-level data including entry/exit prices, PnL, and trade status.
42  *
43  * @interface GroupedTrade
44  */
45  export interface GroupedTrade {
46    position_Id: string;
47    quantity?: number;
48    pnl?: number;
49    buy_price?: string;
50    sell_price?: string;
51    totalSpend?: number;
52    updatedAt: string;
53    // Nuevas propiedades para claridad
54    entryPrice?: string;
55    exitPrice?: string;
56    side?: string; // 'buy' o 'sell' de la posición
57    isWon?: boolean;
58    isOpen?: boolean;
59    stopLoss?: string;
60    takeProfit?: string;
61    allTrades?: historyTrade[]; // Todos los trades de esta posición
62  }
63
64 /**
65  * Interface representing a final processed trade after grouping by position.
66  *
67  * This is the final structure used throughout the application to display trade information.
68  * It includes all original trade data plus calculated fields like PnL and win status.
69  *
70  * Used in:
71  * - ReportComponent: Main component displaying trades
72  * - CalendarComponent: Calendar view of trades
73  * - PnlGraphComponent: PnL chart visualization
74  * - WinLossChartComponent: Win/loss ratio visualization
75  *
76  * @interface GroupedTradeFinal
77  */
78  export interface GroupedTradeFinal {
79    id: string; // id
80    tradableInstrumentId: string; // tradableInstrumentId
81    routeId: string; // routeId
82    qty: string; // qty
83    side: string; // side
84    type: string; // type
85    status: string; // status
86    filledQty: string; // filledQty
87    avgPrice: string; // avgPrice
88    price: string; // price
89    stopPrice: string; // stopPrice
90    validity: string; // validity
91    expireDate: string; // expireDate
92    createdDate: string; // createdDate
93    lastModified: string; // lastModified
94    isOpen: boolean; // isOpen
95    positionId: string; // positionId
96    stopLoss: string; // stopLoss
97    stopLossType: string; // stopLossType
98    takeProfit: string; // takeProfit
99    takeProfitType: string; // takeProfitType
100   strategyId: string; // strategyId
101   instrument?: string; // instrument
102   pnl?: number; // pnl
103   isWon?: boolean; // isWon
104 }
105
106 /**
107  * Interface representing account balance and margin data from the trading API.
108  *

```

```

109 * Contains comprehensive balance information including available funds, margin requirements,
110 * daily trading statistics, and open position data.
111 *
112 * Used in:
113 * - ReportComponent: Displaying account balance information
114 * - ReportService: Processing balance data from API
115 *
116 * @interface BalanceData
117 */
118 export interface BalanceData {
119   balance: number, // balance
120   projectedBalance: number, // projectedBalance
121   availableFunds: number, // availableFunds
122   blockedBalance: number, // blockedBalance
123   cashBalance: number, // cashBalance
124   unsettledCash: number, // unsettledCash
125   withdrawalAvailable: number, // withdrawalAvailable
126   stocksValue: number, // stocksValue
127   optionValue: number, // optionValue
128   initialMarginReq: number, // initialMarginReq
129   maintMarginReq: number, // maintMarginReq
130   marginWarningLevel: number, // marginWarningLevel
131   blockedForStocks: number, // blockedForStocks
132   stockOrdersReq: number, // stockOrdersReq
133   stopOutLevel: number, // stopOutLevel
134   warningMarginReq: number, // warningMarginReq
135   marginBeforeWarning: number, // marginBeforeWarning
136   todayGross: number, // todayGross - A gross profit for today
137   todayNet: number, // todayNet - A total profit or loss realized from positions today
138   todayFees: number, // todayFees - Fees paid today
139   todayVolume: number, // todayVolume - A total volume traded for today
140   todayTradesCount: number, // todayTradesCount - A number of trades done for today
141   openGrossPnL: number, // openGrossPnL - A profit or loss on all currently opened positions
142   openNetPnL: number, // openNetPnL - A net profit or loss on open positions
143   positionsCount: number, // positionsCount - A number of currently opened positions
144   ordersCount: number // ordersCount - A number of currently placed pending orders
145 }
146 /**
147 * Interface representing detailed information about a trading instrument.
148 *
149 * Contains instrument metadata including currency, lot size, trading hours,
150 * leverage, and market information.
151 *
152 * Used in:
153 * - ReportService: Fetching instrument details for trade processing
154 * - CalendarComponent: Displaying instrument names in calendar view
155 *
156 * @interface InstrumentDetails
157 */
158 export interface InstrumentDetails {
159   barSource: string; // "BID"
160   baseCurrency: string; // "XMR"
161   betSize: number | null; // null
162   betStep: number | null; // null
163   bettingCurrency: string | null; // null
164   contractMonth: string | null; // null
165   country: string | null; // null
166   deliveryStatus: string | null; // null
167   description: string; // ""
168   exerciseStyle: string | null; // null
169   firstTradeDate: string | null; // null
170   hasDaily: boolean; // true
171   hasIntraday: boolean; // true
172   industry: string | null; // null
173   isin: string; // ""
174   lastTradeDate: string | null; // null
175   leverage: string; // "2.00"
176   localizedName: string; // "XMRUSD"
177   logoUrl: string | null; // null
178 }
```

```

179   lotSize: number; // 10
180   lotStep: number; // 0.01
181   marginHedgingType: string; // "none"
182   marketCap: number | null; // null
183   marketDataExchange: string; // "Cryptos"
184   maxLot: number | null; // null
185   minLot: number; // 0.01
186   name: string; // "XMRUSD"
187   noticeDate: string | null; // null
188   quotingCurrency: string; // "USD"
189   sector: string | null; // null
190   settlementDate: string | null; // null
191   settlementSystem: string; // "Immediate"
192   strikePrice: string | null; // null
193   strikeType: string | null; // null
194   symbolStatus: string; // "FULLY_OPEN"
195   tickCost: Array<{
196     leftRangeLimit: number | null;
197     tickCost: number;
198   }>; // Array(1) [{leftRangeLimit: null, tickCost: 0}]
199   tickSize: Array<{
200     leftRangeLimit: number | null;
201     tickSize: number;
202   }>; // Array(1) [{leftRangeLimit: null, tickSize: 0.01}]
203   tradeSessionId: number; // 1547
204   tradeSessionStatusId: number; // 20
205   tradingExchange: string; // "Crypto"
206   type: string; // "CRYPTO"
207 }
208 /**
209 * Interface representing a trading instrument with basic information.
210 *
211 * Contains essential instrument data including ID, name, routes, and market information.
212 * This is a simplified version compared to InstrumentDetails.
213 *
214 * Used in:
215 * - ReportService: Fetching available instruments
216 *
217 * @interface Instrument
218 */
219
220 export interface Instrument {
221   barSource: string;
222   continuous: boolean;
223   contractMonth: string;
224   country: number;
225   description: string;
226   hasDaily: boolean;
227   hasIntraday: boolean;
228   id: number;
229   localizedName: string;
230   routes: Array<{
231     id: string;
232     type: string;
233   }>;
234   logoUrl: string;
235   marketDataExchange: string;
236   name: string;
237   strikePrice: number;
238   strikeType: string;
239   tradableInstrumentId: number;
240   tradingExchange: string;
241   type: string;
242   underlierId: number;
243 }
244 /**
245 * Interface representing trading statistics configuration.
246 *
247 * Contains calculated trading metrics including PnL, win rate, profit factor,

```

```

249 * and trade counts. Used to display statistical cards in the report view.
250 *
251 * Used in:
252 * - ReportComponent: Displaying trading statistics
253 * - statCardComponent: Individual statistic card display
254 *
255 * @interface StatConfig
256 */
257 export interface StatConfig {
258   netPnl: number;
259   tradeWinPercent: number;
260   profitFactor: number;
261   avgWinLossTrades: number;
262   totalTrades: number;
263   activePositions: number;
264 }
265 /**
266 * Interface representing display configuration for trading rules.
267 *
268 * Contains information about which trading rules are active and should be displayed
269 * in the report interface.
270 *
271 * Used in:
272 * - ReportComponent: Managing rule display configuration
273 *
274 * @interface displayConfigData
275 */
276 export interface displayConfigData {
277   title: string;
278   type: RuleType;
279   isActive: boolean;
280 }
281 /**
282 * Interface representing a day in the trading calendar.
283 *
284 * Contains aggregated trade data for a specific day including total PnL,
285 * trade count, win percentage, and strategy compliance information.
286 *
287 * Used in:
288 * - CalendarComponent: Calendar view of trades
289 * - TradesPopupComponent: Displaying trades for a selected day
290 *
291 * @interface CalendarDay
292 */
293 export interface CalendarDay {
294   date: Date;
295   trades: GroupedTradeFinal[];
296   pnlTotal: number;
297   tradesCount: number;
298   followedStrategy: boolean;
299   tradeWinPercent: number;
300   strategyName?: string | null; // Nombre de la estrategia seguida en este día
301   isCurrentMonth: boolean; // Indica si el día pertenece al mes actual
302 }
303 /**
304 * Interface representing the NgRx store state for the report module.
305 *
306 * Contains all report-related state including grouped trades, statistics,
307 * and user key for API authentication.
308 *
309 * Used in:
310 * - report.reducer.ts: Reducer managing report state
311 * - report.selectors.ts: Selectors for accessing report state
312 * - report.actions.ts: Actions for updating report state
313 *
314 * @interface ReportState
315 */

```

```

319 export interface ReportState {
320   groupedTrades: GroupedTradeFinal[];
321   netPnL: number;
322   tradeWin: number;
323   profitFactor: number;
324   AvgWnL: number;
325   totalTrades: number;
326   userKey: string;
327 }
328 /**
329 * Interface representing a monthly trading report.
330 *
331 * Contains aggregated monthly trading statistics including profit, trades count,
332 * and strategy compliance percentage. Used for storing and displaying monthly summaries.
333 *
334 * Used in:
335 * - ReportService: Updating monthly reports
336 * - MonthlyReportsService: Managing monthly report data
337 *
338 * @interface MonthlyReport
339 */
340
341 export interface MonthlyReport {
342   best_trade: string;
343   netPnL: number;
344   number_trades: number;
345   profit: number;
346   strategy_followed: number;
347   total_spend: number;
348   month: number;
349   year: number;
350   id: string;
351 }
352 /**
353 * Interface representing plugin usage history record.
354 *
355 * Tracks when the trading plugin was active or inactive, including date ranges
356 * and token requirements. Used to determine strategy compliance for trades.
357 *
358 * Used in:
359 * - CalendarComponent: Determining if trades followed strategies
360 * - PluginHistoryService: Managing plugin history data
361 *
362 * @interface PluginHistoryRecord
363 */
364
365 export interface PluginHistoryRecord {
366   isActive: boolean;
367   updatedOn: string;
368   id: string;
369   tokenNeeded?: boolean;
370   dateActive: string[];
371   dateInactive: string[];
372 }
373

```

Ø=ÜÁ features\report\service

Ø=ÜÁ features\report\service\report.service.ts

```

1 import { TradeLocker ApiService } from '../../../../../shared/services/tradelocker-api.service';
2 import { MonthlyReportsService } from '../../../../../shared/services/monthly-reports.service';
3 import { Injectable } from '@angular/core';

```

```

4 import { map, Observable, switchMap } from 'rxjs';
5 import { ApplicationContextService } from '../../../../../shared/context';
6 import {
7   GroupedTrade,
8   GroupedTradeFinal,
9   BalanceData,
10  historyTrade,
11  MonthlyReport,
12  InstrumentDetails,
13  Instrument,
14 } from '../models/report.model';
15 import {
16   arrayToHistoryTrade,
17   groupOrdersByPosition,
18 } from '../utils/normalization-utils';
19
20 /**
21 * Service for managing report data and API interactions.
22 *
23 * This service acts as an intermediary between the ReportComponent and external services,
24 * handling data fetching, transformation, and context updates for trading reports.
25 *
26 * Responsibilities:
27 * - Fetching trading history from TradeLocker API
28 * - Fetching account balance data
29 * - Fetching instrument details
30 * - Updating monthly reports
31 * - Managing loading states in ApplicationContextService
32 *
33 * Relations:
34 * - TradeLocker ApiService: Direct API communication
35 * - MonthlyReportsService: Monthly report data management
36 * - ApplicationContextService: Global state and loading management
37 *
38 * @injectable
39 * @providedIn root
40 */
41 @Injectable({ providedIn: 'root' })
42 export class ReportService {
43   /**
44    * Constructor for ReportService.
45    *
46    * @param tradeLocker ApiService - Service for TradeLocker API interactions
47    * @param monthlyReportsService - Service for monthly report management
48    * @param applicationContext - Application context service for global state
49    */
50   constructor(
51     private tradeLocker ApiService: TradeLocker ApiService,
52     private monthlyReportsService: MonthlyReportsService,
53     private applicationContext: ApplicationContextService
54   ) {}
55
56   /**
57    * Updates a monthly report in the database.
58    *
59    * @param monthlyReport - The monthly report data to update
60    * @returns Promise that resolves when the update is complete
61    * @memberof ReportService
62    */
63   async updateMonthlyReport(monthlyReport: MonthlyReport) {
64     return this.monthlyReportsService.updateMonthlyReport(monthlyReport);
65   }
66
67   /**
68    * Gets user authentication key from TradeLocker API.
69    *
70    * Authenticates user credentials and returns an access token for API requests.
71    *
72    * @param email - Trading account email
73    * @param password - Trading account password

```

```

74     * @param server - Trading server name
75     * @returns Observable that emits the user key (access token)
76     * @memberof ReportService
77     */
78     getUserKey(
79         email: string,
80         password: string,
81         server: string
82     ): Observable<string> {
83         return this.tradeLocker ApiService.getUserKey(email, password, server);
84     }
85
86     /**
87      * Fetches trading history data for an account.
88      *
89      * Retrieves order history from TradeLocker API, transforms it into GroupedTradeFinal
90      objects and updates the application context with the processed data.
91      *
92      * Process:
93      * 1. Sets loading state in ApplicationContextService
94      * 2. Fetches order history from API
95      * 3. Transforms array data to historyTrade objects
96      * 4. Groups orders by position using groupOrdersByPosition
97      * 5. Updates ApplicationContextService with grouped trades
98      * 6. Clears loading state
99      *
100     * Related to:
101     * - arrayToHistoryTrade(): Transforms API array to historyTrade
102     * - groupOrdersByPosition(): Groups trades by position
103     * - ApplicationContextService.updateReportHistory(): Updates global state
104     *
105     * @param accountId - Trading account ID
106     * @param accessToken - User authentication token
107     * @param accNum - Account number
108     * @returns Observable that emits an array of GroupedTradeFinal objects
109     * @memberof ReportService
110     */
111     getHistoryData(
112         accountId: string,
113         accessToken: string,
114         accNum: number
115     ): Observable<GroupedTradeFinal[]> {
116         this.appContext.setLoading('report', true);
117         this.appContext.setError('report', null);
118
119         return this.tradeLocker ApiService.getTradingHistory(accessToken, accountId, accNum)
120             .pipe(
121                 switchMap(async (details) => {
122                     // Verificar si hay datos válidos
123                     if (!details || !details.d || !details.d.ordersHistory) {
124                         this.appContext.updateReportHistory([]);
125                         this.appContext.setLoading('report', false);
126                         return [];
127                     }
128
129                     const historyTrades: historyTrade[] =
130                         details.d.ordersHistory.map(arrayToHistoryTrade);
131
132                     // Pasar accessToken y accNum a la función
133                     const groupedTrades = await groupOrdersByPosition(historyTrades, this,
134                     accessToken, accNum);
135
136                     // Actualizar contexto con los datos del historial
137                     this.appContext.updateReportHistory(groupedTrades);
138                     this.appContext.setLoading('report', false);
139
140                     return groupedTrades;
141                 })
142             );
143     }

```

```

144     /**
145      * Fetches account balance data from TradeLocker API.
146      *
147      * Retrieves comprehensive balance information including available funds, margin
148      * requirements, trading statistics, and open position data.
149      *
150      * Process:
151      * 1. Sets loading state in ApplicationContextService
152      * 2. Fetches balance data from API
153      * 3. Maps array data to BalanceData interface
154      * 4. Updates ApplicationContextService with balance data
155      * 5. Clears loading state
156      *
157      * Related to:
158      * - ApplicationContextService.updateReportBalance(): Updates global balance state
159      *
160      * @param accountId - Trading account ID
161      * @param accessToken - User authentication token
162      * @param accNum - Account number
163      * @returns Observable that emits BalanceData object
164      * @memberof ReportService
165      */
166      getBalanceData(
167        accountId: string,
168        accessToken: string,
169        accNum: number
170      ): Observable<any> {
171        this.appContext.setLoading('report', true);
172        this.appContext.setError('report', null);
173
174        return this.tradeLocker ApiService.getAccountBalance(accountId, accessToken, accNum)
175          .pipe(
176            map((details) => {
177              // Verificar si hay datos válidos
178              if (!details || !details.d || !details.d.accountDetailsData) {
179                const emptyBalanceData: BalanceData = {
180                  balance: 0,
181                  projectedBalance: 0,
182                  availableFunds: 0,
183                  blockedBalance: 0,
184                  cashBalance: 0,
185                  unsettledCash: 0,
186                  withdrawalAvailable: 0,
187                  stocksValue: 0,
188                  optionValue: 0,
189                  initialMarginReq: 0,
190                  maintMarginReq: 0,
191                  marginWarningLevel: 0,
192                  blockedForStocks: 0,
193                  stockOrdersReq: 0,
194                  stopOutLevel: 0,
195                  warningMarginReq: 0,
196                  marginBeforeWarning: 0,
197                  todayGross: 0,
198                  todayNet: 0,
199                  todayFees: 0,
200                  todayVolume: 0,
201                  todayTradesCount: 0,
202                  openGrossPnL: 0,
203                  openNetPnL: 0,
204                  positionsCount: 0,
205                  ordersCount: 0
206                };
207
208                this.appContext.updateReportBalance(emptyBalanceData);
209                this.appContext.setLoading('report', false);
210                return emptyBalanceData;
211              }
212
213              // Extract all balance data for calculations

```

```

214     const accountData = details.d;
215
216     // Mapear el array accountDetailsData a las propiedades específicas
217     const accountDetailsData = accountData.accountDetailsData;
218     const balanceData: BalanceData = {
219         balance: accountDetailsData[0] || 0,
220         projectedBalance: accountDetailsData[1] || 0,
221         availableFunds: accountDetailsData[2] || 0,
222         blockedBalance: accountDetailsData[3] || 0,
223         cashBalance: accountDetailsData[4] || 0,
224         unsettledCash: accountDetailsData[5] || 0,
225         withdrawalAvailable: accountDetailsData[6] || 0,
226         stocksValue: accountDetailsData[7] || 0,
227         optionValue: accountDetailsData[8] || 0,
228         initialMarginReq: accountDetailsData[9] || 0,
229         maintMarginReq: accountDetailsData[10] || 0,
230         marginWarningLevel: accountDetailsData[11] || 0,
231         blockedForStocks: accountDetailsData[12] || 0,
232         stockOrdersReq: accountDetailsData[13] || 0,
233         stopOutLevel: accountDetailsData[14] || 0,
234         warningMarginReq: accountDetailsData[15] || 0,
235         marginBeforeWarning: accountDetailsData[16] || 0,
236         todayGross: accountDetailsData[17] || 0,
237         todayNet: accountDetailsData[18] || 0,
238         todayFees: accountDetailsData[19] || 0,
239         todayVolume: accountDetailsData[20] || 0,
240         todayTradesCount: accountDetailsData[21] || 0,
241         openGrossPnL: accountDetailsData[22] || 0,
242         openNetPnL: accountDetailsData[23] || 0,
243         positionsCount: accountDetailsData[24] || 0,
244         ordersCount: accountDetailsData[25] || 0
245     };
246
247     // Actualizar contexto con los datos de balance
248     this.appContext.updateReportBalance(balanceData);
249     this.appContext.setLoading('report', false);
250
251     return balanceData;
252 }
253 );
254 }
255
256 /**
257 * Fetches detailed information about a trading instrument.
258 *
259 * Retrieves instrument metadata including lot size, name, currency, and trading
260 * specifications. Mainly for calculating accurate PnL and displaying instrument names.
261 *
262 * @param accessToken - User authentication token
263 * @param tradableInstrumentId - Unique identifier for the instrument
264 * @param routeId - Route ID for the instrument
265 * @param accNum - Account number
266 * @returns Observable that emits InstrumentDetails object
267 * @memberof ReportService
268 */
269 getInstrumentDetails(
270     accessToken: string,
271     tradableInstrumentId: string,
272     routeId: string,
273     accNum: number
274 ): Observable<InstrumentDetails> {
275     return this.tradeLocker ApiService.getInstrumentDetails(accessToken,
276     tradableInstrumentId, routeId, accNum)
277     map((details) => {
278         // Extract all instrument data for calculations
279         const instrumentData = details.d;
280         const instrumentDetailsData: InstrumentDetails = instrumentData;
281
282         return instrumentDetailsData;
283     })

```

```

284     );
285 }
286
287 /**
288  * Fetches all available trading instruments for an account.
289  *
290  * Retrieves a list of all instruments that can be traded on the account,
291  * including basic information like ID, name, and routes.
292  *
293  * @param accessToken - User authentication token
294  * @param accNum - Account number
295  * @param accountId - Trading account ID
296  * @returns Observable that emits an array of Instrument objects
297  * @memberof ReportService
298  */
299 getAllInstruments(
300   accessToken: string,
301   accNum: number,
302   accountId: string
303 ): Observable<Instrument[]> {
304   return this.tradeLocker ApiService.getAllInstruments(accessToken, accountId, accNum)
305     .pipe(
306       map((details) => {
307         return details.d.instruments;
308       })
309     )
310     .pipe(
311       map((instruments) => {
312         return instruments.map((instrument: Instrument) => {
313           return instrument;
314         });
315       })
316     );
317 }
318 }
319

```

Ø=ÜÁ features\report\store

Ø=ÜÁ features\report\store\report.actions.ts

```

1 import { createAction, props } from '@ngrx/store';
2 import { GroupedTrade, GroupedTradeFinal } from '../models/report.model';
3
4 /**
5  * Action to trigger fetching report history trades.
6  *
7  * @action getReportHistory
8  */
9 export const getReportHistory = createAction(
10   '[Report] get report history trades',
11   props<{ userId: string }>()
12 );
13
14 /**
15  * Action to get user authentication key.
16  *
17  * @action getUserKey
18  */
19 export const getUserKey = createAction(
20   '[Report] Get User Key',
21   props<{ email: string; password: string; server: string }>()
22 );

```

```

23
24  /**
25   * Action to set user authentication key in store.
26   *
27   * @action setUserKey
28   */
29  export const setUserKey = createAction(
30    '[Report] Set User Key',
31    props<{ userKey: string }>()
32  );
33
34  /**
35   * Action to set grouped trades in store.
36   *
37   * @action setGroupedTrades
38   */
39  export const setGroupedTrades = createAction(
40    '[Report] Set Grouped Trades',
41    props<{ groupedTrades: GroupedTradeFinal[] }>()
42  );
43
44  /**
45   * Action to set net PnL value in store.
46   *
47   * @action setNetPnL
48   */
49  export const setNetPnL = createAction(
50    '[Report] Set Net PnL',
51    props<{ netPnL: number }>()
52  );
53
54  /**
55   * Action to set trade win percentage in store.
56   *
57   * @action setTradeWin
58   */
59  export const setTradeWin = createAction(
60    '[Report] Set Trade Win',
61    props<{ tradeWin: number }>()
62  );
63
64  /**
65   * Action to set profit factor value in store.
66   *
67   * @action setProfitFactor
68   */
69  export const setProfitFactor = createAction(
70    '[Report] Set Profit Factor',
71    props<{ profitFactor: number }>()
72  );
73
74  /**
75   * Action to set average win/loss ratio in store.
76   *
77   * @action setAvgWnL
78   */
79  export const setAvgWnL = createAction(
80    '[Report] Set Avg Win/Loss',
81    props<{ avgWnL: number }>()
82  );
83
84  /**
85   * Action to set total trades count in store.
86   *
87   * @action setTotalTrades
88   */
89  export const setTotalTrades = createAction(
90    '[Report] Set Total Trades',
91    props<{ totalTrades: number }>()
92  );

```

Ø=ÜÄ features\report\store\report.reducer.ts

```

1 import { createReducer, on } from '@ngrx/store';
2
3 import { ReportState } from '../models/report.model';
4 import {
5   getUserKey,
6   setAvgWnL,
7   setGroupedTrades,
8   setNetPnL,
9   setProfitFactor,
10  setTotalTrades,
11  setTradeWin,
12  setUserKey,
13 } from './report.actions';
14
15 /**
16  * Initial state for the report feature.
17  *
18  * @constant initialReportState
19  */
20 export const initialReportState: ReportState = {
21   groupedTrades: [],
22   netPnL: 0,
23   tradeWin: 0,
24   profitFactor: 0,
25   AvgWnL: 0,
26   totalTrades: 0,
27   userKey: '',
28 };
29
30 /**
31  * Reducer for managing report state.
32  *
33  * Handles all report-related actions and updates the state accordingly.
34  * Actions handled:
35  * - setUserKey: Updates user authentication key
36  * - setGroupedTrades: Updates grouped trades array
37  * - setNetPnL: Updates net PnL value
38  * - setTradeWin: Updates trade win percentage
39  * - setProfitFactor: Updates profit factor
40  * - setAvgWnL: Updates average win/loss ratio
41  * - setTotalTrades: Updates total trades count
42  *
43  * @reducer reportReducer
44  */
45 export const reportReducer = createReducer(
46   initialReportState,
47   on(setUserKey, (state, { userKey }) => ({
48     ...state,
49     userKey,
50   })),
51   on(setGroupedTrades, (state, { groupedTrades }) => ({
52     ...state,
53     groupedTrades,
54   })),
55   on(setNetPnL, (state, { netPnL }) => ({
56     ...state,
57     netPnL,
58   })),
59   on(setTradeWin, (state, { tradeWin }) => ({
60     ...state,
61     tradeWin,
62   })),

```

```

63     on(setProfitFactor, (state, { profitFactor }) => ({
64       ...state,
65       profitFactor,
66     })),
67     on(setAvgWnL, (state, { avgWnL }) => ({
68       ...state,
69       AvgWnL: avgWnL,
70     })),
71     on(setTotalTrades, (state, { totalTrades }) => ({
72       ...state,
73       totalTrades,
74     }))
75   );
76

```

Ø=ÜÄ features\report\store\report.selectors.ts

```

1  import { createFeatureSelector, createSelector } from '@ngrx/store';
2  import { ReportState } from '../models/report.model';
3
4  /**
5   * Feature selector for the report state.
6   *
7   * @selector selectReport
8   */
9  export const selectReport = createFeatureSelector<ReportState>('report');
10
11 /**
12  * Selector for grouped trades from report state.
13  *
14  * @selector selectGroupedTrades
15  */
16 export const selectGroupedTrades = createSelector(
17   selectReport,
18   (state: ReportState) => state.groupedTrades
19 );
20
21 /**
22  * Selector for user key from report state.
23  *
24  * @selector selectUserKey
25  */
26 export const selectUserKey = createSelector(
27   selectReport,
28   (state: ReportState) => state.userKey
29 );
30
31 /**
32  * Selector for net PnL from report state.
33  *
34  * @selector selectNetPnL
35  */
36 export const selectNetPnL = createSelector(
37   selectReport,
38   (state: ReportState) => state.netPnL
39 );
40
41 /**
42  * Selector for trade win percentage from report state.
43  *
44  * @selector selectTradeWin
45  */
46 export const selectTradeWin = createSelector(
47   selectReport,
48   (state: ReportState) => state.tradeWin
49 );

```

```

50
51 /**
52  * Selector for profit factor from report state.
53 *
54 * @selector selectProfitFactor
55 */
56 export const selectProfitFactor = createSelector(
57   selectReport,
58   (state: ReportState) => state.profitFactor
59 );
60
61 /**
62  * Selector for average win/loss ratio from report state.
63 *
64 * @selector selectAvgWnL
65 */
66 export const selectAvgWnL = createSelector(
67   selectReport,
68   (state: ReportState) => state.AvgWnL
69 );
70
71 /**
72  * Selector for total trades count from report state.
73 *
74 * @selector selectTotalTrades
75 */
76 export const selectTotalTrades = createSelector(
77   selectReport,
78   (state: ReportState) => state.totalTrades
79 );
80

```

Ø=ÜÁ features\report\utils

Ø=ÜÁ features\report\utils\firebase-data-utils.ts

```

1 import { GroupedTrade, GroupedTradeFinal } from '../models/report.model';
2
3 /**
4  * Gets the best trade (highest PnL) from an array of trades.
5  *
6  * Iterates through all trades and finds the one with the maximum PnL value.
7  * Returns null if there are no trades or if all trades have undefined PnL.
8  *
9  * @param groupedTrades - Array of GroupedTradeFinal objects
10 * @returns The highest PnL value rounded to nearest integer, or null if no valid trades
11 */
12 export const getBestTrade = (groupedTrades: GroupedTradeFinal[]): number | null => {
13   if (!groupedTrades || groupedTrades.length === 0) {
14     return null;
15   }
16
17   const tradeWithMaxPnl = groupedTrades.reduce((maxTrade, currentTrade) => {
18     if (currentTrade.pnl === undefined) return maxTrade;
19     if (maxTrade.pnl === undefined || currentTrade.pnl > maxTrade.pnl) {
20       return currentTrade;
21     }
22     return maxTrade;
23   }, groupedTrades[0]);
24
25   return Math.round(tradeWithMaxPnl.pnl ?? 0);
26 };
27

```

```

28 /**
29 * Calculates the total amount spent on trades.
30 *
31 * Multiplies price by quantity for each trade and sums the results.
32 * Used for calculating total investment or capital used in trading.
33 *
34 * @param groupedTrades - Array of GroupedTradeFinal objects
35 * @returns Total spend amount rounded down to nearest integer
36 */
37 export const getTotalSpend = (groupedTrades: GroupedTradeFinal[]): number | null => {
38   const totalSpend = groupedTrades.reduce((total, trade) => {
39     const price = Number(trade.price) || 0;
40     const qty = Number(trade.qty) || 0;
41     return total + (price * qty);
42   }, 0);
43   return Math.floor(totalSpend);
44 };
45
46 /**
47 * Generates a unique ID for monthly report data.
48 *
49 * Creates an ID by combining a base ID with formatted month and year.
50 * Format: "{baseId}-{MM}-{YYYY}"
51 *
52 * @param baseId - Base identifier (typically user ID or account ID)
53 * @param month - Month number (1-12)
54 * @param year - Year number (e.g., 2024)
55 * @returns Formatted ID string
56 */
57 export function newDataId(baseId: string, month: number, year: number): string {
58   const monthFormatted = month.toString().padStart(2, '0');
59   return `${baseId}-${monthFormatted}-${year}`;
60 }
61

```

Ø=ÜÄ features\report\utils\normalization-utils.ts

```

1 import { GroupedTrade, GroupedTradeFinal, historyTrade } from '../models/report.model';
2
3 /**
4 * Converts an array from the TradeLocker API into a historyTrade object.
5 *
6 * The API returns trade data as arrays where each index corresponds to a specific field.
7 * This function maps the array indices to named properties for easier access.
8 *
9 * Array mapping:
10 * - [0]: id
11 * - [1]: tradableInstrumentId
12 * - [2]: routeId
13 * - [3]: qty
14 * - [4]: side
15 * - [5]: type
16 * - [6]: status
17 * - [7]: filledQty
18 * - [8]: avgPrice
19 * - [9]: price
20 * - [10]: stopPrice
21 * - [11]: validity
22 * - [12]: expireDate
23 * - [13]: createdDate
24 * - [14]: lastModified
25 * - [15]: isOpen
26 * - [16]: positionId
27 * - [17]: stopLoss
28 * - [18]: stopLossType
29 * - [19]: takeProfit

```

```

30 * - [20]: takeProfitType
31 * - [21]: strategyId
32 *
33 * @param arr - Array from TradeLocker API containing trade data
34 * @returns historyTrade object with named properties
35 */
36 export function arrayToHistoryTrade(arr: any[]): historyTrade {
37     // Mapeo según la configuración de la API:
38     // 0: id, 1: tradableInstrumentId, 2: routeId, 3: qty, 4: side, 5: type, 6: status, 7:
39     // filledQty, 8: avgPrice, 9: price, 10: stopPrice, 11: validity, 12: expireDate, 13:
40     // createdDate, 14: lastModified, 15: isOpen, 16: positionId, 17: stopLoss, 18: stopLossType,
41     // 19: takeProfit, 20: takeProfitType, 21: strategyId
42     return {
43         id: arr[0], // id
44         tradableInstrumentId: arr[1], // tradableInstrumentId
45         routeId: arr[2], // routeId
46         qty: arr[3], // qty
47         side: arr[4], // side
48         type: arr[5], // type
49         status: arr[6], // status
50         filledQty: arr[7], // filledQty
51         avgPrice: arr[8], // avgPrice
52         price: arr[9], // price
53         stopPrice: arr[10], // stopPrice
54         validity: arr[11], // validity
55         expireDate: arr[12], // expireDate
56         createdDate: arr[13], // createdDate
57         lastModified: arr[14], // lastModified
58         isOpen: arr[15], // isOpen
59         positionId: arr[16], // positionId
60         stopLoss: arr[17], // stopLoss
61         stopLossType: arr[18], // stopLossType
62         takeProfit: arr[19], // takeProfit
63         takeProfitType: arr[20], // takeProfitType
64         strategyId: arr[21], // strategyId
65     };
66 }
67 /**
68 * Groups trading orders by position ID and calculates PnL for each position.
69 *
70 * This function processes raw order history and groups orders that belong to the same
71 * position (identified by positionId). It then calculates the PnL for each position by matching
72 * opening and closing orders, and fetches instrument details to ensure accurate
73 * calculations.
74 * Process:
75 * 1. Filters valid orders (status 'Filled' and valid positionId)
76 * 2. Groups orders by positionId
77 * 3. Fetches instrument details for all unique instruments
78 * 4. For each position:
79 *     - Finds opening order (isOpen: 'true')
80 *     - Finds closing order (isOpen: 'false')
81 *     - Calculates PnL based on entry/exit prices and lot size
82 *     - Determines if trade was won or lost
83 * 5. Returns array of GroupedTradeFinal objects
84 *
85 * Related to:
86 * - fetchInstrumentDetails(): Fetches lot size and instrument names
87 * - ReportService.getInstrumentDetails(): API call for instrument data
88 *
89 * @param orders - Array of historyTrade objects to group
90 * @param reportService - ReportService instance for fetching instrument details
91 * @param accessToken - User authentication token
92 * @param accNum - Account number
93 * @returns Promise that resolves to an array of GroupedTradeFinal objects
94 */
95 export async function groupOrdersByPosition(orders: historyTrade[], reportService: any,
96 accessToken: string, trades: number): Promise<GroupedTradeFinal[]> {
97     const insensitiveValidOrders = orders.filter(order => {
98         const hasValidStatus = order.status && (
99             order.status.toLowerCase() === 'filled' ||

```

```

100     order.status === 'filled'
101   );
102   const hasValidPositionId = order.positionId &&
103     order.positionId !== 'null' &&
104     order.positionId !== '' &&
105     order.positionId !== null &&
106     order.positionId !== undefined;
107
108   return hasValidStatus && hasValidPositionId;
109 });
110
111 // Agrupar órdenes por positionId
112 const ordersByPosition: { [positionId: string]: historyTrade[] } = {};
113
114 validOrders.forEach(order => {
115   const positionId = order.positionId;
116   if (!ordersByPosition[positionId]) {
117     ordersByPosition[positionId] = [];
118   }
119   ordersByPosition[positionId].push(order);
120 });
121
122 // Obtener lotSize de todos los instrumentos únicos ANTES de procesar trades
123 const instrumentDetailsMap = await fetchInstrumentDetails(validOrders, reportService,
124 accessToken, accNum);
125 // Procesar cada posición para crear un trade final
126 const finalTrades: GroupedTradeFinal[] = [];
127 let totalWins = 0;
128 let totalLosses = 0;
129
130 Object.entries(ordersByPosition).forEach(([positionId, positionOrders]) => {
131   // Ordenar por fecha de creación para asegurar orden correcto
132   positionOrders.sort((a, b) => new Date(a.createdAt).getTime() - new
133 Date(b.createdAt).getTime());
134   // Buscar la orden de apertura (isOpen: true) y la de cierre (isOpen: false)
135   const openOrder = positionOrders.find(order => order.isOpen === 'true');
136   const closeOrder = positionOrders.find(order => order.isOpen === 'false');
137
138   if (openOrder && closeOrder) {
139     // Calcular P&L
140     const tradeKey = `${openOrder.tradeableInstrumentId}-${openOrder.routeId}`;
141     const instrumentDetails = instrumentDetailsMap.get(tradeKey);
142     const lotSize = instrumentDetails?.lotSize || 1;
143
144     const entryPrice = Number(openOrder.price);
145     const exitPrice = Number(closeOrder.price);
146     const quantity = Number(openOrder.qty);
147
148     // Calcular P&L según el tipo de trade
149     let pnl: number;
150     let isWin = false;
151
152     if (openOrder.side === 'buy' && closeOrder.side === 'sell') {
153       // BUY !' SELL: Ganas si vendes más caro de lo que compraste
154       pnl = (exitPrice - entryPrice) * (quantity * lotSize);
155       isWin = exitPrice > entryPrice;
156     } else if (openOrder.side === 'sell' && closeOrder.side === 'buy') {
157       // SELL !' BUY: Ganas si compras más barato de lo que vendiste
158       pnl = (entryPrice - exitPrice) * (quantity * lotSize);
159       isWin = entryPrice > exitPrice;
160     } else {
161       // Fallback (no debería pasar)
162       pnl = (exitPrice - entryPrice) * (quantity * lotSize);
163       isWin = pnl > 0;
164     }
165
166     // Crear el trade final usando la orden de apertura como base
167     const finalTrade: GroupedTradeFinal = {
168       ...openOrder,
169       instrument: instrumentDetails?.name || openOrder.tradeableInstrumentId,

```

```

170     pnl: pnl,
171     isWon: isWin,
172     isOpen: false, // Ya está cerrado
173     lastModified: closeOrder.lastModified, // Usar la fecha de cierre
174   };
175
176   // Validar que el trade final tenga positionId válido
177   if (finalTrade.positionId && finalTrade.positionId !== 'null' &&
178     finalTrade.positionId.push(finalTrade));
179 }
180
181   // Contar wins y losses
182   if (isWin) {
183     totalWins++;
184   } else {
185     totalLosses++;
186   }
187 } else {
188   // Si no se encuentran ambos trades, agregar la orden de apertura si existe
189   if (openOrder) {
190     const tradeKey = `${openOrder.tradeableInstrumentId}-${openOrder.routeId}`;
191     const instrumentDetails = instrumentDetailsMap.get(tradeKey);
192
193     const finalTrade: GroupedTradeFinal = {
194       ...openOrder,
195       instrument: instrumentDetails?.name || openOrder.tradeableInstrumentId,
196       pnl: 0,
197       isWon: false,
198       isOpen: true,
199     };
200
201     // Validar que el trade final tenga positionId válido
202     if (finalTrade.positionId && finalTrade.positionId !== 'null' &&
203       finalTrade.positionId.push(finalTrade));
204   }
205 }
206 }
207 });
208
209 return finalTrades;
210 }
211
212 /**
213 * Fetches instrument details for all unique instruments in the orders.
214 *
215 * This helper function extracts unique instrument combinations (tradeableInstrumentId +
216 * routeId) orders and fetches their details (lot size and name) from the API.
217 * It processes instruments sequentially with a small delay to avoid rate limiting.
218 *
219 * @param orders - Array of historyTrade objects
220 * @param reportService - ReportService instance for API calls
221 * @param accessToken - User authentication token
222 * @param accNum - Account number
223 * @returns Promise that resolves to a Map with instrument keys and their details
224 * @private
225 */
226 async function fetchInstrumentDetails(
227   orders: historyTrade[],
228   reportService: any,
229   accessToken: string,
230   accNum: number
231 ): Promise<Map<string, { lotSize: number, name: string }>> {
232   // Extraer tradeableInstrumentId y routeId únicos
233   const uniqueInstruments = new Map<string, { tradeableInstrumentId: string, routeId: string }>();
234
235   orders.forEach(order => {
236     if (order.tradeableInstrumentId && order.routeId) {
237       const key = `${order.tradeableInstrumentId}-${order.routeId}`;
238       if (!uniqueInstruments.has(key)) {
239         uniqueInstruments.set(key, {

```

```

240         tradableInstrumentId: order.tradableInstrumentId,
241         routeId: order.routeId
242     });
243 }
244 );
245 );
246
247 // Hacer consultas a la API para obtener lotSize y name de cada instrumento
248 const instrumentDetailsMap = new Map<string, { lotSize: number, name: string }>(); // key:
249 tradableInstrumentId-routeId, value: {lotSize, name}
250 // Procesar cada instrumento único de forma secuencial
251 for (const [key, instrument] of uniqueInstruments) {
252     try {
253         // Hacer petición individual para cada instrumento
254         const instrumentDetails = await reportService.getInstrumentDetails(
255             accessToken,
256             instrument.tradableInstrumentId,
257             instrument.routeId,
258             accNum
259         ).toPromise();
260
261         const lotSize = instrumentDetails.lotSize;
262         const name = instrumentDetails.name;
263
264         instrumentDetailsMap.set(key, { lotSize, name });
265
266         // Pequeña pausa entre peticiones para evitar rate limiting
267         await new Promise(resolve => setTimeout(resolve, 100));
268
269     } catch (error) {
270         console.error(`'L Error querying instrument ${key}:`, error);
271         instrumentDetailsMap.set(key, { lotSize: 1, name:
272             instrument.tradableInstrumentId }); // Default values si falla la consulta
273     }
274
275     return instrumentDetailsMap;
276 }
277
278
279 /**
280 * Calculates the total net PnL from an array of trades.
281 *
282 * Sums all PnL values from trades and rounds to the nearest integer.
283 *
284 * @param trades - Array of trade objects with optional pnl property
285 * @returns Total net PnL rounded to nearest integer
286 */
287 export function calculateNetPnl(trades: { pnl?: number }[]): number {
288     const total = trades.reduce((sum, t) => sum + (t.pnl ?? 0), 0);
289     return Math.round(total);
290 }
291
292 /**
293 * Calculates the percentage of winning trades.
294 *
295 * Counts trades with positive PnL and calculates the percentage
296 * relative to total trades. Returns 0 if there are no trades.
297 *
298 * @param trades - Array of trade objects with optional pnl property
299 * @returns Win percentage rounded to nearest integer (0-100)
300 */
301 export function calculateTradeWinPercent(trades: { pnl?: number }[]): number {
302     const wins = trades.filter(t => t.pnl !== undefined && t.pnl > 0).length;
303     return trades.length > 0 ? Math.round((wins / trades.length) * 100) : 0;
304 }
305
306 /**
307 * Calculates the profit factor from an array of trades.
308 *
309 * Profit factor is calculated as: gross profit / gross loss

```

```

310 * - If there are no losses, returns 999.99 (infinite profit factor)
311 * - If there are no profits, returns 0
312 * - Otherwise, returns the ratio rounded to 2 decimal places
313 *
314 * @param trades - Array of trade objects with optional pnl property
315 * @returns Profit factor rounded to 2 decimal places
316 */
317 export function calculateProfitFactor(trades: { pnl?: number }[]): number {
318   const grossProfit = trades
319     .filter((t) => (t.pnl ?? 0) > 0)
320     .reduce((sum, t) => sum + t.pnl!, 0);
321   const grossLoss = trades
322     .filter((t) => (t.pnl ?? 0) < 0)
323     .reduce((sum, t) => sum + Math.abs(t.pnl!), 0);
324
325   // If there are no losses, profit factor should be calculated differently
326   if (grossLoss === 0) {
327     // If there are profits and no losses, return a high but finite number
328     return grossProfit > 0 ? 999.99 : 0;
329   }
330
331   const profitFactor = grossProfit / grossLoss;
332   return Math.round(profitFactor * 100) / 100;
333 }
334
335 /**
336 * Calculates the average win/loss ratio.
337 *
338 * Calculates the ratio of average winning trade to average losing trade.
339 * - If there are no losses, returns 999.99 (infinite ratio)
340 * - If there are no wins, returns 0
341 * - Otherwise, returns the ratio rounded to 2 decimal places
342 *
343 * @param trades - Array of trade objects with optional pnl property
344 * @returns Average win/loss ratio rounded to 2 decimal places
345 */
346 export function calculateAvgWinLossTrades(trades: { pnl?: number }[]): number {
347   const wins = trades.filter((t) => t.pnl !== undefined && t.pnl > 0);
348   const losses = trades.filter((t) => t.pnl !== undefined && t.pnl < 0);
349
350   // If there are no losses, we can't calculate a ratio
351   if (losses.length === 0) {
352     // If there are wins but no losses, return a high ratio
353     return wins.length > 0 ? 999.99 : 0;
354   }
355
356   const grossProfit = wins.reduce((sum, t) => sum + t.pnl!, 0);
357   const grossLoss = losses.reduce((sum, t) => sum + Math.abs(t.pnl!), 0);
358
359   const avgWin = wins.length > 0 ? grossProfit / wins.length : 0;
360   const avgLoss = losses.length > 0 ? grossLoss / losses.length : 0;
361
362   return avgLoss > 0 ? Math.round((avgWin / avgLoss) * 100) / 100 : 0;
363 }
364
365 /**
366 * Calculates the total number of trades.
367 *
368 * Simply returns the length of the trades array.
369 *
370 * @param trades - Array of trade objects
371 * @returns Total number of trades
372 */
373 export function calculateTotalTrades(trades: any[]): number {
374   return trades.length;
375 }
376
377 /**
378 * Groups trades by month and calculates total PnL per month.
379 *

```

```

380 * Processes trades and aggregates PnL values by month (format: "YYYY-MM").
381 * Used for generating monthly PnL charts and reports.
382 *
383 * @param trades - Array of GroupedTrade objects with updatedAt and pnl properties
384 * @returns Object with month keys (YYYY-MM) and total PnL values
385 */
386 export function getMonthlyPnL(trades: GroupedTrade[]): {
387   [month: string]: number;
388 } {
389   const monthlyPnL: { [month: string]: number } = {};
390
391   trades.forEach((trade) => {
392     const d = new Date(trade.updatedAt);
393     const key = `${d.getFullYear()}-${String(d.getMonth() + 1).padStart(
394       2,
395       '0'
396     )}`;
397     monthlyPnL[key] = (monthlyPnL[key] ?? 0) + (trade.pnl ?? 0);
398   });
399
400   return monthlyPnL;
401 }
402

```

Ø=ÜÁ features\revenue

Ø=ÜÁ features\revenue\revenue.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { FormsModule } from '@angular/forms';
3 import { LoadingPopupComponent } from '../../../../../shared/pop-ups/loading-pop-up/loading-
4 popup-component';
5 import {
6   DailyRevenueData,
7   MonthlyRevenueData,
8   OrderTableRow,
9   RevenueSummary,
10  RevenueTableRow,
11  SubscriptionTableRow,
12 } from './models/revenue';
13 import { Store } from '@ngrx/store';
14 import {
15   dailyRevenueMock,
16   mockRevenueSummary,
17   monthlyRevenueMock,
18   orderTableMock,
19   revenueTableMock,
20   subscriptionTableMock,
21 } from './mocks/revenue_mock';
22 import { statCardComponent } from '../report/components/statCard/stat_card.component';
23 import { RevenueGraphComponent } from './components/revenueGraph/revenue-graph.component';
24 import { RevenueTableComponent } from './components/revenue-table/revenue-table.component';
25 import { OrdersTableComponent } from './components/orders-table/orders-table.component';
26 import { SubscriptionsTableComponent } from './components/subscriptions-table/subscriptions-
27 table-component';
28 import { selectUser } from '../auth/store/user.selectios';
29 import { User } from '../overview/models/overview';
30 import { AccountData } from '../auth/models/userModel';
31
32 /**
33 * Main component for displaying revenue analytics and data.
34 *
35 * This component displays revenue-related information including:

```

```

36 * - Revenue summary statistics (gross revenue, returns, coupons, net revenue)
37 * - Revenue charts (daily and monthly)
38 * - Revenue table with filtering and pagination
39 * - Orders table with filtering and pagination
40 * - Subscriptions table with filtering and pagination
41 *
42 * Currently uses mock data for display. Future implementation will fetch
43 * real data from APIs based on user accounts and access tokens.
44 *
45 * Relations:
46 * - RevenueGraphComponent: Displays revenue charts
47 * - RevenueTableComponent: Displays revenue table
48 * - OrdersTableComponent: Displays orders table
49 * - SubscriptionsTableComponent: Displays subscriptions table
50 * - ReportService: For fetching user keys and historical data (future implementation)
51 * - Store (NgRx): For accessing user data
52 *
53 * @component
54 * @selector app-revenue
55 * @standalone true
56 */
57 @Component({
58   selector: 'app-revenue',
59   imports: [
60     CommonModule,
61     LoadingPopupComponent,
62     FormsModule,
63     statCardComponent,
64     RevenueGraphComponent,
65     RevenueTableComponent,
66     OrdersTableComponent,
67     SubscriptionsTableComponent,
68   ],
69   templateUrl: './revenue.component.html',
70   styleUrls: ['./revenue.component.scss'],
71   standalone: true,
72 })
73 export class RevenueComponent {
74   revenueSummary: RevenueSummary | null = null;
75   revenueDailyData: DailyRevenueData[] | null = null;
76   revenueMonthlyData: MonthlyRevenueData[] | null = null;
77   revenueTableData: RevenueTableRow[] | null = null;
78   loading = false;
79   orderTableData: OrderTableRow[] | null = null;
80   subscriptionsTableData: SubscriptionTableRow[] | null = null;
81
82   // Propiedades para el accessToken
83   accessToken: string | null = null;
84   user: User | null = null;
85   accountsData: AccountData[] = [];
86
87   constructor(private store: Store, private reportService: ReportService) {}
88
89   /**
90    * Initializes the component on load.
91    *
92    * Loads configuration (mock data) and fetches user data from the store.
93    *
94    * @memberof RevenueComponent
95    */
96   ngOnInit(): void {
97     this.loadConfig();
98     this.getUserData();
99   }
100
101  /**
102   * Loads configuration data (currently using mock data).
103   *
104   * Initializes all revenue-related data from mock sources:
105   * - Revenue summary

```

```

106     * - Daily and monthly revenue data
107     * - Revenue table data
108     * - Orders table data
109     * - Subscriptions table data
110     *
111     * NOTE: In production, this should fetch data from APIs.
112     *
113     * @memberof RevenueComponent
114     */
115     loadConfig() {
116         this.revenueSummary = mockRevenueSummary;
117         this.revenueDailyData = dailyRevenueMock;
118         this.revenueMonthlyData = monthlyRevenueMock;
119         this.revenueTableData = revenueTableMock;
120         this.orderTableData = orderTableMock;
121         this.subscriptionsTableData = subscriptionTableMock;
122     }
123
124     /**
125      * Fetches user data from the NgRx store.
126      *
127      * Subscribes to the selectUser selector to get current user information.
128      * If user has trading accounts, attempts to fetch access token for API calls.
129      * Currently falls back to mock data if no accounts are available.
130      *
131      * Related to:
132      * - Store.select(selectUser): Gets user from NgRx store
133      * - fetchUserKey(): Fetches access token for API calls
134      *
135      * @memberof RevenueComponent
136      */
137     getUserData() {
138         // Obtener datos del usuario desde el store
139         this.store.select(selectUser).subscribe((userState) => {
140             if (userState && userState.user) {
141                 this.user = userState.user;
142                 // Por ahora, usar datos mock ya que el modelo User no tiene accountsData
143                 // TODO: Agregar accountsData al modelo User o obtenerlo de otra fuente
144                 this.accountsData = []; // Temporalmente vacío
145
146                 // Si hay cuentas, obtener el accessToken de la primera cuenta
147                 if (this.accountsData.length > 0) {
148                     this.fetchUserKey(this.accountsData[0]);
149                 } else {
150                     console.warn('No hay cuentas de trading disponibles - usando datos mock');
151                     this.orderTableData = orderTableMock;
152                 }
153             }
154         });
155     }
156
157     /**
158      * Fetches user authentication key for API access.
159      *
160      * Uses ReportService to authenticate with trading account credentials
161      * and obtain an access token. On success, triggers fetching historical data.
162      *
163      * Related to:
164      * - ReportService.getUserKey(): Authenticates and gets access token
165      * - getHistoricalData(): Fetches historical data after authentication
166      *
167      * @param account - Trading account data with credentials
168      * @memberof RevenueComponent
169      */
170     fetchUserKey(account: AccountData) {
171         this.reportService
172             .getUserKey(
173                 account.emailTradingAccount,
174                 account.brokerPassword,
175                 account.server

```

```

176     )
177     .subscribe({
178       next: (accessToken: string) => {
179         this.accessToken = accessToken;
180         this.getHistoricalData();
181       },
182       error: (error) => {
183         console.error('Error al obtener el accessToken:', error);
184         this.orderTableData = orderTableMock;
185       }
186     });
187   }
188
189 /**
190 * Fetches historical revenue data from the API.
191 *
192 * Currently a placeholder method. In production, this should:
193 * - Use the access token to authenticate API requests
194 * - Fetch historical order and revenue data
195 * - Update orderTableData with real data
196 *
197 * NOTE: This method is not yet fully implemented.
198 *
199 * @memberof RevenueComponent
200 */
201 getHistoricalData() {
202   if (!this.accessToken || this.accountsData.length === 0) {
203     console.warn('No hay accessToken o cuentas disponibles');
204     this.orderTableData = orderTableMock;
205     return;
206   }
207
208   // Valores de ejemplo para probar el endpoint
209   // TODO: Reemplazar con valores reales
210   const routeId = 1;
211   const from = Date.now() - (30 * 24 * 60 * 60 * 1000); // 30 días atrás
212   const to = Date.now(); // Ahora
213   const resolution = '1D'; // Diario
214   const tradableInstrumentId = 1; // TODO: Usar ID real del instrumento
215   const accNum = this.accountsData[0].accountNumber || 1; // Usar número de cuenta real
216
217
218 }
219 }
220

```

Ø=ÜÁ features\revenue\components\orders-table

Ø=ÜÄ features\revenue\components\orders-table\orders-table.component.ts

```

1 import { Component, Input, Output } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 import { FormsModule } from '@angular/forms';
5 import {
6   OrderFilter,
7   OrderStatus,
8   OrderTableRow,
9   RevenueFilter,
10  RevenueTableRow,
11 } from '../../../../../models/revenue';
12
13 /**

```

```

14 * Component for displaying orders data in a table format.
15 *
16 * This component provides a paginated and filterable table for order data.
17 * It supports filtering by search term, order status, and total amount,
18 * as well as sorting and pagination.
19 *
20 * Features:
21 * - Search by order ID or user name
22 * - Filter by order status (Completed, Pending, Cancelled, Failed)
23 * - Filter by total amount range
24 * - Sort by date (ascending/descending)
25 * - Pagination
26 *
27 * @component
28 * @selector app-orders-table
29 * @standalone true
30 */
31 @Component({
32   selector: 'app-orders-table',
33   standalone: true,
34   imports: [CommonModule, FormsModule],
35   templateUrl: './orders-table.component.html',
36   styleUrls: ['./orders-table.component.scss'],
37 })
38 export class OrdersTableComponent {
39   @Input() orderRows: OrderTableRow[] = [];
40
41   showFilter = false;
42   currentPage: number = 1;
43   itemsPerPage: number = 10;
44   sortField: 'date' = 'date';
45   sortAsc: boolean = true;
46
47   orderStatus = OrderStatus;
48
49   filter: OrderFilter = {};
50
51   private _searchTerm = '';
52   get searchTerm(): string {
53     return this._searchTerm;
54   }
55   set searchTerm(val: string) {
56     this._searchTerm = val;
57     this.goToPage(1);
58   }
59
60   get filteredRows(): OrderTableRow[] {
61     let result = this.filterOrderRows(this.orderRows, this.filter);
62
63     result = result.sort((a, b) => {
64       const fieldA = a[this.sortField].toLowerCase();
65       const fieldB = b[this.sortField].toLowerCase();
66       if (fieldA < fieldB) return this.sortAsc ? -1 : 1;
67       if (fieldA > fieldB) return this.sortAsc ? 1 : -1;
68       return 0;
69     });
70
71     return result;
72   }
73
74   get paginatedRows(): OrderTableRow[] {
75     const start = (this.currentPage - 1) * this.itemsPerPage;
76     const end = start + this.itemsPerPage;
77     return this.filteredRows.slice(start, end);
78   }
79
80   get totalPages(): number {
81     return Math.ceil(this.filteredRows.length / this.itemsPerPage);
82   }
83

```

```

84     statusClass(status: string) {
85         return status;
86     }
87
88     openFilter() {
89         this.showFilter = !this.showFilter;
90     }
91
92     closeFilter() {
93         this.showFilter = false;
94     }
95
96     apply() {
97         this.showFilter = false;
98
99         this.applyFilters();
100    }
101
102    applyFilters() {
103        this.goToPage(1);
104    }
105
106    goToPage(page: number) {
107        if (page < 1) page = 1;
108        if (page > this.totalPages) page = this.totalPages;
109        this.currentPage = page;
110    }
111
112    prevPage() {
113        this.goToPage(this.currentPage - 1);
114    }
115
116    nextPage() {
117        this.goToPage(this.currentPage + 1);
118    }
119
120    toggleSort() {
121        this.sortAsc = !this.sortAsc;
122    }
123
124    filterOrderRows(rows: OrderTableRow[], filter: OrderFilter): OrderTableRow[] {
125        const lowerSearch = filter.searchTerm?.toLowerCase() ?? '';
126
127        return rows.filter((row) => {
128            const matchesSearch =
129                lowerSearch === '' ||
130                `${row.orderId} ${row.user}`.toLowerCase().includes(lowerSearch);
131
132            const matchesStatus =
133                filter.status === undefined ||
134                filter.status === null ||
135                filter.status === ('' as OrderStatus) ||
136                row.status === filter.status;
137
138            const matchesMinTotal =
139                filter.minTotal === undefined ||
140                filter.minTotal === null ||
141                row.total >= filter.minTotal;
142            const matchesMaxTotal =
143                filter.maxTotal === undefined ||
144                filter.maxTotal === null ||
145                row.total <= filter.maxTotal;
146
147            if (this.showFilter) {
148                return matchesSearch;
149            }
150
151            return (
152                matchesSearch && matchesStatus && matchesMinTotal && matchesMaxTotal
153            );

```

```
154     });
155 }
156 }
157
```

Ø=ÜÄ features\revenue\components\revenue-table

Ø=ÜÄ features\revenue\components\revenue-table\revenue-table.component.ts

```
1 import { Component, Input, Output, Injectable } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 import { FormsModule } from '@angular/forms';
5 import { RevenueFilter, RevenueTableRow } from '../../../../../models/revenue';
6 import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';
7
8 /**
9  * Component for displaying revenue data in a table format.
10 *
11 * This component provides a paginated and filterable table for revenue data.
12 * It supports filtering by search term, order count, gross revenue, and total sales,
13 * as well as sorting and pagination.
14 *
15 * Features:
16 * - Search by date
17 * - Filter by orders, gross revenue, and total sales ranges
18 * - Sort by date (ascending/descending)
19 * - Pagination
20 * - Currency formatting
21 *
22 * Relations:
23 * - NumberFormatterService: Currency formatting
24 *
25 * @component
26 * @selector app-revenue-table
27 * @standalone true
28 */
29 @Component({
30   selector: 'app-revenue-table',
31   standalone: true,
32   imports: [CommonModule, FormsModule],
33   templateUrl: './revenue-table.component.html',
34   styleUrls: ['./revenue-table.component.scss'],
35 })
36 @Injectable()
37 export class RevenueTableComponent {
38   @Input() revenueRows: RevenueTableRow[] = [];
39
40   showFilter = false;
41   currentPage: number = 1;
42   itemsPerPage: number = 10;
43   sortField: 'date' = 'date';
44   sortAsc: boolean = true;
45
46   filter: RevenueFilter = {};
47
48   private numberFormatter = new NumberFormatterService();
49
50   private _searchTerm = '';
51   get searchTerm(): string {
52     return this._searchTerm;
53   }
54   set searchTerm(val: string) {
```

```

55     this._searchTerm = val;
56     this.goToPage(1);
57 }
58
59 get filteredRows(): RevenueTableRow[] {
60   let result = this.filterRevenueRows(this.revenueRows, this.filter);
61
62   result = result.sort((a, b) => {
63     const fieldA = a[this.sortField].toLowerCase();
64     const fieldB = b[this.sortField].toLowerCase();
65     if (fieldA < fieldB) return this.sortAsc ? -1 : 1;
66     if (fieldA > fieldB) return this.sortAsc ? 1 : -1;
67     return 0;
68   });
69
70   return result;
71 }
72
73 get paginatedRows(): RevenueTableRow[] {
74   const start = (this.currentPage - 1) * this.itemsPerPage;
75   const end = start + this.itemsPerPage;
76   return this.filteredRows.slice(start, end);
77 }
78
79 get totalPages(): number {
80   return Math.ceil(this.filteredRows.length / this.itemsPerPage);
81 }
82
83 openFilter() {
84   this.showFilter = !this.showFilter;
85 }
86
87 closeFilter() {
88   this.showFilter = false;
89 }
90
91 apply() {
92   this.showFilter = false;
93
94   this.applyFilters();
95 }
96
97 applyFilters() {
98   this.goToPage(1);
99 }
100
101 goToPage(page: number) {
102   if (page < 1) page = 1;
103   if (page > this.totalPages) page = this.totalPages;
104   this.currentPage = page;
105 }
106
107 prevPage() {
108   this.goToPage(this.currentPage - 1);
109 }
110
111 nextPage() {
112   this.goToPage(this.currentPage + 1);
113 }
114
115 toggleSort() {
116   this.sortAsc = !this.sortAsc;
117 }
118
119 filterRevenueRows(
120   rows: RevenueTableRow[],
121   filter: RevenueFilter
122 ): RevenueTableRow[] {
123   const lowerSearch = filter.searchTerm?.toLowerCase() ?? '';
124 }
```

```

125     return rows.filter((row) => {
126       const matchesSearch =
127         lowerSearch === '' || row.date.toLowerCase().includes(lowerSearch);
128
129       const matchesMinOrders =
130         filter.minOrders === undefined ||
131         filter.minOrders === null ||
132         row.orders >= filter.minOrders;
133       const matchesMaxOrders =
134         filter.maxOrders === undefined ||
135         filter.maxOrders === null ||
136         row.orders <= filter.maxOrders;
137
138       const matchesMinGrossRevenue =
139         filter.minGrossRevenue === undefined ||
140         filter.minGrossRevenue === null ||
141         row.grossRevenue >= filter.minGrossRevenue;
142       const matchesMaxGrossRevenue =
143         filter.maxGrossRevenue === undefined ||
144         filter.maxGrossRevenue === null ||
145         row.grossRevenue <= filter.maxGrossRevenue;
146
147       const matchesMinTotalSales =
148         filter.minTotalSales === undefined ||
149         filter.minTotalSales === null ||
150         row.totalSales >= filter.minTotalSales;
151       const matchesMaxTotalSales =
152         filter.maxTotalSales === undefined ||
153         filter.maxTotalSales === null ||
154         row.totalSales <= filter.maxTotalSales;
155
156       if (this.showFilter) {
157         return matchesSearch;
158       }
159
160       return (
161         matchesSearch &&
162         matchesMinOrders &&
163         matchesMaxOrders &&
164         matchesMinGrossRevenue &&
165         matchesMaxGrossRevenue &&
166         matchesMinTotalSales &&
167         matchesMaxTotalSales
168       );
169     });
170   }
171
172   formatCurrency(value: number | null | undefined): string {
173     return this.numberFormatter.formatCurrency(value);
174   }
175 }
176

```

Ø=ÜÁ features\revenue\components\revenueGraph

Ø=ÜÄ features\revenue\components\revenueGraph\revenue-graph.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { Component, Inject, Input, PLATFORM_ID } from '@angular/core';
3 import { NgApexchartsModule } from 'ng-apexcharts';
4 import { FormsModule } from '@angular/forms';
5 import { DailyRevenueData, MonthlyRevenueData } from '../../../../../models/revenue';
6 import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';

```

```

7
8 /**
9  * Component for displaying revenue charts.
10 *
11 * This component displays revenue data in either bar or area chart format,
12 * with support for filtering by day, month, or year. It uses ApexCharts
13 * for visualization and includes interactive tooltips showing percentage changes.
14 *
15 * Features:
16 * - Bar and area chart types
17 * - Filter by day, month, or year
18 * - Interactive tooltips with percentage changes
19 * - Automatic data sorting and formatting
20 * - Gross revenue calculation
21 *
22 * Relations:
23 * - NgApexchartsModule: Chart rendering
24 * - NumberFormatterService: Value formatting
25 *
26 * @component
27 * @selector app-revenue-graph
28 * @standalone true
29 */
30 @Component({
31   selector: 'app-revenue-graph',
32   templateUrl: './revenue-graph.component.html',
33   styleUrls: ['./revenue-graph.component.scss'],
34   standalone: true,
35   imports: [CommonModule, NgApexchartsModule, FormsModule],
36 })
37 export class RevenueGraphComponent {
38   @Input() dailyData!: DailyRevenueData[];
39   @Input() monthlyData!: MonthlyRevenueData[];
40
41   actualYear: number = new Date().getFullYear();
42   private numberFormatter = new NumberFormatterService();
43
44   filterType: 'day' | 'month' | 'year' = 'day';
45
46   chartType: 'bar' | 'area' = 'bar';
47
48   public chartOptions: any;
49
50   constructor(@Inject(PLATFORM_ID) private platformId: any) {}
51
52   ngOnInit() {}
53
54   get filterTypeLabel(): string {
55     switch (this.filterType) {
56       case 'day':
57         return 'By day';
58       case 'month':
59         return 'By month';
60       case 'year':
61         return 'By year';
62     }
63   }
64
65   setChartType(type: 'bar' | 'area') {
66     this.chartType = type;
67     this.changeChartData();
68   }
69
70   setFilterType(type: 'day' | 'month' | 'year') {
71     this.filterType = type;
72     this.changeChartData();
73   }
74
75   onFilterTypeChange(event: any) {
76     this.filterType = event.target.value;

```

```

77     this.changeChartData();
78 }
79
80 ngOnChanges() {
81   if (this.dailyData) {
82     this.chartOptions = this.getChartOptions(this.dailyData, this.chartType);
83   }
84 }
85
86 changeChartData() {
87   if (this.filterType === 'day') {
88     this.chartOptions = this.getChartOptions(this.dailyData, this.chartType);
89   } else if (this.filterType === 'month') {
90     this.chartOptions = this.getChartOptions(
91       this.monthlyData,
92       this.chartType
93     );
94   }
95 }
96
97 getChartOptions(
98   data: DailyRevenueData[] | MonthlyRevenueData[],
99   chartType: 'bar' | 'area'
100 ): any {
101   const isDailyData = 'date' in data[0];
102   let categories: string[] = [];
103
104   if (isDailyData) {
105     categories = data
106       .map((d) => new Date(d as DailyRevenueData).date).getDate().toString()
107       .sort((a, b) => parseInt(a) - parseInt(b));
108   } else {
109     categories = (data as MonthlyRevenueData[])
110       .map((d) => {
111         const dateObj = new Date(d.year, d.month - 1);
112         return dateObj.toLocaleString('en-US', {
113           month: 'short',
114           year: 'numeric',
115         });
116       });
117   }
118
119   let sortedData = data.slice();
120   if (isDailyData) {
121     sortedData = sortedData.sort(
122       (a, b) =>
123         new Date((a as DailyRevenueData).date).getDate() -
124         new Date((b as DailyRevenueData).date).getDate()
125     );
126   } else {
127     sortedData = sortedData.sort((a, b) =>
128       (a as MonthlyRevenueData).year === (b as MonthlyRevenueData).year
129       ? (a as MonthlyRevenueData).month - (b as MonthlyRevenueData).month
130       : (a as MonthlyRevenueData).year - (b as MonthlyRevenueData).year
131     );
132   }
133
134   const chartData = sortedData.map((d) => d.grossRevenue);
135   const actualYear = this.actualYear;
136
137   if (chartType === 'area') {
138     return {
139       chart: {
140         type: 'area',
141         height: 350,
142         toolbar: { show: false },
143         foreColor: '#fff',
144         fontFamily: 'Inter, Arial, sans-serif',
145         background: 'transparent',
146       },
147       series: [

```

```

147     },
148     name: 'months',
149     data: chartData,
150   },
151 ],
152 xaxis: {
153   categories: categories,
154   labels: {
155     style: { colors: '#d8d8d8' },
156   },
157   axisBorder: { show: false },
158   axisTicks: { show: false },
159 },
160 yaxis: {
161   labels: {
162     style: { colors: '#d8d8d8' },
163   },
164 },
165 grid: {
166   borderColor: '#333',
167   strokeDashArray: 4,
168   xaxis: {
169     lines: {
170       show: true,
171     },
172   },
173 },
174 dataLabels: { enabled: false },
175 stroke: {
176   curve: 'straight',
177   width: 1,
178   colors: ['#EAF2F8'],
179 },
180 fill: {
181   type: 'gradient',
182   gradient: {
183     shade: 'dark',
184     type: 'vertical',
185     gradientToColors: ['#3967D7'],
186     opacityFrom: 0.9,
187     opacityTo: 0,
188   },
189 },
190 tooltip: {
191   theme: 'dark',
192   x: { show: true },
193   custom: function ({ series, seriesIndex, dataPointIndex, w }: any) {
194     const value = series[seriesIndex][dataPointIndex];
195
196     const prevValue =
197       dataPointIndex > 0
198         ? series[seriesIndex][dataPointIndex - 1]
199         : null;
200     let percentDiff: number | null = 0;
201     let direction = null;
202     let cardClass = '';
203     let validatorClass = '';
204
205     if (prevValue !== null && prevValue !== 0) {
206       percentDiff = Math.round(
207         ((value - prevValue) / Math.abs(prevValue)) * 100
208       );
209       direction = percentDiff > 0 ? 'up' : 'down';
210       if (direction === 'up') {
211         cardClass = 'positive-container';
212         validatorClass = 'positive-validator';
213       } else {
214         cardClass = 'negative-container';
215         validatorClass = 'negative-validator';
216       }

```

```

217     } else {
218         percentDiff = null;
219         direction = null;
220     }
221
222     return `<div class=" ${cardClass} regularText color-background d-flex flex-col
223 toolTip-contain items-center">$actualYear</p>
224     <div class="d-flex text-container items-center ">
225         <p class= "subtitle">
226             $$value
227         </p>
228     ${
229         percentDiff != null
230         ? `<span class="smallText py-4 px-6 d-flex justify-center items-center
231 ${validatorClass}"${percentDiff}><span class="${
232             ? 'icon-status-arrow-up'
233             : 'icon-status-arrow'
234         } ml-3"></span></span>` :
235         ''
236     }
237     </div>
238
239
240     </div>`;
241 },
242     position: function (data: any, opts: any) {
243         return {
244             left: data.point.x,
245             top: data.point.y - 160,
246         };
247     },
248 },
249 );
250
251
252     return {
253         chart: {
254             type: 'bar',
255             height: 350,
256             toolbar: { show: false },
257             foreColor: '#fff',
258             background: 'transparent',
259             fontFamily: 'Inter, Arial, sans-serif',
260         },
261         series: [
262             {
263                 name: 'Gross Revenue',
264                 data: chartData,
265             },
266         ],
267         fill: {
268             colors: ['#F44336', '#E91E63', '#9C27B0'],
269         },
270         xaxis: {
271             categories: categories,
272             labels: { style: { colors: '#d8d8d8' } },
273             axisBorder: { show: false },
274             axisTicks: { show: false },
275         },
276         yaxis: {
277             labels: { style: { colors: '#d8d8d8' } },
278         },
279         grid: {
280             borderColor: '#333',
281             strokeDashArray: 4,
282             xaxis: { lines: { show: true } },
283         },
284         dataLabels: { enabled: false },
285         tooltip: {

```

```

287     theme: 'dark',
288     x: { show: true },
289     custom: ({ series, seriesIndex, dataPointIndex, w }: any) => {
290       const value = series[seriesIndex][dataPointIndex];
291       const prevValue =
292         dataPointIndex > 0 ? series[seriesIndex][dataPointIndex - 1] : null;
293       let percentDiff: number | null = 0;
294       let direction = null;
295       let cardClass = '';
296       let validatorClass = '';
297
298       if (prevValue !== null && prevValue !== 0) {
299         percentDiff = Math.round(
300           ((value - prevValue) / Math.abs(prevValue)) * 100
301         );
302         direction = percentDiff > 0 ? 'up' : 'down';
303         if (direction === 'up') {
304           cardClass = 'positive-container';
305           validatorClass = 'positive-validator';
306         } else {
307           cardClass = 'negative-container';
308           validatorClass = 'negative-validator';
309         }
310       } else {
311         percentDiff = null;
312         direction = null;
313       }
314
315       const formattedValue = this.numberFormatter.formatCurrency(value);
316       const formattedPercent = this.numberFormatter.formatPercentageValue(percentDiff);
317
318       return `<div class="${cardClass} regularText color-background d-flex flex-col
319 toolTip-containsText mainText" style="color-text-gray">${actualYear}</p>
320         <div class="d-flex text-container items-center">
321           <p class="subtitle">${formattedValue}</p>
322           ${
323             percentDiff != null
324             ? <span class="smallText py-4 px-6 d-flex justify-center items-center
325 ${validatorClass}"> ${formattedPercent}% <span class="${
326               direction === 'up'
327                 ? 'icon-status-arrow-up'
328                 : 'icon-status-arrow'
329             } ml-3"></span>
330             </span>
331           :
332             }
333           </div>
334         </div>`;
335       },
336     },
337   );
338 }
339
340 capitalizeFirstLetter(str: string): string {
341   if (!str) return '';
342   return str.charAt(0).toUpperCase() + str.slice(1);
343 }
344
345 get grossRevenue(): number {
346   const result = this.dailyData.reduce(
347     (acc, curr) => acc + (curr.grossRevenue ?? 0),
348     0
349   );
350   return result;
351 }
352 }
353

```

Ø=ÜÁ features\revenue\components\subscriptions-table

Ø=ÜÄ features\revenue\components\subscriptions-table\subscriptions-table.component.ts

```
1 import { Component, Input, Output } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 import { FormsModule } from '@angular/forms';
5 import {
6   SubscriptionFilter,
7   SubscriptionStatus,
8   SubscriptionTableRow,
9 } from '../../../../../models/revenue';
10
11 /**
12 * Component for displaying subscriptions data in a table format.
13 *
14 * This component provides a paginated and filterable table for subscription data.
15 * It supports filtering by search term, subscription status, and total amount,
16 * as well as sorting and pagination.
17 *
18 * Features:
19 * - Search by subscription name
20 * - Filter by subscription status (Active, Pending, Failed)
21 * - Filter by total amount range (parses currency strings)
22 * - Sort by start date (ascending/descending)
23 * - Pagination
24 *
25 * @component
26 * @selector app-subscriptions-table
27 * @standalone true
28 */
29 @Component({
30   selector: 'app-subscriptions-table',
31   standalone: true,
32   imports: [CommonModule, FormsModule],
33   templateUrl: './subscriptions-table.component.html',
34   styleUrls: ['./subscriptions-table.component.scss'],
35 })
36 export class SubscriptionsTableComponent {
37   @Input() subscriptionRows: SubscriptionTableRow[] = [];
38
39   showFilter = false;
40   currentPage: number = 1;
41   itemsPerPage: number = 10;
42   sortField: 'startDate' = 'startDate';
43   sortAsc: boolean = true;
44
45   subscriptionStatus = SubscriptionStatus;
46
47   filter: SubscriptionFilter = {};
48
49   private _searchTerm = '';
50   get searchTerm(): string {
51     return this._searchTerm;
52   }
53   set searchTerm(val: string) {
54     this._searchTerm = val;
55     this.goToPage(1);
56   }
57
58   get filteredRows(): SubscriptionTableRow[] {
59     let result = this.filterOrderRows(this.subscriptionRows, this.filter);
60
61     result = result.sort((a, b) => {
```

```

62     const fieldA = a[this.sortField].toLowerCase();
63     const fieldB = b[this.sortField].toLowerCase();
64     if (fieldA < fieldB) return this.sortAsc ? -1 : 1;
65     if (fieldA > fieldB) return this.sortAsc ? 1 : -1;
66     return 0;
67   });
68
69   return result;
70 }
71
72 get paginatedRows(): SubscriptionTableRow[] {
73   const start = (this.currentPage - 1) * this.itemsPerPage;
74   const end = start + this.itemsPerPage;
75   return this.filteredRows.slice(start, end);
76 }
77
78 get totalPages(): number {
79   return Math.ceil(this.filteredRows.length / this.itemsPerPage);
80 }
81
82 statusClass(status: string) {
83   return status;
84 }
85
86 openFilter() {
87   this.showFilter = !this.showFilter;
88 }
89
90 closeFilter() {
91   this.showFilter = false;
92 }
93
94 apply() {
95   this.showFilter = false;
96
97   this.applyFilters();
98 }
99
100 applyFilters() {
101   this.goToPage(1);
102 }
103
104 goToPage(page: number) {
105   if (page < 1) page = 1;
106   if (page > this.totalPages) page = this.totalPages;
107   this.currentPage = page;
108 }
109
110 prevPage() {
111   this.goToPage(this.currentPage - 1);
112 }
113
114 nextPage() {
115   this.goToPage(this.currentPage + 1);
116 }
117
118 toggleSort() {
119   this.sortAsc = !this.sortAsc;
120 }
121
122 filterOrderRows(
123   rows: SubscriptionTableRow[],
124   filter: SubscriptionFilter
125 ): SubscriptionTableRow[] {
126   const lowerSearch = filter.searchTerm?.toLowerCase() ?? '';
127
128   return rows.filter((row) => {
129     const matchesSearch =
130       lowerSearch === '' ||
131       row.subscription.toLowerCase().includes(lowerSearch);

```

```

132
133     const matchesStatus =
134         filter.status === undefined ||
135         filter.status === null ||
136         filter.status === ('' as SubscriptionStatus) ||
137         row.status === filter.status;
138
139     const matchesMinTotal =
140         filter.minTotal === undefined ||
141         filter.minTotal === null ||
142         parseFloat(row.total.replace('$', '')).split('/')[0]) >= filter.minTotal;
143
144     const matchesMaxTotal =
145         filter.maxTotal === undefined ||
146         filter.maxTotal === null ||
147         parseFloat(row.total.replace('$', '')).split('/')[0]) <= filter.maxTotal;
148
149     if (this.showFilter) {
150         return matchesSearch;
151     }
152
153     return (
154         matchesSearch && matchesStatus && matchesMinTotal && matchesMaxTotal
155     );
156 );
157 }
158 }
159

```

Ø=ÜÁ features\revenue\mocks

Ø=ÜÁ features\revenue\mocks\revenue_mock.ts

```

1 import {
2     DailyRevenueData,
3     MonthlyRevenueData,
4     OrderStatus,
5     OrderTableRow,
6     RevenueSummary,
7     RevenueTableRow,
8     SubscriptionStatus,
9     SubscriptionTableRow,
10    YearlyRevenueData,
11 } from '../models/revenue';
12
13 /**
14 * Mock data for revenue summary.
15 *
16 * Used for development and testing purposes. Contains aggregated revenue metrics.
17 *
18 * @constant mockRevenueSummary
19 * @type {RevenueSummary}
20 */
21 export const mockRevenueSummary: RevenueSummary = {
22     grossRevenue: 45750,
23     returns: 56,
24     coupons: 1870,
25     netRevenue: 24780,
26     totalRevenue: 24780,
27 };
28
29 /**
30 * Mock data for daily revenue.

```

```

31  *
32  * Contains daily revenue data for a month (August 2025).
33  * Used for chart visualization in development.
34  *
35  * @constant dailyRevenueMock
36  * @type {DailyRevenueData[]}
37  */
38 export const dailyRevenueMock: DailyRevenueData[] = [
39  { date: '2025-08-01', grossRevenue: 400 },
40  { date: '2025-08-02', grossRevenue: 700 },
41  { date: '2025-08-03', grossRevenue: 1100 },
42  { date: '2025-08-04', grossRevenue: 1700 },
43  { date: '2025-08-05', grossRevenue: 3000 },
44  { date: '2025-08-06', grossRevenue: 2350 },
45  { date: '2025-08-07', grossRevenue: 900 },
46  { date: '2025-08-08', grossRevenue: 1200 },
47  { date: '2025-08-09', grossRevenue: 1500 },
48  { date: '2025-08-10', grossRevenue: 1800 },
49  { date: '2025-08-11', grossRevenue: 400 },
50  { date: '2025-08-12', grossRevenue: 700 },
51  { date: '2025-08-13', grossRevenue: 1100 },
52  { date: '2025-08-14', grossRevenue: 1700 },
53  { date: '2025-08-15', grossRevenue: 3000 },
54  { date: '2025-08-16', grossRevenue: 2350 },
55  { date: '2025-08-17', grossRevenue: 900 },
56  { date: '2025-08-18', grossRevenue: 1200 },
57  { date: '2025-08-19', grossRevenue: 1500 },
58  { date: '2025-08-20', grossRevenue: 1800 },
59  { date: '2025-08-21', grossRevenue: 400 },
60  { date: '2025-08-22', grossRevenue: 700 },
61  { date: '2025-08-23', grossRevenue: 1100 },
62  { date: '2025-08-24', grossRevenue: 1700 },
63  { date: '2025-08-25', grossRevenue: 3000 },
64  { date: '2025-08-26', grossRevenue: 2350 },
65  { date: '2025-08-27', grossRevenue: 900 },
66  { date: '2025-08-28', grossRevenue: 1200 },
67  { date: '2025-08-29', grossRevenue: 1500 },
68  { date: '2025-08-30', grossRevenue: 1800 },
69  { date: '2025-08-31', grossRevenue: 1800 },
70 ];
71 /**
72  * Mock data for monthly revenue.
73  *
74  * Contains monthly revenue data for the year 2025.
75  * Used for chart visualization in development.
76  *
77  * @constant monthlyRevenueMock
78  * @type {MonthlyRevenueData[]}
79  */
80 export const monthlyRevenueMock: MonthlyRevenueData[] = [
81  { year: 2025, month: 1, grossRevenue: 21000 },
82  { year: 2025, month: 2, grossRevenue: 19800 },
83  { year: 2025, month: 3, grossRevenue: 23050 },
84  { year: 2025, month: 4, grossRevenue: 25500 },
85  { year: 2025, month: 5, grossRevenue: 24800 },
86  { year: 2025, month: 6, grossRevenue: 26706.52 },
87  { year: 2025, month: 7, grossRevenue: 28000 },
88  { year: 2025, month: 8, grossRevenue: 29000 },
89  { year: 2025, month: 9, grossRevenue: 30000 },
90  { year: 2025, month: 10, grossRevenue: 31000 },
91  { year: 2025, month: 11, grossRevenue: 32000 },
92  { year: 2025, month: 12, grossRevenue: 33000 },
93 ];
94 /**
95  * Mock data for revenue table.
96  *
97  * Contains sample revenue table rows with daily revenue breakdown.
98  * Used for table display in development.
99
100
```

```

101  *
102  * @constant revenueTableMock
103  * @type {RevenueTableRow[]}
104  */
105 export const revenueTableMock: RevenueTableRow[] = [
106  {
107    date: 'June 23,2025',
108    orders: 2,
109    grossRevenue: 665.0,
110    returns: 0.0,
111    coupons: 52.0,
112    netSales: 603.0,
113    taxes: 0.0,
114    shipping: 0.0,
115    totalSales: 603.0,
116  },
117  {
118    date: 'June 22,2025',
119    orders: 2,
120    grossRevenue: 377.5,
121    returns: 0.0,
122    coupons: 0.0,
123    netSales: 377.5,
124    taxes: 0.0,
125    shipping: 0.0,
126    totalSales: 377.5,
127  },
128  {
129    date: 'June 21,2025',
130    orders: 0,
131    grossRevenue: 0.0,
132    returns: 0.0,
133    coupons: 0.0,
134    netSales: 0.0,
135    taxes: 0.0,
136    shipping: 0.0,
137    totalSales: 0.0,
138  },
139  {
140    date: 'June 20,2025',
141    orders: 3,
142    grossRevenue: 189.0,
143    returns: 0.0,
144    coupons: 7.0,
145    netSales: 182.0,
146    taxes: 0.0,
147    shipping: 0.0,
148    totalSales: 182.0,
149  },
150  {
151    date: 'June 19,2025',
152    orders: 0,
153    grossRevenue: 0.0,
154    returns: 0.0,
155    coupons: 0.0,
156    netSales: 0.0,
157    taxes: 0.0,
158    shipping: 0.0,
159    totalSales: 0.0,
160  },
161];
162
163 /**
164  * Mock data for orders table.
165  *
166  * Contains sample order rows with order details.
167  * Used for orders table display in development.
168  *
169  * @constant orderTableMock
170  * @type {OrderTableRow[]}

```

```

171  */
172 export const orderTableMock: OrderTableRow[] = [
173 {
174   orderId: '#3686',
175   user: 'Andrés Valdes',
176   date: '3 hours ago',
177   status: OrderStatus.Completed,
178   total: 127.0,
179   affiliateReferral: '#532',
180   origin: 'Referral:Tx3funding.com',
181 },
182 {
183   orderId: '#3686',
184   user: 'Marvin McKinney',
185   date: '3 hours ago',
186   status: OrderStatus.Pending,
187   total: 127.0,
188   affiliateReferral: null,
189   origin: 'Referral:Tx3funding.com',
190 },
191 {
192   orderId: '#3686',
193   user: 'Guy Hawkins',
194   date: '5 hours ago',
195   status: OrderStatus.Completed,
196   total: 127.0,
197   affiliateReferral: '#532',
198   origin: 'Referral:Tx3funding.com',
199 },
200 {
201   orderId: '#3686',
202   user: 'Ralph Edwards',
203   date: '6 hours ago',
204   status: OrderStatus.Cancelled,
205   total: 127.0,
206   affiliateReferral: '#532',
207   origin: 'Referral:Tx3funding.com',
208 },
209 {
210   orderId: '#3686',
211   user: 'Jenny Wilson',
212   date: '6 hours ago',
213   status: OrderStatus.Failed,
214   total: 127.0,
215   affiliateReferral: '#532',
216   origin: 'Referral:Tx3funding.com',
217 },
218 ];
219
220 /**
221 * Mock data for subscriptions table.
222 *
223 * Contains sample subscription rows with subscription details.
224 * Used for subscriptions table display in development.
225 *
226 * @constant subscriptionTableMock
227 * @type {SubscriptionTableRow[]}
228 */
229 export const subscriptionTableMock: SubscriptionTableRow[] = [
230 {
231   status: SubscriptionStatus.Active,
232   subscription: '#3686 Andrés Valdes',
233   items: 'Pro Model',
234   total: '$157.00/month',
235   startDate: '3 hours ago',
236   trialEnd: '-',
237   nextPayment: 'July 23, 2025',
238   lastOrderDate: '-',
239   endDate: '-',
240   orders: 1,

```

```

241     },
242     {
243       status: SubscriptionStatus.Pending,
244       subscription: '#3686 Andrés Valdes',
245       items: 'Pro Model',
246       total: '$157.00/month',
247       startDate: '3 hours ago',
248       trialEnd: '-',
249       nextPayment: 'July 23, 2025',
250       lastOrderDate: '-',
251       endDate: '-',
252       orders: 1,
253     },
254     {
255       status: SubscriptionStatus.Active,
256       subscription: '#3686 Andrés Valdes',
257       items: 'Pro Model',
258       total: '$157.00/month',
259       startDate: '3 hours ago',
260       trialEnd: '-',
261       nextPayment: 'July 23, 2025',
262       lastOrderDate: '-',
263       endDate: '-',
264       orders: 1,
265     },
266     {
267       status: SubscriptionStatus.Failed,
268       subscription: '#3686 Andrés Valdes',
269       items: 'Pro Model',
270       total: '$157.00/month',
271       startDate: '3 hours ago',
272       trialEnd: '-',
273       nextPayment: 'July 23, 2025',
274       lastOrderDate: '-',
275       endDate: '-',
276       orders: 1,
277     },
278   {
279     status: SubscriptionStatus.Active,
280     subscription: '#3686 Andrés Valdes',
281     items: 'Pro Model',
282     total: '$157.00/month',
283     startDate: '3 hours ago',
284     trialEnd: '-',
285     nextPayment: 'July 23, 2025',
286     lastOrderDate: '-',
287     endDate: '-',
288     orders: 1,
289   },
290 ];
291

```

Ø=ÜÁ features\revenue\models

Ø=ÜÁ features\revenue\models\revenue.ts

```

1  /**
2   * Enum representing the possible statuses of an order.
3   *
4   * @enum {string}
5   */
6  export enum OrderStatus {
7    Completed = 'Completed',

```

```

8     Pending = 'Pending',
9     Cancelled = 'Cancelled',
10    Failed = 'Failed',
11 }
12
13 /**
14  * Enum representing the possible statuses of a subscription.
15  *
16  * @enum {string}
17  */
18 export enum SubscriptionStatus {
19     Active = 'Active',
20     Pending = 'Pending',
21     Failed = 'Failed',
22 }
23
24 /**
25  * Interface representing a summary of revenue data.
26  *
27  * Contains aggregated revenue metrics including gross revenue, returns, coupons,
28  * net revenue, and total revenue.
29  *
30  * @interface RevenueSummary
31  */
32 export interface RevenueSummary {
33     grossRevenue: number;
34     returns: number;
35     coupons: number;
36     netRevenue: number;
37     totalRevenue: number;
38 }
39
40 /**
41  * Interface representing a row in the revenue table.
42  *
43  * Contains daily revenue data including orders count, gross revenue, returns,
44  * coupons, net sales, taxes, shipping, and total sales.
45  *
46  * @interface RevenueTableRow
47  */
48 export interface RevenueTableRow {
49     date: string;
50     orders: number;
51     grossRevenue: number;
52     returns: number;
53     coupons: number;
54     netSales: number;
55     taxes: number;
56     shipping: number;
57     totalSales: number;
58 }
59
60 /**
61  * Interface representing daily revenue data.
62  *
63  * Used for chart visualization showing revenue trends by day.
64  *
65  * @interface DailyRevenueData
66  */
67 export interface DailyRevenueData {
68     date: string;
69     grossRevenue: number;
70 }
71
72 /**
73  * Interface representing monthly revenue data.
74  *
75  * Used for chart visualization showing revenue trends by month.
76  *
77  * @interface MonthlyRevenueData

```

```

78  /*
79  export interface MonthlyRevenueData {
80   year: number;
81   month: number;
82   grossRevenue: number;
83 }
84
85 /**
86 * Interface representing yearly revenue data.
87 *
88 * Used for chart visualization showing revenue trends by year.
89 *
90 * @interface YearlyRevenueData
91 */
92 export interface YearlyRevenueData {
93   year: number;
94   grossRevenue: number;
95 }
96
97 /**
98 * Interface representing a row in the orders table.
99 *
100 * Contains order information including order ID, user, date, status, total,
101 * affiliate referral, and origin.
102 *
103 * @interface OrderTableRow
104 */
105 export interface OrderTableRow {
106   orderId: string;
107   user: string;
108   date: string;
109   status: OrderStatus;
110   total: number;
111   affiliateReferral: string | null;
112   origin: string;
113 }
114
115 /**
116 * Interface representing a row in the subscriptions table.
117 *
118 * Contains subscription information including status, subscription details,
119 * items, total, dates, and order count.
120 *
121 * @interface SubscriptionTableRow
122 */
123 export interface SubscriptionTableRow {
124   status: SubscriptionStatus;
125   subscription: string;
126   items: string;
127   total: string;
128   startDate: string;
129   trialEnd: string;
130   nextPayment: string;
131   lastOrderDate: string;
132   endDate: string;
133   orders: number;
134 }
135
136 /**
137 * Interface representing filter criteria for revenue table.
138 *
139 * Used to filter revenue data by search term, order count, gross revenue, and total sales.
140 *
141 * @interface RevenueFilter
142 */
143 export interface RevenueFilter {
144   searchTerm?: string;
145   minOrders?: number;
146   maxOrders?: number;
147   minGrossRevenue?: number;

```

```

148     maxGrossRevenue?: number;
149     minTotalSales?: number;
150     maxTotalSales?: number;
151 }
152 /**
153 * Interface representing filter criteria for orders table.
154 *
155 * Used to filter orders by search term, status, and total amount.
156 *
157 * @interface OrderFilter
158 */
159 export interface OrderFilter {
160     searchTerm?: string;
161     status?: OrderStatus;
162     minTotal?: number;
163     maxTotal?: number;
164 }
165 }
166 /**
167 * Interface representing filter criteria for subscriptions table.
168 *
169 * Used to filter subscriptions by search term, status, and total amount.
170 *
171 * @interface SubscriptionFilter
172 */
173 export interface SubscriptionFilter {
174     searchTerm?: string;
175     status?: SubscriptionStatus;
176     minTotal?: number;
177     maxTotal?: number;
178 }
179 }
180

```

Ø=ÜÁ features\strategy

Ø=ÜÁ features\strategy\strategy.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { Component, OnInit, OnDestroy } from '@angular/core';
3 import { Router } from '@angular/router';
4 import { FormsModule } from '@angular/forms';
5 import { User } from '../overview/models/overview';
6 import { selectUser } from '../auth/store/user.selectors';
7 import { SettingsService } from './service/strategy.service';
8 import { ConfigurationOverview } from './models/strategy.model';
9 import { TextInputComponent, StrategyCardComponent, StrategyCardData,
10 StrategyCardModel, popComponentLoad, spinnerComponentPop, planComponentPop } from './...
11 shared/components';
12 import { ReportService } from '../report/service/report.service';
13 import { AuthService } from '../auth/service/authService';
14 import { initialStrategyState } from '../strategy/store/strategy.reducer';
15 import { selectTradeWin } from '../report/store/report.selectors';
16 import { selectTotalTrades } from '../report/store/report.selectors';
17 import { resetConfig } from '../strategy/store/strategy.actions';
18 import { StrategyState } from '../strategy/models/strategy.model';
19 import { AccountData } from '../auth/models/userModel';
20 import { setUserKey } from '../report/store/report.actions';
21 import { selectUserKey } from '../report/store/report.selectors';
22 import { GlobalStrategyUpdaterService } from '....shared/services/global-strategy-
23 upater[serviçes] from '../strategy/store/strategy.selectors';
24 import { selectNetPnL } from '../report/store/report.selectors';
25 import { PlanLimitationsGuard } from '....guards/plan-limitations.guard';

```

```

26 import { ApplicationContextService } from '../../../../../shared/context';
27 import { StrategyCacheService } from './services/strategy-cache.service';
28 import { BalanceCacheService } from './services/balance-cache.service';
29 import { AlertService } from '../../../../../shared/services/alert.service';
30
31 /**
32  * Main component for managing trading strategies.
33  *
34  * This component provides comprehensive strategy management functionality including:
35  * - Creating, editing, activating, copying, and deleting strategies
36  * - Displaying strategy cards with statistics
37  * - Managing multiple strategies per user
38  * - Plan limitation detection and enforcement
39  * - Strategy guide modal for first-time users
40  * - Caching strategies for performance
41  * - Real-time balance updates for risk calculations
42  *
43  * Key Features:
44  * - Multiple strategy support with active/inactive states
45  * - Strategy caching for fast access
46  * - Balance caching for risk calculations
47  * - Plan limitation checks (max strategies per plan)
48  * - Strategy guide for new users
49  * - Search and filter strategies
50  * - Strategy cards with win rate, active rules, and days active
51  *
52  * Data Flow:
53  * 1. Component initializes and loads user data
54  * 2. Loads all user accounts
55  * 3. Loads all strategies (overview + configuration) into cache
56  * 4. Sets up listeners for context data and report data
57  * 5. Updates strategy cards with real-time data
58  *
59  * Relations:
60  * - SettingsService: CRUD operations for strategies
61  * - StrategyCacheService: Caching strategy data
62  * - BalanceCacheService: Caching account balances
63  * - PlanLimitationsGuard: Checking plan limitations
64  * - ApplicationContextService: Global state management
65  * - ReportService: Fetching account balances
66  * - Store (NgRx): Local state for strategy rules
67  * - EditStrategyComponent: Editing strategy details
68  * - StrategyCardComponent: Displaying strategy cards
69  *
70  * @component
71  * @selector app-strategy
72  * @standalone true
73 */
74
75 @Component({
76   selector: 'app-strategy',
77   imports: [
78     CommonModule,
79     FormsModule,
80     TextInputComponent,
81     StrategyCardComponent,
82     StrategyGuideModalComponent,
83     LoadingSpinnerComponent,
84     PlanBannerComponent,
85     ConfirmPopupComponent,
86   ],
87   templateUrl: './strategy.component.html',
88   styleUrls: ['./strategy.component.scss'],
89   standalone: true,
90 })
91 export class Strategy implements OnInit, OnDestroy {
92   config: any = {};
93   user: User | null = null;
94   loading = false;
95   initialLoading = true;

```

```

96
97 // Plan detection and banner
98 accountsData: AccountData[] = [];
99 showPlanBanner = false;
100 planBannerMessage = '';
101 planBannerType = 'info'; // 'info', 'warning', 'success'
102
103
104 // Button state
105 isAddStrategyDisabled = false;
106
107 // Strategy guide modal
108 showStrategyGuide = false;
109
110 // Loading state for strategy creation
111 isCreatingStrategy = false;
112
113 // Loading state for strategy operations (activate, delete, copy)
114 isProcessingStrategy = false;
115
116 // Delete strategy confirmation popup
117 showDeleteConfirmPopup = false;
118 strategyToDeleteId: string = '';
119
120 // Report data for strategy card
121 tradeWin: number = 0;
122 totalTrades: number = 0;
123 netPnL: number = 0;
124
125 // Nuevas propiedades para múltiples estrategias
126 userStrategies: ConfigurationOverview[] = [];
127 activeStrategy: ConfigurationOverview | null = null;
128 filteredStrategies: ConfigurationOverview[] = [];
129 strategyCardsData: StrategyCardData[] = [];
130 searchTerm = '';
131
132 // Strategy Card Data - Dynamic
133 strategyCard: StrategyCardData = {
134   id: '1',
135   name: 'My Trading Strategy',
136   status: true,
137   lastModified: 'Never',
138   rules: 0,
139   days_active: 0,
140   winRate: 0,
141   isFavorite: false,
142   created_at: null,
143   updated_at: null,
144   userId: '',
145   configurationId: ''
146 };
147
148 // CACHE CENTRALIZADO - Usar servicio de cache
149
150 constructor(
151   private store: Store,
152   private router: Router,
153   private strategySvc: SettingsService,
154   private reportSvc: ReportService,
155   private authService: AuthService,
156   private globalStrategyUpdater: GlobalStrategyUpdaterService,
157   private planLimitationsGuard: PlanLimitationsGuard,
158   private appContext: AppContextService,
159   private strategyCacheService: StrategyCacheService,
160   private balanceCacheService: BalanceCacheService,
161   private alertService: AlertService
162 ) {}
163
164 async ngOnInit(): Promise<void> {
165   this.initialLoading = true;

```

```

166
167     try {
168         // FLUJO SIMPLIFICADO: Una sola secuencia de inicialización
169         await this.initializeEverything();
170     } catch (error) {
171         console.error('Error during initialization:', error);
172     } finally {
173         this.initialLoading = false;
174     }
175 }
176 /**
177 * FLUJO SIMPLIFICADO DE INICIALIZACIÓN
178 * 1. Obtener usuario
179 * 2. Cargar cuentas
180 * 3. Cargar TODAS las estrategias completas (overview + configuration) en cache
181 * 4. Configurar listeners
182 * 5. Cargar datos de reporte
183 */
184
185 private async initializeEverything(): Promise<void> {
186     // 1. Obtener usuario
187     await this.initializeUserData();
188
189     if (!this.user?.id) {
190         console.error('No user ID available');
191         return;
192     }
193
194     // 2. Cargar cuentas
195     await this.initializeAccounts();
196
197     // 3. Cargar TODAS las estrategias completas en cache
198     await this.loadAllStrategiesToCache();
199
200     // 4. Configurar listeners
201     this.setupListeners();
202
203     // 5. Cargar datos de reporte
204     await this.initializeReportData();
205 }
206
207 private subscribeToContextData() {
208     // Suscribirse a los datos del usuario
209     this.appContext.currentUser$.subscribe(user => {
210         this.user = user;
211     });
212
213     // Suscribirse a las cuentas del usuario
214     this.appContext.userAccounts$.subscribe(accounts => {
215         this.accountsData = accounts;
216     });
217
218     // Suscribirse a las estrategias del usuario
219     this.appContext.userStrategies$.subscribe(strategies => {
220         this.userStrategies = strategies;
221         this.filteredStrategies = strategies;
222         this.updateStrategyCard();
223     });
224
225     // Suscribirse a los estados de carga
226     this.appContext.isLoading$.subscribe/loading => {
227         this.loading = loading.strategies;
228     });
229
230     // Suscribirse a los errores
231     this.appContext.errors$.subscribe(errors => {
232         if (errors.strategies) {
233             console.error('Error en estrategias:', errors.strategies);
234         }
235     });

```

```

236     }
237
238     /**
239      * Inicializa los datos del usuario
240     */
241     private async initializeUserData(): Promise<void> {
242       return new Promise((resolve) => {
243         this.store.select(selectUser).subscribe({
244           next: (user) => {
245             this.user = user.user;
246             resolve();
247           },
248           error: (err) => {
249             console.error('Error fetching user data', err);
250             resolve();
251           },
252         });
253       });
254     }
255
256     /**
257      * Inicializa las cuentas del usuario
258     */
259     private async initializeAccounts(): Promise<void> {
260       if (this.user?.id) {
261         try {
262           const accounts = await this.authService.getUserAccounts(this.user.id);
263           this.accountsData = accounts || [];
264           await this.checkPlanLimitations();
265           this.fetchUserKey();
266         } catch (error) {
267           console.error('Error loading accounts:', error);
268         }
269       }
270     }
271
272     /**
273      * MÉTODO PRINCIPAL: Cargar TODAS las estrategias completas al cache
274      * Este método carga tanto el overview como la configuration de cada estrategia
275      * y las almacena en el cache para acceso rápido
276     */
277     private async loadAllStrategiesToCache(): Promise<void> {
278       if (!this.user?.id) return;
279
280       try {
281         // Limpiar cache anterior
282         this.strategyCacheService.clearCache();
283
284         // 1. Obtener todas las estrategias (overviews)
285         const allStrategies = await this.strategySvc.getUserStrategyViews(this.user.id);
286
287         if (!allStrategies || allStrategies.length === 0) {
288           this.userStrategies = [];
289           this.activeStrategy = null;
290           this.filteredStrategies = [];
291           return;
292         }
293
294         // 2. Para cada estrategia, cargar su configuración completa
295         const strategiesWithConfigs = await Promise.all(
296           allStrategies.map(async (strategy) => {
297             try {
298               // Obtener la configuración completa
299               const strategyData = await this.strategySvc.getStrategyView((strategy as
any).id);
300               if (strategyData && strategyData.configuration) {
301                 // Almacenar en cache usando el servicio
302                 this.strategyCacheService.setStrategy(
303                   (strategy as any).id,

```

```

306             strategyData.overview,
307             strategyData.configuration
308         );
309
310         return {
311             overview: strategyData.overview,
312             configuration: strategyData.configuration
313         };
314     } else {
315         return null;
316     }
317 } catch (error) {
318     console.error(`'L Error loading strategy ${strategy.name}:`, error);
319     return null;
320 }
321 )
322 );
323
324 // 3. Filtrar estrategias válidas y separar activa de inactivas
325 const validStrategies = strategiesWithConfigs.filter(s => s !== null);
326
327 this.activeStrategy = validStrategies.find(s => s.overview.status === true)?.overview
328 || null this.userStrategies = validStrategies.filter(s => s.overview.status !== true).map(s =>
329 s.overview);
330 filteredStrategies = [...this.userStrategies];
331
332 // 4. Cargar datos de las cards
333 await this.loadStrategyCardsData();
334
335 // 5. Actualizar strategy card si hay estrategia activa
336 if (this.activeStrategy) {
337     this.updateStrategyCardWithActiveStrategy();
338 }
339
340 // 6. Verificar limitaciones
341 this.checkStrategyLimitations();
342
343 } catch (error) {
344     console.error('L Error loading strategies to cache:', error);
345     this.userStrategies = [];
346     this.activeStrategy = null;
347     this.filteredStrategies = [];
348 }
349
350 /**
351 * Inicializa los datos de reporte
352 */
353 private async initializeReportData(): Promise<void> {
354     try {
355         this.getActualBalance();
356     } catch (error) {
357         console.error('Error loading report data:', error);
358     }
359 }
360
361 /**
362 * Configurar todos los listeners necesarios
363 */
364 private setupListeners(): void {
365     // Suscribirse a los datos del contexto
366     this.subscribeToContextData();
367
368     // Configurar listeners de reporte
369     this.listenReportData();
370
371     // Inicializar servicio global de actualización
372     if (this.user?.id) {
373         this.globalStrategyUpdater.updateAllStrategies(this.user.id);
374     }
375 }

```

```

376
377     ngOnDestroy(): void {
378         // Limpiar recursos si es necesario
379     }
380
381     fetchUserKey() {
382         if (this.user?.email && this.accountsData.length > 0) {
383             // Use the first account's credentials
384             const firstAccount = this.accountsData[0];
385             this.reportSvc
386                 .getUserKey(firstAccount.emailTradingAccount, firstAccount.brokerPassword,
387                 firstAccount.subscriber);
388             next: (key: string) => {
389                 this.store.dispatch(setUserKey({ userKey: key }));
390             },
391             error: (err) => {
392                 console.error('Error fetching user key:', err);
393                 this.store.dispatch(setUserKey({ userKey: '' }));
394             },
395         );
396     } else {
397         console.warn('No user email or accounts available for fetching user key');
398         this.store.dispatch(setUserKey({ userKey: '' }));
399     }
400 }
401
402     getActualBalance() {
403         // Usar el método optimizado que carga desde cache primero
404         this.loadConfigWithCachedBalance();
405     }
406
407     getUserData() {
408         this.store.select(selectUser).subscribe({
409             next: (user) => {
410                 this.user = user.user;
411             },
412             error: (err) => {
413                 console.error('Error fetching user data', err);
414             },
415         });
416     }
417
418 /**
419 * MÉTODO SIMPLIFICADO: Cargar configuración usando el cache
420 * Ya no hace peticiones a Firebase, solo lee del cache
421 */
422     async loadConfig(balance: number) {
423         this.loading = true;
424
425         try {
426             // Si hay estrategia activa, usar su configuración del cache
427             if (this.activeStrategy) {
428                 const cachedStrategy = this.strategyCacheService.getStrategy((this.activeStrategy as
429                     any).id);
430                 if (cachedStrategy) {
431                     // Actualizar balance en riskPerTrade
432                     const configWithBalance = {
433                         ...cachedStrategy.configuration,
434                         riskPerTrade: {
435                             ...cachedStrategy.configuration.riskPerTrade,
436                             balance: balance
437                         }
438                     };
439
440                     this.config = configWithBalance;
441                 } else {
442                     this.config = initialStrategyState;
443                 }
444             } else {
445                 // No hay estrategia activa, usar configuración inicial

```

```

446     this.config = {
447       ...initialStrategyState,
448       riskPerTrade: {
449         ...initialStrategyState.riskPerTrade,
450         balance: balance
451       }
452     };
453   }
454
455   await this.checkPlanLimitations();
456 } catch (error) {
457   console.error('Error loading config:', error);
458   this.config = initialStrategyState;
459 } finally {
460   this.loading = false;
461 }
462 }
463
464 /**
465 * MÉTODO OPTIMIZADO: Cargar configuración con balance desde cache
466 */
467 async loadConfigWithCachedBalance() {
468   this.loading = true;
469
470   try {
471     // Obtener balance desde cache
472     let balance = 0;
473     if (this.accountsData.length > 0) {
474       const firstAccount = this.accountsData[0];
475       balance = this.balanceCacheService.getBalance(firstAccount.accountID);
476     }
477
478     // Cargar configuración con balance del cache
479     await this.loadConfig(balance);
480
481     // Si necesita actualización, hacer petición en background
482     if (this.accountsData.length > 0) {
483       const firstAccount = this.accountsData[0];
484       if (this.balanceCacheService.needsUpdate(firstAccount.accountID)) {
485         this.updateBalanceInBackground(firstAccount);
486       }
487     }
488   } catch (error) {
489     console.error('Error loading config with cached balance:', error);
490     this.config = initialStrategyState;
491   } finally {
492     this.loading = false;
493   }
494 }
495
496 /**
497 * Actualizar balance en background sin bloquear la UI
498 */
499 private updateBalanceInBackground(account: AccountData) {
500   this.store.select(selectUserKey).pipe().subscribe({
501     next: (userKey) => {
502       if (userKey && userKey !== '') {
503         this.reportSvc.getBalanceData(account.accountID as string, userKey,
504         account.accountNumber).pipe().subscribe({
505           next: (balance) => {
506             // Actualizar cache con nuevo balance
507             this.balanceCacheService.setBalance(account.accountID, balance);
508             // Recargar configuración con nuevo balance
509             this.loadConfig(balance);
510           },
511           error: (err) => {
512             console.error('Error fetching balance data in background:', err);
513           },
514         });
515       }
516     },
517   });
518 }

```

```

516     });
517 }
518
519 /**
520 * MÉTODO PÚBLICO: Invalidar cache y recargar estrategias
521 * Se llama cuando hay cambios en las estrategias (crear, actualizar, eliminar)
522 */
523 public async invalidateCacheAndReload(): Promise<void> {
524     await this.loadAllStrategiesToCache();
525 }
526
527 /**
528 * MÉTODO PÚBLICO: Obtener estrategia del cache
529 * Para que edit-strategy pueda acceder a los datos sin hacer peticiones
530 */
531 public getStrategyFromCache(strategyId: string): { overview: ConfigurationOverview;
532 configuration: StrategyServiceStrategy; } {
533     return this.strategyCacheService.getStrategy(strategyId);
534 }
535
536 /**
537 * MÉTODO PÚBLICO: Verificar si el cache está cargado
538 */
539 public isCacheLoaded(): boolean {
540     return this.strategyCacheService.isCacheLoaded();
541 }
542
543 openEditPopup() {
544     this.router.navigate(['/edit-strategy']);
545 }
546
547 onSearchChange(event: Event) {
548     const value = (event.target as HTMLInputElement).value;
549     this.searchTerm = value;
550     this.filterStrategies();
551 }
552
553 filterStrategies() {
554     if (!this.searchTerm.trim()) {
555         this.filteredStrategies = [...this.userStrategies];
556     } else {
557         this.filteredStrategies = this.userStrategies.filter(strategy =>
558             strategy.name.toLowerCase().includes(this.searchTerm.toLowerCase())
559         );
560     }
561 }
562
563 // Obtener datos de card por ID de estrategia
564 getCardDataByStrategyId(strategyId: string): StrategyCardData | undefined {
565     return this.strategyCardsData.find(card => card.id === strategyId);
566 }
567
568 // Contar el total de estrategias del usuario (activas + inactivas) sin duplicar
569 // IMPORTANTE: Solo cuenta estrategias NO eliminadas (deleted !== true)
570 // Las estrategias eliminadas (deleted: true) NO se incluyen en este conteo
571 private getTotalStrategiesCount(): number {
572     const uniqueIds = new Set<string>();
573     this.userStrategies.forEach(s => uniqueIds.add((s as any).id));
574     if (this.activeStrategy) uniqueIds.add((this.activeStrategy as any).id);
575
576     const total = uniqueIds.size;
577
578     return total;
579 }
580
581 async onNewStrategy() {
582     if (!this.user?.id || this.isCreatingStrategy) return;
583
584     try {
585         // Activar loading
586         this.isCreatingStrategy = true;

```

```

586     // Contar el total de estrategias del usuario (activas + inactivas)
587     const totalStrategies = this.getTotalStrategiesCount();
588     const accessCheck = await
589     this.planLimitationsGuard.checkStrategyCreationWithModal(this.user.id, totalStrategies);
590
591     if (!accessCheck.canCreate) {
592         // Verificar si es el plan Pro con 8 estrategias (límite máximo)
593         const limitations = await
594         this.planLimitationsGuard.checkUserLimitations(this.user.id);
595         limitations.planName.toLowerCase().includes('pro') &&
596             limitations.maxStrategies === 8 &&
597             totalStrategies >= 8;
598
599         if (isProPlanWithMaxStrategies) {
600             // Para plan Pro con 8 estrategias: desactivar botón
601             this.isAddStrategyDisabled = true;
602             return;
603         } else {
604             // Para otros planes: redirigir a la página de cuenta
605             this.router.navigate(['/account'], {
606                 queryParams: { tab: 'plan' }
607             });
608             return;
609         }
610     }
611
612     // Verificar si es la primera estrategia del usuario
613     if (totalStrategies === 0) {
614         // Primera estrategia: mostrar modal de guía de estrategias
615         this.showStrategyGuide = true;
616     } else {
617         // Estrategias adicionales: crear automáticamente con nombre genérico
618         await this.createGenericStrategy();
619     }
620     } finally {
621         // Desactivar loading
622         this.isCreatingStrategy = false;
623     }
624 }
625
626 // Strategy Card Event Handlers
627 onStrategyEdit(strategyId: string) {
628     // Verificar si el usuario no ha marcado "don't show again"
629     const dontShowAgain = localStorage.getItem('strategy-guide-dont-show');
630
631     // Si no ha marcado "don't show again", mostrar el modal de guía
632     if (!dontShowAgain) {
633         this.showStrategyGuide = true;
634         return;
635     }
636
637     // Si ya marcó "don't show again", navegar directamente a edit-strategy
638     if (strategyId) {
639         this.router.navigate(['/edit-strategy'], { queryParams: { strategyId: strategyId } });
640     } else {
641         this.alertService.showWarning('No strategy found. Please create a strategy first.', 'No Strategy Found');
642     }
643
644     onStrategyFavorite(strategyId: string) {
645         // TODO: Implementar funcionalidad de favoritos en la base de datos
646         // Por ahora solo actualizar el estado local
647         const strategy = this.userStrategies.find(s => (s as any).id === strategyId);
648         if (strategy) {
649         }
650     }
651
652     onStrategyMoreOptions(strategyId: string) {
653         // TODO: Implementar menú de opciones (copiar, eliminar, etc.)
654         // Por ahora mostrar opciones básicas
655         const strategy = this.userStrategies.find(s => (s as any).id === strategyId);

```

```

656     if (strategy) {
657       const action = confirm(`Options for "${strategy.name}"`:\n\nOK - Copy strategy\nCancel
658 - Delete strategy`);
659       this.copyStrategy(strategyId);
660     } else {
661       this.deleteStrategy(strategyId);
662     }
663   }
664 }
665
666 onStrategyCustomize(strategyId: string) {
667   this.router.navigate(['/edit-strategy'], { queryParams: { strategyId: strategyId } });
668 }
669
670 // Plan detection and banner methods
671 fetchUserAccounts() {
672   if (this.user?.id) {
673     this.authService.getUserAccounts(this.user.id).then(async (accounts) => {
674       this.accountsData = accounts || [];
675       await this.checkPlanLimitations();
676       // After loading accounts, try to fetch user key
677       this.fetchUserKey();
678     });
679   }
680 }
681
682 listenReportData() {
683   // Listen to trade win percentage
684   this.store.select(selectTradeWin).subscribe((tradeWin) => {
685     this.tradeWin = tradeWin;
686     this.updateStrategyCard();
687   });
688
689   // Listen to total trades
690   this.store.select(selectTotalTrades).subscribe((totalTrades) => {
691     this.totalTrades = totalTrades;
692     this.updateStrategyCard();
693   });
694
695   // Listen to net PnL
696   this.store.select(selectNetPnL).subscribe((netPnL) => {
697     this.netPnL = netPnL;
698     this.updateStrategyCard();
699   });
700 }
701
702 updateStrategyCard() {
703   // Solo actualizar si hay una estrategia activa
704   if (this.activeStrategy) {
705     this.updateStrategyCardWithActiveStrategy();
706   }
707 }
708
709 countActiveRules(config: any): number {
710   let count = 0;
711   if (config.maxDailyTrades?.isActive) count++;
712   if (config.riskReward?.isActive) count++;
713   if (config.riskPerTrade?.isActive) count++;
714   if (config.daysAllowed?.isActive) count++;
715   if (config.hoursAllowed?.isActive) count++;
716   if (config.assetsAllowed?.isActive) count++;
717   return count;
718 }
719
720 getLastModifiedDate(): string {
721   // For now, return current date. In the future, this could come from a timestamp in the
722   config
723   const now = new Date();
724   const months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
725   'Nov', 'Dec'];
726   const month = months[now.getMonth()];
727   const day = now.getDate();

```

```

726     const year = now.getFullYear();
727     return `${month} ${day}, ${year}`;
728 }
729
730 // Check if the current plan allows multiple strategies
731 canCreateMultipleStrategies(): boolean {
732     // This will be determined by the plan limitations guard
733     return true; // Default to true, let the guard handle the actual validation
734 }
735
736 private getActiveStrategyCount(): number {
737     if (!this.config) return 0;
738
739     let count = 0;
740     Object.values(this.config).forEach(rule => {
741         if (rule && typeof rule === 'object' && 'isActive' in rule && rule.isActive) {
742             count++;
743         }
744     });
745     return count;
746 }
747
748 private async checkPlanLimitations() {
749     if (!this.user?.id) {
750         this.showPlanBanner = false;
751         return;
752     }
753
754     try {
755         // Get user's plan limitations from the guard
756         const limitations = await this.planLimitationsGuard.checkUserLimitations(this.user.id);
757         const totalStrategies = this.getTotalStrategiesCount();
758
759         this.showPlanBanner = false;
760         this.planBannerMessage = '';
761         this.planBannerType = 'info';
762
763         // If user needs subscription or is banned/cancelled
764         if (limitations.needsSubscription || limitations.isBanned || limitations.isCancelled) {
765             // Only show banner if user has trading accounts (not first-time user with plan)
766             if (this.accountsData.length > 0) {
767                 this.showPlanBanner = true;
768                 this.planBannerMessage = this.getBlockedMessage(limitations);
769                 this.planBannerType = 'warning';
770             }
771         }
772     }
773
774     // Check if user has reached strategy limit
775     if (totalStrategies >= limitations.maxStrategies) {
776         this.showPlanBanner = true;
777         this.planBannerMessage = `You've reached the strategy limit for your
778 ${limitations.planName} plan. Please upgrade to a higher plan and keep growing your account.`;
779     }
780     } catch (error) {
781         console.error('Error checking plan limitations:', error);
782         this.showPlanBanner = false;
783     }
784 }
785
786 private getBlockedMessage(limitations: any): string {
787     if (limitations.isBanned) {
788         return 'Your account has been banned. Please contact support for assistance.';
789     }
790
791     if (limitations.isCancelled) {
792         return 'Your subscription has been cancelled. Please purchase a plan to access this
793 functionality.';
794
795     if (limitations.needsSubscription) {

```

```

796     return 'You need to purchase a plan to access this functionality.';
797 }
798
799 return 'Access denied. Please contact support for assistance.';
800 }
801
802 onUpgradePlan() {
803   this.router.navigate(['/account'], {
804     queryParams: { tab: 'plan' }
805   });
806 }
807
808 onCloseBanner() {
809   this.showPlanBanner = false;
810 }
811
812 // Check strategy limitations and update button state
813 async checkStrategyLimitations() {
814   if (!this.user?.id) {
815     this.isAddStrategyDisabled = true;
816     return;
817   }
818
819   try {
820     // Contar el total de estrategias del usuario (activas + inactivas)
821     const totalStrategies = this.getTotalStrategiesCount();
822     const accessCheck = await
823     this.planLimitationsGuard.checkStrategyCreationWithModal(this.user.id, totalStrategies);
824     this.isAddStrategyDisabled = !accessCheck.canCreate;
825
826     // El banner se actualiza automáticamente en checkPlanLimitations()
827   } catch (error) {
828     console.error('Error checking strategy limitations:', error);
829     this.isAddStrategyDisabled = true;
830   }
831 }
832
833 // ===== MÉTODOS PARA MÚLTIPLES ESTRATEGIAS =====
834
835
836
837 // Cargar datos de las cards para todas las estrategias
838 async loadStrategyCardsData() {
839   this.strategyCardsData = [];
840
841   for (const strategy of this.userStrategies) {
842     try {
843       const cardData = await this.getStrategyCardData(strategy);
844       this.strategyCardsData.push(cardData);
845     } catch (error) {
846       // Agregar datos básicos en caso de error
847       this.strategyCardsData.push({
848         id: (strategy as any).id,
849         name: strategy.name,
850         status: strategy.status,
851         lastModified: this.formatDate(strategy.updated_at.toDate()),
852         rules: 0,
853         days_active: strategy.days_active || 0,
854         winRate: 0,
855         isFavorite: false,
856         created_at: strategy.created_at,
857         updated_at: strategy.updated_at,
858         userId: strategy.userId,
859         configurationId: strategy.configurationId
860       });
861     }
862   }
863 }
864
865 // Generar nombre único para estrategia

```

```

866     private generateUniqueStrategyName(baseName: string): string {
867         // Obtener todas las estrategias existentes (activas e inactivas)
868         const allStrategies = [
869             ...this.activeStrategy ? [this.activeStrategy] : [],
870             ...this.userStrategies
871         ];
872
873         // Extraer solo los nombres
874         const existingNames = allStrategies.map(strategy => strategy.name);
875
876         // Si el nombre base no existe, usarlo tal como está
877         if (!existingNames.includes(baseName)) {
878             return baseName;
879         }
880
881         // Si el nombre base termina con "copy", agregar número secuencial
882         if (baseName.toLowerCase().endsWith('copy')) {
883             let counter = 1;
884             let newName = `${baseName} ${counter}`;
885
886             while (existingNames.includes(newName)) {
887                 counter++;
888                 newName = `${baseName} ${counter}`;
889             }
890
891             return newName;
892         }
893
894         // Si el nombre base no termina con "copy", agregar "copy" primero
895         let copyName = `${baseName} copy`;
896
897         if (!existingNames.includes(copyName)) {
898             return copyName;
899         }
900
901         // Si "copy" ya existe, agregar número secuencial
902         let counter = 1;
903         let newName = `${baseName} copy ${counter}`;
904
905         while (existingNames.includes(newName)) {
906             counter++;
907             newName = `${baseName} copy ${counter}`;
908         }
909
910         return newName;
911     }
912
913
914
915     // Crear estrategia genérica automáticamente (para estrategias adicionales)
916     async createGenericStrategy() {
917         if (!this.user?.id) return;
918
919         try {
920             // 1. Primero recargar las strategies para tener el estado actualizado
921             await this.invalidateCacheAndReload();
922
923             // 2. Generar nombre único para la estrategia genérica
924             const genericName = this.generateUniqueStrategyName('Strategy');
925
926             // 3. Verificar si ya hay una estrategia activa
927             const hasActiveStrategy = this.activeStrategy !== null;
928
929             // 4. Verificar si es la primera estrategia del usuario
930             // NOTA: getTotalStrategiesCount() solo cuenta estrategias NO eliminadas (deleted !==
931             true) // Las estrategias eliminadas no cuentan para determinar si es la primera
932             const totalStrategies = this.getTotalStrategiesCount();
933             const isFirstStrategy = totalStrategies === 0;
934
935             // 5. Crear configuración vacía con reglas por defecto (todas inactivas)

```

```

936     const emptyStrategyConfig: StrategyState = {
937       maxDailyTrades: {
938         isActive: false,
939         maxDailyTrades: 0,
940         type: 'MAX DAILY TRADES' as any,
941       },
942       riskReward: {
943         isActive: false,
944         riskRewardRatio: '1:2',
945         type: 'RISK REWARD RATIO' as any,
946       },
947       riskPerTrade: {
948         isActive: false,
949         review_type: 'MAX',
950         number_type: 'PERCENTAGE',
951         percentage_type: 'NULL',
952         risk_ammount: 0,
953         type: 'MAX RISK PER TRADE' as any,
954         balance: 0,
955         actualBalance: 0,
956       },
957       daysAllowed: {
958         isActive: false,
959         type: 'DAYS ALLOWED' as any,
960         tradingDays: [],
961       },
962       hoursAllowed: {
963         isActive: false,
964         tradingOpenTime: '',
965         tradingCloseTime: '',
966         timezone: '',
967         type: 'TRADING HOURS' as any,
968       },
969       assetsAllowed: {
970         isActive: false,
971         type: 'ASSETS ALLOWED' as any,
972         assetsAllowed: [],
973       },
974     };
975
976     // 6. Crear la nueva estrategia genérica
977     // La primera estrategia siempre es activa, las adicionales siempre inactivas
978     const strategyId = await this.strategySvc.createStrategyView(
979       this.user.id,
980       genericName,
981       emptyStrategyConfig,
982       isFirstStrategy ? true : false // Primera estrategia activa, el resto inactivas
983     );
984
985     // 6.1. Si NO es la primera estrategia, agregar dateInactive inmediatamente
986     // Esto evita problemas en los reports al tener estrategias con dateActive sin
987     dateInactive
988     if (!isFirstStrategy) {
989       const inactiveTime = new Date(Date.now() + 2000); // +2 segundos desde ahora
990
991       // Solo agregar dateInactive para cerrar el ciclo de activación/desactivación
992       await this.strategySvc.updateStrategyDates(
993         this.user.id,
994         strategyId,
995         undefined, // No agregar dateActive (ya existe uno)
996         inactiveTime // Agregar dateInactive en 2 segundos
997       );
998     }
999
1000    // 7. Actualizar el estado del plan en tiempo real después de crear
1001    await this.checkPlanLimitations();
1002
1003    // 8. Redirigir directamente a edit-strategy con la nueva estrategia
1004    this.router.navigate(['/edit-strategy'], { queryParams: { strategyId: strategyId } });
1005  } catch (error) {

```

```

1006     console.error('Error creating generic strategy:', error);
1007     this.alertService.showError('Error creating strategy. Please try again.', 'Strategy
1008 Creation Error');
1009 }
1010
1011 // Activar una estrategia
1012 async activateStrategy(strategyId: string) {
1013     if (!this.user?.id) return;
1014
1015     try {
1016         // Mostrar loading completo durante la activación
1017         this.isProcessingStrategy = true;
1018
1019         const currentTimestamp = new Date();
1020
1021         // Verificar si hay una estrategia activa actualmente
1022         if (this.activeStrategy && (this.activeStrategy as any).id !== strategyId) {
1023             // Hay una estrategia activa diferente, desactivarla primero
1024             await this.strategySvc.updateStrategyDates(
1025                 this.user.id,
1026                 (this.activeStrategy as any).id,
1027                 undefined, // No agregar a dateActive
1028                 currentTimestamp // Agregar timestamp actual a dateInactive
1029             );
1030         }
1031
1032         // Activar la nueva estrategia
1033         await this.strategySvc.updateStrategyDates(
1034             this.user.id,
1035             strategyId,
1036             currentTimestamp, // Agregar timestamp actual a dateActive
1037             undefined // No agregar a dateInactive
1038         );
1039
1040         // Invalidar cache y recargar estrategias
1041         await this.invalidateCacheAndReload();
1042
1043         // Recargar configuración con la nueva estrategia activa
1044         // Obtener balance desde la configuración actual
1045         const currentConfig = this.config;
1046         const balance = currentConfig?.riskPerTrade?.balance || 0;
1047         this.loadConfig(balance);
1048     } catch (error) {
1049         console.error('Error activating strategy:', error);
1050         this.alertService.showError('Error activating strategy. Please try again.', 'Strategy
1051 Activation Error');
1052         // Ocultar loading al finalizar
1053         this.isProcessingStrategy = false;
1054     }
1055 }
1056
1057 // Eliminar estrategia
1058 deleteStrategy(strategyId: string) {
1059     // Guardar el ID de la estrategia a eliminar y mostrar el popup de confirmación
1060     this.strategyToDeleteId = strategyId;
1061     this.showDeleteConfirmPopup = true;
1062 }
1063
1064 // Confirmar eliminación de estrategia (marcar como deleted)
1065 confirmDeleteStrategy = async () => {
1066     this.showDeleteConfirmPopup = false;
1067
1068     try {
1069         // Mostrar loading completo durante la eliminación
1070         this.isProcessingStrategy = true;
1071
1072         // Marcar la estrategia como deleted en lugar de borrarla
1073         await this.strategySvc.markStrategyAsDeleted(this.strategyToDeleteId);
1074
1075         // Invalidar cache y recargar estrategias

```

```

1076     await this.invalidateCacheAndReload();
1077
1078     // Actualizar el estado del plan en tiempo real después de eliminar
1079     await this.checkPlanLimitations();
1080
1081     // Verificar si se debe reactivar el botón (para plan Pro que ya no está en el límite
1082     // máximo)
1083     if (this.user?.id) {
1084         const limitations = await
1085         this.planLimitationsForUser();
1086         const isProPlan = limitations.planName.toLowerCase().includes('pro') &&
1087         limitations.maxStrategies === 8;
1088         if (isProPlan && currentTotalStrategies < 8) {
1089             // Reactivar el botón si ya no está en el límite máximo
1090             this.isAddStrategyDisabled = false;
1091         }
1092     }
1093
1094     // Si se eliminó la estrategia activa, cargar la primera disponible o estado inicial
1095     if (!this.activeStrategy) {
1096         if (this.userStrategies.length > 0) {
1097             await this.activateStrategy((this.userStrategies[0] as any).id);
1098         } else {
1099             this.config = initialStrategyState;
1100         }
1101     }
1102     } catch (error) {
1103         this.alertService.showError('Error marking strategy as deleted. Please try again.', 'Strategy Deletion Error');
1104         // Ocultar loading al finalizar
1105         this.isProcessingStrategy = false;
1106         this.strategyToDeleteId = '';
1107     };
1108 }
1109
1110 // Cancelar eliminación de estrategia
1111 cancelDeleteStrategy = () => {
1112     this.showDeleteConfirmPopup = false;
1113     this.strategyToDeleteId = '';
1114 };
1115
1116 // Actualizar nombre de estrategia
1117 async updateStrategyName(strategyId: string, newName: string) {
1118     if (!newName.trim()) {
1119         this.alertService.showWarning('Please enter a valid strategy name', 'Invalid Strategy Name');
1120         return;
1121     }
1122
1123     try {
1124         await this.strategySvc.updateStrategyView(strategyId, { name: newName.trim() });
1125
1126         // Invalidar cache y recargar estrategias
1127         await this.invalidateCacheAndReload();
1128     } catch (error) {
1129         this.alertService.showError('Error updating strategy name. Please try again.', 'Strategy Name Update Error');
1130     }
1131 }
1132
1133 // Actualizar strategy card con la estrategia activa
1134 async updateStrategyCardWithActiveStrategy() {
1135     if (!this.activeStrategy || !this.user?.id) return;
1136
1137     try {
1138         // Obtener la estrategia completa (configurations + configuration-overview)
1139         const strategyData = await this.strategySvc.getStrategyView((this.activeStrategy as any).id);
1140         if (!strategyData) {
1141             console.error('No strategy data found for active strategy');
1142             return;
1143         }
1144     }
1145

```

```

1146     // Calcular win rate (por ahora 0, necesitamos implementar la lógica)
1147     const winRate = 0; // TODO: Implementar cálculo de win rate
1148
1149     // Contar reglas activas de la configuración
1150     const activeRules = strategyData.configuration ?
1151       this.countActiveRules(strategyData.configuration) : 0;
1152     // Formatear fecha de actualización
1153     const lastModified = this.formatDate(strategyData.overview.updated_at.toDate());
1154
1155     this.strategyCard = {
1156       id: (this.activeStrategy as any).id,
1157       name: strategyData.overview.name, // Nombre de la estrategia desde overview
1158       status: strategyData.overview.status, // Estado desde overview
1159       lastModified: lastModified, // updated_at formateado desde overview
1160       rules: activeRules, // Reglas activas de configuration
1161       days_active: strategyData.overview.days_active || 0, // days_active desde overview
1162       winRate: winRate, // Win rate (por implementar)
1163       isFavorite: this.strategyCard.isFavorite,
1164       created_at: strategyData.overview.created_at,
1165       updated_at: strategyData.overview.updated_at,
1166       userId: strategyData.overview.userId,
1167       configurationId: strategyData.overview.configurationId
1168     };
1169   } catch (error) {
1170     // Error silencioso
1171   }
1172 }
1173
1174 // Formatear fecha
1175 formatDate(date: Date): string {
1176   const months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
1177   'Nov', 'Dec'];
1178   const month = months[date.getMonth()];
1179   const day = date.getDate();
1180   const year = date.getFullYear();
1181   return `${month} ${day}, ${year}`;
1182 }
1183
1184
1185
1186 // Obtener ID de la estrategia para el track
1187 getStrategyId(strategy: ConfigurationOverview): string {
1188   return (strategy as any).id;
1189 }
1190
1191 // Convertir ConfigurationOverview a StrategyCardData
1192 async getStrategyCardData(strategy: ConfigurationOverview): Promise<StrategyCardData> {
1193   try {
1194     // Obtener la estrategia completa (configurations + configuration-overview)
1195     const strategyData = await this.strategySvc.getStrategyView((strategy as any).id);
1196
1197     if (!strategyData) {
1198       // Retornar datos básicos en caso de error
1199       return {
1200         id: (strategy as any).id,
1201         name: strategy.name,
1202         status: strategy.status,
1203         lastModified: this.formatDate(strategy.updated_at.toDate()),
1204         rules: 0,
1205         days_active: strategy.days_active || 0,
1206         winRate: 0,
1207         isFavorite: false,
1208         created_at: strategy.created_at,
1209         updated_at: strategy.updated_at,
1210         userId: strategy.userId,
1211         configurationId: strategy.configurationId
1212       };
1213     }
1214
1215     // Calcular win rate (por ahora 0, necesitamos implementar la lógica)

```

```

1216     const winRate = 0; // TODO: Implementar cálculo de win rate
1217
1218     // Contar reglas activas de la configuración
1219     const activeRules = strategyData.configuration ?
1220       this.countActiveRules(strategyData.configuration) : 0;
1221
1222     // Formatear fecha de actualización
1223     const lastModified = this.formatDate(strategyData.overview.updated_at.toDate());
1224
1225     const cardData: StrategyCardData = {
1226       id: (strategy as any).id,
1227       name: strategyData.overview.name, // Nombre desde overview
1228       status: strategyData.overview.status, // Estado desde overview
1229       lastModified: lastModified, // updated_at desde overview
1230       rules: activeRules, // Reglas activas de configuration
1231       days_active: strategyData.overview.days_active || 0, // days_active desde overview
1232       winRate: winRate, // Win rate (por implementar)
1233       isFavorite: false,
1234       created_at: strategyData.overview.created_at,
1235       updated_at: strategyData.overview.updated_at,
1236       userId: strategyData.overview.userId,
1237       configurationId: strategyData.overview.configurationId
1238     };
1239
1240     return cardData;
1241   } catch (error) {
1242     // Retornar datos básicos en caso de error
1243     return {
1244       id: (strategy as any).id,
1245       name: strategy.name,
1246       status: strategy.status,
1247       lastModified: this.formatDate(strategy.updated_at.toDate()),
1248       rules: 0,
1249       days_active: strategy.days_active || 0,
1250       winRate: 0,
1251       isFavorite: false,
1252       created_at: strategy.created_at,
1253       updated_at: strategy.updated_at,
1254       userId: strategy.userId,
1255       configurationId: strategy.configurationId
1256     };
1257   }
1258 }
1259
1260 // Copiar estrategia
1261 async copyStrategy(strategyId: string) {
1262   if (!this.user?.id) return;
1263
1264   try {
1265     // Mostrar loading completo durante la copia
1266     this.isProcessingStrategy = true;
1267
1268     // Verificar límites del plan antes de duplicar
1269     const totalStrategies = this.getTotalStrategiesCount();
1270     const accessCheck = await
1271       this.planLimitationsGuard.checkStrategyCreationWithModal(this.user.id, totalStrategies);
1272     if (!accessCheck.canCreate) {
1273       // Verificar si es el plan Pro con 8 estrategias (límite máximo)
1274       const limitations = await
1275         this.planLimitationsGuard.checkUserLimitations(this.user.id);
1276       limitations.planName.toLowerCase().includes('pro') && limitations.maxStrategies === 8 &&
1277           totalStrategies >= 8;
1278
1279       if (isProPlanWithMaxStrategies) {
1280         // Para plan Pro con 8 estrategias: desactivar botón y mostrar mensaje
1281         this.isAddStrategyDisabled = true;
1282         this.alertService.showWarning('You have reached the maximum number of strategies
1283 (8) for your Pro plan.', 'Strategy Limit Reached');
1284       } else {
1285         // Para otros planes: redirigir a la página de cuenta
1286         this.router.navigate(['/account'], {

```

```

1286         queryParams: { tab: 'plan' }
1287     });
1288     return;
1289 }
1290 }
1291
1292 // Buscar en estrategias inactivas primero
1293 let strategy = this.userStrategies.find(s => (s as any).id === strategyId);
1294
1295 // Si no se encuentra en inactivas, buscar en la estrategia activa
1296 if (!strategy && this.activeStrategy && (this.activeStrategy as any).id ===
1297 strategyId) {
1298     strategy = this.activeStrategy;
1299 }
1300
1301 if (!strategy) {
1302     console.error('Strategy not found');
1303     return;
1304 }
1305
1306 // Determinar el nombre de la copia usando la lógica de nombres únicos
1307 const isActiveStrategy = this.activeStrategy && (this.activeStrategy as any).id ===
1308 strategyId;
1309 const baseName = strategy.name;
1310 const newName = this.generateUniqueStrategyName(baseName);
1311
1312 // Obtener la estrategia completa (configuration-overview + configurations)
1313 const strategyData = await this.strategySvc.getStrategyView((strategy as any).id);
1314 if (!strategyData || !strategyData.configuration) {
1315     console.error('Strategy configuration not found');
1316     return;
1317 }
1318
1319 // Crear configuración con los campos requeridos para la copia
1320 const strategyConfig: StrategyState = {
1321     ...strategyData.configuration
1322 };
1323
1324 // Si es la estrategia activa, crear la copia como inactiva
1325 const newStrategyId = await this.strategySvc.createStrategyView(
1326     this.user.id,
1327     newName,
1328     strategyConfig,
1329     isActiveStrategy ? false : undefined // false = inactiva, undefined = mantener
1330     estadoOriginal
1331 );
1332
1333 // Si se está copiando una estrategia activa (isActiveStrategy = true),
1334 // agregar dateInactive inmediatamente para cerrar el ciclo de activación
1335 if (isActiveStrategy) {
1336     const inactiveTime = new Date(Date.now() + 2000); // +2 segundos desde ahora
1337
1338     // Solo agregar dateInactive para cerrar el ciclo de activación/desactivación
1339     await this.strategySvc.updateStrategyDates(
1340         this.user.id,
1341         newStrategyId,
1342         undefined, // No agregar dateActive (ya existe uno)
1343         inactiveTime // Agregar dateInactive en 2 segundos
1344     );
1345 }
1346
1347 // Invalidar cache y recargar estrategias
1348 await this.invalidateCacheAndReload();
1349
1350 // Actualizar el estado del plan en tiempo real después de copiar
1351 await this.checkPlanLimitations();
1352
1353 // Verificar si se debe reactivar el botón (para plan Pro que ya no está en el límite
1354 // máximo)
1355 if (this.user?.id) {
1356     const limitations = await
1357     this.planLimitationsGuard.checkUserLimitations(this.user.id.length + (this.activeStrategy as any).length +
1358     1 : 0);
1359     const isProPlan = limitations.planName.toLowerCase().includes('pro') &&
1360     limitations.maxStrategies === 8;

```

```

1356     if (isProPlan && currentTotalStrategies < 8) {
1357         // Reactivar el botón si ya no está en el límite máximo
1358         this.isAddStrategyDisabled = false;
1359     }
1360 }
1361
1362 } catch (error) {
1363     this.alertService.showError('Error copying strategy. Please try again.', 'Strategy
1364 Copy Error');
1365     // Ocultar loading al finalizar
1366     this.isProcessingStrategy = false;
1367 }
1368 }
1369
1370 // Navegar a trading accounts
1371 navigateToTradingAccounts() {
1372     const accountsCount = this.accountsData?.length || 0;
1373     if (accountsCount >= 8) {
1374         return; // botón ya estará deshabilitado; no hacer nada
1375     }
1376     // Redirigir a Plan Management en Account
1377     this.router.navigate(['/account'], { queryParams: { tab: 'plan' } });
1378 }
1379
1380 // Strategy guide modal methods
1381 private checkShowStrategyGuide(): void {
1382     const dontShowAgain = localStorage.getItem('strategy-guide-dont-show');
1383     if (!dontShowAgain) {
1384         this.showStrategyGuide = true;
1385     }
1386 }
1387
1388 onCloseStrategyGuide(): void {
1389     this.showStrategyGuide = false;
1390 }
1391
1392 onDontShowStrategyGuideAgain(): void {
1393     localStorage.setItem('strategy-guide-dont-show', 'true');
1394     this.showStrategyGuide = false;
1395 }
1396
1397 async onEditStrategyFromGuide(): Promise<void> {
1398     // Cerrar el modal de guía
1399     this.showStrategyGuide = false;
1400
1401     // Activar loading
1402     this.isCreatingStrategy = true;
1403
1404     try {
1405         // Crear automáticamente la primera estrategia genérica
1406         await this.createGenericStrategy();
1407     } finally {
1408         // Desactivar loading
1409         this.isCreatingStrategy = false;
1410     }
1411 }
1412 }
1413

```

Ø=ÜÁ features\strategy\components\assets-allowed

Ø=ÜÁ features\strategy\components\assets-allowed\assets-allowed.component.ts

```

1 import { Component, HostListener, OnInit } from '@angular/core';
2 import { Store } from '@ngrx/store';
3 import { SettingsService } from '../../../../../service/strategy.service';
4 import {
5   assetsAllowed,
6   daysAllowed,
7   selectMaxDailyTrades,
8 } from '../../../../../store/strategy.selectors';
9 import {
10   AssetsAllowedConfig,
11   Days,
12   DaysAllowedConfig,
13   MaxDailyTradesConfig,
14   RuleType,
15 } from '../../../../../models/strategy.model';
16 import {
17   setAssetsAllowedConfig,
18   setDaysAllowedConfig,
19   setMaxDailyTradesConfig,
20 } from '../../../../../store/strategy.actions';
21 import { CommonModule } from '@angular/common';
22
23 /**
24 * Component for configuring the assets allowed for trading rule.
25 *
26 * This component allows users to select which trading instruments/assets are
27 * permitted for trading. It includes a searchable dropdown for selecting
28 * instruments and displays selected assets as removable chips.
29 *
30 * Features:
31 * - Toggle rule active/inactive
32 * - Searchable dropdown for instrument selection
33 * - Display selected assets as chips
34 * - Remove asset functionality
35 * - Syncs with NgRx store
36 *
37 * Relations:
38 * - Store (NgRx): Reads and updates assetsAllowed configuration
39 * - SettingsService: Strategy service (injected but not directly used)
40 *
41 * @component
42 * @selector app-assets-allowed
43 * @standalone true
44 */
45 @Component({
46   selector: 'app-assets-allowed',
47   templateUrl: './assets-allowed.component.html',
48   styleUrls: ['./assets-allowed.component.scss'],
49   imports: [CommonModule],
50   standalone: true,
51 })
52 export class AssetsAllowedComponent implements OnInit {
53   config: AssetsAllowedConfig = {
54     isActive: false,
55     type: RuleType.ASSETS_ALLOWED,
56     assetsAllowed: ['XMRUSD', 'BTCUSD'],
57   };
58
59   symbols: string[] = [];
60   availableSymbolsOptions: string[] = [];
61   selectedInstrument: string | undefined = undefined;
62   searchTerm: string = '';
63
64   dropdownOpen = false;
65
66   constructor(private store: Store, private settingsService: SettingsService) {}
67
68   closeDropdown() {
69     this.dropdownOpen = false;
70   }

```

```

71  toggleDropdown() {
72    if (this.config.isActive) {
73      this.dropdownOpen = !this.dropdownOpen;
74    } else {
75      this.dropdownOpen = false;
76    }
77  }
78}
79
80 onSearchInput(event: Event) {
81   this.searchTerm = (event.target as HTMLInputElement).value;
82   this.dropdownOpen = true;
83 }
84
85 onSearchFocus() {
86   if (this.config.isActive) {
87     this.dropdownOpen = true;
88   }
89 }
90
91 onSearchBlur() {
92   // Delay para permitir click en dropdown
93   setTimeout(() => {
94     this.dropdownOpen = false;
95   }, 200);
96 }
97
98 selectInstrument(instrument: string) {
99   this.selectedInstrument = instrument;
100  this.addSymbol(instrument);
101  this.dropdownOpen = false;
102  this.selectedInstrument = undefined;
103  this.searchTerm = ''; // Limpiar búsqueda
104 }
105
106 getFilteredSymbols(): string[] {
107   if (!this.searchTerm) {
108     return this.availableSymbolsOptions;
109   }
110   return this.availableSymbolsOptions.filter(symbol =>
111     symbol.toLowerCase().includes(this.searchTerm.toLowerCase())
112   );
113 }
114
115 ngOnInit(): void {
116   this.listenRuleConfiguration();
117 }
118
119 addSymbol(symbol: string) {
120   if (symbol && !this.symbols.includes(symbol)) {
121     this.symbols = [...this.symbols, symbol];
122   }
123   this.updateConfig({
124     ...this.config,
125     assetsAllowed: this.symbols,
126   });
127 }
128
129 removeSymbol(symbol: string) {
130   if (this.config.isActive) {
131     this.symbols = this.symbols.filter((s) => s !== symbol);
132     this.updateConfig({
133       ...this.config,
134       assetsAllowed: this.symbols,
135     });
136   }
137 }
138
139 onToggleActive(event: Event) {
140   const isActive = (event.target as HTMLInputElement).checked;

```

```

141     const newConfig = {
142       ...this.config,
143       isActive: isActive,
144       // Reiniciar assets cuando se desactiva
145       assetsAllowed: isActive ? this.config.assetsAllowed : [],
146     };
147
148     // Reiniciar símbolos seleccionados
149     if (!isActive) {
150       this.symbols = [];
151     }
152
153     this.updateConfig(newConfig);
154   }
155
156   listenRuleConfiguration() {
157     this.store
158       .select(assetsAllowed)
159       .pipe()
160       .subscribe((config) => {
161         this.config = { ...config };
162         this.symbols = this.config.assetsAllowed;
163       });
164   }
165
166   private updateConfig(config: AssetsAllowedConfig) {
167     this.store.dispatch(setAssetsAllowedConfig({ config }));
168   }
169 }
170

```

Ø=ÜÁ features\strategy\components\days-allowed

Ø=ÜÄ features\strategy\components\days-allowed\days-allowed.component.ts

```

1 import { Component, OnInit } from '@angular/core';
2 import { Store } from '@ngrx/store';
3 import { SettingsService } from '../../../../../service/strategy.service';
4 import {
5   daysAllowed,
6   selectMaxDailyTrades,
7 } from '../../../../../store/strategy.selectors';
8 import {
9   Days,
10  DaysAllowedConfig,
11  MaxDailyTradesConfig,
12  RuleType,
13 } from '../../../../../models/strategy.model';
14 import {
15   setDaysAllowedConfig,
16   setMaxDailyTradesConfig,
17 } from '../../../../../store/strategy.actions';
18 import { CommonModule } from '@angular/common';
19
20 /**
21  * Component for configuring the days allowed for trading rule.
22  *
23  * This component allows users to select which days of the week trading is
24  * permitted. It displays buttons for each day (Monday through Sunday) that
25  * can be toggled on/off.
26  *
27  * Features:
28  * - Toggle rule active/inactive

```

```

29 * - Day buttons for each day of the week
30 * - Visual feedback for selected days
31 * - Syncs with NgRx store
32 *
33 * Relations:
34 * - Store (NgRx): Reads and updates daysAllowed configuration
35 * - SettingsService: Strategy service (injected but not directly used)
36 *
37 * @component
38 * @selector app-days-allowed
39 * @standalone true
40 */
41 @Component({
42   selector: 'app-days-allowed',
43   templateUrl: './days-allowed.component.html',
44   styleUrls: ['./days-allowed.component.scss'],
45   imports: [CommonModule],
46   standalone: true,
47 })
48 export class DaysAllowedComponent implements OnInit {
49   config: DaysAllowedConfig = {
50     isActive: false,
51     type: RuleType.DAYS_ALLOWED,
52     tradingDays: [],
53   };
54
55   daysButtons = [
56     { day: Days.MONDAY, isActive: false },
57     { day: Days.TUESDAY, isActive: false },
58     { day: Days.WEDNESDAY, isActive: false },
59     { day: Days.THURSDAY, isActive: false },
60     { day: Days.FRIDAY, isActive: false },
61     { day: Days.SATURDAY, isActive: false },
62     { day: Days.SUNDAY, isActive: false },
63   ];
64
65   constructor(private store: Store, private settingsService: SettingsService) {}
66
67   ngOnInit(): void {
68     this.listenRuleConfiguration();
69   }
70   onToggleActive(event: Event) {
71     const isActive = (event.target as HTMLInputElement).checked;
72     const newConfig = {
73       ...this.config,
74       isActive: isActive,
75       // Reiniciar días cuando se desactiva
76       tradingDays: isActive ? this.config.tradingDays : [],
77     };
78
79     // Reiniciar botones de días
80     if (!isActive) {
81       this.daysButtons.forEach(day => {
82         day.isActive = false;
83       });
84     }
85
86     this.updateConfig(newConfig);
87   }
88
89   onChangeValue(day: { day: Days; isActive: boolean }) {
90     if (this.config.isActive) {
91       this.daysButtons.forEach((d) => {
92         if (d.day === day.day) {
93           d.isActive = !d.isActive;
94         }
95       });
96
97       const newConfig: DaysAllowedConfig = {
98         ...this.config,

```

```

99         tradingDays: this.transformDaysActive(),
100     );
101     this.updateConfig(newConfig);
102 }
103 }
104
105 transformDaysActive(): string[] {
106     let daysArr: string[] = [];
107
108     this.daysButtons.forEach((d) => {
109         if (d.isActive) {
110             daysArr.push(d.day);
111         }
112     });
113
114     return daysArr;
115 }
116
117 listenRuleConfiguration() {
118     this.store
119         .select(daysAllowed)
120         .pipe()
121         .subscribe((config) => {
122             this.config = config;
123             this.daysButtons.forEach((dayOption) => {
124                 const findedDay = config.tradingDays.find((d) => d === dayOption.day);
125                 if (findedDay) {
126                     dayOption.isActive = true;
127                 }
128             });
129         });
130 }
131
132 private updateConfig(config: DaysAllowedConfig) {
133     this.store.dispatch(setDaysAllowedConfig({ config }));
134 }
135 }
136

```

Ø=ÜÁ features\strategy\components\hours-allowed

Ø=ÜÄ features\strategy\components\hours-allowed\hours-allowed.component.ts

```

1 import { Component, OnInit } from '@angular/core';
2 import { Store } from '@ngrx/store';
3 import { SettingsService } from '../../../../../service/strategy.service';
4 import {
5     hoursAllowed,
6     selectMaxDailyTrades,
7 } from '../../../../../store/strategy.selectors';
8 import {
9     HoursAllowedConfig,
10    MaxDailyTradesConfig,
11    RuleType,
12 } from '../../../../../models/strategy.model';
13 import {
14     setHoursAllowedConfig,
15     setMaxDailyTradesConfig,
16 } from '../../../../../store/strategy.actions';
17 import { CommonModule } from '@angular/common';
18 import { FormsModule } from '@angular/forms';
19 import { NgxMaterialTimepickerModule } from 'ngx-material-timepicker';
20 import * as moment from 'moment-timezone';

```

```

21 import { AlertService } from '../../../../../shared/services/alert.service';
22 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
23
24 /**
25 * Component for configuring the trading hours allowed rule.
26 *
27 * This component allows users to set the time window during which trading is
28 * permitted. It includes time pickers for opening and closing times, and a
29 * timezone selector with all available timezones.
30 *
31 * Features:
32 * - Toggle rule active/inactive
33 * - Time pickers for opening and closing times
34 * - Timezone selector with GMT offsets
35 * - Validation: opening time must be before closing time
36 * - Minimum 30-minute difference between times
37 *
38 * Relations:
39 * - Store (NgRx): Reads and updates hoursAllowed configuration
40 * - NgxMaterialTimepickerModule: Time picker UI
41 * - AlertService: Shows validation warnings
42 *
43 * @component
44 * @selector app-hours-allowed
45 * @standalone true
46 */
47 @Component({
48   selector: 'app-hours-allowed',
49   templateUrl: './hours-allowed.component.html',
50   styleUrls: ['./hours-allowed.component.scss'],
51   imports: [CommonModule, FormsModule, NgxMaterialTimepickerModule],
52   standalone: true,
53 })
54 export class HoursAllowedComponent implements OnInit {
55   config: HoursAllowedConfig = {
56     isActive: false,
57     tradingOpenTime: '09:30',
58     tradingCloseTime: '17:00',
59     timezone: 'Zulu',
60     type: RuleType.TRADING_HOURS,
61   };
62
63   timezones = Array.from(
64     new Map(
65       moment.tz.names().map((tz) => {
66         const offset = moment.tz(tz).utcOffset();
67         const offsetSign = offset >= 0 ? '+' : '-';
68         const absOffset = Math.abs(offset);
69         const hours = Math.floor(absOffset / 60);
70         const mins = absOffset % 60;
71
72         const formattedOffset = `(GMT${offsetSign}${hours
73           .toString()
74           .padStart(2, '0')}:${mins.toString().padStart(2, '0')})`;
75
76         const abbreviation = moment.tz(tz).zoneAbbr();
77         const key = `${abbreviation} ${formattedOffset}`;
78
79         return [key, { value: tz, label: key, offsetMinutes: offset }] as [
80           string,
81           { value: string; label: string; offsetMinutes: number }
82         ];
83       })
84     ).values()
85   )
86   .sort((a, b) => a.offsetMinutes - b.offsetMinutes)
87   .map(({ value, label }) => ({ value, label }));
88
89   constructor(private store: Store, private settingsService: SettingsService, private
90   alertService: AlertService) {}

```

```

91  ngOnInit(): void {
92    this.listenRuleConfiguration();
93  }
94
95  onToggleActive(event: Event) {
96    const isActive = (event.target as HTMLInputElement).checked;
97    const newConfig = {
98      ...this.config,
99      isActive: isActive,
100     // Reiniciar valores cuando se desactiva
101    tradingOpenTime: isActive ? this.config.tradingOpenTime : '09:30',
102    tradingCloseTime: isActive ? this.config.tradingCloseTime : '17:00',
103    timezone: isActive ? this.config.timezone : 'UTC',
104  };
105  this.updateConfig(newConfig);
106}
107onTimezoneChange(newTz: string) {
108  // Validar que la timezone sea válida
109  if (this.isValidTimezone(newTz)) {
110    const newConfig = { ...this.config, timezone: newTz };
111    this.updateConfig(newConfig);
112  } else {
113    console.warn('Invalid timezone selected:', newTz);
114  }
115}
116
117isValidTimezone(timezone: string): boolean {
118  return this.timezones.some(tz => tz.value === timezone);
119}
120
121onTimeChange(field: 'tradingOpenTime' | 'tradingCloseTime', value: string) {
122  const tempConfig = { ...this.config, [field]: value };
123  const openMinutes = this.toMinutes(tempConfig.tradingOpenTime);
124  const closeMinutes = this.toMinutes(tempConfig.tradingCloseTime);
125  if (openMinutes >= closeMinutes) {
126    this.alertService.showWarning('Opening time must be earlier than closing time.', 'Invalid Time Range');
127  }
128  if (closeMinutes - openMinutes < 30) {
129    this.alertService.showWarning('There must be at least a 30-minute difference between opening and closing times.', 'Minimum Time Difference');
130  }
131  this.updateConfig(tempConfig);
132}
133
134
135listenRuleConfiguration() {
136  this.store
137    .select(hoursAllowed)
138    .pipe()
139    .subscribe((config) => {
140      this.config = { ...config };
141    });
142}
143
144
145  private updateConfig(config: HoursAllowedConfig) {
146    this.store.dispatch(setHoursAllowedConfig({ config }));
147  }
148
149  private toMinutes(time: string): number {
150    const is12hFormat =
151      time.toUpperCase().includes('AM') || time.toUpperCase().includes('PM');
152
153    let hours: number;
154    let minutes: number;
155
156    if (is12hFormat) {
157      const [timePart, period] = time.split(' ');
158      [hours, minutes] = timePart.split(':').map(Number);
159      if (period.toUpperCase() === 'PM' && hours !== 12) {
160        hours += 12;
161      }
162    }
163  }

```

```

161     }
162     if (period.toUpperCase() === 'AM' && hours === 12) {
163       hours = 0;
164     }
165   } else {
166   [hours, minutes] = time.split(':').map(Number);
167 }
168
169   return hours * 60 + minutes;
170 }
171
172 }
```

Ø=ÜÁ features\strategy\components\max-daily-trades

Ø=ÜÁ features\strategy\components\max-daily-trades\max-daily-trades.component.ts

```

1 import { Component, OnInit } from '@angular/core';
2 import { Store } from '@ngrx/store';
3 import { SettingsService } from '../../../../../service/strategy.service';
4 import { selectMaxDailyTrades } from '../../../../../store/strategy.selectors';
5 import { MaxDailyTradesConfig, RuleType } from '../../../../../models/strategy.model';
6 import { setMaxDailyTradesConfig } from '../../../../../store/strategy.actions';
7 import { CommonModule } from '@angular/common';
8
9 /**
10  * Component for configuring the maximum daily trades rule.
11  *
12  * This component allows users to set the maximum number of trades that can be
13  * executed per day. It includes increment/decrement buttons and ensures the
14  * minimum value is 1.
15  *
16  * Features:
17  * - Toggle rule active/inactive
18  * - Number input with validation (minimum 1)
19  * - Increment/decrement buttons
20  * - Syncs with NgRx store
21  *
22  * Relations:
23  * - Store (NgRx): Reads and updates maxDailyTrades configuration
24  * - SettingsService: Strategy service (injected but not directly used)
25  *
26  * @component
27  * @selector app-max-daily-trades
28  * @standalone true
29  */
30 @Component({
31   selector: 'app-max-daily-trades',
32   templateUrl: './max-daily-trades.component.html',
33   styleUrls: ['./max-daily-trades.component.scss'],
34   imports: [CommonModule],
35   standalone: true,
36 })
37 export class MaxDailyTradesComponent implements OnInit {
38   config: MaxDailyTradesConfig = {
39     isActive: false,
40     maxDailyTrades: 1,
41     type: RuleType.MAX_DAILY_TRADES,
42   };
43
44   constructor(private store: Store, private settingsService: SettingsService) {}
45
46   ngOnInit(): void {
```

```

47     this.listenRuleConfiguration();
48 }
49 onToggleActive(event: Event) {
50   const isActive = (event.target as HTMLInputElement).checked;
51   const newConfig = {
52     ...this.config,
53     isActive: isActive,
54     // Mantener el valor actual, no resetear
55   };
56   this.updateConfig(newConfig);
57 }
58 onChangeValue(event: Event) {
59   const numValue = Number((event.target as HTMLInputElement).value);
60   const newConfig: MaxDailyTradesConfig = {
61     ...this.config,
62     maxDailyTrades: numValue < 1 ? 1 : numValue,
63   };
64   this.updateConfig(newConfig);
65 }
66 }
67
68 // Métodos para spinner (solo incrementar/decrementar)
69 incrementValue() {
70   if (this.config.isActive) {
71     const newConfig: MaxDailyTradesConfig = {
72       ...this.config,
73       maxDailyTrades: this.config.maxDailyTrades + 1,
74     };
75     this.updateConfig(newConfig);
76   }
77 }
78
79 decrementValue() {
80   if (this.config.isActive && this.config.maxDailyTrades > 1) {
81     const newConfig: MaxDailyTradesConfig = {
82       ...this.config,
83       maxDailyTrades: this.config.maxDailyTrades - 1,
84     };
85     this.updateConfig(newConfig);
86   }
87 }
88
89 listenRuleConfiguration() {
90   this.store
91     .select(selectMaxDailyTrades)
92     .pipe()
93     .subscribe((config) => {
94       this.config = config;
95     });
96 }
97
98 private updateConfig(config: MaxDailyTradesConfig) {
99   this.store.dispatch(setMaxDailyTradesConfig({ config }));
100 }
101 }
102

```

Ø=ÜÁ features\strategy\components\risk-per-trade

Ø=ÜÁ features\strategy\components\risk-per-trade\risk-per-trade.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { Component, HostListener, OnInit } from '@angular/core';

```

```

3 import {
4   MaxDailyTradesConfig,
5   RiskPerTradeConfig,
6   RiskRewardConfig,
7   RuleType,
8 } from '../../../../../models/strategy.model';
9 import { Store } from '@ngrx/store';
10 import { SettingsService } from '../../../../../service/strategy.service';
11 import { riskPerTrade, riskReward } from '../../../../../store/strategy.selectors';
12 import {
13   setRiskPerTradeConfig,
14   setRiskRewardConfig,
15 } from '../../../../../store/strategy.actions';
16 import { currencies } from './models/risk-per-trade.model';
17 import { ApplicationContextService } from '../../../../../shared/context';
18 import { AuthService } from '../../../../../auth/service/authService';
19 import { ReportService } from '../../../../../report/service/report.service';
20
21 /**
22 * Component for configuring the maximum risk per trade rule.
23 *
24 * This component allows users to set the maximum amount of risk that can be
25 * taken per trade. It supports multiple configuration options:
26 * - Review type: MAX (maximum allowed) or FIXED (fixed amount)
27 * - Number type: PERCENTAGE or MONEY
28 * - Percentage type: INITIAL_B (initial balance), ACTUAL_B (actual balance), or NULL
29 *
30 * Features:
31 * - Toggle rule active/inactive
32 * - Currency selection dropdown
33 * - Percentage or money amount input
34 * - Real-time balance fetching from API
35 * - Calculated amount display
36 *
37 * Relations:
38 * - Store (NgRx): Reads and updates riskPerTrade configuration
39 * - ApplicationContextService: Gets report data and balance
40 * - AuthService: Gets authentication tokens
41 * - ReportService: Fetches account balance
42 *
43 * @component
44 * @selector app-risk-per-trade
45 * @standalone true
46 */
47 @Component({
48   selector: 'app-risk-per-trade',
49   templateUrl: './risk-per-trade.component.html',
50   styleUrls: ['./risk-per-trade.component.scss'],
51   imports: [CommonModule],
52   standalone: true,
53 })
54 export class RiskPerTradeComponent implements OnInit {
55   config: RiskPerTradeConfig = {
56     isActive: false,
57     review_type: 'MAX',
58     number_type: 'PERCENTAGE',
59     percentage_type: 'NULL',
60     risk_ammount: 0,
61     type: RuleType.MAX_RISK_PER_TRADE,
62     balance: 1,
63     actualBalance: 0,
64   };
65
66   actualBalance: number = 0;
67   calculatedAmount: number = 0;
68   inputFirstRatioValue: number = 0;
69
70   inputSecondRatioValue: number = 0;
71
72   initialRiskTrade: number | undefined;

```

```

73     selectedCurrency = currencies[0];
74
75     dropdownOpen = false;
76
77     currencies = currencies;
78
79     constructor(
80       private store: Store,
81       private settingsService: SettingsService,
82       private applicationContext: ApplicationContextService,
83       private authService: AuthService,
84       private reportService: ReportService
85     ) {}
86
87     closeDropdown() {
88       this.dropdownOpen = false;
89     }
90
91     async ngOnInit(): Promise<void> {
92       this.listenRuleConfiguration();
93       await this.loadData();
94     }
95
96     async loadData() {
97       await this.loadActualBalance();
98       this.calculatedAmount = await this.getCalculatedAmount();
99     }
100
101    toggleDropdown() {
102      if (this.config.isActive) {
103        this.dropdownOpen = !this.dropdownOpen;
104      } else {
105        this.dropdownOpen = false;
106      }
107    }
108
109    selectCurrency(currency: { code: string; country: string }) {
110      this.selectedCurrency = currency;
111      this.dropdownOpen = false;
112    }
113
114    onToggleActive(event: Event) {
115      const newConfig = {
116        ...this.config,
117        isActive: (event.target as HTMLInputElement).checked,
118      };
119      this.updateConfig(newConfig);
120    }
121
122    onChangePercentage(event: Event) {
123      const percentage = Number((event.target as HTMLInputElement).value);
124      this.actualBalance = this.applicationContext.reportData()?.balanceData?.balance || 0;
125      const moneyRisk = Number(
126        ((percentage / 100) * this.actualBalance).toFixed(2)
127      );
128
129      const newConfig: RiskPerTradeConfig = {
130        ...this.config,
131        risk_ammount: percentage,
132        balance: -1,
133        actualBalance: this.actualBalance,
134      };
135      this.updateConfig(newConfig);
136    }
137
138    async getCurrentBalance(): Promise<number> {
139      if (this.config.percentage_type === 'ACTUAL_B') {
140        // Usar el actualBalance guardado si está disponible, sino cargar
141        if (this.config.actualBalance && this.config.actualBalance > 0) {

```

```

143     return this.config.actualBalance;
144 } else if (this.actualBalance === 0) {
145     await this.loadActualBalance();
146     return this.actualBalance;
147 }
148     return this.actualBalance;
149 } else if (this.config.percentage_type === 'INITIAL_B') {
150     return this.config.balance;
151 }
152     return 0;
153 }
154
155 // Método para cargar el balance actual desde el servicio
156 async loadActualBalance() {
157     try {
158         // Obtener datos del usuario para hacer la petición
159         const currentUser = this.appContext.currentUser();
160         if (!currentUser) {
161             console.warn('No hay usuario autenticado');
162             this.actualBalance = 0;
163             return;
164         }
165
166         // Obtener la primera cuenta del usuario
167         const userAccounts = this.appContext.userAccounts();
168         if (!userAccounts || userAccounts.length === 0) {
169             console.warn('No hay cuentas disponibles');
170             this.actualBalance = 0;
171             return;
172         }
173
174         const account = userAccounts[0];
175         const accessToken = await this.authService.getBearerTokenFirebase(currentUser.id);
176
177         // Hacer la petición al servicio de reportes para obtener el balance
178         const balanceData = await this.reportService.getBalanceData(
179             account.accountID,
180             accessToken,
181             account.accountNumber
182         ).toPromise();
183
184         if (balanceData && balanceData.balance) {
185             this.actualBalance = balanceData.balance;
186             // Actualizar el contexto con los datos obtenidos
187             this.appContext.updateReportBalance(balanceData);
188         } else {
189             this.actualBalance = 0;
190         }
191     } catch (error) {
192         console.error('Error loading actual balance:', error);
193         this.actualBalance = 0;
194     }
195 }
196
197 async getCalculatedAmount(): Promise<number> {
198     const balance = await this.getCurrentBalance();
199     return (this.config.risk_ammount / 100) * balance;
200 }
201
202 onChangeAmount(event: Event) {
203     const moneyRisk = Number((event.target as HTMLInputElement).value);
204     this.actualBalance = this.appContext.reportData()?.balanceData?.balance || 0;
205     const percentage = Number(
206         ((moneyRisk / this.actualBalance) * 100).toFixed(2)
207     );
208
209     const newConfig: RiskPerTradeConfig = {
210         ...this.config,
211         risk_ammount: moneyRisk,
212         balance: 0,

```

```

213     actualBalance: 0,
214   };
215   this.updateConfig(newConfig);
216 }
217
218 listenRuleConfiguration() {
219   this.store
220     .select(riskPerTrade)
221     .pipe()
222     .subscribe((config) => {
223       this.config = config;
224       if (!this.initialRiskTrade) {
225         this.initialRiskTrade = config.risk_ammount;
226       }
227
228       // Si la regla está inactiva, resetear todos los valores
229       if (!config.isActive) {
230         this.actualBalance = 0;
231         this.calculatedAmount = 0;
232       } else {
233         // Si está activa, cargar actualBalance si está disponible
234         if (config.actualBalance && config.actualBalance > 0) {
235           this.actualBalance = config.actualBalance;
236         }
237       }
238     });
239   }
240
241   private updateConfig(config: RiskPerTradeConfig) {
242     this.store.dispatch(setRiskPerTradeConfig({ config }));
243   }
244 }
245

```

Ø=ÜÁ features\strategy\components\risk-per-trade\models

Ø=ÜÁ features\strategy\components\risk-per-trade\models\risk-per-trade.model.ts

```

1  export const currencies = [
2    { code: 'USD', country: 'US' },
3    { code: 'EUR', country: 'EU' },
4    { code: 'JPY', country: 'JP' },
5    { code: 'GBP', country: 'GB' },
6    { code: 'AUD', country: 'AU' },
7    { code: 'CAD', country: 'CA' },
8    { code: 'CHF', country: 'CH' },
9    { code: 'CNY', country: 'CN' },
10   { code: 'HKD', country: 'HK' },
11   { code: 'NZD', country: 'NZ' },
12 ];
13

```

Ø=ÜÁ features\strategy\components\risk-reward

Ø=ÜÁ features\strategy\components\risk-reward\risk-reward.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { Component, OnInit } from '@angular/core';
3 import {
4   MaxDailyTradesConfig,
5   RiskRewardConfig,
6   RuleType,
7 } from '../../../../../models/strategy.model';
8 import { Store } from '@ngrx/store';
9 import { SettingsService } from '../../../../../service/strategy.service';
10 import { riskReward } from '../../../../../store/strategy.selectors';
11 import { setRiskRewardConfig } from '../../../../../store/strategy.actions';
12
13 /**
14  * Component for configuring the risk/reward ratio rule.
15  *
16  * This component allows users to set a minimum risk/reward ratio (e.g., "1:2")
17  * that trades must meet. It provides input fields for both parts of the ratio
18  * and includes increment/decrement buttons for the second value.
19  *
20  * Features:
21  * - Toggle rule active/inactive
22  * - Input fields for risk and reward values
23  * - Increment/decrement buttons for reward value
24  * - Syncs with NgRx store
25  *
26  * Relations:
27  * - Store (NgRx): Reads and updates riskReward configuration
28  * - SettingsService: Strategy service (injected but not directly used)
29  *
30  * @component
31  * @selector app-risk-reward-ratio
32  * @standalone true
33  */
34 @Component({
35   selector: 'app-risk-reward-ratio',
36   templateUrl: './risk-reward.component.html',
37   styleUrls: ['./risk-reward.component.scss'],
38   imports: [CommonModule],
39   standalone: true,
40 })
41 export class RiskRewardComponent implements OnInit {
42   config: RiskRewardConfig = {
43     isActive: false,
44     riskRewardRatio: '1:2',
45     type: RuleType.RISK_REWARD_RATIO,
46   };
47
48   inputFirstRatioValue: number = 0;
49
50   inputSecondRatioValue: number = 0;
51
52   initialRatio: string | undefined;
53
54   constructor(private store: Store, private settingsService: SettingsService) {}
55
56   ngOnInit(): void {
57     this.listenRuleConfiguration();
58   }
59   onToggleActive(event: Event) {
60     const isActive = (event.target as HTMLInputElement).checked;
61     const newConfig = {
62       ...this.config,
63       isActive: isActive,
64       // Reiniciar a 1:2 cuando se desactiva
65       riskRewardRatio: isActive ? this.config.riskRewardRatio : '1:2',
66     };
67     this.updateConfig(newConfig);
68   }
69
70   onChangeValue(event: Event, isFirst: boolean) {

```

```

71  const numValue = Number((event.target as HTMLInputElement).value);
72  const numberArray = this.config.riskRewardRatio
73    .split(':')
74    .map((number) => parseInt(number, 10));
75
76  if (isFirst) {
77    numberArray[0] = numValue;
78  } else {
79    numberArray[1] = numValue;
80  }
81
82  const newConfig: RiskRewardConfig = {
83    ...this.config,
84    riskRewardRatio: numberArray.join(':'),
85  };
86  this.updateConfig(newConfig);
87}
88
89 // Métodos para spinner (solo para el segundo número)
90 incrementSecondValue() {
91  if (this.config.isActive) {
92    const numberArray = this.config.riskRewardRatio
93      .split(':')
94      .map((number) => parseInt(number, 10));
95
96    numberArray[1] = numberArray[1] + 1;
97
98    const newConfig: RiskRewardConfig = {
99      ...this.config,
100     riskRewardRatio: numberArray.join(':'),
101   };
102   this.updateConfig(newConfig);
103 }
104}
105
106 decrementSecondValue() {
107  if (this.config.isActive) {
108    const numberArray = this.config.riskRewardRatio
109      .split(':')
110      .map((number) => parseInt(number, 10));
111
112    if (numberArray[1] > 2) {
113      numberArray[1] = numberArray[1] - 1;
114
115    const newConfig: RiskRewardConfig = {
116      ...this.config,
117      riskRewardRatio: numberArray.join(':'),
118    };
119    this.updateConfig(newConfig);
120  }
121 }
122}
123
124 listenRuleConfiguration() {
125  this.store
126    .select(riskReward)
127    .pipe()
128    .subscribe((config) => {
129      this.config = config;
130      if (!this.initialRatio) {
131        this.initialRatio = config.riskRewardRatio;
132      }
133      const numberArray = config.riskRewardRatio
134        .split(':')
135        .map((number) => parseInt(number, 10));
136      this.inputFirstRatioValue = numberArray[0];
137      this.inputSecondRatioValue = numberArray[1];
138    });
139}
140

```

```

141     private updateConfig(config: RiskRewardConfig) {
142       this.store.dispatch(setRiskRewardConfig({ config }));
143     }
144   }
145

```

Ø=ÜÁ features\strategy\edit-strategy

Ø=ÜÁ features\strategy\edit-strategy\edit-strategy.component.ts

```

1 import { Component, OnInit, OnDestroy, ViewChild } from '@angular/core';
2 import { Router, ActivatedRoute } from '@angular/router';
3 import { Store } from '@ngrx/store';
4 import { Observable } from 'rxjs';
5 import { FormsModule } from '@angular/forms';
6 import { NgIf } from '@angular/common';
7 import { allRules } from '../store/strategy.selectors';
8 import { StrategyState, RuleType } from '../models/strategy.model';
9 import { resetConfig } from '../store/strategy.actions';
10 import { SettingsService } from '../service/strategy.service';
11 import { ReportService } from '../../../../../report/service/report.service';
12 import { User } from '../../../../../overview/models/overview';
13 import { selectUser } from '../../../../../auth/store/user.selectios';
14 import { selectUserKey } from '../../../../../report/store/report.selectors';
15 import { setUserKey } from '../../../../../report/store/report.actions';
16 import { initialStrategyState } from '../store/strategy.reducer';
17 import { EditPopupComponent } from '../../../../../shared/pop-ups/edit-pop-up/edit-
18 paper-component';
19 import { ConfirmPopupComponent } from '../../../../../shared/pop-ups/confirm-pop-up/confirm-
20 paper-component';
21 // Importar componentes de reglas edit-*
22 import { EditMaxDailyTradesComponent } from './components/edit-max-daily-trades/edit-max-
23 dailytradesriskrewardComponent' from './components/edit-risk-reward/edit-risk-
24 reward.componeriskperTradeComponent' from './components/edit-risk-per-trade/edit-risk-per-
25 tradecomponent';
26 import { EditDaysAllowedComponent } from './components/edit-days-allowed/edit-days-
27 allowed';
28 import { EditAssetsAllowedComponent } from './components/edit-assets-allowed/edit-assets-
29 allowed';
30 import { EditHoursAllowedComponent } from './components/edit-hours-allowed/edit-hours-
31 allowed';
32 import { AuthService } from '../../../../../auth/service/authService';
33 import { AccountData } from '../../../../../auth/models/userModel';
34 import { PluginHistoryService, PluginHistory } from '../../../../../shared/services/plugin-
35 historyservice';
36 /**
37 * Component for editing trading strategy configurations.
38 *
39 * This component provides a comprehensive interface for editing all aspects
40 * of a trading strategy, including all six trading rules. It supports both
41 * creating new strategies and editing existing ones.
42 *
43 * Features:
44 * - Edit all six trading rules (risk/reward, risk per trade, max daily trades,
45 * days allowed, hours allowed, assets allowed)
46 * - Strategy name editing
47 * - Save and cancel functionality
48 * - Validation before saving
49 * - Loading states
50 * - Plugin history integration
51 * - Instrument fetching for assets allowed
52 *
53 * Relations:

```

```

54 * - EditMaxDailyTradesComponent: Max daily trades rule editor
55 * - EditRiskRewardComponent: Risk/reward ratio rule editor
56 * - EditRiskPerTradeComponent: Risk per trade rule editor
57 * - EditDaysAllowedComponent: Days allowed rule editor
58 * - EditHoursAllowedComponent: Trading hours rule editor
59 * - EditAssetsAllowedComponent: Assets allowed rule editor
60 * - SettingsService: Saving strategy configurations
61 * - StrategyCacheService: Loading strategies from cache
62 * - BalanceCacheService: Getting account balances
63 * - Store (NgRx): Managing strategy state
64 *
65 * @component
66 * @selector app-edit-strategy
67 * @standalone true
68 */
69 @Component({
70   selector: 'app-edit-strategy',
71   imports: [
72     FormsModule,
73     NgIf,
74     EditMaxDailyTradesComponent,
75     EditRiskRewardComponent,
76     EditRiskPerTradeComponent,
77     EditDaysAllowedComponent,
78     EditAssetsAllowedComponent,
79     EditHoursAllowedComponent,
80     EditPopupComponent,
81     ConfirmPopupComponent,
82     LoadingPopupComponent
83   ],
84   templateUrl: './edit-strategy.component.html',
85   styleUrls: ['./edit-strategy.component.scss'],
86   standalone: true,
87 })
88 export class EditStrategyComponent implements OnInit, OnDestroy {
89   config$: Observable<StrategyState>;
90   config: StrategyState | null = null;
91   myChoices: StrategyState | null = null;
92   loading = false;
93   initialLoading = true; // Loading global para evitar tambaleo
94   user: User | null = null;
95   editPopupVisible = false;
96   confirmPopupVisible = false;
97   strategyId: string | null = null;
98
99   // Mini card properties
100  currentStrategyName: string = 'My Strategy';
101  lastModifiedText: string = 'Never modified';
102  isFavorited: boolean = false;
103  isEditingName: boolean = false;
104  editingStrategyName: string = '';
105  accountsData: AccountData[] = [];
106  currentAccount: AccountData | null = null;
107  availableInstruments: string[] = [];
108
109  // Plugin history properties
110  pluginHistory: PluginHistory[] = [];
111  isPluginActive: boolean = false;
112  private pluginSubscription: any = null;
113
114  // Referencias a componentes de reglas para validación
115  @ViewChild('hoursAllowedRef') hoursAllowedRef?: EditHoursAllowedComponent;
116  @ViewChild('daysAllowedRef') daysAllowedRef?: EditDaysAllowedComponent;
117  @ViewChild('assetsAllowedRef') assetsAllowedRef?: EditAssetsAllowedComponent;
118  @ViewChild('maxDailyTradesRef') maxDailyTradesRef?: EditMaxDailyTradesComponent;
119  @ViewChild('riskPerTradeRef') riskPerTradeRef?: EditRiskPerTradeComponent;
120
121  constructor(
122    private store: Store,
123    private router: Router,

```

```

124     private route: ActivatedRoute,
125     private strategySvc: SettingsService,
126     private reportSvc: ReportService,
127     private authService: AuthService,
128     private pluginHistoryService: PluginHistoryService,
129     private strategyCacheService: StrategyCacheService,
130     private balanceCacheService: BalanceCacheService,
131     private alertService: AlertService
132   ) {
133     this.config$ = this.store.select(allRules);
134   }
135
136   async ngOnInit() {
137     this.initialLoading = true;
138
139     try {
140       // FLUJO SIMPLIFICADO: Cargar todo antes de mostrar la UI
141       await this.initializeEverything();
142     } catch (error) {
143       console.error('Error during initialization:', error);
144     } finally {
145       this.initialLoading = false;
146     }
147   }
148
149   /**
150    * Inicializar todo antes de mostrar la UI
151    */
152   private async initializeEverything(): Promise<void> {
153     // 1. Obtener datos del usuario
154     await this.getUserDataAsync();
155
156     // 2. Configurar listeners
157     this.listenConfigurations();
158
159     // 3. Obtener ID de estrategia y cargar configuración
160     this.getStrategyId();
161   }
162
163   /**
164    * MÉTODO SIMPLIFICADO: Obtener ID de estrategia desde la URL
165    * - Si hay strategyId: Cargar desde cache
166    * - Si no hay strategyId: Nueva estrategia
167    */
168   getStrategyId() {
169     this.route.queryParams.subscribe(params => {
170       const newStrategyId = params['strategyId'] || null;
171
172       // Si cambió la estrategia, limpiar el estado
173       if (this.strategyId !== newStrategyId) {
174         this.clearState();
175       }
176
177       this.strategyId = newStrategyId;
178
179       // Cargar configuración usando el cache
180       this.loadStrategyFromCache();
181     });
182   }
183
184   /**
185    * Limpia el estado al cambiar entre estrategias
186    */
187   clearState() {
188     // Limpiar variables del componente
189     this.myChoices = null;
190     this.config = null;
191     this.currentStrategyName = 'My Strategy';
192     this.lastModifiedText = 'Never modified';
193     this.isFavorited = false;

```

```

194     this.isEditingName = false;
195     this.editingStrategyName = '';
196
197     // LIMPIAR EL STORE para evitar datos de estrategias anteriores
198     // Usar un estado inicial vacío en lugar de null
199     const emptyState: StrategyState = {
200       maxDailyTrades: { isActive: false, type: RuleType.MAX_DAILY_TRADES, maxDailyTrades:
201         0 }, riskReward: { isActive: false, type: RuleType.RISK_REWARD_RATIO, riskRewardRatio:
202         '1:2' }, riskPerTrade: { isActive: false, type: RuleType.MAX_RISK_PER_TRADE, review_type:
203         'MAX', dayRateType: { 'PERCENTAGE': false, type: RuleType.DAYS_ALLOWED, tradingDays: 1, balance: 0 },
204         actualBalance: 0 }, assetsAllowed: { isActive: false, type: RuleType.ASSETS_ALLOWED, assetsAllowed: [] },
205         hoursAllowed: { isActive: false, type: RuleType.TRAILING_HOURS, tradingOpenTime: '',
206         tradingCloseTime: '', timezone: '' }
207
208     this.store.dispatch(resetConfig({ config: emptyState }));
209   }
210
211   /**
212    * MÉTODO PRINCIPAL: Cargar estrategia desde cache
213    * FLUJO SIMPLIFICADO:
214    * - Si hay strategyId: Cargar desde cache del componente Strategy
215    * - Si no hay strategyId: Inicializar como nueva estrategia
216    */
217   loadStrategyFromCache() {
218     if (this.strategyId) {
219       // Cargar estrategia existente desde cache
220       this.loadExistingStrategyFromCache();
221     } else {
222       // Inicializar como nueva estrategia
223       this.initializeAsNewStrategy();
224     }
225   }
226
227   /**
228    * Cargar estrategia existente desde cache o Firebase
229    */
230   loadExistingStrategyFromCache() {
231     if (!this.strategyId) return;
232
233     // Verificar si el cache está disponible
234     if (!this.strategyCacheService.isCacheLoaded()) {
235       // Si el cache no está disponible, cargar directamente desde Firebase
236       this.loadStrategyFromFirebase();
237       return;
238     }
239
240     // Obtener estrategia del cache
241     const cachedStrategy = this.strategyCacheService.getStrategy(this.strategyId);
242
243     if (!cachedStrategy) {
244       // Si no está en cache, cargar desde Firebase
245       this.loadStrategyFromFirebase();
246       return;
247     }
248
249     // Actualizar mini card
250     this.currentStrategyName = cachedStrategy.overview.name;
251     this.lastModifiedText = this.formatDate(cachedStrategy.overview.updated_at.toDate());
252     this.isFavorited = false;
253
254     // Cargar balance si es necesario
255     this.loadBalanceAndInitializeWithConfig(cachedStrategy.configuration);
256   }
257
258   /**
259    * Cargar estrategia directamente desde Firebase (fallback cuando cache no está disponible)
260    */
261   async loadStrategyFromFirebase() {
262     if (!this.strategyId || !this.user?.id) {
263       this.initializeAsNewStrategy();

```

```

264     return;
265   }
266
267   try {
268     // Cargar estrategia directamente desde Firebase
269     const strategyData = await this.strategySvc.getStrategyView(this.strategyId);
270
271     if (!strategyData || !strategyData.configuration) {
272       this.initializeAsNewStrategy();
273       return;
274     }
275
276     // Actualizar mini card
277     this.currentStrategyName = strategyData.overview.name;
278     this.lastModifiedText = this.formatDate(strategyData.overview.updated_at.toDate());
279     this.isFavorited = false;
280
281     // Cargar balance y inicializar con configuración
282     this.loadBalanceAndInitializeWithConfig(strategyData.configuration);
283
284   } catch (error) {
285     console.error('Error loading strategy from Firebase:', error);
286     this.initializeAsNewStrategy();
287   }
288 }
289
290 /**
291 * Cargar balance e inicializar con configuración específica
292 */
293 loadBalanceAndInitializeWithConfig(configuration: StrategyState) {
294   // Obtener balance desde cache primero
295   let balance = 0;
296   if (this.accountsData.length > 0) {
297     const firstAccount = this.accountsData[0];
298     balance = this.balanceCacheService.getBalance(firstAccount.accountID);
299   }
300
301   // Inicializar con balance del cache
302   this.initializeWithConfigAndBalance(configuration, balance);
303
304   // Si necesita actualización, hacer petición en background
305   if (this.accountsData.length > 0) {
306     const firstAccount = this.accountsData[0];
307     if (this.balanceCacheService.needsUpdate(firstAccount.accountID)) {
308       this.updateBalanceInBackground(firstAccount);
309     }
310   }
311 }
312
313 /**
314 * Actualizar balance en background sin bloquear la UI
315 */
316 private updateBalanceInBackground(account: AccountData) {
317   this.store.select(selectUserKey).pipe().subscribe({
318     next: (userKey) => {
319       if (userKey && userKey !== '') {
320         this.reportSvc.getBalanceData(account.accountID as string, userKey,
321 account.accountNumber).pipe().subscribe({
322           // Actualizar cache con nuevo balance
323           this.balanceCacheService.setBalance(account.accountID, balance);
324         },
325         error: (err) => {
326           console.error('Error fetching balance data in background:', err);
327         },
328       });
329     }
330   },
331   });
332 }
333

```

```

334 /**
335 * Inicializar con configuración y balance específicos
336 */
337 initializeWithConfigAndBalance(configuration: StrategyState, balance: number) {
338     // Solo actualizar balance si es para balance actual (ACTUAL_B)
339     // Para balance inicial (INITIAL_B), mantener el valor de Firebase
340     let configWithBalance = { ...configuration };
341
342     if (configuration.riskPerTrade.percentage_type === 'ACTUAL_B') {
343         // Para balance actual, usar el balance del cache
344         configWithBalance = {
345             ...configuration,
346             riskPerTrade: {
347                 ...configuration.riskPerTrade,
348                 actualBalance: balance
349             }
350         };
351     } else if (configuration.riskPerTrade.percentage_type === 'INITIAL_B') {
352         // Para balance inicial, mantener el valor de Firebase (no modificar)
353         configWithBalance = configuration;
354     } else {
355         // Para otros casos, usar el balance del cache
356         configWithBalance = {
357             ...configuration,
358             riskPerTrade: {
359                 ...configuration.riskPerTrade,
360                 balance: balance
361             }
362         };
363     }
364
365     // Cargar en el store
366     this.store.dispatch(resetConfig({ config: configWithBalance }));
367
368     // Distribuir reglas entre paneles
369     this.distributeRulesBetweenPanels(configWithBalance);
370 }
371
372 /**
373 * MÉTODO SIMPLIFICADO: Inicializar como nueva estrategia
374 * FLUJO PARA NUEVA ESTRATEGIA:
375 * - Available Rules: Todas las reglas (isActive = true para mostrar)
376 * - My Choices: Vacío (isActive = false)
377 */
378 initializeAsNewStrategy() {
379     // Cargar configuración por defecto con balance
380     this.loadBalanceAndInitialize();
381 }
382
383 /**
384 * MÉTODO SIMPLIFICADO: Distribuir reglas entre paneles
385 * LÓGICA UNIFICADA:
386 * - My Choices: Reglas con isActive = true
387 * - Available Rules: Reglas con isActive = false
388 */
389 distributeRulesBetweenPanels(configurationData: StrategyState) {
390     // My Choices: Solo reglas activas (isActive = true)
391     this.myChoices = {
392         maxDailyTrades: {
393             isActive: configurationData.maxDailyTrades.isActive,
394             type: configurationData.maxDailyTrades.type,
395             maxDailyTrades: configurationData.maxDailyTrades.maxDailyTrades
396         },
397         riskReward: {
398             isActive: configurationData.riskReward.isActive,
399             type: configurationData.riskReward.type,
400             riskRewardRatio: configurationData.riskReward.riskRewardRatio
401         },
402         riskPerTrade: {
403             isActive: configurationData.riskPerTrade.isActive,

```

```

404     type: configurationData.riskPerTrade.type,
405     review_type: configurationData.riskPerTrade.review_type,
406     number_type: configurationData.riskPerTrade.number_type,
407     percentage_type: configurationData.riskPerTrade.percentage_type,
408     risk_amount: configurationData.riskPerTrade.risk_amount,
409     balance: configurationData.riskPerTrade.balance,
410     actualBalance: configurationData.riskPerTrade.actualBalance
411   },
412   daysAllowed: {
413     isActive: configurationData.daysAllowed.isActive,
414     type: configurationData.daysAllowed.type,
415     tradingDays: configurationData.daysAllowed.tradingDays
416   },
417   assetsAllowed: {
418     isActive: configurationData.assetsAllowed.isActive,
419     type: configurationData.assetsAllowed.type,
420     assetsAllowed: configurationData.assetsAllowed.assetsAllowed
421   },
422   hoursAllowed: {
423     isActive: configurationData.hoursAllowed.isActive,
424     type: configurationData.hoursAllowed.type,
425     tradingOpenTime: configurationData.hoursAllowed.tradingOpenTime,
426     tradingCloseTime: configurationData.hoursAllowed.tradingCloseTime,
427     timezone: configurationData.hoursAllowed.timezone
428   }
429 };
430
431 // Available Rules: Solo reglas NO activas (isActive = false)
432 this.config = {
433   maxDailyTrades: {
434     isActive: !configurationData.maxDailyTrades.isActive,
435     type: configurationData.maxDailyTrades.type,
436     maxDailyTrades: configurationData.maxDailyTrades.maxDailyTrades
437   },
438   riskReward: {
439     isActive: !configurationData.riskReward.isActive,
440     type: configurationData.riskReward.type,
441     riskRewardRatio: configurationData.riskReward.riskRewardRatio
442   },
443   riskPerTrade: {
444     isActive: !configurationData.riskPerTrade.isActive,
445     type: configurationData.riskPerTrade.type,
446     review_type: configurationData.riskPerTrade.review_type,
447     number_type: configurationData.riskPerTrade.number_type,
448     percentage_type: configurationData.riskPerTrade.percentage_type,
449     risk_amount: configurationData.riskPerTrade.risk_amount,
450     balance: configurationData.riskPerTrade.balance,
451     actualBalance: configurationData.riskPerTrade.actualBalance
452   },
453   daysAllowed: {
454     isActive: !configurationData.daysAllowed.isActive,
455     type: configurationData.daysAllowed.type,
456     tradingDays: configurationData.daysAllowed.tradingDays
457   },
458   assetsAllowed: {
459     isActive: !configurationData.assetsAllowed.isActive,
460     type: configurationData.assetsAllowed.type,
461     assetsAllowed: configurationData.assetsAllowed.assetsAllowed
462   },
463   hoursAllowed: {
464     isActive: !configurationData.hoursAllowed.isActive,
465     type: configurationData.hoursAllowed.type,
466     tradingOpenTime: configurationData.hoursAllowed.tradingOpenTime,
467     tradingCloseTime: configurationData.hoursAllowed.tradingCloseTime,
468     timezone: configurationData.hoursAllowed.timezone
469   }
470 };
471 }
472 /**
473 */

```

```
474 * Cargar cuentas de forma asíncrona
475 */
476 private async fetchUserAccountsAsync(): Promise<void> {
477   if (this.user?.id) {
478     try {
479       const accounts = await this.authService.getUserAccounts(this.user.id);
480       this.accountsData = accounts || [];
481       // After loading accounts, try to fetch user key
482       await this.fetchUserKeyAsync();
483     } catch (error) {
484       console.error('Error loading accounts:', error);
485       this.accountsData = [];
486     }
487   }
488 }
489
490 fetchUserAccounts() {
491   if (this.user?.id) {
492     this.authService.getUserAccounts(this.user.id).then((accounts) => {
493       this.accountsData = accounts || [];
494       // After loading accounts, try to fetch user key
495       this.fetchUserKey();
496     });
497   }
498 }
499
500 /**
501 * Obtener userKey de forma asíncrona
502 */
503 private async fetchUserKeyAsync(): Promise<void> {
504   if (this.user?.email && this.accountsData.length > 0) {
505     // Use the first account's credentials
506     const firstAccount = this.accountsData[0];
507
508     try {
509       const key = await this.reportSvc.getUserKey(
510         firstAccount.emailTradingAccount,
511         firstAccount.brokerPassword,
512         firstAccount.server
513       ).toPromise();
514
515       this.store.dispatch(setUserKey({ userKey: key || '' }));
516       // After getting userKey, load instruments
517       this.loadInstruments(key || '', firstAccount);
518     } catch (err) {
519       console.error('Error fetching user key:', err);
520       this.store.dispatch(setUserKey({ userKey: '' }));
521     }
522   } else {
523     this.store.dispatch(setUserKey({ userKey: '' }));
524   }
525 }
526
527 fetchUserKey() {
528   if (this.user?.email && this.accountsData.length > 0) {
529     // Use the first account's credentials
530     const firstAccount = this.accountsData[0];
531     this.reportSvc
532       .getUserKey(firstAccount.emailTradingAccount, firstAccount.brokerPassword,
533       firstAccount.server)
534       .subscribe({
535         next: (key: string) => {
536           this.store.dispatch(setUserKey({ userKey: key }));
537           // After getting userKey, load instruments
538           this.loadInstruments(key, firstAccount);
539         },
540         error: (err) => {
541           console.error('Error fetching user key:', err);
542           this.store.dispatch(setUserKey({ userKey: '' }));
543         },
544       });
545   }
546 }
```

```

544     } else {
545       this.store.dispatch(setUserKey({ userKey: '' }));
546     }
547   }
548
549   loadInstruments(userKey: string, account: AccountData) {
550     this.reportSvc.getAllInstruments(
551       userKey,
552       account.accountNumber,
553       account.accountID
554     ).subscribe({
555       next: (instruments: Instrument[]) => {
556         // Extraer solo los nombres de los instrumentos
557         this.availableInstruments = instruments.map(instrument => instrument.name);
558       },
559       error: (err) => {
560         console.error('Error loading instruments:', err);
561         this.availableInstruments = [];
562       }
563     });
564   }
565
566   getActualBalance() {
567     this.store
568       .select(selectUserKey)
569       .pipe()
570       .subscribe({
571         next: (userKey) => {
572           if (userKey === '') {
573             this.fetchUserKey();
574           } else {
575             // Use the first account's data dynamically
576             if (this.accountsData.length > 0) {
577               const firstAccount = this.accountsData[0];
578               // El servicio ya actualiza el contexto automáticamente
579               this.reportSvc.getBalanceData(firstAccount.accountID as string, userKey,
580               firstAccount.accountNumber as number).subscribe({
581                 initializeWithConfigAndBalance(initialStrategyState, balance);
582               },
583               error: (err) => {
584                 console.error('Error fetching balance data', err);
585                 this.initializeWithConfigAndBalance(initialStrategyState, 0);
586               },
587             });
588           } else {
589             this.initializeWithConfigAndBalance(initialStrategyState, 0);
590           }
591         }
592       },
593     });
594   }
595
596 /**
597  * Obtener datos del usuario de forma asíncrona
598 */
599 private async getUserDataAsync(): Promise<void> {
600   return new Promise((resolve) => {
601     this.store.select(selectUser).subscribe({
602       next: async (user) => {
603         this.user = user.user;
604
605         // Cargar cuentas y configurar plugin history cuando el usuario esté disponible
606         if (this.user?.id) {
607           await this.fetchUserAccountsAsync();
608           this.setupPluginHistoryListener();
609         } else {
610           console.warn('User ID not available yet');
611         }
612         resolve();
613     },

```

```

614     error: (err) => {
615       console.error('Error fetching user data', err);
616       resolve();
617     },
618   });
619 });
620 }
621
622 getUserData() {
623   this.store.select(selectUser).subscribe({
624     next: (user) => {
625       this.user = user.user;
626
627       // Cargar cuentas y configurar plugin history cuando el usuario esté disponible
628       if (this.user?.id) {
629         this.fetchUserAccounts();
630         this.setupPluginHistoryListener();
631       } else {
632         console.warn('User ID not available yet');
633       }
634     },
635     error: (err) => {
636       console.error('Error fetching user data', err);
637     },
638   });
639 }
640
641 /**
642 * MÉTODO SIMPLIFICADO: Cargar balance e inicializar como nueva estrategia
643 */
644 loadBalanceAndInitialize() {
645   // Obtener balance desde cache primero
646   let balance = 0;
647   if (this.accountsData.length > 0) {
648     const firstAccount = this.accountsData[0];
649     balance = this.balanceCacheService.getBalance(firstAccount.accountID);
650   }
651
652   // Inicializar con balance del cache
653   this.initializeWithConfigAndBalance(initialStrategyState, balance);
654
655   // Si necesita actualización, hacer petición en background
656   if (this.accountsData.length > 0) {
657     const firstAccount = this.accountsData[0];
658     if (this.balanceCacheService.needsUpdate(firstAccount.accountID)) {
659       this.updateBalanceInBackground(firstAccount);
660     }
661   }
662 }
663
664 /**
665 * MÉTODO SIMPLIFICADO: Escuchar cambios en el store de reglas
666 */
667 listenConfigurations() {
668   this.store
669     .select(allRules)
670     .pipe()
671     .subscribe((config) => {
672       // Actualizar UI en tiempo real
673       this.listenToStoreChanges();
674     });
675 }
676
677
678 /**
679 * MÉTODO SIMPLIFICADO: Escuchar cambios en tiempo real
680 */
681 listenToStoreChanges() {
682   this.config$.subscribe(config => {
683     if (this.config && this.myChoices) {

```

```

684         // Actualizar UI en tiempo real
685         this.updateMyChoicesFromConfig(config);
686     }
687   });
688 }
689 /**
690 * MÉTODO SIMPLIFICADO: Actualizar UI en tiempo real
691 */
692 updateMyChoicesFromConfig(newConfig: StrategyState) {
693   if (!this.myChoices || !this.config) return;
694
695   // Usar el método unificado para distribuir reglas
696   this.distributeRulesBetweenPanels(newConfig);
697 }
698
699 saveStrategy() {
700   // Verificar si se puede guardar (plugin no activo)
701   if (!this.canSaveStrategy()) {
702     this.alertService.showWarning('Cannot save strategy while plugin is active. Please
703 deactivate the plugin first.', 'Plugin Active');
704   }
705 }
706
707 // Validar todas las reglas activas
708 if (!this.validateActiveRules()) {
709   return;
710 }
711
712 this.confirmPopupVisible = true;
713 }
714
715 /**
716 * MÉTODO 7: Guardar configuración
717 * FLUJO DE GUARDADO:
718 * - Solo se guardan las reglas que están en My Choices (isActive = true)
719 * - Si hay strategyId: Actualizar estrategia existente
720 * - Si no hay strategyId: Crear nueva estrategia
721 */
722 save = () => {
723   if (this.myChoices && this.user?.id) {
724     this.loading = true;
725
726     // GUARDADO: Crear configuración solo con las reglas de My Choices
727     // Las reglas que están en My Choices son las que se guardarán como activas
728     const newConfig: StrategyState = {
729       maxDailyTrades: { isActive: this.myChoices.maxDailyTrades.isActive, type:
730         this.myChoices.maxDailyTrades.type, riskRewardRatio: this.myChoices.maxDailyTrades.riskRewardRatio },
731       myChoices.maxDailyTrades.type, riskRewardRatio },
732       this.myChoices.maxDailyTrades.type, riskRewardRatio },
733       this.myChoices.maxDailyTrades.type, riskRewardRatio },
734       this.myChoices.maxDailyTrades.type, riskRewardRatio },
735       this.myChoices.maxDailyTrades.type, riskRewardRatio },
736       this.myChoices.maxDailyTrades.type, riskRewardRatio },
737       this.myChoices.hoursAllowed.tradingOpenTime, tradingCloseTime,
738       this.myChoices.hoursAllowed.timeZone },
739       this.store.dispatch(resetConfig({ config: newConfig }));
740
741       // CASO A: Estrategia existente - Actualizar en Firebase
742       if (this.strategyId) {
743         this.strategySvc
744           .updateStrategyView(this.strategyId, {
745             configuration: newConfig
746           })
747           .then(async () => {
748             // Actualizar fecha de modificación en la mini card
749             this.lastModifiedText = this.formatDate(new Date());
750
751             // Limpiar cache para forzar recarga
752             this.strategyCacheService.clearCache();
753
754             this.router.navigate(['/strategy']);

```

```

754     })
755     .catch((err) => {
756       console.error('Update Error:', err);
757       this.alertService.showError('Error Updating Strategy', 'Update Error');
758     })
759     .finally(() => {
760       this.loading = false;
761       this.closeConfirmModal();
762     });
763   } else {
764     // CASO B: Nueva estrategia - Crear en Firebase
765     this.strategySvc
766       .createStrategyView(this.user.id, this.currentStrategyName, newConfig)
767       .then(async (strategyId) => {
768         // Limpiar cache para forzar recarga
769         this.strategyCacheService.clearCache();
770
771         this.router.navigate(['/strategy']);
772       })
773       .catch((err) => {
774         console.error('Create Error:', err);
775         this.alertService.showError('Error Creating Strategy', 'Creation Error');
776       })
777       .finally(() => {
778         this.loading = false;
779         this.closeConfirmModal();
780       });
781   }
782 }
783 };
784
785 closePopup = () => {
786   this.editPopupVisible = false;
787 };
788
789 openConfirmPopup() {
790   this.confirmPopupVisible = true;
791 }
792
793 closeConfirmModal = () => {
794   this.confirmPopupVisible = false;
795 };
796
797 /**
798 * Método que se ejecuta cuando se confirma el guardado
799 * Hace la validación final antes de proceder con el guardado
800 */
801 confirmSave = () => {
802   // Validar todas las reglas activas una vez más
803   if (!this.validateActiveRules()) {
804     return; // No proceder con el guardado
805   }
806
807   this.save();
808 };
809
810 /**
811 * Valida todas las reglas activas antes de permitir guardar
812 */
813 private validateActiveRules(): boolean {
814   if (!this.myChoices) {
815     return true; // Si no hay reglas activas, permitir guardar
816   }
817
818   const validationErrors: string[] = [];
819
820   // Validar regla de horas permitidas
821   if (this.myChoices.hoursAllowed.isActive && this.hoursAllowedRef) {
822     if (!this.hoursAllowedRef.isRuleValid()) {
823       validationErrors.push(`Hours Allowed: ${this.hoursAllowedRef.getMessage()}`);

```

```

824     }
825 }
826
827 // Validar regla de días permitidos
828 if (this.myChoices.daysAllowed.isActive && this.daysAllowedRef) {
829     if (!this.daysAllowedRef.isRuleValid()) {
830         validationErrors.push(`Days Allowed: ${this.daysAllowedRef.getErrorMessage()}`);
831     }
832 }
833
834 // Validar regla de assets permitidos
835 if (this.myChoices.assetsAllowed.isActive && this.assetsAllowedRef) {
836     if (!this.assetsAllowedRef.isRuleValid()) {
837         validationErrors.push(`Assets Allowed: ${this.assetsAllowedRef.getErrorMessage()}`);
838     }
839 }
840
841 // Validar regla de máximo de trades diarios
842 if (this.myChoices.maxDailyTrades.isActive && this.maxDailyTradesRef) {
843     if (!this.maxDailyTradesRef.isRuleValid()) {
844         validationErrors.push(`Max Daily Trades: ${this.maxDailyTradesRef.getErrorMessage()}`);
845     }
846 }
847
848 // Validar regla de riesgo por trade
849 if (this.myChoices.riskPerTrade.isActive && this.riskPerTradeRef) {
850     if (!this.riskPerTradeRef.isRuleValid()) {
851         validationErrors.push(`Risk Per Trade: ${this.riskPerTradeRef.getErrorMessage()}`);
852     }
853 }
854
855 if (validationErrors.length > 0) {
856     const errorMessage = 'Cannot save strategy. Please complete the following rules:\n\n' +
857     validationErrors.join('\n') +
858     '\n\nAll required fields must be filled before saving.';
859     this.alertService.showError(errorMessage, 'Validation Error');
860     return false;
861 }
862
863 return true;
864 }
865
866 // Mini card methods
867 startEditName() {
868     this.isEditingName = true;
869     this.editingStrategyName = this.currentStrategyName;
870     // Focus on input after view update
871     setTimeout(() => {
872         const input = document.querySelector('.strategy-name-input') as HTMLInputElement;
873         if (input) {
874             input.focus();
875             input.select();
876         }
877     }, 0);
878 }
879
880 async saveStrategyName() {
881     if (!this.editingStrategyName.trim()) {
882         this.cancelEditName();
883         return;
884     }
885
886     if (this.editingStrategyName.trim() === this.currentStrategyName) {
887         this.cancelEditName();
888         return;
889     }
890
891     try {
892         if (this.strategyId) {
893             // Actualizar nombre en configuration-overview

```

```

894     await this.strategySvc.updateConfigurationOverview(this.strategyId, {
895         name: this.editingStrategyName.trim()
896     });
897
898     // Actualizar UI
899     this.currentStrategyName = this.editingStrategyName.trim();
900     this.lastModifiedText = this.formatDate(new Date());
901 } else {
902     // Si no hay strategyId, solo actualizar localmente
903     this.currentStrategyName = this.editingStrategyName.trim();
904     this.lastModifiedText = this.formatDate(new Date());
905 }
906
907     this.isEditingName = false;
908 } catch (error) {
909     console.error('Error updating strategy name:', error);
910     this.alertService.showError('Error updating strategy name', 'Name Update Error');
911     this.cancelEditName();
912 }
913 }
914
915 cancelEditName() {
916     this.isEditingName = false;
917     this.editingStrategyName = '';
918 }
919
920 toggleFavorite() {
921     this.isFavorited = !this.isFavorited;
922     // TODO: Implementar persistencia de favoritos en Firebase
923     console.log('Strategy favorited:', this.isFavorited);
924 }
925
926 formatDate(date: Date): string {
927     const months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
928     'Nov', 'Dec'];
929     const month = months[date.getMonth()];
930     const day = date.getDate();
931     const year = date.getFullYear();
932     return `Last modified: ${month} ${day}, ${year}`;
933 }
934
935 discardChanges() {
936     this.router.navigate(['/strategy']);
937 }
938
939 resetStrategy() {
940     // Verificar si se puede resetear (plugin no activo)
941     if (!this.canSaveStrategy()) {
942         this.alertService.showWarning('Cannot reset strategy while plugin is active. Please
943 deactivate it first.', 'Plugin Active');
944     }
945
946     // Confirmar la acción
947     const confirmed = confirm('Are you sure you want to reset the strategy? This will move
948 all rules back to Available Rules and cannot be undone.');
949     if (confirmed) {
950         // Resetear a configuración inicial (todas las reglas en Available Rules)
951         this.initializeWithConfigAndBalance(initialStrategyState, 0);
952     }
953 }
954
955 /**
956 * MÉTODO 8: Cargar plugin history y verificar si está activo
957 * FLUJO DE VERIFICACIÓN:
958 * - Cargar plugin history desde Firebase
959 * - Verificar si algún plugin está activo
960 * - Bloquear botón de guardar si está activo
961 */
962 setupPluginHistoryListener() {
963     if (!this.user?.id) {

```

```

964         return;
965     }
966
967     try {
968
969         // Suscribirse al Observable del servicio con userId
970         this.pluginSubscription =
971         this.pluginHistoryService.getPluginHistoryRealtime(this.userId).subscribe({
972             next: (pluginHistory) => {
973                 this.pluginHistory = pluginHistory;
974
975                 // Verificar si el plugin está activo usando la nueva lógica de fechas
976                 if (pluginHistory.length > 0) {
977                     const plugin = pluginHistory[0];
978                     this.isPluginActive = this.pluginHistoryService.isPluginActiveByDates(plugin);
979                 } else {
980                     // No hay plugin para este usuario
981                     this.isPluginActive = false;
982                     console.log('No plugin found for user');
983                 }
984             },
985             error: (error) => {
986                 console.error('Error in plugin history subscription:', error);
987                 this.isPluginActive = false;
988             }
989         });
990
991     } catch (error) {
992         console.error('Error setting up plugin history listener:', error);
993         this.isPluginActive = false;
994     }
995 }
996
997 /**
998 * MÉTODO 9: Verificar si se puede guardar la estrategia
999 * FLUJO DE VALIDACIÓN:
1000 * - Si el plugin está activo, no se puede guardar
1001 * - Si el plugin no está activo, se puede guardar normalmente
1002 */
1003 canSaveStrategy(): boolean {
1004     const canSave = !this.isPluginActive;
1005     return canSave;
1006 }
1007
1008 /**
1009 * MÉTODO 10: Limpiar recursos al destruir el componente
1010 * FLUJO DE LIMPIEZA:
1011 * - Desuscribirse del listener de Firebase
1012 * - Evitar memory leaks
1013 */
1014 ngOnDestroy() {
1015     if (this.pluginSubscription) {
1016         this.pluginSubscription.unsubscribe();
1017         this.pluginSubscription = null;
1018     }
1019 }
1020 }
1021
1022

```

Ø=ÜÁ features\strategy\edit-strategy\components\edit-assets-allowed

Ø=ÜÄ features\strategy\edit-strategy\components\edit-assets-allowed\edit-assets-allowed.component.ts

```
1 import { Component, HostListener, OnInit, Input } from '@angular/core';
2 import { Store } from '@ngrx/store';
3 import { SettingsService } from '../../../../../service/strategy.service';
4 import {
5   assetsAllowed,
6   daysAllowed,
7   selectMaxDailyTrades,
8 } from '../../../../../store/strategy.selectors';
9 import {
10   AssetsAllowedConfig,
11   Days,
12   DaysAllowedConfig,
13   MaxDailyTradesConfig,
14   RuleType,
15 } from '../../../../../models/strategy.model';
16 import {
17   setAssetsAllowedConfig,
18   setDaysAllowedConfig,
19   setMaxDailyTradesConfig,
20 } from '../../../../../store/strategy.actions';
21 import { CommonModule } from '@angular/common';
22
23 @Component({
24   selector: 'app-edit-assets-allowed',
25   templateUrl: './edit-assets-allowed.component.html',
26   styleUrls: ['./edit-assets-allowed.component.scss'],
27   imports: [CommonModule],
28   standalone: true,
29 })
30 export class EditAssetsAllowedComponent implements OnInit {
31   @Input() availableSymbolsOptions: string[] = [];
32
33   config: AssetsAllowedConfig = {
34     isActive: false,
35     type: RuleType.ASSETS_ALLOWED,
36     assetsAllowed: ['XMRUSD', 'BTCUSD'],
37   };
38
39   symbols: string[] = [];
40   selectedInstrument: string | undefined = undefined;
41   searchTerm: string = '';
42
43   dropdownOpen = false;
44
45   // Estado de validación
46   isValid: boolean = true;
47   errorMessage: string = '';
48
49   constructor(
50     private store: Store,
51     private settingsService: SettingsService
52   ) {}
53
54   closeDropdown() {
55     this.dropdownOpen = false;
56   }
57
58   toggleDropdown() {
59     if (this.config.isActive) {
60       this.dropdownOpen = !this.dropdownOpen;
61     } else {
62       this.dropdownOpen = false;
63     }
64 }
```

```

65
66    onSearchInput(event: Event) {
67        this.searchTerm = (event.target as HTMLInputElement).value;
68        this.dropdownOpen = true;
69    }
70
71    onSearchFocus() {
72        if (this.config.isActive) {
73            this.dropdownOpen = true;
74        }
75    }
76
77    onSearchBlur() {
78        // Delay para permitir click en dropdown
79        setTimeout(() => {
80            this.dropdownOpen = false;
81        }, 200);
82    }
83
84    selectInstrument(instrument: string) {
85        this.selectedInstrument = instrument;
86        this.addSymbol(instrument);
87        this.dropdownOpen = false;
88        this.selectedInstrument = undefined;
89        this.searchTerm = ''; // Limpiar búsqueda
90    }
91
92    getFilteredSymbols(): string[] {
93        if (!this.searchTerm) {
94            return this.availableSymbolsOptions;
95        }
96        return this.availableSymbolsOptions.filter(symbol =>
97            symbol.toLowerCase().includes(this.searchTerm.toLowerCase())
98        );
99    }
100
101   ngOnInit(): void {
102       this.listenRuleConfiguration();
103   }
104
105   addSymbol(symbol: string) {
106       if (symbol && !this.symbols.includes(symbol)) {
107           this.symbols = [...this.symbols, symbol];
108       }
109       this.updateConfig({
110           ...this.config,
111           assetsAllowed: this.symbols,
112       });
113   }
114
115   removeSymbol(symbol: string) {
116       if (this.config.isActive) {
117           this.symbols = this.symbols.filter((s) => s !== symbol);
118           this.updateConfig({
119               ...this.config,
120               assetsAllowed: this.symbols,
121           });
122       }
123   }
124
125   onToggleActive(event: Event) {
126       const isActive = (event.target as HTMLInputElement).checked;
127       const newConfig = {
128           ...this.config,
129           isActive: isActive,
130           // Reiniciar assets cuando se desactiva
131           assetsAllowed: isActive ? this.config.assetsAllowed : [],
132       };
133
134       // Reiniciar símbolos seleccionados

```

```

135     if (!isActive) {
136         this.symbols = [];
137     }
138
139     this.updateConfig(newConfig);
140 }
141
142 listenRuleConfiguration() {
143     this.store
144         .select(assetsAllowed)
145         .pipe()
146         .subscribe((config) => {
147             this.config = { ...config };
148             this.symbols = this.config.assetsAllowed;
149
150             // Validar la configuración después de actualizarla
151             this.validateConfig(this.config);
152         });
153 }
154
155 private updateConfig(config: AssetsAllowedConfig) {
156     this.store.dispatch(setAssetsAllowedConfig({ config }));
157     this.validateConfig(config);
158 }
159
160 private validateConfig(config: AssetsAllowedConfig) {
161
162     if (!config.isActive) {
163         this.isValid = true;
164         this.errorMessage = '';
165         return;
166     }
167
168     if (!config.assetsAllowed || config.assetsAllowed.length === 0) {
169         this.isValid = false;
170         this.errorMessage = 'You must select at least one asset';
171     } else {
172         this.isValid = true;
173         this.errorMessage = '';
174     }
175 }
176
177 // Método público para verificar si la regla es válida
178 public isRuleValid(): boolean {
179     return this.isValid;
180 }
181
182 // Método público para obtener el mensaje de error
183 public getErrorMessage(): string {
184     return this.errorMessage;
185 }
186
187 }
188

```

Ø=ÜÁ features\strategy\edit-strategy\components\edit-days-allowed

Ø=ÜÁ features\strategy\edit-strategy\components\edit-days-allowed\edit-days-allowed.component.ts

```

1 import { Component, OnInit } from '@angular/core';
2 import { Store } from '@ngrx/store';
3 import { SettingsService } from '../../../../../service/strategy.service';
4 import {

```

```

5     daysAllowed,
6     selectMaxDailyTrades,
7   } from '../../../../../store/strategy.selectors';
8   import {
9     Days,
10    DaysAllowedConfig,
11    MaxDailyTradesConfig,
12    RuleType,
13  } from '../../../../../models/strategy.model';
14  import {
15    setDaysAllowedConfig,
16    setMaxDailyTradesConfig,
17  } from '../../../../../store/strategy.actions';
18  import { CommonModule } from '@angular/common';
19
20  @Component({
21    selector: 'app-edit-days-allowed',
22    templateUrl: './edit-days-allowed.component.html',
23    styleUrls: ['./edit-days-allowed.component.scss'],
24    imports: [CommonModule],
25    standalone: true,
26  })
27  export class EditDaysAllowedComponent implements OnInit {
28    config: DaysAllowedConfig = {
29      isActive: false,
30      type: RuleType.DAYS_ALLOWED,
31      tradingDays: [],
32    };
33
34    daysButtons = [
35      { day: Days.MONDAY, isActive: false },
36      { day: Days.TUESDAY, isActive: false },
37      { day: Days.WEDNESDAY, isActive: false },
38      { day: Days.THURSDAY, isActive: false },
39      { day: Days.FRIDAY, isActive: false },
40      { day: Days.SATURDAY, isActive: false },
41      { day: Days.SUNDAY, isActive: false },
42    ];
43
44    // Estado de validación
45    isValid: boolean = true;
46    errorMessage: string = '';
47
48    constructor(private store: Store, private settingsService: SettingsService) {}
49
50    ngOnInit(): void {
51      this.listenRuleConfiguration();
52    }
53    onToggleActive(event: Event) {
54      const isActive = (event.target as HTMLInputElement).checked;
55      const newConfig = {
56        ...this.config,
57        isActive: isActive,
58        // Reiniciar días cuando se desactiva
59        tradingDays: isActive ? this.config.tradingDays : [],
60      };
61
62      // Reiniciar botones de días
63      if (!isActive) {
64        this.daysButtons.forEach(day => {
65          day.isActive = false;
66        });
67      }
68
69      this.updateConfig(newConfig);
70    }
71
72    onChangeValue(day: { day: Days; isActive: boolean }) {
73      if (this.config.isActive) {
74        this.daysButtons.forEach((d) => {

```

```

75         if (d.day === day.day) {
76             d.isActive = !d.isActive;
77         }
78     });
79
80     const newConfig: DaysAllowedConfig = {
81         ...this.config,
82         tradingDays: this.transformDaysActive(),
83     };
84     this.updateConfig(newConfig);
85 }
86
87 transformDaysActive(): string[] {
88     let daysArr: string[] = [];
89
90     this.daysButtons.forEach((d) => {
91         if (d.isActive) {
92             daysArr.push(d.day);
93         }
94     });
95
96
97     return daysArr;
98 }
99
100 listenRuleConfiguration() {
101     this.store
102         .select(daysAllowed)
103         .pipe()
104         .subscribe((config) => {
105             this.config = config;
106             this.daysButtons.forEach((dayOption) => {
107                 const findedDay = config.tradingDays.find((d) => d === dayOption.day);
108                 if (findedDay) {
109                     dayOption.isActive = true;
110                 }
111             });
112
113             // Validar la configuración después de actualizarla
114             this.validateConfig(this.config);
115         });
116     }
117
118     private updateConfig(config: DaysAllowedConfig) {
119         this.store.dispatch(setDaysAllowedConfig({ config }));
120         this.validateConfig(config);
121     }
122
123     private validateConfig(config: DaysAllowedConfig) {
124         if (!config.isActive) {
125             this.isValid = true;
126             this.errorMessage = '';
127             return;
128         }
129
130         if (!config.tradingDays || config.tradingDays.length === 0) {
131             this.isValid = false;
132             this.errorMessage = 'You must select at least one day';
133         } else {
134             this.isValid = true;
135             this.errorMessage = '';
136         }
137     }
138
139     // Método público para verificar si la regla es válida
140     public isRuleValid(): boolean {
141         return this.isValid;
142     }
143
144     // Método público para obtener el mensaje de error

```

```

145     public getErrorMessage(): string {
146         return this.errorMessage;
147     }
148 }
149

```

Ø=ÜÁ features\strategy\edit-strategy\components\edit-hours-allowed

Ø=ÜÁ features\strategy\edit-strategy\components\edit-hours-allowed\edit-hours-allowed.component.ts

```

1 import { Component, OnInit } from '@angular/core';
2 import { AlertService } from '../../../../../shared/services/alert.service';
3 import { Store } from '@ngrx/store';
4 import { SettingsService } from '../../../../../service/strategy.service';
5 import {
6     hoursAllowed,
7     selectMaxDailyTrades,
8 } from '../../../../../store/strategy.selectors';
9 import {
10     HoursAllowedConfig,
11     MaxDailyTradesConfig,
12     RuleType,
13 } from '../../../../../models/strategy.model';
14 import {
15     setHoursAllowedConfig,
16     setMaxDailyTradesConfig,
17 } from '../../../../../store/strategy.actions';
18 import { CommonModule } from '@angular/common';
19 import { FormsModule } from '@angular/forms';
20 import { NgxMaterialTimepickerModule } from 'ngx-material-timepicker';
21 import * as moment from 'moment-timezone';
22 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
23
24 @Component({
25     selector: 'app-edit-hours-allowed',
26     templateUrl: './edit-hours-allowed.component.html',
27     styleUrls: ['./edit-hours-allowed.component.scss'],
28     imports: [CommonModule, FormsModule, NgxMaterialTimepickerModule],
29     standalone: true,
30 })
31 export class EditHoursAllowedComponent implements OnInit {
32     config: HoursAllowedConfig = {
33         isActive: false,
34         tradingOpenTime: '', // Inicialmente vacío para mostrar el placeholder
35         tradingCloseTime: '', // Inicialmente vacío para mostrar el placeholder
36         timezone: '', // Inicialmente vacío para mostrar el placeholder
37         type: RuleType.TRAILING_HOURS,
38     };
39
40     // Placeholder opcional para el select de timezone
41     timezonePlaceholder: string = 'Select a timezone';
42
43     // Estado de validación
44     isValid: boolean = true;
45     errorMessage: string = '';
46
47     timezones = [
48         { value: 'Pacific/Auckland', label: 'Auckland (GMT+12:00)' },
49         { value: 'Australia/Sydney', label: 'Sydney (GMT+10:00)' },
50         { value: 'Australia/Melbourne', label: 'Melbourne (GMT+10:00)' },
51         { value: 'Asia/Tokyo', label: 'Tokyo (GMT+09:00)' },
52         { value: 'Asia/Seoul', label: 'Seoul (GMT+09:00)' },
53         { value: 'Asia/Shanghai', label: 'Shanghai (GMT+08:00)' },

```

```

54     { value: 'Asia/Hong_Kong', label: 'Hong Kong (GMT+08:00)' },
55     { value: 'Asia/Singapore', label: 'Singapore (GMT+08:00)' },
56     { value: 'Asia/Manila', label: 'Manila (GMT+08:00)' },
57     { value: 'Asia/Bangkok', label: 'Bangkok (GMT+07:00)' },
58     { value: 'Asia/Jakarta', label: 'Jakarta (GMT+07:00)' },
59     { value: 'Asia/Kolkata', label: 'Mumbai (GMT+05:30)' },
60     { value: 'Asia/Dubai', label: 'Dubai (GMT+04:00)' },
61     { value: 'Europe/Moscow', label: 'Moscow (GMT+03:00)' },
62     { value: 'Africa/Cairo', label: 'Cairo (GMT+02:00)' },
63     { value: 'Africa/Johannesburg', label: 'Johannesburg (GMT+02:00)' },
64     { value: 'Europe/Paris', label: 'Paris (GMT+01:00)' },
65     { value: 'Europe/Berlin', label: 'Berlin (GMT+01:00)' },
66     { value: 'Europe/Madrid', label: 'Madrid (GMT+01:00)' },
67     { value: 'Europe/Rome', label: 'Rome (GMT+01:00)' },
68     { value: 'UTC', label: 'UTC (GMT+00:00)' },
69     { value: 'Europe/London', label: 'London (GMT+00:00)' },
70     { value: 'America/Sao_Paulo', label: 'São Paulo (GMT-03:00)' },
71     { value: 'America/New_York', label: 'New York (GMT-05:00)' },
72     { value: 'America/Toronto', label: 'Toronto (GMT-05:00)' },
73     { value: 'America/Chicago', label: 'Chicago (GMT-06:00)' },
74     { value: 'America/Mexico_City', label: 'Mexico City (GMT-06:00)' },
75     { value: 'America/Denver', label: 'Denver (GMT-07:00)' },
76     { value: 'America/Los_Angeles', label: 'Los Angeles (GMT-08:00)' },
77     { value: 'America/Vancouver', label: 'Vancouver (GMT-08:00)' }
78   ];
79
80   constructor(private store: Store, private settingsService: SettingsService, private
81 alertService: AlertService) {}
82   ngOnInit(): void {
83     this.listenRuleConfiguration();
84   }
85
86   onToggleActive(event: Event) {
87     const isActive = (event.target as HTMLInputElement).checked;
88     const newConfig = {
89       ...this.config,
90       isActive: isActive,
91       // Reiniciar valores cuando se desactiva
92       tradingOpenTime: isActive ? this.config.tradingOpenTime : '', // Vacío para mostrar
93       placeholderTradingCloseTime: isActive ? this.config.tradingCloseTime : '', // Vacío para mostrar
94       placeholderTimezone: isActive ? this.config.timezone : '', // Vacío para mostrar placeholder
95     };
96     this.updateConfig(newConfig);
97   }
98   onTimezoneChange(newTz: string) {
99     // Validar que la timezone sea válida
100    if (this.isValidTimezone(newTz)) {
101      const newConfig = { ...this.config, timezone: newTz };
102      this.updateConfig(newConfig);
103    } else {
104      console.warn('Invalid timezone selected:', newTz);
105    }
106  }
107
108  isValidTimezone(timezone: string): boolean {
109    return this.timezones.some(tz => tz.value === timezone);
110  }
111
112  getTimezoneWithCountry(timezone: string): string {
113    // Buscar la zona horaria en la lista para obtener el label completo
114    const timezoneObj = this.timezones.find(tz => tz.value === timezone);
115    return timezoneObj ? timezoneObj.label : timezone;
116  }
117
118  onTimeChange(field: 'tradingOpenTime' | 'tradingCloseTime', value: string) {
119    const tempConfig = { ...this.config, [field]: value };
120
121    // Solo validar si ambos valores están presentes
122    if (tempConfig.tradingOpenTime && tempConfig.tradingCloseTime) {
123      const openMinutes = this.toMinutes(tempConfig.tradingOpenTime);

```

```

124     const closeMinutes = this.toMinutes(tempConfig.tradingCloseTime);
125
126     if (openMinutes >= closeMinutes) {
127         this.alertService.showWarning('Opening time must be earlier than closing time.', 'Invalid Time Range');
128     }
129
130     if (closeMinutes - openMinutes < 30) {
131         this.alertService.showWarning('There must be at least a 30-minute difference between opening and closing times.', 'Minimum Time Difference');
132     }
133 }
134
135 this.updateConfig(tempConfig);
136 }
137
138 listenRuleConfiguration() {
139     this.store
140         .select(hoursAllowed)
141         .pipe()
142         .subscribe((config) => {
143             // Si la configuración no está activa o es la primera vez, usar valores vacíos para
144             // mostrar el placeholder
145             if (!config.isActive) {
146                 this.config = {
147                     ...config,
148                     tradingOpenTime: '',
149                     tradingCloseTime: '',
150                     timezone: ''
151                 };
152             } else {
153                 this.config = { ...config };
154             }
155
156             // Validar la configuración después de actualizarla
157             this.validateConfig(this.config);
158         });
159     }
160
161     private updateConfig(config: HoursAllowedConfig) {
162         this.store.dispatch(setHoursAllowedConfig({ config }));
163         this.validateConfig(config);
164     }
165
166     private validateConfig(config: HoursAllowedConfig) {
167
168         if (!config.isActive) {
169             this.isValid = true;
170             this.errorMessage = '';
171             return;
172         }
173
174         const missingFields = [];
175
176         if (!config.tradingOpenTime || config.tradingOpenTime.trim() === '') {
177             missingFields.push('start time');
178         }
179
180         if (!config.tradingCloseTime || config.tradingCloseTime.trim() === '') {
181             missingFields.push('end time');
182         }
183
184         if (!config.timezone || config.timezone.trim() === '') {
185             missingFields.push('timezone');
186         }
187
188         if (missingFields.length > 0) {
189             this.isValid = false;
190             this.errorMessage = `Please fill in the following fields: ${missingFields.join(', ')}`;
191         } else {
192             this.isValid = true;
193             this.errorMessage = '';

```

```

194     }
195   }
196
197   private toMinutes(time: string): number {
198     if (!time || time.trim() === '') {
199       return 0;
200     }
201
202     const is12hFormat =
203       time.toUpperCase().includes('AM') || time.toUpperCase().includes('PM');
204
205     let hours: number;
206     let minutes: number;
207
208     if (is12hFormat) {
209       const [timePart, period] = time.split(' ');
210       [hours, minutes] = timePart.split(':').map(Number);
211       if (period.toUpperCase() === 'PM' && hours !== 12) {
212         hours += 12;
213       }
214       if (period.toUpperCase() === 'AM' && hours === 12) {
215         hours = 0;
216       }
217     } else {
218       [hours, minutes] = time.split(':').map(Number);
219     }
220
221     return hours * 60 + minutes;
222   }
223
224   // Método público para verificar si la regla es válida
225   public isValid(): boolean {
226     return this.isValid;
227   }
228
229   // Método público para obtener el mensaje de error
230   public getMessage(): string {
231     return this.errorMessage;
232   }
233 }
234

```

Ø=ÜÁ features\strategy\edit-strategy\components\edit-max-daily-trades

Ø=ÜÄ features\strategy\edit-strategy\components\edit-max-daily-trades\edit-max-daily-trades.component.ts

```

1 import { Component, OnInit } from '@angular/core';
2 import { Store } from '@ngrx/store';
3 import { SettingsService } from '../../../../../service/strategy.service';
4 import { selectMaxDailyTrades } from '../../../../../store/strategy.selectors';
5 import { MaxDailyTradesConfig, RuleType } from '../../../../../models/strategy.model';
6 import { setMaxDailyTradesConfig } from '../../../../../store/strategy.actions';
7 import { CommonModule } from '@angular/common';
8
9 @Component({
10   selector: 'app-edit-max-daily-trades',
11   templateUrl: './edit-max-daily-trades.component.html',
12   styleUrls: ['./edit-max-daily-trades.component.scss'],
13   imports: [CommonModule],
14   standalone: true,
15 })
16 export class EditMaxDailyTradesComponent implements OnInit {
17   config: MaxDailyTradesConfig = {

```

```

18     isActive: false,
19     maxDailyTrades: 0, // Cambiar el valor por defecto a 0
20     type: RuleType.MAX_DAILY_TRADES,
21   };
22
23   // Estado de validación
24   isValid: boolean = true;
25   errorMessage: string = '';
26
27   constructor(private store: Store, private settingsService: SettingsService) {}
28
29   ngOnInit(): void {
30     this.listenRuleConfiguration();
31   }
32
33   onToggleActive(event: Event) {
34     const isActive = (event.target as HTMLInputElement).checked;
35     const newConfig = {
36       ...this.config,
37       isActive: isActive,
38       // Resetear a 0 cuando se desactiva
39       maxDailyTrades: isActive ? this.config.maxDailyTrades : 0,
40     };
41     this.updateConfig(newConfig);
42   }
43
44   onChangeValue(event: Event) {
45     const numValue = Number((event.target as HTMLInputElement).value);
46     const newConfig: MaxDailyTradesConfig = {
47       ...this.config,
48       maxDailyTrades: numValue < 1 ? 1 : numValue,
49     };
50     this.updateConfig(newConfig);
51   }
52
53   // Métodos para spinner (solo incrementar/decrementar)
54   incrementValue() {
55     if (this.config.isActive) {
56       const newConfig: MaxDailyTradesConfig = {
57         ...this.config,
58         maxDailyTrades: this.config.maxDailyTrades + 1,
59       };
60       this.updateConfig(newConfig);
61     }
62   }
63
64   decrementValue() {
65     if (this.config.isActive && this.config.maxDailyTrades > 0) {
66       const newConfig: MaxDailyTradesConfig = {
67         ...this.config,
68         maxDailyTrades: this.config.maxDailyTrades - 1,
69       };
70       this.updateConfig(newConfig);
71     }
72   }
73
74   listenRuleConfiguration() {
75     this.store
76       .select(selectMaxDailyTrades)
77       .pipe()
78       .subscribe((config) => {
79         // Si la configuración no está activa, usar valor 0 para mostrar placeholder
80         if (!config.isActive) {
81           this.config = {
82             ...config,
83             maxDailyTrades: 0
84           };
85         } else {
86           this.config = config;
87         }

```

```

88         // Validar la configuración después de actualizarla
89         this.validateConfig(this.config);
90     });
91 }
92
93
94 private updateConfig(config: MaxDailyTradesConfig) {
95     this.store.dispatch(setMaxDailyTradesConfig({ config }));
96     this.validateConfig(config);
97 }
98
99 private validateConfig(config: MaxDailyTradesConfig) {
100    if (!config.isActive) {
101        this.isValid = true;
102        this.errorMessage = '';
103        return;
104    }
105
106    if (config.maxDailyTrades <= 0) {
107        this.isValid = false;
108        this.errorMessage = 'You must have at least one trade per day';
109    } else {
110        this.isValid = true;
111        this.errorMessage = '';
112    }
113 }
114
115 // Método público para verificar si la regla es válida
116 public isRuleValid(): boolean {
117     return this.isValid;
118 }
119
120 // Método público para obtener el mensaje de error
121 public getErrorMessage(): string {
122     return this.errorMessage;
123 }
124 }
125

```

Ø=ÜÁ features\strategy\edit-strategy\components\edit-risk-per-trade

Ø=ÜÁ features\strategy\edit-strategy\components\edit-risk-per-trade\edit-risk-per-trade.component.ts

```

1 import { CommonModule } from '@angular/common';
2 import { Component, HostListener, OnInit, ViewChild, ElementRef, Input } from '@angular/
3 import { take } from 'rxjs/operators';
4 import {
5     MaxDailyTradesConfig,
6     RiskPerTradeConfig,
7     RiskRewardConfig,
8     RuleType,
9 } from '../../../../../models/strategy.model';
10 import { Store } from '@ngrx/store';
11 import { SettingsService } from '../../../../../service/strategy.service';
12 import { riskPerTrade, riskReward } from '../../../../../store/strategy.selectors';
13 import {
14     setRiskPerTradeConfig,
15     setRiskRewardConfig,
16 } from '../../../../../store/strategy.actions';
17 import { currencies } from './models/risk-per-trade.model';
18 import { ReportService } from '../../../../../report/service/report.service';
19 import { ApplicationContextService } from '../../../../../shared/context/context';
20 import { AuthService } from '../../../../../auth/service/authService';

```

```

21 import { AccountData } from '../../../../../auth/models/userModel';
22 import { NumberFormatterService } from '../../../../../shared/utils/number-
23 formatter.service';
24 @Component({
25   selector: 'app-edit-risk-per-trade',
26   templateUrl: './edit-risk-per-trade.component.html',
27   styleUrls: ['./edit-risk-per-trade.component.scss'],
28   imports: [CommonModule],
29   standalone: true,
30 })
31 export class EditRiskPerTradeComponent implements OnInit {
32   @Input() userAccounts: AccountData[] = [];
33
34   config: RiskPerTradeConfig = {
35     isActive: false,
36     review_type: 'MAX',
37     number_type: 'PERCENTAGE',
38     percentage_type: 'NULL',
39     risk_ammount: 0,
40     type: RuleType.MAX_RISK_PER_TRADE,
41     balance: 0,
42     actualBalance: 0,
43   };
44
45   // Nuevas propiedades para la lógica del componente
46   selectedSizeType: 'max-size' | 'fixed' | null = null;
47   selectedCalculationType: 'by percentage' | 'by price' | null = null;
48   selectedBalanceType: 'by actual balance' | 'by initial balance' | null = null;
49
50   // Valores del balance actual (se obtiene del servicio)
51   actualBalance: number = 0;
52   initialBalance: number = 0;
53   // Balances por cuenta
54   accountActualBalances: Record<string, number> = {};
55   selectedAccount: AccountData | null = null;
56
57   // Variables para el balance inicial
58   initialBalanceValue: number = 0;
59   displayInitialBalanceValue: string = '';
60   isInitialBalanceEditing: boolean = true; // Iniciar en modo edición
61   isInitialBalanceConfirmed: boolean = false;
62
63   // Valores de entrada
64   percentageValue: number = 0;
65   priceValue: number = 0;
66
67   // Valor mostrado en el input de precio (con formato)
68   displayPriceValue: string = '';
69
70   // Valor del input de precio sin formato (para edición)
71   priceInputValue: string = '';
72
73   // Propiedad para el precio formateado
74   price: string = '';
75
76   inputFirstRatioValue: number = 0;
77   inputSecondRatioValue: number = 0;
78   initialRiskTrade: number | undefined;
79   selectedCurrency = currencies[0];
80   dropdownOpen = false;
81   currencies = currencies;
82
83   // Estado de validación
84   isValid: boolean = true;
85   errorMessage: string = '';
86
87   // Rastrear el estado inicial de Firebase
88   private initialFirebaseState: boolean | null = null;
89
90   // Control de peticiones para evitar llamadas infinitas

```

```

91     private accountLoadAttempts: number = 0;
92     private maxAccountLoadAttempts: number = 2;
93
94     // Referencia al dropdown de cuenta
95     @ViewChild('accountSelect') accountSelect?: ElementRef<HTMLSelectElement>;
96
97     constructor(
98         private store: Store,
99         private settingsService: SettingsService,
100        private reportService: ReportService,
101        private applicationContext: ApplicationContextService,
102        private authService: AuthService,
103        private numberFormatter: NumberFormatterService
104    ) {}
105
106    closeDropdown() {
107        this.dropdownOpen = false;
108    }
109
110    ngOnInit(): void {
111        // Capturar el estado inicial ANTES de suscribirse al observable
112        this.captureInitialFirebaseState();
113        this.listenRuleConfiguration();
114
115        // Inicializar cuentas si vienen como input
116        this.initializeUserAccounts();
117    }
118
119    /**
120     * Inicializar cuentas del usuario desde el input
121     */
122    private initializeUserAccounts(): void {
123        if (this.userAccounts && this.userAccounts.length > 0) {
124            // Las cuentas ya vienen cargadas desde el componente padre
125            console.log('User accounts loaded from parent:', this.userAccounts);
126        } else {
127            // Si no hay cuentas, cargar desde el servicio (fallback)
128            this.loadUserAccounts();
129        }
130    }
131
132
133    /**
134     * Capturar el estado inicial de Firebase desde el store actual
135     * Esto evita que se capture múltiples veces durante las emisiones del observable
136     */
137    private captureInitialFirebaseState(): void {
138        // Usar take(1) para obtener solo el primer valor y luego desuscribirse automáticamente
139        this.store.select(riskPerTrade).pipe(take(1)).subscribe(config => {
140            this.initialFirebaseState = config.isActive;
141        });
142    }
143
144    toggleDropdown() {
145        if (this.config.isActive) {
146            this.dropdownOpen = !this.dropdownOpen;
147        } else {
148            this.dropdownOpen = false;
149        }
150    }
151
152    selectCurrency(currency: { code: string; country: string }) {
153        this.selectedCurrency = currency;
154        this.dropdownOpen = false;
155    }
156
157    onToggleActive(event: Event) {
158        const isActive = (event.target as HTMLInputElement).checked;
159
160        if (!isActive) {

```

```

161     // Si se desactiva, resetear todos los valores a 0
162     const newConfig = {
163         ...this.config,
164         isActive: false,
165         balance: 0,
166         actualBalance: 0,
167         risk_ammount: 0,
168         review_type: 'MAX' as const,
169         number_type: 'PERCENTAGE' as const,
170         percentage_type: 'NULL' as const,
171     };
172     this.updateConfig(newConfig);
173 } else {
174     // Si se activa, mantener la configuración actual
175     const newConfig = {
176         ...this.config,
177         isActive: true,
178     };
179     this.updateConfig(newConfig);
180 }
181 }
182
183 // Nuevos métodos para manejar las opciones
184 selectSizeType(type: 'max-size' | 'fixed') {
185     this.selectedSizeType = type;
186     this.selectedCalculationType = null;
187     this.selectedBalanceType = null;
188     this.saveConfiguration();
189 }
190
191 selectCalculationType(type: 'by percentage' | 'by price') {
192     this.selectedCalculationType = type;
193
194     // Resetear todos los campos relacionados cuando se cambia el tipo de cálculo
195     this.selectedBalanceType = null;
196     this.selectedAccount = null;
197     this.percentageValue = 0;
198     this.priceValue = 0;
199     this.displayPriceValue = '';
200     this.priceInputValue = '';
201     this.initialBalance = 0;
202     this.initialBalanceValue = 0;
203     this.displayInitialBalanceValue = '';
204     this.isInitialBalanceConfirmed = false;
205     this.isInitialBalanceEditing = true;
206
207     // Resetear el dropdown para mostrar el placeholder
208     setTimeout(() => {
209         if (this.accountSelect) {
210             this.accountSelect.nativeElement.selectedIndex = 0; // Seleccionar la primera opción
211             (placeholder)
212         }, 0);
213
214         this.saveConfiguration();
215     }
216
217 selectBalanceType(type: 'by actual balance' | 'by initial balance') {
218     this.selectedBalanceType = type;
219
220     // Resetear cuenta seleccionada y valores cuando se cambia el tipo de balance
221     this.selectedAccount = null;
222     this.percentageValue = 0;
223     this.priceValue = 0;
224     this.displayPriceValue = '';
225     this.priceInputValue = '';
226     this.initialBalance = 0;
227     this.initialBalanceValue = 0;
228     this.displayInitialBalanceValue = '';
229     this.isInitialBalanceConfirmed = false;
230     this.isInitialBalanceEditing = true;

```

```

231
232     // Resetear el dropdown para mostrar el placeholder
233     setTimeout(() => {
234         if (this.accountSelect) {
235             this.accountSelect.nativeElement.selectedIndex = 0; // Seleccionar la primera opción
236             (placeholder)
237             , 0);
238
239         // Solo cargar el balance actual cuando el usuario seleccione "by actual balance"
240         if (type === 'by actual balance') {
241             if (this.userAccounts.length > 0) {
242                 this.loadActualBalancesForAccounts();
243             } else {
244                 this.loadUserAccounts().then(() => this.loadActualBalancesForAccounts());
245             }
246         } else {
247             // by initial balance -> las cuentas ya están disponibles desde el input
248             if (this.userAccounts.length === 0) {
249                 this.loadUserAccounts();
250             }
251         }
252
253         this.saveConfiguration();
254     }
255
256     async loadUserAccounts() {
257         try {
258             // Preferir UID de Firebase
259             const firebaseUser = this.authService.getAuth().currentUser;
260             const userId = firebaseUser?.uid || this.appContext.currentUser()?.id;
261             if (!userId) {
262                 this.userAccounts = [];
263                 return;
264             }
265             const accounts = await this.authService.getUserAccounts(userId);
266             this.userAccounts = accounts || [];
267         } catch {
268             this.userAccounts = [];
269         }
270     }
271
272     async loadActualBalancesForAccounts() {
273         try {
274             const balances: Record<string, number> = {};
275             for (const acc of this.userAccounts) {
276                 try {
277                     // Obtener token de TradeLocker por cuenta (igual que en ReportService)
278                     const token = await this.reportService
279                         .getUserKey(acc.emailTradingAccount, acc.brokerPassword, acc.server)
280                         .toPromise();
281                     if (!token) {
282                         balances[acc.accountID] = 0;
283                         continue;
284                     }
285                     // Con ese token, obtener el balance de la cuenta
286                     const data = await this.reportService
287                         .getBalanceData(acc.accountID, token, acc.accountNumber)
288                         .toPromise();
289                     balances[acc.accountID] = data?.balance || 0;
290                 } catch {
291                     balances[acc.accountID] = 0;
292                 }
293             }
294             this.accountActualBalances = balances;
295         } catch {
296             this.accountActualBalances = {};
297         }
298     }
299
300     onSelectAccount(event: Event) {

```

```

301 const accountID = (event.target as HTMLSelectElement).value;
302 this.selectedAccount = this.userAccounts.find(a => a.accountID === accountID) || null;
303
304 // Asegurar que se tengan los tipos necesarios para mostrar el input de porcentaje
305 if (!this.selectedSizeType) {
306   this.selectedSizeType = 'max-size'; // Valor por defecto
307 }
308 if (!this.selectedCalculationType) {
309   this.selectedCalculationType = 'by percentage'; // Valor por defecto
310 }
311
312 // Ajustar balances visibles según selección
313 if (this.selectedBalanceType === 'by initial balance') {
314   this.initialBalance = this.selectedAccount?.initialBalance || 0;
315   // Mostrar inmediatamente el balance inicial seleccionado en los resúmenes
316   this.isInitialBalanceConfirmed = true;
317   this.isInitialBalanceEditing = false;
318 } else if (this.selectedBalanceType === 'by actual balance' && this.selectedAccount) {
319   this.actualBalance = this.accountActualBalances[this.selectedAccount.accountID] || 0;
320 }
321 this.saveConfiguration();
322 }
323
324 onPercentageChange(event: Event) {
325   this.percentageValue = Number((event.target as HTMLInputElement).value);
326   this.saveConfiguration();
327 }
328
329 formatCurrency(event: any) {
330   const input = event.target.value;
331   const formatted = this.numberFormatter.formatInputValue(input);
332
333   if (formatted === '') {
334     this.price = '';
335     this.priceValue = 0;
336     this.displayPriceValue = '';
337     return;
338   }
339
340   // Convertir a número
341   const value = parseFloat(this.numberFormatter.cleanNumericInput(input));
342   if (isNaN(value)) {
343     this.price = '';
344     this.priceValue = 0;
345     this.displayPriceValue = '';
346     return;
347   }
348
349   this.priceValue = value;
350   this.price = formatted;
351   this.displayPriceValue = this.price;
352   this.saveConfiguration();
353 }
354
355 onPriceInput(event: Event) {
356   const target = event.target as HTMLInputElement;
357   this.priceInputValue = target.value;
358 }
359
360 onPriceFocus() {
361   // Cuando el usuario hace focus, mostrar solo el número sin formato para edición
362   if (this.priceValue > 0) {
363     this.priceInputValue = this.priceValue.toString();
364   }
365 }
366
367 onPriceBlur() {
368   // Convertir el valor a número usando el servicio centralizado
369   const numericValue = this.numberFormatter.parseCurrencyValue(this.priceInputValue);
370   if (!isNaN(numericValue) && numericValue > 0) {

```

```

371     // Guardar el valor sin formato
372     this.priceValue = numericValue;
373
374     // Mostrar formato visual (solo para display)
375     this.displayPriceValue = this.numberFormatter.formatCurrencyDisplay(numericValue);
376
377     // Actualizar el input para mostrar el formato visual
378     this.priceInputValue = this.displayPriceValue;
379
380     this.saveConfiguration();
381 } else {
382     // Si no es un número válido, limpiar
383     this.priceInputValue = '';
384     this.displayPriceValue = '';
385     this.priceValue = 0;
386 }
387 }
388
389 onBlur() {
390     // Método legacy - mantener para compatibilidad
391     this.onPriceBlur();
392 }
393
394 // Métodos para el balance inicial
395 formatInitialBalance(event: any) {
396     const input = event.target.value;
397     const formatted = this.numberFormatter.formatInputValue(input);
398
399     if (formatted === '') {
400         this.initialBalanceValue = 0;
401         this.displayInitialBalanceValue = '';
402         return;
403     }
404
405     // Convertir a número
406     const value = parseFloat(this.numberFormatter.cleanNumericInput(input));
407     if (isNaN(value)) {
408         this.initialBalanceValue = 0;
409         this.displayInitialBalanceValue = '';
410         return;
411     }
412
413     this.initialBalanceValue = value;
414     this.displayInitialBalanceValue = formatted;
415 }
416
417 onInitialBalanceBlur() {
418     if (!this.displayInitialBalanceValue) return;
419
420     const num = this.numberFormatter.parseCurrencyValue(this.displayInitialBalanceValue);
421     if (!isNaN(num)) {
422         this.initialBalanceValue = num;
423         this.displayInitialBalanceValue = this.numberFormatter.formatNumber(num, 2);
424     }
425 }
426
427 clearInitialBalance() {
428     this.initialBalanceValue = 0;
429     this.displayInitialBalanceValue = '';
430     this.initialBalance = 0;
431     this.isInitialBalanceConfirmed = false;
432     this.isInitialBalanceEditing = true; // Habilitar edición después de borrar
433 }
434
435 editInitialBalance() {
436     this.isInitialBalanceEditing = true;
437     this.isInitialBalanceConfirmed = false;
438 }
439
440 confirmInitialBalance() {

```

```

441     this.initialBalance = this.initialBalanceValue;
442     this.isInitialBalanceEditing = false;
443     this.isInitialBalanceConfirmed = true;
444     this.saveConfiguration();
445   }
446
447   // Método legacy para cargar un balance actual (se mantiene por compatibilidad, ahora
448   // usamos loadActualBalancesForAccounts)
449   try {
450     // Obtener el usuario actual de Firebase directamente
451     const currentUser = this.authService.getAuth().currentUser;
452     if (!currentUser) {
453       this.actualBalance = 0;
454       return;
455     }
456
457     // Obtener el token fresco
458     const accessToken = await currentUser.getIdToken(true); // true = force refresh
459
460     // Obtener las cuentas del usuario desde el contexto
461     const userAccounts = this.appContext.userAccounts();
462     if (!userAccounts || userAccounts.length === 0) {
463       this.actualBalance = 0;
464       return;
465     }
466
467     const account = userAccounts[0];
468
469     // Hacer la petición al servicio de reportes para obtener el balance
470     const balanceData = await this.reportService.getBalanceData(
471       account.accountID,
472       accessToken,
473       account.accountNumber
474     ).toPromise();
475
476     if (balanceData && balanceData.balance) {
477       this.actualBalance = balanceData.balance;
478       // Actualizar el contexto con los datos obtenidos
479       this.appContext.updateReportBalance(balanceData);
480     } else {
481       this.actualBalance = 0;
482     }
483   } catch (error) {
484     console.error('Error loading actual balance:', error);
485     this.actualBalance = 0;
486   }
487 }
488
489 getCurrentBalance(): number {
490   if (this.selectedBalanceType === 'by actual balance') {
491     return this.actualBalance;
492   } else if (this.selectedBalanceType === 'by initial balance') {
493     return this.isInitialBalanceConfirmed ? this.initialBalance : 0;
494   }
495   return 0;
496 }
497
498 getCalculatedAmount(): number {
499   const balance = this.getCurrentBalance();
500   return (this.percentageValue / 100) * balance;
501 }
502
503 onChangePercentage(event: Event) {
504   const percentage = Number((event.target as HTMLInputElement).value);
505   const moneyRisk = Number(
506     ((percentage / 100) * this.getCurrentBalance()).toFixed(2)
507   );
508
509   // Usar la misma lógica que saveConfiguration para determinar el balance
510   let balanceToSave: number;

```

```

511     let actualBalanceToSave: number | undefined;
512     if (this.selectedCalculationType === 'by percentage' && this.selectedBalanceType === 'by
513     initialBalanceToSave = this.selectedAccount?.initialBalance || 0;
514     actualBalanceToSave = 0;
515   } else {
516     balanceToSave = -1;
517     actualBalanceToSave = this.actualBalance;
518   }
519
520   const newConfig: RiskPerTradeConfig = {
521     ...this.config,
522     risk_ammount: percentage,
523     balance: balanceToSave,
524     actualBalance: actualBalanceToSave,
525   };
526   this.updateConfig(newConfig);
527 }
528
529 onChangeAmount(event: Event) {
530   const moneyRisk = Number((event.target as HTMLInputElement).value);
531   const percentage = Number(
532     ((moneyRisk / this.getCurrentBalance()) * 100).toFixed(2)
533   );
534
535   // Cuando es money, siempre guardar 0 en ambos
536   const newConfig: RiskPerTradeConfig = {
537     ...this.config,
538     risk_ammount: moneyRisk,
539     balance: 0,
540     actualBalance: 0,
541   };
542   this.updateConfig(newConfig);
543 }
544
545 listenRuleConfiguration() {
546   this.store
547     .select(riskPerTrade)
548     .pipe()
549     .subscribe((config) => {
550       this.config = config;
551
552       // Resetear contador de intentos cuando se carga una nueva configuración
553       this.accountLoadAttempts = 0;
554
555       // El estado inicial ya fue capturado en ngOnInit, solo procesar la lógica
556
557       if (config.isActive) {
558         // Usar el estado inicial de Firebase para determinar el comportamiento
559         if (this.initialFirebaseState === false) {
560           // Regla vino inactiva de Firebase: SIEMPRE empezar desde cero, ignorar valores
561           de config
562           this.selectedSizeType = null;
563           this.selectedCalculationType = null;
564           this.selectedBalanceType = null;
565           this.percentageValue = 0;
566           this.priceValue = 0;
567           this.displayPriceValue = '';
568           this.priceInputValue = '';
569           this.initialBalance = 0;
570           this.initialBalanceValue = 0;
571           this.displayInitialBalanceValue = '';
572           this.isInitialBalanceConfirmed = false;
573           this.isInitialBalanceEditing = true;
574           this.selectedAccount = null;
575           this.actualBalance = 0;
576         } else {
577           // Regla vino activa de Firebase: cargar valores desde config
578           this.selectedSizeType = config.review_type === 'MAX' ? 'max-size' : 'fixed';
579           this.selectedCalculationType = config.number_type === 'PERCENTAGE' ? 'by
percentage' : 'by price';
580           // Manejar el percentage_type correctamente

```

```

581     if (config.percentage_type === 'NULL') {
582         // Cuando es money, no hay balance type específico
583         this.selectedBalanceType = null;
584     } else {
585         this.selectedBalanceType = config.percentage_type === 'ACTUAL_B' ? 'by actual
balance' : 'by initial balance';
586
587         // Cargar el valor de riesgo
588         if (config.risk_ammount) {
589             if (config.number_type === 'PERCENTAGE') {
590                 this.percentageValue = config.risk_ammount;
591             } else {
592                 this.priceValue = config.risk_ammount;
593                 this.price = config.risk_ammount.toLocaleString('en-US', {
594                     style: 'currency',
595                     currency: 'USD',
596                     minimumFractionDigits: 2,
597                     maximumFractionDigits: 2
598                 });
599                 this.displayPriceValue = this.price;
600                 this.priceInputValue = this.displayPriceValue;
601             }
602         }
603     }
604
605     // Cargar balance y seleccionar cuenta según el tipo
606     if (config.percentage_type === 'INITIAL_B' && config.balance && config.balance >
607         0) {
608         // Configurar el tipo de balance y cálculo para balance inicial
609         this.selectedBalanceType = 'by initial balance';
610         this.selectedCalculationType = 'by percentage';
611
612         // Cargar balance inicial y seleccionar la cuenta correspondiente
613         this.initialBalance = config.balance;
614         this.initialBalanceValue = config.balance;
615
616         this.displayInitialBalanceValue = config.balance.toLocaleString('en-US', {
617             minimumFractionDigits: 2,
618             maximumFractionDigits: 2
619         });
620         this.isInitialBalanceConfirmed = true;
621         this.isInitialBalanceEditing = false;
622
623         // Usar cuentas disponibles o cargarlas si es necesario
624         if (this.userAccounts.length > 0) {
625
626             // Buscar la cuenta que coincida con el balance inicial
627             const matchingAccount = this.userAccounts.find((account) => {
628                 // Asegurar que ambos valores sean números
629                 const accountBalance = Number(account.initialBalance) || 0;
630                 const configBalance = Number(config.balance) || 0;
631                 const difference = Math.abs(accountBalance - configBalance);
632                 return difference < 0.01; // Tolerancia para decimales
633             });
634
635             if (matchingAccount) {
636                 this.selectedAccount = matchingAccount;
637                 // Seleccionar en el dropdown
638                 setTimeout(() => {
639                     if (this.accountSelect) {
640                         const selectElement = this.accountSelect.nativeElement;
641                         const optionIndex = this.userAccounts.findIndex(account =>
642                             account.accountID === matchingAccount.accountID
643                         );
644                         if (optionIndex !== -1) {
645                             selectElement.selectedIndex = optionIndex + 1;
646                         }
647                     }, 100);
648             } else {
649                 this.selectedAccount = null;
650                 this.isValid = false;

```

```

651         this.errorMessage = 'Please complete this rule. The saved balance does not
652 match any available account.';
653     } else {
654         // Controlar intentos de carga para evitar peticiones infinitas
655         if (this.accountLoadAttempts < this.maxAccountLoadAttempts) {
656             this.accountLoadAttempts++;
657             this.loadUserAccounts().then(() => {
658                 // Buscar la cuenta que coincida con el balance inicial
659                 const matchingAccount = this.userAccounts.find((account) => {
660                     // Asegurar que ambos valores sean números
661                     const accountBalance = Number(account.initialBalance) || 0;
662                     const configBalance = Number(config.balance) || 0;
663                     const difference = Math.abs(accountBalance - configBalance);
664                     return difference < 0.01; // Tolerancia para decimales
665                 });
666
667                 if (matchingAccount) {
668                     this.selectedAccount = matchingAccount;
669                     // Seleccionar en el dropdown
670                     setTimeout(() => {
671                         if (this.accountSelect) {
672                             const selectElement = this.accountSelect.nativeElement;
673                             const optionIndex = this.userAccounts.findIndex(account =>
674                                 account.accountID === matchingAccount.accountID
675                             );
676                             if (optionIndex !== -1) {
677                                 selectElement.selectedIndex = optionIndex + 1; // +1 porque el
678 primer option es el placeholder
679                             }
680                         }, 100);
681                     } else {
682                         this.selectedAccount = null;
683                         this.isValid = false;
684                         this.errorMessage = 'Please complete this rule. The saved balance does
685 not match any available account.';
686                     }).catch(() => {
687                         // Si falla la carga, mostrar error
688                         this.selectedAccount = null;
689                         this.isValid = false;
690                         this.errorMessage = 'Error loading accounts. Please try again.';
691                     });
692                 } else {
693                     // Máximo de intentos alcanzado, mostrar error
694                     this.selectedAccount = null;
695                     this.isValid = false;
696                     this.errorMessage = 'Please complete this rule. The saved balance does not
697 match any available account.';
698                 }
699
700             } else if (config.percentage_type === 'ACTUAL_B' && config.actualBalance.balance
701 && config.actualBalance.balance !== null) {
702                 // Usar balance y cálculo para balance actual
703                 this.selectedBalanceType = 'by actual balance';
704                 this.selectedCalculationType = 'by percentage';
705
706                 // Cargar balance actual y seleccionar la cuenta correspondiente
707                 this.actualBalance = config.actualBalance.balance || 0;
708
709                 // Usar cuentas disponibles o cargarlas si es necesario
710                 if (this.userAccounts.length > 0) {
711                     this.loadActualBalancesForAccounts().then(() => {
712                         // Buscar la cuenta que tenga el balance actual que coincide
713                         const matchingAccount = this.userAccounts.find(account => {
714                             // Asegurar que ambos valores sean números
715                             const accountBalance =
716                             Number(this.accountActualBalance[account.accountID].actualBalance.balance) || 0;
717                             const difference = Math.abs(accountBalance - configBalance);
718                             return difference < 0.01; // Tolerancia para decimales
719                         });
720
721                         if (matchingAccount) {

```

```

721         this.selectedAccount = matchingAccount;
722         // Seleccionar en el dropdown
723         setTimeout(() => {
724             if (this.accountSelect) {
725                 const selectElement = this.accountSelect.nativeElement;
726                 const optionIndex = this.userAccounts.findIndex(account =>
727                     account.accountID === matchingAccount.accountID
728                 );
729                 if (optionIndex !== -1) {
730                     selectElement.selectedIndex = optionIndex + 1; // +1 porque el
731                     primer option es el placeholder
732                 }
733             }, 100);
734         } else {
735             this.selectedAccount = null;
736             this.isValid = false;
737             this.errorMessage = 'Please complete this rule. The saved balance does
738             not match any available account.';
739         });
740     } else {
741         // Controlar intentos de carga para evitar peticiones infinitas
742         if (this.accountLoadAttempts < this.maxAccountLoadAttempts) {
743             this.accountLoadAttempts++;
744             this.loadUserAccounts().then(() => {
745                 this.loadActualBalancesForAccounts().then(() => {
746                     // Buscar la cuenta que tenga el balance actual que coincida
747                     const matchingAccount = this.userAccounts.find(account => {
748                         // Asegurar que ambos valores sean números
749                         const accountBalance =
750                             Number(this.accountActualBalances[account.accountID]) || config.actualBalance.balance;
751                         const difference = Math.abs(accountBalance - configBalance);
752                         return difference < 0.01; // Tolerancia para decimales
753                     });
754
755                     if (matchingAccount) {
756                         this.selectedAccount = matchingAccount;
757                         // Seleccionar en el dropdown
758                         setTimeout(() => {
759                             if (this.accountSelect) {
760                                 const selectElement = this.accountSelect.nativeElement;
761                                 const optionIndex = this.userAccounts.findIndex(account =>
762                                     account.accountID === matchingAccount.accountID
763                                 );
764                                 if (optionIndex !== -1) {
765                                     selectElement.selectedIndex = optionIndex + 1; // +1 porque el
766                                     primer option es el placeholder
767                                 }
768                             }, 100);
769                         } else {
770                             this.selectedAccount = null;
771                             this.isValid = false;
772                             this.errorMessage = 'Please complete this rule. The saved balance
773                             does not match any available account.';
774                         }).catch(() => {
775                             // Si falla la carga de balances, mostrar error
776                             this.selectedAccount = null;
777                             this.isValid = false;
778                             this.errorMessage = 'Error loading account balances. Please try
779                             again.';
780                         }).catch(() => {
781                             // Si falla la carga de cuentas, mostrar error
782                             this.selectedAccount = null;
783                             this.isValid = false;
784                             this.errorMessage = 'Error loading accounts. Please try again.';
785                         });
786                     } else {
787                         // Máximo de intentos alcanzado, mostrar error
788                         this.selectedAccount = null;
789                         this.isValid = false;
790                         this.errorMessage = 'Please complete this rule. The saved balance does not
791                         match any available account.';
792                     }
793                 });
794             });
795         }
796     }
797 }

```

```

791         }
792     }
793 }
794 }
795 } else {
796     // Si la regla está desactivada, resetear UI a null
797     this.selectedSizeType = null;
798     this.selectedCalculationType = null;
799     this.selectedBalanceType = null;
800     this.percentageValue = 0;
801     this.priceValue = 0;
802     this.displayPriceValue = '';
803     this.priceInputValue = '';
804     this.initialBalance = 0;
805     this.initialBalanceValue = 0;
806     this.displayInitialBalanceValue = '';
807     this.isInitialBalanceConfirmed = false;
808     this.isInitialBalanceEditing = true;
809     this.selectedAccount = null;
810     this.actualBalance = 0;
811 }
812 }
813
814     // Validar la configuración después de actualizarla (solo si no hay selección
815     // automática pendiente) para casos con selección automática se maneja dentro de los
816     setTimeouts(!config.isActive || config.percentage_type != 'INITIAL_B' && config.percentage_type != 'ACTUAL_B') this.validateConfig(this.config);
817     }
818     });
819 }
820 );
821 }
822
823 private updateConfig(config: RiskPerTradeConfig) {
824     this.store.dispatch(setRiskPerTradeConfig({ config }));
825     this.validateConfig(config);
826 }
827
828 private validateConfig(config: RiskPerTradeConfig) {
829     if (!config.isActive) {
830         this.isValid = true;
831         this.errorMessage = '';
832         return;
833     }
834
835     // Validar que se haya seleccionado un tipo de tamaño
836     if (!this.selectedSizeType) {
837         this.isValid = false;
838         this.errorMessage = 'You must select a size type';
839         return;
840     }
841
842     // Validar que se haya seleccionado un tipo de cálculo
843     if (!this.selectedCalculationType) {
844         this.isValid = false;
845         this.errorMessage = 'You must select a calculation type';
846         return;
847     }
848
849     // Validar según el tipo de cálculo seleccionado
850     if (this.selectedCalculationType === 'by percentage') {
851         // Validar que se haya seleccionado un tipo de balance
852         if (!this.selectedBalanceType) {
853             this.isValid = false;
854             this.errorMessage = 'You must select a balance type';
855             return;
856         }
857
858         // Validar que se haya seleccionado una cuenta
859         if (!this.selectedAccount) {
860             this.isValid = false;

```

```

861         this.errorMessage = 'You must select an account';
862         return;
863     }
864
865     // Validar que se haya ingresado un porcentaje válido
866     if (!this.percentageValue || this.percentageValue <= 0) {
867         this.isValid = false;
868         this.errorMessage = 'You must enter a valid percentage value';
869         return;
870     }
871 } else if (this.selectedCalculationType === 'by price') {
872     // Validar que se haya ingresado un precio válido
873     if (!this.priceValue || this.priceValue <= 0) {
874         this.isValid = false;
875         this.errorMessage = 'You must enter a valid price value';
876         return;
877     }
878 }
879
880 // Si llegamos aquí, la validación pasó
881 this.isValid = true;
882 this.errorMessage = '';
883 }
884
885 // Método público para verificar si la regla es válida
886 public isRuleValid(): boolean {
887     return this.isValid;
888 }
889
890 // Método público para obtener el mensaje de error
891 public getErrorMessage(): string {
892     return this.errorMessage;
893 }
894
895 // Método para guardar la configuración con la nueva estructura
896 private saveConfiguration() {
897     if (!this.selectedSizeType || !this.selectedCalculationType) {
898         return; // No guardar si faltan selecciones básicas
899     }
900
901     // Para percentage, también necesitamos selectedBalanceType
902     if (this.selectedCalculationType === 'by percentage' && !this.selectedBalanceType) {
903         return;
904     }
905
906     // Determinar el balance a guardar según la lógica especificada
907     let balanceToSave: number;
908     let actualBalanceToSave: number | undefined;
909     let percentageType: "INITIAL_B" | "ACTUAL_B" | "NULL";
910
911     if (this.selectedCalculationType === 'by percentage') {
912         // Cuando es percentage
913         if (this.selectedBalanceType === 'by initial balance') {
914             // percentage + initialBalance: guardar el balance en balance, actualBalance = 0
915             balanceToSave = this.selectedAccount?.initialBalance || 0;
916             actualBalanceToSave = 0;
917             percentageType = 'INITIAL_B';
918         } else {
919             // percentage + actualBalance: guardar -1 en balance, valor en actualBalance
920             balanceToSave = -1;
921             actualBalanceToSave = this.actualBalance;
922             percentageType = 'ACTUAL_B';
923         }
924     } else {
925         // Cuando es money: guardar 0 en ambos
926         balanceToSave = 0;
927         actualBalanceToSave = 0;
928         percentageType = 'NULL';
929     }
930 }
```

```

931     const newConfig: RiskPerTradeConfig = {
932       ...this.config,
933       review_type: this.selectedSizeType === 'max-size' ? 'MAX' : 'FIXED',
934       number_type: this.selectedCalculationType === 'by percentage' ? 'PERCENTAGE' : 'MONEY',
935       percentage_type: percentageType,
936       risk_amount: this.selectedCalculationType === 'by percentage' ?
937         this.percentageValueToSave,
938         actualBalance: actualBalanceToSave
939     };
940
941     this.updateConfig(newConfig);
942   }
943 }
944

```

Ø=ÜÁ features\strategy\edit-strategy\components\edit-risk-per-trade\models

Ø=ÜÁ features\strategy\edit-strategy\components\edit-risk-per-trade\models\risk-per-trade.model.ts

```

1  export const currencies = [
2    { code: 'USD', country: 'US' },
3    { code: 'EUR', country: 'EU' },
4    { code: 'JPY', country: 'JP' },
5    { code: 'GBP', country: 'GB' },
6    { code: 'AUD', country: 'AU' },
7    { code: 'CAD', country: 'CA' },
8    { code: 'CHF', country: 'CH' },
9    { code: 'CNY', country: 'CN' },
10   { code: 'HKD', country: 'HK' },
11   { code: 'NZD', country: 'NZ' },
12 ];
13

```

Ø=ÜÁ features\strategy\edit-strategy\components\edit-risk-reward

Ø=ÜÁ features\strategy\edit-strategy\components\edit-risk-reward\edit-risk-reward.component.ts

```

1  import { CommonModule } from '@angular/common';
2  import { Component, OnInit } from '@angular/core';
3  import {
4    MaxDailyTradesConfig,
5    RiskRewardConfig,
6    RuleType,
7  } from '../../../../../models/strategy.model';
8  import { Store } from '@ngrx/store';
9  import { SettingsService } from '../../../../../service/strategy.service';
10 import { riskReward } from '../../../../../store/strategy.selectors';
11 import { setRiskRewardConfig } from '../../../../../store/strategy.actions';
12
13 @Component({
14   selector: 'app-edit-risk-reward-ratio',
15   templateUrl: './edit-risk-reward.component.html',
16   styleUrls: ['./edit-risk-reward.component.scss'],
17   imports: [CommonModule],
18   standalone: true,
19 })

```

```

20 export class EditRiskRewardComponent implements OnInit {
21   config: RiskRewardConfig = {
22     isActive: false,
23     riskRewardRatio: '1:2',
24     type: RuleType.RISK_REWARD_RATIO,
25   };
26
27   inputFirstRatioValue: number = 0;
28
29   inputSecondRatioValue: number = 0;
30
31   initialRatio: string | undefined;
32
33   constructor(private store: Store, private settingsService: SettingsService) {}
34
35   ngOnInit(): void {
36     this.listenRuleConfiguration();
37   }
38   onToggleActive(event: Event) {
39     const isActive = (event.target as HTMLInputElement).checked;
40     const newConfig = {
41       ...this.config,
42       isActive,
43       // Reiniciar a 1:2 cuando se desactiva
44       riskRewardRatio: isActive ? this.config.riskRewardRatio : '1:2',
45     };
46     this.updateConfig(newConfig);
47   }
48
49   onChangeValue(event: Event, isFirst: boolean) {
50     const numValue = Number((event.target as HTMLInputElement).value);
51     const numberArray = this.config.riskRewardRatio
52       .split(':')
53       .map((number) => parseInt(number, 10));
54
55     if (isFirst) {
56       numberArray[0] = numValue;
57     } else {
58       numberArray[1] = numValue;
59     }
60
61     const newConfig: RiskRewardConfig = {
62       ...this.config,
63       riskRewardRatio: numberArray.join(':'),
64     };
65     this.updateConfig(newConfig);
66   }
67
68   // Métodos para spinner (solo para el segundo número)
69   incrementSecondValue() {
70     if (this.config.isActive) {
71       const numberArray = this.config.riskRewardRatio
72         .split(':')
73         .map((number) => parseInt(number, 10));
74
75       numberArray[1] = numberArray[1] + 1;
76
77       const newConfig: RiskRewardConfig = {
78         ...this.config,
79         riskRewardRatio: numberArray.join(':'),
80       };
81       this.updateConfig(newConfig);
82     }
83   }
84
85   decrementSecondValue() {
86     if (this.config.isActive) {
87       const numberArray = this.config.riskRewardRatio
88         .split(':')
89         .map((number) => parseInt(number, 10));

```

```

90     if (numberArray[1] > 2) {
91         numberArray[1] = numberArray[1] - 1;
92     }
93
94     const newConfig: RiskRewardConfig = {
95         ...this.config,
96         riskRewardRatio: numberArray.join(''),
97     };
98     this.updateConfig(newConfig);
99 }
100 }
101 }
102
103 listenRuleConfiguration() {
104     this.store
105         .select(riskReward)
106         .pipe()
107         .subscribe((config) => {
108             this.config = config;
109             if (!this.initialRatio) {
110                 this.initialRatio = config.riskRewardRatio;
111             }
112             const numberArray = config.riskRewardRatio
113                 .split(':')
114                 .map((number) => parseInt(number, 10));
115             this.inputFirstRatioValue = numberArray[0];
116             this.inputSecondRatioValue = numberArray[1];
117         });
118     }
119
120     private updateConfig(config: RiskRewardConfig) {
121         this.store.dispatch(setRiskRewardConfig({ config }));
122     }
123 }
124

```

Ø=ÜÁ features\strategy\models

Ø=ÜÁ features\strategy\models\strategy.model.ts

```

1 import { Timestamp } from "firebase/firestore";
2
3 /**
4  * Enum representing the types of trading rules available in the system.
5  *
6  * Each rule type corresponds to a specific trading constraint or configuration.
7  *
8  * @enum {string}
9 */
10 export enum RuleType {
11     ASSETS_ALLOWED = 'ASSETS ALLOWED',
12     DAYS_ALLOWED = 'DAYS ALLOWED',
13     MAX_DAILY_TRADES = 'MAX DAILY TRADES',
14     MAX_RISK_PER_TRADE = 'MAX RISK PER TRADE',
15     RISK_REWARD_RATIO = 'RISK REWARD RATIO',
16     TRADING_HOURS = 'TRADING HOURS',
17 }
18
19 /**
20  * Enum representing the days of the week.
21  *
22  * Used for configuring which days trading is allowed.
23 */

```

```

24     * @enum {string}
25     */
26 export enum Days {
27     MONDAY = 'Monday',
28     TUESDAY = 'Tuesday',
29     WEDNESDAY = 'Wednesday',
30     THURSDAY = 'Thursday',
31     FRIDAY = 'Friday',
32     SATURDAY = 'Saturday',
33     SUNDAY = 'Sunday',
34 }
35 /**
36  * Interface representing the maximum daily trades configuration.
37  *
38  * Limits the number of trades that can be executed per day.
39  *
40  * @interface MaxDailyTradesConfig
41  */
42 export interface MaxDailyTradesConfig {
43     isActive: boolean;
44     type: RuleType;
45     maxDailyTrades: number;
46 }
47 /**
48  * Interface representing the days allowed for trading configuration.
49  *
50  * Specifies which days of the week trading is permitted.
51  *
52  * @interface DaysAllowedConfig
53  */
54 export interface DaysAllowedConfig {
55     isActive: boolean;
56     type: RuleType;
57     tradingDays: string[];
58 }
59 /**
60  * Interface representing the risk/reward ratio configuration.
61  *
62  * Defines the minimum risk/reward ratio required for trades (e.g., "1:2").
63  *
64  * @interface RiskRewardConfig
65  */
66 export interface RiskRewardConfig {
67     isActive: boolean;
68     riskRewardRatio: string;
69     type: RuleType;
70 }
71 /**
72  * Interface representing the maximum risk per trade configuration.
73  *
74  * Defines how much risk can be taken per trade, with options for:
75  * - Review type: MAX (maximum allowed) or FIXED (fixed amount)
76  * - Number type: PERCENTAGE or MONEY
77  * - Percentage type: INITIAL_B (initial balance), ACTUAL_B (actual balance), or NULL
78  *
79  * @interface RiskPerTradeConfig
80  */
81 export interface RiskPerTradeConfig {
82     isActive: boolean;
83     review_type: "MAX" | "FIXED";
84     number_type: "PERCENTAGE" | "MONEY";
85     percentage_type: "INITIAL_B" | "ACTUAL_B" | "NULL";
86     risk_amount: number;
87     balance: number;
88     actualBalance?: any;
89     type: RuleType;
90 }
91 /**
92  */
93 /**

```

```

94 * Interface representing the trading hours configuration.
95 *
96 * Defines the time window during which trading is allowed, including
97 * timezone information.
98 *
99 * @interface HoursAllowedConfig
100 */
101 export interface HoursAllowedConfig {
102   isActive: boolean;
103   tradingOpenTime: string;
104   tradingCloseTime: string;
105   timezone: string;
106   type: RuleType;
107 }
108 /**
109 * Interface representing the assets allowed for trading configuration.
110 *
111 * Specifies which trading instruments/assets are permitted for trading.
112 *
113 * @interface AssetsAllowedConfig
114 */
115 export interface AssetsAllowedConfig {
116   isActive: boolean;
117   assetsAllowed: string[];
118   type: RuleType;
119 }
120 /**
121 * Interface representing the complete strategy state (all trading rules).
122 *
123 * This interface contains all six trading rule configurations that make up
124 * a complete trading strategy. It is stored in the 'configurations' collection
125 * in Firebase, containing only the rules themselves.
126 *
127 * Used in:
128 * - StrategyComponent: Managing strategy configurations
129 * - SettingsService: CRUD operations for strategies
130 * - NgRx Store: Local state management
131 *
132 * @interface StrategyState
133 */
134 export interface StrategyState {
135   assetsAllowed: AssetsAllowedConfig;
136   hoursAllowed: HoursAllowedConfig;
137   riskReward: RiskRewardConfig;
138   maxDailyTrades: MaxDailyTradesConfig;
139   riskPerTrade: RiskPerTradeConfig;
140   daysAllowed: DaysAllowedConfig;
141 }
142 /**
143 * Interface representing strategy metadata (overview information).
144 *
145 * This interface contains metadata about a strategy, stored in the
146 * 'configuration-overview' collection in Firebase. It includes information
147 * like name, status, creation dates, and references to the actual configuration.
148 *
149 * The actual rules are stored separately in the 'configurations' collection
150 * and referenced via configurationId.
151 *
152 * Features:
153 * - Tracks activation/deactivation dates
154 * - Supports soft delete (deleted flag)
155 * - Links to configuration via configurationId
156 *
157 * Used in:
158 * - StrategyComponent: Displaying strategy cards
159 * - SettingsService: Managing strategy metadata
160 * - CalendarComponent: Determining strategy compliance for trades
161 */
162 */
163 
```

```

164  *
165  * @interface ConfigurationOverview
166  */
167 export interface ConfigurationOverview {
168   userId: string;
169   name: string;
170   status: boolean;
171   created_at: any; // Timestamp de Firebase
172   updated_at: any; // Timestamp de Firebase
173   days_active: number;
174   configurationId: string; // ID del documento en la colección 'configurations'
175   dateActive?: string[]; // ISO 8601 strings - Fechas cuando se activó la estrategia
176   dateInactive?: string[]; // ISO 8601 strings - Fechas cuando se desactivó la estrategia
177   deleted?: boolean; // Indica si la estrategia está marcada como eliminada
178   deleted_at?: any; // Timestamp de Firebase cuando se marcó como eliminada
179 }

```

Ø=ÜÁ features\strategy\service

Ø=ÜÁ features\strategy\service\strategy.service.ts

```

1 import { Injectable } from '@angular/core';
2 import { MaxDailyTradesConfig, StrategyState, ConfigurationOverview } from '../models/
3 strategy.strategyOperationsService' from '../../../../../shared/services/strategy-
4 operationsAppContextService' from '../../../../../shared/context';
5 import { AuthService } from '../../../../../shared/services/auth.service';
6
7 /**
8  * Service for managing trading strategies.
9  *
10 * This service acts as a facade for strategy operations, providing methods
11 * to create, read, update, and delete strategies. It manages both the
12 * strategy metadata (ConfigurationOverview) and the actual rules (StrategyState).
13 *
14 * The service uses a two-collection approach:
15 * - 'configuration-overview': Stores metadata (name, status, dates)
16 * - 'configurations': Stores actual trading rules
17 *
18 * Responsibilities:
19 * - Creating complete strategies (overview + configuration)
20 * - Fetching strategies with their configurations
21 * - Updating strategy metadata and rules
22 * - Activating/deactivating strategies
23 * - Soft deleting strategies
24 * - Managing user strategy counts
25 *
26 * Relations:
27 * - StrategyOperationsService: Direct Firebase operations
28 * - AppContextService: Global state management
29 * - AuthService: User count updates
30 *
31 * @injectable
32 * @providedIn root
33 */
34 @Injectable({ providedIn: 'root' })
35 export class SettingsService {
36 /**
37  * Constructor for SettingsService.
38  *
39  * @param strategyOperationsService - Service for direct Firebase operations
40  * @param applicationContext - Application context service for global state
41  * @param authService - Authentication service for user operations
42  */

```

```

43     constructor(
44       private strategyOperationsService: StrategyOperationsService,
45       private applicationContext: ApplicationContextService,
46       private authService: AuthService
47     ) {}
48
49     // ===== CONFIGURATION-OVERVIEW (colección de metadatos) =====
50
51     // Crear configuration-overview (solo metadatos)
52     async createConfigurationOverview(userId: string, name: string): Promise<string> {
53       return this.strategyOperationsService.createConfigurationOverview(userId, name);
54     }
55
56     // Obtener configuration-overview por ID (solo metadatos)
57     async getConfigurationOverview(overviewId: string): Promise<ConfigurationOverview | null> {
58       return this.strategyOperationsService.getConfigurationOverview(overviewId);
59     }
60
61     // Actualizar configuration-overview
62     async updateConfigurationOverview(overviewId: string, updates: Partial<ConfigurationOverview>): Promise<ConfigurationOverview> {
63       return this.strategyOperationsService.updateConfigurationOverview(overviewId, updates);
64     }
65
66     // Eliminar configuration-overview
67     async deleteConfigurationOverview(overviewId: string): Promise<void> {
68       return this.strategyOperationsService.deleteConfigurationOverview(overviewId);
69     }
70
71     // ===== CONFIGURATIONS (colección de reglas) =====
72
73     // Crear configuración (solo reglas + IDs)
74     async createConfiguration(userId: string, configurationOverviewId: string, configuration: StrategyState): Promise<Configuration> {
75       return this.strategyOperationsService.createConfiguration(userId, configurationOverviewId, configuration);
76     }
77
78     // Obtener configuración por userId
79     async getConfiguration(userId: string): Promise<StrategyState | null> {
80       return this.strategyOperationsService.getConfiguration(userId);
81     }
82
83     // Actualizar configuración
84     async updateConfiguration(userId: string, configuration: StrategyState): Promise<void> {
85       return this.strategyOperationsService.updateConfiguration(userId, configuration);
86     }
87
88     // ===== MÉTODOS INDIVIDUALES NUEVOS =====
89
90     // Crear solo configuration (sin userId ni configurationOverviewId)
91     async createConfigurationOnly(configuration: StrategyState): Promise<string> {
92       return this.strategyOperationsService.createConfigurationOnly(configuration);
93     }
94
95     // Crear configuration-overview con configurationId
96     async createConfigurationOverviewWithConfigId(userId: string, name: string, configurationId: string): Promise<ConfigurationOverview> {
97       return this.strategyOperationsService.createConfigurationOverviewWithConfigId(userId, name, configurationId, shouldBeActive);
98     }
99
100    // Actualizar configuration por ID
101    async updateConfigurationById(configurationId: string, configuration: StrategyState): Promise<void> {
102      return this.strategyOperationsService.updateConfigurationById(configurationId, configuration);
103    }
104
105    // ===== MÉTODOS COMBINADOS =====
106
107    // Crear estrategia completa (configurations + configuration-overview)
108    async createStrategyView(userId: string, name: string, configuration: StrategyState, shouldBeActive: boolean): Promise<string> {
109      this.appContext.setLoading(true);
110      this.appContext.setError('strategies', null);
111
112      try {

```

```

113     // 1. Crear configuration primero para obtener el ID
114     const configurationId = await this.createConfigurationOnly(configuration);
115
116     // 2. Crear configuration-overview con el configurationId
117     const overviewId = await this.createConfigurationOverviewWithConfigId(userId, name,
118     configurationId, shouldBeActive);
119     // 3. Actualizar contexto con la nueva estrategia
120     const newStrategy = await this.getConfigurationOverview(overviewId);
121     if (newStrategy) {
122         this.appContext.addStrategy({ ...newStrategy, id: overviewId });
123     }
124
125     // 4. Actualizar conteos del usuario
126     await this.authService.updateUserCounts(userId);
127
128     this.appContext.setLoading('strategies', false);
129     return overviewId;
130 } catch (error) {
131     this.appContext.setLoading('strategies', false);
132     this.appContext.setError('strategies', 'Error al crear estrategia');
133     throw error;
134 }
135 }
136
137 // Obtener estrategia completa (configuration-overview + configurations)
138 async getStrategyView(overviewId: string): Promise<{ overview: ConfigurationOverview;
139 configuration: StrategyState } | null> {
140     // 1. Primero obtener configuration-overview
141     const overview = await this.getConfigurationOverview(overviewId);
142
143     if (!overview) {
144         return null;
145     }
146
147     // 2. Luego obtener configuration usando el configurationId
148     const configuration = await this.getConfigurationById(overview.configurationId);
149
150     if (!configuration) {
151         return null;
152     }
153
154     return { overview, configuration };
155 }
156
157 // Obtener configuración por ID
158 async getConfigurationById(configurationId: string): Promise<StrategyState | null> {
159     return this.strategyOperationsService.getConfigurationById(configurationId);
160 }
161
162 // Obtener configuración por configurationOverviewId (método legacy para compatibilidad)
163 async getConfigurationByOverviewId(overviewId: string): Promise<StrategyState | null> {
164     return this.strategyOperationsService.getConfigurationByOverviewId(overviewId);
165 }
166
167 // Actualizar estrategia completa
168 async updateStrategyView(overviewId: string, updates: { name?: string; configuration?:
169     StrategyState; configurationId?: string }): Promise<void> {
170     // Si hay cambios de nombre
171     if (updates.name) {
172         await this.updateConfigurationOverview(overviewId, { name: updates.name });
173     }
174
175     // 2. Luego actualizar configuration si hay cambios de reglas
176     if (updates.configuration) {
177         // Obtener el configurationId del overview
178         const overview = await this.getConfigurationOverview(overviewId);
179         if (overview && overview.configurationId) {
180             await this.updateConfigurationById(overview.configurationId, updates.configuration);
181         }
182     }
183 }

```

```

183 // Obtener todas las estrategias de un usuario
184 async getUserStrategyViews(userId: string): Promise<ConfigurationOverview[]> {
185   this.appContext.setLoading('strategies', true);
186   this.appContext.setError('strategies', null);
187
188   try {
189     const strategies = await this.strategyOperationsService.getUserStrategyViews(userId);
190     this.appContext.setUserStrategies(strategies);
191     this.appContext.setLoading('strategies', false);
192     return strategies;
193   } catch (error) {
194     this.appContext.setLoading('strategies', false);
195     this.appContext.setError('strategies', 'Error al obtener estrategias del usuario');
196     throw error;
197   }
198 }
199
200 // Obtener configuración activa (método legacy para compatibilidad)
201 async getActiveConfiguration(userId: string): Promise<ConfigurationOverview | null> {
202   return this.strategyOperationsService.getActiveConfiguration(userId);
203 }
204
205 // Método legacy para compatibilidad
206 async getStrategyConfig(userId: string) {
207   return await this.getConfiguration(userId);
208 }
209
210 // Método legacy para compatibilidad
211 async saveStrategyConfig(userId: string, configurationOverviewId: string) {
212   // Este método ya no es necesario con la nueva estructura
213   console.warn('saveStrategyConfig is deprecated. Use createConfiguration instead.');
214 }
215
216 // Activar una estrategia
217 async activateStrategyView(userId: string, strategyId: string): Promise<void> {
218   return this.strategyOperationsService.activateStrategyView(userId, strategyId);
219 }
220
221 // Actualizar fechas de activación/desactivación de estrategias
222 async updateStrategyDates(userId: string, strategyId: string, dateActive?: Date,
223 dateInactive?: Date): Promise<void> {
224   this.strategyOperationsService.updateStrategyDates(userId, strategyId, dateActive, dateInactive);
225 }
226
227 // Eliminar una estrategia
228 async deleteStrategyView(strategyId: string): Promise<void> {
229   return this.strategyOperationsService.deleteStrategyView(strategyId);
230 }
231
232 // Marcar una estrategia como deleted (soft delete)
233 async markStrategyAsDeleted(strategyId: string): Promise<void> {
234   // 1. Obtener el userId de la estrategia antes de marcarla como eliminada
235   const strategy = await this.getConfigurationOverview(strategyId);
236   if (!strategy) {
237     throw new Error('Strategy not found');
238   }
239
240   const userId = strategy.userId;
241
242   // 2. Marcar la estrategia como eliminada
243   await this.strategyOperationsService.markStrategyAsDeleted(strategyId);
244
245   // 3. Actualizar conteos del usuario
246   if (userId) {
247     await this.authService.updateUserCounts(userId);
248   }
249 }
250 }
```

Ø=ÜÁ features\strategy\services

Ø=ÜÄ features\strategy\services\balance-cache.service.ts

```
1 import { Injectable } from '@angular/core';
2 import { BehaviorSubject, Observable } from 'rxjs';
3
4 interface BalanceData {
5   balance: number;
6   timestamp: number;
7   accountId: string;
8 }
9
10 /**
11 * Service for caching account balance data.
12 *
13 * This service provides caching for account balances with a 5-minute expiration.
14 * It uses both in-memory cache and localStorage for persistence across page reloads.
15 * Includes an Observable for real-time balance updates.
16 *
17 * Features:
18 * - In-memory and localStorage caching
19 * - 5-minute cache expiration
20 * - Observable for balance changes
21 * - Automatic cache cleanup
22 *
23 * Used in:
24 * - StrategyComponent: Caching balances for risk calculations
25 * - TradingAccountsComponent: Caching account balances
26 *
27 * @injectable
28 * @providedIn root
29 */
30 @Injectable({
31   providedIn: 'root'
32 })
33 export class BalanceCacheService {
34   private balanceCache = new Map<string, BalanceData>();
35   private readonly CACHE_DURATION = 5 * 60 * 1000; // 5 minutos
36   private balanceSubject = new BehaviorSubject<number>(0);
37
38 /**
39 * Obtener balance desde cache o localStorage
40 */
41 getBalance(accountId: string): number {
42   // 1. Verificar cache en memoria
43   const cachedBalance = this.balanceCache.get(accountId);
44   if (cachedBalance && this.isCacheValid(cachedBalance.timestamp)) {
45     return cachedBalance.balance;
46   }
47
48   // 2. Verificar localStorage
49   const localStorageKey = `balance_${accountId}`;
50   const storedBalance = localStorage.getItem(localStorageKey);
51   if (storedBalance) {
52     try {
53       const balanceData: BalanceData = JSON.parse(storedBalance);
54       if (this.isCacheValid(balanceData.timestamp)) {
55         // Actualizar cache en memoria
56         this.balanceCache.set(accountId, balanceData);
57         return balanceData.balance;
58       }
59     } catch (error) {
60       console.warn('Error parsing stored balance:', error);
61     }
62   }
63 }
```

```

62     }
63
64     return 0; // Valor por defecto
65   }
66
67   /**
68    * Guardar balance en cache y localStorage
69    */
70   setBalance(accountId: string, balance: number): void {
71     const balanceData: BalanceData = {
72       balance,
73       timestamp: Date.now(),
74       accountId
75     };
76
77     // Guardar en cache en memoria
78     this.balanceCache.set(accountId, balanceData);
79
80     // Guardar en localStorage
81     const localStorageKey = `balance_${accountId}`;
82     localStorage.setItem(localStorageKey, JSON.stringify(balanceData));
83
84     // Emitir cambio
85     this.balanceSubject.next(balance);
86   }
87
88   /**
89    * Verificar si el cache es válido
90    */
91   private isCacheValid(timestamp: number): boolean {
92     return Date.now() - timestamp < this.CACHE_DURATION;
93   }
94
95   /**
96    * Limpiar cache expirado
97    */
98   clearExpiredCache(): void {
99     const now = Date.now();
100    this.balanceCache.forEach((value, key) => {
101      if (!this.isCacheValid(value.timestamp)) {
102        this.balanceCache.delete(key);
103        localStorage.removeItem(`balance_${key}`);
104      }
105    });
106  }
107
108 /**
109  * Limpiar todo el cache
110 */
111 clearAllCache(): void {
112   this.balanceCache.clear();
113   // Limpiar localStorage
114   Object.keys(localStorage).forEach(key => {
115     if (key.startsWith('balance_')) {
116       localStorage.removeItem(key);
117     }
118   });
119 }
120
121 /**
122  * Observable para cambios de balance
123 */
124 getBalanceObservable(): Observable<number> {
125   return this.balanceSubject.asObservable();
126 }
127
128 /**
129  * Verificar si necesita actualizar el balance
130 */
131 needsUpdate(accountId: string): boolean {

```

```

132     const cachedBalance = this.balanceCache.get(accountId);
133     if (!cachedBalance) return true;
134
135     return !this.isCacheValid(cachedBalance.timestamp);
136   }
137 }
138

```

Ø=ÜÄ features\strategy\services\strategy-cache.service.ts

```

1 import { Injectable } from '@angular/core';
2 import { ConfigurationOverview } from '../models/strategy.model';
3 import { StrategyState } from '../models/strategy.model';
4
5 /**
6  * Service for caching strategy data in memory.
7  *
8  * This service provides a centralized cache for storing complete strategy data
9  * (both overview and configuration) to avoid redundant Firebase queries.
10 * Strategies are cached by their ID for quick access.
11 *
12 * Features:
13 * - In-memory caching of strategies
14 * - Cache size tracking
15 * - Cache clearing functionality
16 *
17 * Used in:
18 * - StrategyComponent: Caching strategies during initialization
19 *
20 * @injectable
21 * @providedIn root
22 */
23 @Injectable({
24   providedIn: 'root'
25 })
26 export class StrategyCacheService {
27   private strategiesCache: Map<string, { overview: ConfigurationOverview; configuration: StrategyState }> = new Map();
28
29 /**
30  * Almacenar estrategia en el cache
31  */
32 setStrategy(strategyId: string, overview: ConfigurationOverview, configuration: StrategyState): void {
33   this.strategiesCache.set(strategyId, { overview, configuration });
34 }
35
36 /**
37  * Obtener estrategia del cache
38  */
39 getStrategy(strategyId: string): { overview: ConfigurationOverview; configuration: StrategyState } | null {
40   return this.strategiesCache.get(strategyId) || null;
41 }
42
43 /**
44  * Verificar si el cache está cargado
45  */
46 isCacheLoaded(): boolean {
47   return this.strategiesCache.size > 0;
48 }
49
50 /**
51  * Limpiar todo el cache
52  */
53 clearCache(): void {
54   this.strategiesCache.clear();
55 }
56

```

```

57  /**
58   * Obtener todas las estrategias del cache
59   */
60  getAllStrategies(): Map<string, { overview: ConfigurationOverview; configuration:
61    StrategyState }> {
62    return this.strategiesCache;
63  }
64  /**
65   * Obtener el tamaño del cache
66   */
67  getCacheSize(): number {
68    return this.strategiesCache.size;
69  }
70}
71
72

```

Ø=ÜÁ features\strategy\store

Ø=ÜÁ features\strategy\store\strategy.actions.ts

```

1 import { createAction, props } from '@ngrx/store';
2 import {
3   AssetsAllowedConfig,
4   DaysAllowedConfig,
5   HoursAllowedConfig,
6   MaxDailyTradesConfig,
7   RiskPerTradeConfig,
8   RiskRewardConfig,
9   StrategyState,
10 } from '../models/strategy.model';
11
12 /**
13  * Action to set the maximum daily trades configuration.
14  *
15  * @action setMaxDailyTradesConfig
16  */
17 export const setMaxDailyTradesConfig = createAction(
18   '[Strategy] Set Max Daily Trades Config',
19   props<{ config: MaxDailyTradesConfig }>()
20 );
21
22 /**
23  * Action to set the risk/reward ratio configuration.
24  *
25  * @action setRiskRewardConfig
26  */
27 export const setRiskRewardConfig = createAction(
28   '[Strategy] Set Risk Reward Ratio Config',
29   props<{ config: RiskRewardConfig }>()
30 );
31
32 /**
33  * Action to set the risk per trade configuration.
34  *
35  * @action setRiskPerTradeConfig
36  */
37 export const setRiskPerTradeConfig = createAction(
38   '[Strategy] Set Risk Per Trade Config',
39   props<{ config: RiskPerTradeConfig }>()
40 );
41
42 /**

```

```

43 * Action to set the days allowed configuration.
44 *
45 * @action setDaysAllowedConfig
46 */
47 export const setDaysAllowedConfig = createAction(
48   '[Strategy] Set Days Allowed Config',
49   props<{ config: DaysAllowedConfig }>()
50 );
51 /**
52 * Action to set the trading hours configuration.
53 *
54 * @action setHoursAllowedConfig
55 */
56 export const setHoursAllowedConfig = createAction(
57   '[Strategy] Set Hours Allowed Config',
58   props<{ config: HoursAllowedConfig }>()
59 );
60 */
61 /**
62 * Action to set the assets allowed configuration.
63 *
64 * @action setAssetsAllowedConfig
65 */
66 export const setAssetsAllowedConfig = createAction(
67   '[Strategy] Set Assets Allowed Config',
68   props<{ config: AssetsAllowedConfig }>()
69 );
70 */
71 /**
72 * Action to reset the entire strategy configuration.
73 *
74 * @action resetConfig
75 */
76 export const resetConfig = createAction(
77   '[Strategy] Reset Config',
78   props<{ config: StrategyState }>()
79 );
80 */
81

```

Ø=ÜÄ features\strategy\store\strategy.reducer.ts

```

1 import { createReducer, on } from '@ngrx/store';
2 import {
3   resetConfig,
4   setAssetsAllowedConfig,
5   setDaysAllowedConfig,
6   setHoursAllowedConfig,
7   setMaxDailyTradesConfig,
8   setRiskPerTradeConfig,
9   setRiskRewardConfig,
10 } from './strategy.actions';
11 import { Days, RuleType, StrategyState } from '../models/strategy.model';
12
13 /**
14 * Initial state for the strategy feature.
15 *
16 * All rules are inactive by default with empty or zero values.
17 *
18 * @constant initialStrategyState
19 */
20 export const initialStrategyState: StrategyState = {
21   maxDailyTrades: {
22     isActive: false,
23     maxDailyTrades: 0,
24     type: RuleType.MAX_DAILY_TRADES,
25   },
26   rules: [
27     {
28       id: 1,
29       active: false,
30       dailyTrades: 0,
31       maxDailyTrades: 0,
32       type: RuleType.MAX_DAILY_TRADES,
33     },
34     {
35       id: 2,
36       active: false,
37       dailyTrades: 0,
38       maxDailyTrades: 0,
39       type: RuleType.MAX_DAILY_TRADES,
40     },
41     {
42       id: 3,
43       active: false,
44       dailyTrades: 0,
45       maxDailyTrades: 0,
46       type: RuleType.MAX_DAILY_TRADES,
47     },
48     {
49       id: 4,
50       active: false,
51       dailyTrades: 0,
52       maxDailyTrades: 0,
53       type: RuleType.MAX_DAILY_TRADES,
54     },
55   ],
56 };
57 
```

```

25     },
26     riskReward: {
27       isActive: false,
28       riskRewardRatio: '1:2',
29       type: RuleType.RISK_REWARD_RATIO,
30     },
31     riskPerTrade: {
32       isActive: false,
33       review_type: 'MAX',
34       number_type: 'PERCENTAGE',
35       percentage_type: 'NULL',
36       risk_amount: 0,
37       balance: 0,
38       actualBalance: 0,
39       type: RuleType.MAX_RISK_PER_TRADE,
40     },
41     daysAllowed: {
42       isActive: false,
43       type: RuleType.DAYS_ALLOWED,
44       tradingDays: [],
45     },
46     hoursAllowed: {
47       isActive: false,
48       tradingOpenTime: '',
49       tradingCloseTime: '',
50       timezone: '',
51       type: RuleType.TRADING_HOURS,
52     },
53     assetsAllowed: {
54       isActive: false,
55       type: RuleType.ASSETS_ALLOWED,
56       assetsAllowed: [],
57     },
58   };
59
60 /**
61 * Reducer for managing strategy state.
62 *
63 * Handles all strategy-related actions and updates the state accordingly.
64 * Actions handled:
65 * - setMaxDailyTradesConfig: Updates max daily trades rule
66 * - setRiskRewardConfig: Updates risk/reward ratio rule
67 * - setRiskPerTradeConfig: Updates risk per trade rule
68 * - setDaysAllowedConfig: Updates days allowed rule
69 * - setHoursAllowedConfig: Updates trading hours rule
70 * - setAssetsAllowedConfig: Updates assets allowed rule
71 * - resetConfig: Resets entire strategy state
72 *
73 * @reducer strategyReducer
74 */
75 export const strategyReducer = createReducer(
76   initialStrategyState,
77   on(setMaxDailyTradesConfig, (state, { config }) => ({
78     ...state,
79     maxDailyTrades: config,
80   })),
81   on(setRiskRewardConfig, (state, { config }) => ({
82     ...state,
83     riskReward: config,
84   })),
85   on(setRiskPerTradeConfig, (state, { config }) => ({
86     ...state,
87     riskPerTrade: config,
88   })),
89   on(setDaysAllowedConfig, (state, { config }) => ({
90     ...state,
91     daysAllowed: config,
92   })),
93   on(setHoursAllowedConfig, (state, { config }) => ({
94     ...state,

```

```

95     hoursAllowed: config,
96   })),
97   on(setAssetsAllowedConfig, (state, { config }) => ({
98     ...state,
99     assetsAllowed: config,
100   })),
101  on(resetConfig, (state, { config }) => ({
102    ...config,
103  }))
104 );
105

```

Ø=ÜÄ features\strategy\store\strategy.selectors.ts

```

1 import { createFeatureSelector, createSelector } from '@ngrx/store';
2 import { StrategyState } from '../models/strategy.model';
3
4 /**
5  * Feature selector for the strategy state.
6  *
7  * @selector selectStrategy
8  */
9 export const selectStrategy = createFeatureSelector<StrategyState>('strategy');
10
11 /**
12  * Selector for maximum daily trades configuration.
13  *
14  * @selector selectMaxDailyTrades
15  */
16 export const selectMaxDailyTrades = createSelector(
17   selectStrategy,
18   (state) => state.maxDailyTrades
19 );
20
21 /**
22  * Selector for risk/reward ratio configuration.
23  *
24  * @selector riskReward
25  */
26 export const riskReward = createSelector(
27   selectStrategy,
28   (state) => state.riskReward
29 );
30
31 /**
32  * Selector for risk per trade configuration.
33  *
34  * @selector riskPerTrade
35  */
36 export const riskPerTrade = createSelector(
37   selectStrategy,
38   (state) => state.riskPerTrade
39 );
40
41 /**
42  * Selector for days allowed configuration.
43  *
44  * @selector daysAllowed
45  */
46 export const daysAllowed = createSelector(
47   selectStrategy,
48   (state) => state.daysAllowed
49 );
50
51 /**
52  * Selector for trading hours configuration.

```

```

53  *
54  * @selector hoursAllowed
55  */
56 export const hoursAllowed = createSelector(
57   selectStrategy,
58   (state) => state.hoursAllowed
59 );
60 /**
61  * Selector for assets allowed configuration.
62  *
63  * @selector assetsAllowed
64  */
65 export const assetsAllowed = createSelector(
66   selectStrategy,
67   (state) => state.assetsAllowed
68 );
69 /**
70  * Selector for all strategy rules.
71  *
72  * @selector allRules
73  */
74 export const allRules = createSelector(selectStrategy, (state) => state);
75
76
77

```

Ø=ÜÁ features\trading-accounts

Ø=ÜÁ features\trading-accounts\trading-accounts.component.ts

```

1 import { Store } from '@ngrx/store';
2 import { CommonModule } from '@angular/common';
3 import { Component } from '@angular/core';
4 import { LoadingSpinnerComponent } from '../../../../../shared/components/loading-spinner/loading-
5 spinner[componentModule] from '@angular/forms';
6 import { OverviewService } from '../overview/services/overview.service';
7 import { AccountsTableComponent } from './components/accounts-table/accounts-
8 table[componentUserStatus] from '../overview/models/overview';
9 import { AuthService } from '../auth/service/authService';
10 import { AccountComponent } from '../account/account.component';
11 import { selectUser } from '../auth/store/user.selectios';
12 import { AccountData } from '../auth/models/userModel';
13 import { ReportService } from '../report/service/report.service';
14 import { setUserKey } from '../report/store/report.actions';
15 import { concatMap, from, catchError, of } from 'rxjs';
16 import { CreateAccountPopupComponent } from '../../../../../shared/components/create-account-popup/
17 CreateAccountPopupComponent[router];
18 import { PlanLimitationsGuard } from '../../../../../guards/plan-limitations.guard';
19 import { PlanLimitationModalData } from '../../../../../shared/interfaces/plan-limitation-
20 modal[interfacePlanLimitationModalComponent] from '../../../../../shared/components/plan-limitation-modal/
21 PlanLimitationModalComponent[planBannerComponent] from '../../../../../shared/components/plan-banner/plan-
22 banner[componentContextService] from '../../../../../shared/context';
23
24 /**
25  * Main component for managing trading accounts.
26  *
27  * This component provides functionality for viewing, adding, editing, and deleting
28  * trading accounts. It also handles plan limitations, account balance fetching,
29  * and displays plan banners when users approach or reach account limits.
30  *
31  * Features:
32  * - Display all user trading accounts in a table
33  * - Add new trading accounts (with plan limitation checks)

```

```

34 * - Edit existing trading accounts
35 * - Delete trading accounts with confirmation
36 * - Fetch and display account balances from API
37 * - Plan limitation detection and warnings
38 * - Plan upgrade prompts
39 *
40 * Relations:
41 * - AccountsTableComponent: Displays accounts in table format
42 * - CreateAccountPopupComponent: Modal for adding/editing accounts
43 * - PlanLimitationModalComponent: Modal for plan upgrade prompts
44 * - PlanBannerComponent: Banner for plan limitation warnings
45 * - AuthService: Account CRUD operations
46 * - ReportService: Fetching account balances and user keys
47 * - PlanLimitationsGuard: Checking plan limitations
48 * - ApplicationContextService: Global state management
49 *
50 * @component
51 * @selector app-trading-accounts
52 * @standalone true
53 */
54 @Component({
55   selector: 'app-trading-accounts',
56   imports: [
57     CommonModule,
58     LoadingSpinnerComponent,
59     FormsModule,
60     AccountsTableComponent,
61     CreateAccountPopupComponent,
62     PlanLimitationModalComponent,
63     PlanBannerComponent,
64   ],
65   templateUrl: './trading-accounts.component.html',
66   styleUrls: ['./trading-accounts.component.scss'],
67   standalone: true,
68 })
69 export class TradingAccountsComponent {
70   user: User | null = null;
71   userKey: string = '';
72   fromDate: string = '';
73   toDate: string = '';
74
75   constructor(
76     private store: Store,
77     private reportSvc: ReportService,
78     private userSvc: AuthService,
79     private router: Router,
80     private planLimitationsGuard: PlanLimitationsGuard,
81     private applicationContext: ApplicationContextService
82   ) {}
83
84   loading = false;
85   usersData: AccountData[] = [];
86
87   // Plan detection and modal
88   showAddAccountModal = false;
89   editMode = false;
90   accountToEdit: AccountData | null = null;
91   planLimitationModal: PlanLimitationModalData = {
92     showModal: false,
93     modalType: 'upgrade',
94     title: '',
95     message: '',
96     primaryButtonText: '',
97     onPrimaryAction: () => {}
98   };
99
100  // Button state
101  isAddAccountDisabled = false;
102
103  // Plan detection and banner

```

```

104     showPlanBanner = false;
105     planBannerMessage = '';
106     planBannerType = 'info'; // 'info', 'warning', 'success'
107
108     /**
109      * Initializes the component on load.
110      *
111      * Subscribes to context data and store to get user information,
112      * then loads configuration and account data.
113      *
114      * @memberof TradingAccountsComponent
115      */
116     ngOnInit(): void {
117       // Suscribirse a los datos del contexto
118       this.subscribeToContextData();
119
120       this.store.select(selectUser).subscribe((userState) => {
121         this.user = userState?.user ?? null;
122         this.loadConfig();
123       });
124     }
125
126     /**
127      * Subscribes to data from ApplicationContextService.
128      *
129      * Listens to changes in:
130      * - Current user
131      * - User accounts
132      * - Loading states
133      * - Error states
134      *
135      * Related to:
136      * - ApplicationContextService: Global state management
137      *
138      * @private
139      * @memberof TradingAccountsComponent
140      */
141     private subscribeToContextData() {
142       // Suscribirse a los datos del usuario
143       this.appContext.currentUser$.subscribe(user => {
144         this.user = user;
145       });
146
147       // Suscribirse a las cuentas del usuario
148       this.appContext.userAccounts$.subscribe(accounts => {
149         this.userData = accounts;
150       });
151
152       // Suscribirse a los estados de carga
153       this.appContext.isLoading$.subscribe/loading => {
154         this.loading = loading.accounts;
155       });
156
157       // Suscribirse a los errores
158       this.appContext.errors$.subscribe(errors => {
159         if (errors.accounts) {
160           console.error('Error en cuentas:', errors.accounts);
161         }
162       });
163     }
164
165     /**
166      * Loads initial configuration.
167      *
168      * Fetches user accounts and checks plan limitations.
169      *
170      * @memberof TradingAccountsComponent
171      */
172     loadConfig() {
173       this.getUserAccounts();

```

```

174     // this.checkPlanLimitations(); // Comentado temporalmente
175 }
176
177 /**
178 * Fetches all trading accounts for the current user.
179 *
180 * Uses AuthService to get accounts from Firebase, then fetches
181 * balance data for each account. Also checks account limitations
182 * after loading.
183 *
184 * Related to:
185 * - AuthService.getUserAccounts(): Fetches accounts from Firebase
186 * - getBalanceForAccounts(): Fetches balances for all accounts
187 * - checkAccountLimitations(): Checks plan limitations
188 *
189 * @memberof TradingAccountsComponent
190 */
191 getUserAccounts() {
192     this.userSvc
193         .getUserAccounts(this.user?.id || '')
194         .then((docSnap) => {
195
196         if (docSnap && docSnap.length > 0) {
197             this.userData = docSnap;
198             // Don't set loading = false here, wait for balances to load
199             this.getBalanceForAccounts();
200         } else {
201             this.userData = [];
202             this.loading = false; // No accounts, no balances to load
203         }
204
205         // Verificar limitaciones después de cargar las cuentas
206         this.checkAccountLimitations();
207     })
208     .catch((err) => {
209         this.loading = false;
210         console.error('Error to get the config', err);
211     });
212 }
213
214 /**
215 * Fetches balance data for all accounts.
216 *
217 * Processes accounts sequentially, fetching user key and balance
218 * for each account. Updates account balance property with real-time
219 * data from the trading API.
220 *
221 * Related to:
222 * - fetchUserKey(): Gets authentication token for each account
223 * - getActualBalance(): Fetches balance from API
224 * - ReportService: API communication
225 *
226 * @memberof TradingAccountsComponent
227 */
228 getBalanceForAccounts() {
229     // Loading is already active from getUserAccounts, don't set it again
230     from(this.userData)
231         .pipe(concatMap((account) => this.fetchUserKey(account)))
232         .subscribe({
233             next: () => {},
234             complete: () => {
235                 this.loading = false; // Only set to false when balances are loaded
236             },
237             error: (err) => {
238                 this.loading = false;
239                 console.error(
240                     'Error en el proceso de actualización de balances',
241                     err
242                 );
243             },

```

```

244     });
245 }
246
247 /**
248 * Fetches user authentication key for an account.
249 *
250 * Authenticates with trading account credentials and stores the key
251 * in NgRx store, then fetches the actual balance for the account.
252 *
253 * Related to:
254 * - ReportService.getUserKey(): Authenticates and gets token
255 * - Store.dispatch(setUserKey()): Stores token in NgRx
256 * - getActualBalance(): Fetches balance after authentication
257 *
258 * @param account - Trading account data with credentials
259 * @returns Observable that completes when balance is fetched
260 * @memberof TradingAccountsComponent
261 */
262 fetchUserKey(account: AccountData) {
263   return this.reportSvc
264     .getUserKey(
265       account.emailTradingAccount,
266       account.brokerPassword,
267       account.server
268     )
269     .pipe(
270       concatMap((key: string) => {
271         this.store.dispatch(setUserKey({ userKey: key }));
272         return this.getActualBalance(key, account);
273       })
274     );
275 }
276
277 /**
278 * Fetches actual balance for an account from the trading API.
279 *
280 * Uses the authentication key to fetch balance data and updates
281 * the account's balance property.
282 *
283 * Related to:
284 * - ReportService.getBalanceData(): Fetches balance from API
285 *
286 * @param key - User authentication token
287 * @param account - Account to fetch balance for
288 * @returns Observable that emits the account with updated balance
289 * @memberof TradingAccountsComponent
290 */
291 getActualBalance(key: string, account: AccountData) {
292   return this.reportSvc
293     .getBalanceData(account.accountID, key, account.accountNumber)
294     .pipe(
295       concatMap((balanceData) => {
296         // Guardar el balance en la cuenta
297         account.balance = balanceData.balance || 0;
298         return [account];
299       })
300     );
301 }
302
303 /**
304 * Deletes a trading account.
305 *
306 * Removes the account from Firebase and reloads the account list.
307 * Also checks plan limitations after deletion.
308 *
309 * Related to:
310 * - AuthService.deleteAccount(): Deletes account from Firebase
311 * - loadConfig(): Reloads accounts after deletion
312 * - checkPlanLimitations(): Updates limitation status
313 */

```

```

314     * @param account - Account to delete
315     * @memberof TradingAccountsComponent
316     */
317     deleteAccount(account: AccountData) {
318         this.loading = true;
319         this.userSvc
320             .deleteAccount(account.id)
321             .then(async () => {
322                 // Reload everything after deletion
323                 this.loadConfig();
324                 await this.checkPlanLimitations(); // Check plan limitations after deleting account
325                 this.userData = [...this.userData];
326             })
327             .catch((err) => {
328                 this.loading = false;
329                 console.error('Error deleting account', err);
330             });
331     }
332
333     /**
334      * Handles the add account button click.
335      *
336      * Checks plan limitations before allowing account creation.
337      * If user has reached limit, shows upgrade modal or redirects to plan management.
338      * If within limits, opens the add account modal.
339      *
340      * Related to:
341      * - PlanLimitationsGuard.checkAccountCreationWithModal(): Checks if user can create
342      accountRouter.navigate(): Redirects to plan management if needed
343      *
344      * @memberof TradingAccountsComponent
345      */
346     // Add account functionality
347     async onAddAccount() {
348         if (!this.user?.id) return;
349
350         const currentAccountCount = this.userData.length;
351         const accessCheck = await
352             this.planLimitationsGuard.checkAccountCreationWithModal(this.user.id, currentAccountCount);
353         if (!accessCheck.canCreate) {
354             // Check if user has Pro plan with maximum accounts (should disable button)
355             const limitations = await this.planLimitationsGuard.checkUserLimitations(this.user.id);
356             const isProPlanWithMaxAccounts = limitations.planName.toLowerCase().includes('pro') &&
357                                         limitations.maxAccounts === 10 &&
358                                         currentAccountCount >= 10;
359
360             if (isProPlanWithMaxAccounts) {
361                 // Pro plan with maximum accounts: button should be disabled (do nothing)
362                 return;
363             } else {
364                 // Other plans: redirect to account/plan-management
365                 this.router.navigate(['/account'], {
366                     queryParams: { tab: 'plan' }
367                 });
368                 return;
369             }
370         }
371
372         // If within limits, show add account modal
373         this.editMode = false;
374         this.accountToEdit = null;
375         this.showAddAccountModal = true;
376     }
377
378     /**
379      * Handles editing an existing account.
380      *
381      * Opens the account creation modal in edit mode with the account data pre-filled.
382      *
383      * @param account - Account to edit

```

```

384     * @memberof TradingAccountsComponent
385     */
386     // Edit account functionality
387     onEditAccount(account: AccountData) {
388         this.editMode = true;
389         this.accountToEdit = account;
390         this.showAddAccountModal = true;
391     }
392     // Modal methods
393     onCloseAddAccountModal() {
394         this.showAddAccountModal = false;
395         this.editMode = false;
396         this.accountToEdit = null;
397     }
398     onAccountCanceled() {
399         this.showAddAccountModal = false;
400         this.editMode = false;
401         this.accountToEdit = null;
402     }
403
404     // Plan limitation modal methods
405     onClosePlanLimitationModal() {
406         this.planLimitationModal.showModal = false;
407     }
408
409     /**
410      * Checks account limitations and updates button state.
411      *
412      * Determines if the "Add Account" button should be disabled based on
413      * the user's plan and current account count. Also shows plan banners
414      * when approaching or at limits.
415      *
416      * Related to:
417      * - PlanLimitationsGuard.checkUserLimitations(): Gets plan limitations
418      * - checkPlanLimitations(): Shows plan banners
419      *
420      * @memberof TradingAccountsComponent
421      */
422     // Check account limitations and update button state
423     async checkAccountLimitations() {
424         if (!this.user?.id) {
425             this.isAddAccountDisabled = false; // Always active, will redirect if needed
426             return;
427         }
428
429         try {
430             const currentAccountCount = this.userData.length;
431             const limitations = await this.planLimitationsGuard.checkUserLimitations(this.user.id);
432
433             // Only disable button for Pro plan with maximum accounts (10 accounts)
434             const isProPlanWithMaxAccounts = limitations.planName.toLowerCase().includes('pro') &&
435                 limitations.maxAccounts === 10 &&
436                 currentAccountCount >= 10;
437
438             this.isAddAccountDisabled = isProPlanWithMaxAccounts;
439
440             // Show banner based on limitations
441             await this.checkPlanLimitations();
442         } catch (error) {
443             console.error('Error checking account limitations:', error);
444             this.isAddAccountDisabled = false; // Default to active
445         }
446     }
447
448     /**
449      * Checks plan limitations and displays appropriate banners.
450      *
451      * Determines if user needs subscription, is banned/cancelled, or has reached

```

```

454     * account limits. Shows warning or info banners accordingly.
455     *
456     * Related to:
457     * - PlanLimitationsGuard.checkUserLimitations(): Gets plan limitations
458     *
459     * @private
460     * @memberof TradingAccountsComponent
461     */
462     // Plan detection and banner methods
463     private async checkPlanLimitations() {
464       if (!this.user?.id) {
465         this.showPlanBanner = false;
466         return;
467       }
468
469       try {
470         // Get user's plan limitations from the guard
471         const limitations = await this.planLimitationsGuard.checkUserLimitations(this.user.id);
472         const currentAccountCount = this.userData.length;
473
474         this.showPlanBanner = false;
475         this.planBannerMessage = '';
476         this.planBannerType = 'info';
477
478         // If user needs subscription or is banned/cancelled
479         if (limitations.needsSubscription || limitations.isBanned || limitations.isCancelled) {
480           this.showPlanBanner = true;
481           this.planBannerMessage = this.getBlockedMessage(limitations);
482           this.planBannerType = 'warning';
483           return;
484         }
485
486         // Check if user has reached account limit
487         if (currentAccountCount >= limitations.maxAccounts) {
488           this.showPlanBanner = true;
489           this.planBannerMessage = `You've reached the account limit for your
490           ${limitations.planName} plan. Move to a higher plan and keep growing your account.`;
491         } else if (currentAccountCount >= limitations.maxAccounts - 1) {
492           // Show warning when close to limit
493           this.showPlanBanner = true;
494           this.planBannerMessage = `You have ${limitations.maxAccounts - currentAccountCount}
495           accounts left. ${this.showPlanBanner ? 'Want more? Upgrade anytime.' : ''}`;
496         }
497       } catch (error) {
498         console.error('Error checking plan limitations:', error);
499         this.showPlanBanner = false;
500       }
501     }
502
503     private getBlockedMessage(limitations: any): string {
504       if (limitations.isBanned) {
505         return 'Your account has been banned. Please contact support for assistance.';
506       }
507
508       if (limitations.isCancelled) {
509         return 'Your subscription has been cancelled. Please purchase a plan to access this
510         functionality.';
511
512       if (limitations.needsSubscription) {
513         return 'You need to purchase a plan to access this functionality.';
514       }
515
516       return 'Access denied. Please contact support for assistance.';
517     }
518
519
520     onUpgradePlan() {
521       this.router.navigate(['/account']);
522     }
523

```

```

524     onCloseBanner() {
525         this.showPlanBanner = false;
526     }
527
528     /**
529      * Handles account creation event from popup.
530      *
531      * Reloads account list and checks plan limitations after a new account is created.
532      *
533      * @param accountData - Created account data (not used, account already in Firebase)
534      * @memberof TradingAccountsComponent
535      */
536     // Popup event handlers
537     async onAccountCreated(accountData: any) {
538         // Account is already created in Firebase by the popup component
539         // Show loading and reload everything
540         this.loading = true;
541         this.loadConfig(); // Reload accounts
542         await this.checkPlanLimitations();
543     }
544
545     async onAccountUpdated(accountData: any) {
546         // Account is already updated in Firebase by the popup component
547         // Show loading and reload everything
548         this.loading = true;
549         this.loadConfig(); // Reload accounts
550         await this.checkPlanLimitations();
551     }
552 }
553
554 }
```

Ø=ÜÁ features\trading-accounts\components\accounts-table

Ø=ÜÄ features\trading-accounts\components\accounts-table\accounts-table.component.ts

```

1 import { Component, Input, Output, OnInit, OnDestroy, OnChanges, SimpleChanges,
2 HostListener, ElementRef, ViewChild } from '@angular/core';
3
4 import { FormsModule } from '@angular/forms';
5 import { User } from '../../../../../overview/models/overview';
6 import { EventEmitter } from '@angular/core';
7 import { Timestamp } from 'firebase/firestore';
8 import { AccountData } from '../../../../../auth/models/userModel';
9 import { ShowConfirmationComponent } from '../show-confirmation/show-confirmation.component';
10 import { Router } from '@angular/router';
11 import { Subscription } from 'rxjs';
12 import { NumberFormatterService } from '../../../../../shared/utils/number-formatter.service';
13
14 /**
15  * Component for displaying trading accounts in a table format.
16  *
17  * This component provides a filterable and sortable table for trading accounts.
18  * It supports filtering by account name and balance range, sorting by creation date,
19  * and includes functionality for editing and deleting accounts.
20  *
21  * Features:
22  * - Search by account name
23  * - Filter by balance range (with formatted currency inputs)
24  * - Sort by creation date (ascending/descending)
25  * - Edit account functionality
26  * - Delete account with confirmation
27  * - Currency formatting for balances
```

```

28  *
29  * Relations:
30  * - ShowConfirmationComponent: Confirmation modal for account deletion
31  * - NumberFormatterService: Currency formatting and parsing
32  * - TradingAccountsComponent: Parent component (receives accounts, emits edit/delete events)
33  *
34  * @component
35  * @selector app-accounts-table
36  * @standalone true
37  */
38 @Component({
39   selector: 'app-accounts-table',
40   standalone: true,
41   imports: [CommonModule, FormsModule, ShowConfirmationComponent],
42   templateUrl: './accounts-table.component.html',
43   styleUrls: ['./accounts-table.component.scss'],
44 })
45 export class AccountsTableComponent implements OnInit, OnDestroy, OnChanges {
46   @Input() accounts: AccountData[] = [];
47   @Output() delete = new EventEmitter<AccountData>();
48   @Output() edit = new EventEmitter<AccountData>();
49
50   initialMinBalance = 0;
51   initialMaxBalance = 1000000;
52   showFilter = false;
53   sortField: 'createdAt' = 'createdAt';
54   sortAsc: boolean = true;
55   showConfirmation = false;
56
57   // Applied filter values (separate from input values)
58   appliedMinBalance = 0;
59   appliedMaxBalance = 1000000;
60
61   // Properties for formatted balance inputs
62   minBalanceDisplay: string = '';
63   maxBalanceDisplay: string = '';
64   minBalanceInput: string = '';
65   maxBalanceInput: string = '';
66
67   // Reference to filter container for click outside detection
68   @ViewChild('filterContainer') filterContainer?: ElementRef;
69
70   // Balance properties (now handled by parent component)
71   private subscriptions: Subscription[] = [];
72
73   constructor(
74     private router: Router,
75     private numberFormatter: NumberFormatterService
76   ) {
77     // Initialize inputs as empty to show placeholders
78     this.minBalanceInput = '';
79     this.maxBalanceInput = '';
80     this.minBalanceDisplay = '';
81     this.maxBalanceDisplay = '';
82   }
83
84   private _searchTerm = '';
85   accountToDelete!: AccountData;
86   get searchTerm(): string {
87     return this._searchTerm;
88   }
89   set searchTerm(val: string) {
90     this._searchTerm = val;
91   }
92
93   get filteredUsers(): AccountData[] {
94     const lower = this._searchTerm.trim().toLowerCase();
95
96     let result = this.accounts.filter((account) => {
97       const matchesSearch = `${account.accountName}`
```

```

98         .toLowerCase()
99         .includes(lower);
100
101        let matchesMinBalance = (account.balance ?? 0) >= this.appliedMinBalance;
102        let matchesMaxBalance = (account.balance ?? 0) <= this.appliedMaxBalance;
103
104        if (account.balance === undefined) {
105            matchesMinBalance = true;
106            matchesMaxBalance = true;
107        }
108
109        return matchesSearch && matchesMinBalance && matchesMaxBalance;
110    });
111
112    result = result.sort((a, b) => {
113        const fieldA =
114            a[this.sortField] instanceof Timestamp
115            ? (a[this.sortField] as Timestamp).toDate().getTime()
116            : String(a[this.sortField]).toLowerCase();
117        const fieldB =
118            b[this.sortField] instanceof Timestamp
119            ? (b[this.sortField] as Timestamp).toDate().getTime()
120            : String(b[this.sortField]).toLowerCase();
121
122        if (fieldA < fieldB) return this.sortAsc ? -1 : 1;
123        if (fieldA > fieldB) return this.sortAsc ? 1 : -1;
124        return 0;
125    });
126
127    return result;
128}
129
130
131    statusClass(status: string) {
132        return status;
133    }
134
135    returnClass(returnValue: number) {
136        return returnValue >= 0 ? 'green' : 'red';
137    }
138
139    openFilter() {
140        this.showFilter = !this.showFilter;
141
142        if (this.showFilter) {
143            // Load current applied values into inputs
144            this.initialMinBalance = this.appliedMinBalance;
145            this.initialMaxBalance = this.appliedMaxBalance;
146
147            // Set input values for display
148            if (this.initialMinBalance > 0) {
149                this.minBalanceInput = this.initialMinBalance.toString();
150                this.minBalanceDisplay =
151                    this.numberFormatter.formatCurrencyDisplay(this.initialMinBalance);
152                this.minBalanceInput = '';
153                this.minBalanceDisplay = '';
154            }
155
156            if (this.initialMaxBalance > 0 && this.initialMaxBalance !== 1000000) {
157                this.maxBalanceInput = this.initialMaxBalance.toString();
158                this.maxBalanceDisplay =
159                    this.numberFormatter.formatCurrencyDisplay(this.initialMaxBalance);
160                this.maxBalanceInput = '';
161                this.maxBalanceDisplay = '';
162            }
163        }
164    }
165
166    closeFilter() {
167        this.showFilter = false;

```

```

168     }
169
170     apply() {
171       this.applyFilters();
172     }
173
174     applyFilters() {
175       // Apply the current input values to the filter
176       this.appliedMinBalance = this.initialMinBalance;
177       this.appliedMaxBalance = this.initialMaxBalance;
178       this.showFilter = false;
179     }
180
181     // Host listener to detect clicks outside the filter modal
182     @HostListener('document:click', ['$event'])
183     onDocumentClick(event: Event): void {
184       if (this.showFilter && this.filterContainer) {
185         const clickedInside = this.filterContainer.nativeElement.contains(event.target as
186 Node); if (!clickedInside) {
187           this.showFilter = false;
188         }
189       }
190     }
191
192     toggleSort() {
193       this.sortAsc = !this.sortAsc;
194     }
195
196     getUserDate(date: number): Date {
197       return new Date(date);
198     }
199
200     deleteAccount(account: AccountData) {
201       this.showConfirmation = true;
202       this.accountToDelete = account;
203     }
204
205     editAccount(account: AccountData) {
206       this.edit.emit(account);
207     }
208
209     confirmDelete() {
210       this.delete.emit(this.accountToDelete);
211       this.accountToDelete = {} as AccountData;
212       this.showConfirmation = false;
213     }
214
215     cancelDelete() {
216       this.showConfirmation = false;
217       this.accountToDelete = {} as AccountData;
218     }
219
220     ngOnInit() {
221       // Cargar balances para todas las cuentas
222       this.loadAccountBalances();
223     }
224
225     ngOnChanges(changes: SimpleChanges) {
226       if (changes['accounts']) {
227         // Recargar balances cuando cambien las cuentas
228         if (this.accounts && this.accounts.length > 0) {
229           this.loadAccountBalances();
230         }
231       }
232     }
233
234     ngOnDestroy() {
235       // Limpiar todas las suscripciones
236       this.subscriptions.forEach(sub => sub.unsubscribe());
237     }

```

```

238
239 /**
240 * Carga el balance real para todas las cuentas usando el ReportService
241 * Nota: Los balances se cargan desde el componente padre (trading-accounts)
242 */
243 loadAccountBalances() {
244     // Los balances se cargan desde el componente padre
245     // Este método se mantiene para compatibilidad pero no hace nada
246 }
247
248 // Methods for balance input formatting
249 onMinBalanceInput(event: Event) {
250     const target = event.target as HTMLInputElement;
251     this.minBalanceInput = target.value;
252 }
253
254 onMaxBalanceInput(event: Event) {
255     const target = event.target as HTMLInputElement;
256     this.maxBalanceInput = target.value;
257 }
258
259 onMinBalanceFocus() {
260     // When user focuses, show only the number without formatting for editing
261     if (this.initialMinBalance > 0) {
262         this.minBalanceInput = this.initialMinBalance.toString();
263     } else {
264         this.minBalanceInput = '';
265     }
266 }
267
268 onMaxBalanceFocus() {
269     // When user focuses, show only the number without formatting for editing
270     if (this.initialMaxBalance > 0 && this.initialMaxBalance !== 1000000) {
271         this.maxBalanceInput = this.initialMaxBalance.toString();
272     } else {
273         this.maxBalanceInput = '';
274     }
275 }
276
277 onMinBalanceBlur() {
278     // Convert the value to number
279     const numericValue = this.numberFormatter.parseCurrencyValue(this.minBalanceInput);
280     if (!isNaN(numericValue) && numericValue >= 0) {
281         // Save the unformatted value
282         this.initialMinBalance = numericValue;
283
284         // Show visual format (only for display)
285         this.minBalanceDisplay = this.numberFormatter.formatCurrencyDisplay(numericValue);
286
287         // Update the input to show the visual format
288         this.minBalanceInput = this.minBalanceDisplay;
289     } else {
290         // If not a valid number, clear
291         this.minBalanceInput = '';
292         this.minBalanceDisplay = '';
293         this.initialMinBalance = 0;
294     }
295 }
296
297 onMaxBalanceBlur() {
298     // Convert the value to number
299     const numericValue = this.numberFormatter.parseCurrencyValue(this.maxBalanceInput);
300     if (!isNaN(numericValue) && numericValue >= 0) {
301         // Save the unformatted value
302         this.initialMaxBalance = numericValue;
303
304         // Show visual format (only for display)
305         this.maxBalanceDisplay = this.numberFormatter.formatCurrencyDisplay(numericValue);
306
307         // Update the input to show the visual format

```

```

308     this.maxBalanceInput = this.maxBalanceDisplay;
309   } else {
310     // If not a valid number, clear
311     this.maxBalanceInput = '';
312     this.maxBalanceDisplay = '';
313     this.initialMaxBalance = 0; // Reset to 0 instead of 1000000
314   }
315 }
316
317
318
319 }
320

```

Ø=ÜÁ features\trading-accounts\components\show-confirmation

Ø=ÜÁ features\trading-accounts\components\show-confirmation\show-confirmation.component.ts

```

1 import { Component, Input, Output } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 import { FormsModule } from '@angular/forms';
5 import { User } from '../../../../../overview/models/overview';
6 import { EventEmitter } from '@angular/core';
7 import { Timestamp } from 'firebase/firestore';
8 import { AccountData } from '../../../../../auth/models/userModel';
9
10 /**
11  * Component for displaying account deletion confirmation modal.
12  *
13  * This component requires the user to type the account name exactly
14  * to confirm deletion, providing an extra safety measure against accidental deletions.
15  *
16  * Features:
17  * - Account name input validation
18  * - Confirm and cancel actions
19  * - Event emitters for parent component handling
20  *
21  * Relations:
22  * - AccountsTableComponent: Uses this component for deletion confirmation
23  *
24  * @component
25  * @selector app-show-confirmation
26  * @standalone true
27  */
28 @Component({
29   selector: 'app-show-confirmation',
30   standalone: true,
31   imports: [CommonModule, FormsModule],
32   templateUrl: './show-confirmation.component.html',
33   styleUrls: ['./show-confirmation.component.scss'],
34 })
35 export class ShowConfirmationComponent {
36   @Input() account: AccountData | null = null;
37   @Output() confirm = new EventEmitter<void>();
38   @Output() cancel = new EventEmitter<void>();
39
40   accountNameInput: string = '';
41
42   get accountName(): string {
43     return this.account?.accountName ?? '';
44   }
45

```

```

46     onConfirm() {
47       if (this.accountNameInput.trim() === this.accountName) {
48         this.confirm.emit();
49       }
50     }
51   }
52   onCancel() {
53     this.cancel.emit();
54   }
55 }
56

```

Ø=ÜÁ features\users-details

Ø=ÜÁ features\users-details\users-details.component.ts

```

1 import { Store } from '@ngrx/store';
2 import { CommonModule } from '@angular/common';
3 import { Component } from '@angular/core';
4 import { LoadingPopupComponent } from '../../../../../shared/pop-ups/loading-pop-up/loading-
5 paper-component.module' from '@angular/forms';
6 import { UserManagementService } from '../../../../../shared/services/user-management.service';
7 import { UsersTableComponent } from './components/users-table/users-table.component';
8 import { User, UserStatus } from '../overview/models/overview';
9 import { Timestamp } from 'firebase/firestore';
10 import { UserModalComponent } from './components/user-modal/user-modal.component';
11 import { AuthService } from '../auth/service/authService';
12 import { AppContextService } from '../../../../../shared/context';
13 import { AlertService } from '../../../../../shared/services/alert.service';
14 import { ReasonsService } from '../../../../../shared/services/reasons.service';
15 import { serverTimestamp } from 'firebase/firestore';
16
17 /**
18 * Component for managing user details and administrative actions.
19 *
20 * This component provides administrators with tools to view, manage, and perform
21 * administrative actions on users. It displays a table of all users (excluding admins)
22 * and allows viewing detailed user information, banning/unbanning users, sending
23 * password reset links, and logging users out from all devices.
24 *
25 * Features:
26 * - Display all non-admin users in a table
27 * - View detailed user information in a modal
28 * - Ban users with reason tracking
29 * - Unban users and update ban reason records
30 * - Send password reset links
31 * - Revoke Firebase tokens (logout everywhere)
32 * - Admin-only access control
33 *
34 * Relations:
35 * - UsersTableComponent: Displays the user table and handles user selection
36 * - UserModalComponent: Shows detailed user information and action buttons
37 * - UserManagementService: Fetches and updates user data
38 * - AuthService: Handles password reset and token revocation
39 * - ReasonsService: Manages ban reason records
40 * - AppContextService: Provides current user data and loading states
41 * - AlertService: Shows success/error notifications
42 *
43 * @component
44 * @selector app-users-details
45 * @standalone true
46 */
47 @Component({

```

```

48     selector: 'app-users-details',
49     imports: [
50       CommonModule,
51       LoadingPopupComponent,
52       FormsModule,
53       UsersTableComponent,
54       UserModalComponent,
55     ],
56     templateUrl: './users-details.component.html',
57     styleUrls: ['./users-details.component.scss'],
58     standalone: true,
59   })
60   export class UsersDetails {
61     topUsers: User[] = [];
62     selectedUser: User | null = null;
63     constructor(
64       private store: Store,
65       private userManagementService: UserManagementService,
66       private userSvc: AuthService,
67       private appContext: ApplicationContextService,
68       private alertService: AlertService,
69       private reasonsService: ReasonsService
70     ) {}
71
72     loading = false;
73     usersData: User[] = [];
74
75     ngOnInit(): void {
76       // Suscribirse a los datos del contexto
77       this.subscribeToContextData();
78       this.loadConfig();
79     }
80
81     private subscribeToContextData() {
82       // Suscribirse a los datos del usuario actual
83       this.appContext.currentUser$.subscribe(user => {
84         // Verificar si el usuario actual es admin
85         if (user && user.isAdmin) {
86           // Solo los admins pueden ver esta página
87           this.loadUsersData();
88         }
89       });
90
91       // Suscribirse a los estados de carga
92       this.appContext.isLoading$.subscribe/loading => {
93         this.loading = loading.user;
94       });
95
96       // Suscribirse a los errores
97       this.appContext.errors$.subscribe(errors => {
98         if (errors.user) {
99           console.error('Error en gestión de usuarios:', errors.user);
100        }
101      });
102    }
103
104    loadConfig() {
105      this.getUsersData();
106    }
107
108    loadUsersData() {
109      this.getUsersData();
110    }
111
112    async getUsersData() {
113      try {
114        const allUsers = await this.userManagementService.getAllUsers();
115        this.usersData = allUsers.filter((user) => !user.isAdmin);
116        this.loading = false;
117      } catch (error) {

```

```

118     this.loading = false;
119     console.error('Error getting users data:', error);
120   }
121 }
122
123 async onBan(event: { username: string; reason: string }) {
124   if (!this.selectedUser) return;
125   this.loading = true;
126   const userId = String(this.selectedUser.id);
127   this.reasonsService
128     .createReason(userId, event?.reason || '')
129     .then(() => this.userManagementService.updateUser(userId, { status:
130       UserStatus.BANNED }))=> {
131     await this.getUsersData();
132     const refreshed = await this.userManagementService.getUserById(userId);
133     if (refreshed) this.selectedUser = refreshed;
134     this.alertService.showSuccess('User banned successfully', 'Ban User');
135   }
136   .catch((err) => {
137     console.error('Error banning user', err);
138     this.alertService.showError('Error banning user', 'Ban User');
139   })
140   .finally(() => (this.loading = false));
141 }
142
143 async onUnban(username: string) {
144   if (!this.selectedUser) return;
145   this.loading = true;
146   const userId = String(this.selectedUser.id);
147   this.userManagementService
148     .updateUser(userId, { status: UserStatus.ACTIVE })
149     .then(async () => {
150       const openReason = await this.reasonsService.getOpenLatestReason(userId);
151       if (openReason?.id) {
152         await this.reasonsService.updateReason(userId, openReason.id, { dateUnban:
153           serverTimestamp() } as any);
154       }
155     .then(async () => {
156       await this.getUsersData();
157       const refreshed = await this.userManagementService.getUserById(userId);
158       if (refreshed) this.selectedUser = refreshed;
159       this.alertService.showSuccess('User unbanned successfully', 'Unban User');
160     })
161     .catch((err) => {
162       console.error('Error unbanning user', err);
163       this.alertService.showError('Error unbanning user', 'Unban User');
164     })
165     .finally(() => (this.loading = false));
166 }
167
168 onSendResetLink(email: string) {
169   if (!email) {
170     this.alertService.showInfo('Email is required', 'Password reset');
171     return;
172   }
173   this.loading = true;
174   this.userSvc
175     .sendPasswordReset(email)
176     .then(() => this.alertService.showSuccess('Reset link sent', 'Password reset'))
177     .catch((err) => {
178       console.error('Error sending reset link', err);
179       this.alertService.showError('Error sending reset link', 'Password reset');
180     })
181     .finally(() => (this.loading = false));
182 }
183
184 onLogoutEverywhere(userId: string) {
185   if (!userId) return;
186   this.loading = true;
187   this.userSvc

```

```

188     .deleteLinkToken(userId as any)
189     .then(() => this.alertService.showSuccess('Firebase token revoked', 'Logout
190 everywhere'))(err) => {
191     console.error('Error revoking token', err);
192     this.alertService.showError('Error revoking token', 'Logout everywhere');
193   }
194   .finally(() => (this.loading = false));
195 }
196 }
```

Ø=ÜÁ features\users-details\components\create-user-role-popup

Ø=ÜÄ features\users-details\components\create-user-role-popup\create-user-role-popup.component.ts

```

1 import { Component, EventEmitter, Input, Output, OnChanges, SimpleChanges } from '@angular/
2 CommonModule { CommonModule } from '@angular/common';
3 import { ReactiveFormsModule, FormBuilder, FormGroup, Validators, AbstractControl,
4 ValidationErrors, FormControl } from '@angular/forms'; // shared/components/text-input/text-
5 input-component/InputComponent } from '../../../../../shared/components/password-input/
6 password-input-component/PasswordInputComponent } from '../../../../../shared/components/phone-input/phone-
7 input-component/BirthdayInputComponent } from '../../../../../shared/components/birthday-input/
8 birthday-input-component/BirthdayInputComponent' from '../../../../../shared/services/auth.service';
9 import { SubscriptionService, Subscription } from '../../../../../shared/services/subscription-
10 service'; // User, UserStatus } from '../../../../../overview/models/overview';
11 import { LinkToken } from '../../../../../shared/services/tokens-operations.service';
12 import { UserCredentials } from '../../../../../auth/models/userModel';
13 import { AlertService } from '../../../../../shared/services/alert.service';
14
15 /**
16 * Component for creating new users with role selection (user or admin).
17 *
18 * This component provides a two-step process for creating new users:
19 * 1. Role selection (user or admin)
20 * 2. User registration form with validation
21 *
22 * It handles complete user creation including Firebase Authentication,
23 * user document creation, link token generation, and free subscription assignment.
24 *
25 * Features:
26 * - Two-step wizard (role selection !' form)
27 * - Role selection (user or admin)
28 * - Form validation (email, password, phone, birthday, name)
29 * - Email uniqueness check
30 * - Age validation (minimum 18 years)
31 * - Phone number validation
32 * - Email format validation
33 * - Confirmation dialogs before creation
34 * - Success state with options to create another or go to list
35 * - Automatic free subscription assignment
36 *
37 * Validation:
38 * - Email: Must be unique and valid format
39 * - Password: Minimum 8 characters
40 * - Phone: Valid international format (10-15 digits)
41 * - Birthday: User must be at least 18 years old
42 * - Name: Minimum 2 characters for first and last name
43 *
44 * Relations:
45 * - AuthService: Creates user in Firebase Auth and Firestore
46 * - SubscriptionService: Creates free subscription for new users
47 * - TextInputComponent: Form input component
48 * - PasswordInputComponent: Password input with validation
49 * - PhoneInputComponent: Phone number input with country code
```

```

50 * - BirthdayInputComponent: Date picker for birthday
51 * - AlertService: Shows error notifications
52 *
53 * @component
54 * @selector app-create-user-role-popup
55 * @standalone true
56 */
57 @Component({
58   selector: 'app-create-user-role-popup',
59   standalone: true,
60   imports: [CommonModule, ReactiveFormsModule, TextInputComponent, PasswordInputComponent,
61             SharedModule, BirthdayInputComponent],
62   styleUrls: ['./create-user-role-popup.component.scss']
63 })
64 export class CreateUserRolePopupComponent implements OnChanges {
65   @Input() visible = false;
66   @Output() close = new EventEmitter<void>();
67   @Output() selectRole = new EventEmitter<'user' | 'admin'>();
68   @Output() created = new EventEmitter<void>();
69
70   step: 'role' | 'form' = 'role';
71   role: 'user' | 'admin' | null = null;
72   form: FormGroup;
73   showCancelConfirm = false;
74   showSuccess = false;
75   showCreateConfirm = false;
76
77   constructor(private fb: FormBuilder, private authService: AuthService, private
78   subscriptionService: SubscriptionService, private alertService: AlertService) {
79     this.form = this.fb.group({
80       firstName: ['', [Validators.required, Validators.minLength(2)]],
81       lastName: ['', [Validators.required, Validators.minLength(2)]],
82       email: ['', [Validators.required, Validators.email, this.emailValidator]],
83       password: ['', [Validators.required, Validators.minLength(8)]],
84       birthday: ['', [Validators.required, this.ageValidator]],
85       phoneNumber: ['', [Validators.required, this.phoneValidator]],
86     });
87   }
88
89   ngOnChanges(changes: SimpleChanges): void {
90     if (changes['visible'] && !this.visible) {
91       // Al cerrarse, dejar todo listo para comenzar siempre desde 'role'
92       this.step = 'role';
93       this.role = null;
94       this.form.reset({ firstName: '', lastName: '', email: '', password: '',
95       phoneNumber: '' });
96       this.showCancelConfirm = false;
97       this.showSuccess = false;
98     }
99   }
100
101   onSelect(role: 'user' | 'admin') {
102     this.role = role;
103     this.step = 'form';
104     this.selectRole.emit(role);
105   }
106
107   onCancel() {
108     this.showCancelConfirm = true;
109   }
110
111   // Cancel confirmation overlay actions
112   confirmCancel() {
113     this.showCancelConfirm = false;
114     this.step = 'role';
115     this.role = null;
116     this.close.emit();
117   }
118
119   keepEditing() {
120     this.showCancelConfirm = false;
121   }

```

```

120 // Pre-confirmation before creating the user
121 submitCreateUser() {
122   if (!this.form.valid || !this.role) return;
123   this.showCreateConfirm = true;
124 }
125
126
127
128
129 async confirmCreate() {
130   try {
131     const email = this.form.value.email;
132     const password = this.form.value.password;
133     // Crear usuario en Firebase Auth
134     const credentials: UserCredentials = { email, password } as { email: string; password: string };
135
136     const existingUser = await this.authService.getUserByEmail(email);
137     if (existingUser) {
138       this.alertService.showError('This email is already registered. Please use a different email or try logging in.', 'Email Already Registered');
139     }
140   }
141
142   const userResponse = await this.authService.register(credentials);
143   const userId = userResponse.user.uid;
144
145   // Token
146   const token: LinkToken = {
147     id: email.split('@')[0] + userId.substring(0, 4),
148     userId,
149   } as any;
150
151   // User doc
152   const user: User = {
153     id: userId,
154     email,
155     tokenId: token.id,
156     firstName: this.form.value.firstName,
157     lastName: this.form.value.lastName,
158     phoneNumber: this.form.value.phoneNumber,
159     birthday: this.form.value.birthday,
160     best_trade: 0,
161     netPnl: 0,
162     number_trades: 0,
163     profit: 0,
164     status: this.role === 'admin' ? UserStatus.ADMIN : UserStatus.ACTIVE,
165     strategy_followed: 0,
166     subscription_date: new Date().getTime(),
167     lastUpdated: new Date().getTime(),
168     total_spend: 0,
169     isAdmin: this.role === 'admin' ? true : false,
170     trading_accounts: 0,
171     strategies: 0,
172   };
173
174   await this.authService.createUser(user as User);
175   await this.authService.createLinkToken(token);
176
177   // Crear suscripción Free
178   const freeSubscriptionData: Omit<Subscription, 'id' | 'created_at' | 'updated_at'> = {
179     planId: 'Cb1B0tpxdE6AP6eMZDo0',
180     status: UserStatus.ACTIVE,
181     userId,
182   };
183   await this.subscriptionService.createSubscription(userId, freeSubscriptionData);
184
185   this.showSuccess = true;
186   this.created.emit();
187 } catch (e) {
188   console.error('Error creating user:', e);
189 }

```

```

190     }
191
192     keepEditingCreate() {
193         this.showCreateConfirm = false;
194     }
195
196     // ===== Validators (ported from signup.ts) =====
197     private phoneValidator(control: AbstractControl): ValidationErrors | null {
198         if (!control.value) return null;
199         const phoneRegex = /^[+]?[1-9][\d]{0,15}$/;
200         const cleanPhone = String(control.value).replace(/[\s\-\(\)]/g, '');
201         if (!phoneRegex.test(cleanPhone)) {
202             return { invalidPhone: true };
203         }
204         if (cleanPhone.length < 10 || cleanPhone.length > 15) {
205             return { invalidPhoneLength: true };
206         }
207         return null;
208     }
209
210     private ageValidator(control: AbstractControl): ValidationErrors | null {
211         if (!control.value) return null;
212         const today = new Date();
213         const birthDate = new Date(control.value);
214         let age = today.getFullYear() - birthDate.getFullYear();
215         const monthDiff = today.getMonth() - birthDate.getMonth();
216         if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) {
217             age--;
218         }
219         if (age < 18) {
220             return { underage: true };
221         }
222         return null;
223     }
224
225     private emailValidator(control: AbstractControl): ValidationErrors | null {
226         if (!control.value) return null;
227         const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$/;
228         if (!emailRegex.test(control.value)) {
229             return { invalidEmailFormat: true };
230         }
231         return null;
232     }
233
234     successCreateAnother() {
235         this.showSuccess = false;
236         this.form.reset({ firstName: '', lastName: '', email: '', password: '', phoneNumber: '' });
237     }
238
239     successGoToList() {
240         this.showSuccess = false;
241         this.step = 'role';
242         this.role = null;
243         this.close.emit();
244     }
245 }

```

Ø=ÜÁ features\users-details\components\user-modal

Ø=ÜÄ features\users-details\components\user-modal\user-modal.component.ts

```

1 import { Component, Input, Output, EventEmitter } from '@angular/core';
2 import { User } from '../../../../../overview/models/overview';

```

```

3 import { FormsModule } from '@angular/forms';
4 import { CommonModule } from '@angular/common';
5 import { Timestamp } from 'firebase/firestore';
6 import { CountryOption } from '../../../../../shared/services/countryService';
7
8 /**
9  * Component for displaying detailed user information in a modal.
10 *
11 * This component shows comprehensive user details including personal information,
12 * trading statistics, account metrics, and provides action buttons for administrative
13 * operations. It includes phone number normalization and country detection from
14 * phone numbers.
15 *
16 * Features:
17 * - Display user personal information (name, email, phone, birthday)
18 * - Show trading statistics (profit, net PnL, trades, spend)
19 * - Calculate and display average order value (AOV)
20 * - Phone number country detection
21 * - Ban user with reason input
22 * - Unban user
23 * - Send password reset link
24 * - Logout user everywhere (revoke tokens)
25 *
26 * Phone Number Handling:
27 * - Normalizes phone numbers with international dial codes
28 * - Detects country from phone number prefix
29 * - Supports fallback country list for detection
30 *
31 * Relations:
32 * - UsersDetails: Parent component that handles ban/unban actions
33 * - CountryService: Provides country options (imported but uses fallback list)
34 *
35 * @component
36 * @selector app-user-modal
37 * @standalone true
38 */
39 @Component({
40   selector: 'app-user-modal',
41   templateUrl: './user-modal.component.html',
42   styleUrls: ['./user-modal.component.scss'],
43   imports: [FormsModule, CommonModule],
44   standalone: true,
45 })
46 export class UserModalComponent {
47   @Input() user!: User;
48   @Output() close = new EventEmitter<void>();
49   @Output() ban = new EventEmitter<{ username: string; reason: string }>();
50   @Output() unban = new EventEmitter<string>();
51   @Output() sendResetLink = new EventEmitter<string>();
52   @Output() logoutEverywhere = new EventEmitter<string>();
53
54   today = new Date();
55
56   usernameToBan = '';
57   banReason = '';
58
59   private readonly DEFAULT_DIAL_CODE = '+57';
60   private readonly fallbackCountries: { dialCode: string; name: string }[] = [
61     // North America
62     { dialCode: '+1', name: 'United States/Canada' },
63     { dialCode: '+1242', name: 'Bahamas' },
64     { dialCode: '+1246', name: 'Barbados' },
65     { dialCode: '+1264', name: 'Anguilla' },
66     { dialCode: '+1268', name: 'Antigua and Barbuda' },
67     { dialCode: '+1284', name: 'British Virgin Islands' },
68     { dialCode: '+1340', name: 'US Virgin Islands' },
69     { dialCode: '+1345', name: 'Cayman Islands' },
70     { dialCode: '+1441', name: 'Bermuda' },
71     { dialCode: '+1473', name: 'Grenada' },
72     { dialCode: '+1649', name: 'Turks and Caicos Islands' },

```

```
73 { dialCode: '+1664', name: 'Montserrat' },
74 { dialCode: '+1670', name: 'Northern Mariana Islands' },
75 { dialCode: '+1671', name: 'Guam' },
76 { dialCode: '+1684', name: 'American Samoa' },
77 { dialCode: '+1758', name: 'Saint Lucia' },
78 { dialCode: '+1767', name: 'Dominica' },
79 { dialCode: '+1784', name: 'Saint Vincent and the Grenadines' },
80 { dialCode: '+1809', name: 'Dominican Republic' },
81 { dialCode: '+1829', name: 'Dominican Republic' },
82 { dialCode: '+1849', name: 'Dominican Republic' },
83 { dialCode: '+1868', name: 'Trinidad and Tobago' },
84 { dialCode: '+1869', name: 'Saint Kitts and Nevis' },
85 { dialCode: '+1876', name: 'Jamaica' },
86 { dialCode: '+1939', name: 'Puerto Rico' },
87 // Central America
88 { dialCode: '+502', name: 'Guatemala' },
89 { dialCode: '+503', name: 'El Salvador' },
90 { dialCode: '+504', name: 'Honduras' },
91 { dialCode: '+505', name: 'Nicaragua' },
92 { dialCode: '+506', name: 'Costa Rica' },
93 { dialCode: '+507', name: 'Panama' },
94 { dialCode: '+501', name: 'Belize' },
95 // South America
96 { dialCode: '+54', name: 'Argentina' },
97 { dialCode: '+55', name: 'Brazil' },
98 { dialCode: '+56', name: 'Chile' },
99 { dialCode: '+57', name: 'Colombia' },
100 { dialCode: '+58', name: 'Venezuela' },
101 { dialCode: '+51', name: 'Peru' },
102 { dialCode: '+593', name: 'Ecuador' },
103 { dialCode: '+595', name: 'Paraguay' },
104 { dialCode: '+598', name: 'Uruguay' },
105 { dialCode: '+591', name: 'Bolivia' },
106 { dialCode: '+592', name: 'Guyana' },
107 { dialCode: '+597', name: 'Suriname' },
108 // Europe
109 { dialCode: '+34', name: 'Spain' },
110 { dialCode: '+351', name: 'Portugal' },
111 { dialCode: '+33', name: 'France' },
112 { dialCode: '+49', name: 'Germany' },
113 { dialCode: '+39', name: 'Italy' },
114 { dialCode: '+44', name: 'United Kingdom' },
115 { dialCode: '+31', name: 'Netherlands' },
116 { dialCode: '+32', name: 'Belgium' },
117 { dialCode: '+352', name: 'Luxembourg' },
118 { dialCode: '+41', name: 'Switzerland' },
119 { dialCode: '+43', name: 'Austria' },
120 { dialCode: '+45', name: 'Denmark' },
121 { dialCode: '+46', name: 'Sweden' },
122 { dialCode: '+47', name: 'Norway' },
123 { dialCode: '+358', name: 'Finland' },
124 { dialCode: '+48', name: 'Poland' },
125 { dialCode: '+420', name: 'Czech Republic' },
126 { dialCode: '+421', name: 'Slovakia' },
127 { dialCode: '+36', name: 'Hungary' },
128 { dialCode: '+386', name: 'Slovenia' },
129 { dialCode: '+385', name: 'Croatia' },
130 { dialCode: '+381', name: 'Serbia' },
131 { dialCode: '+382', name: 'Montenegro' },
132 { dialCode: '+389', name: 'North Macedonia' },
133 { dialCode: '+387', name: 'Bosnia and Herzegovina' },
134 { dialCode: '+30', name: 'Greece' },
135 { dialCode: '+357', name: 'Cyprus' },
136 { dialCode: '+353', name: 'Ireland' },
137 { dialCode: '+354', name: 'Iceland' },
138 { dialCode: '+371', name: 'Latvia' },
139 { dialCode: '+370', name: 'Lithuania' },
140 { dialCode: '+372', name: 'Estonia' },
141 { dialCode: '+375', name: 'Belarus' },
142 { dialCode: '+380', name: 'Ukraine' },
```

```

143 { dialCode: '+373', name: 'Moldova' },
144 { dialCode: '+7', name: 'Russia/Kazakhstan' },
145 { dialCode: '+376', name: 'Andorra' },
146 { dialCode: '+377', name: 'Monaco' },
147 { dialCode: '+378', name: 'San Marino' },
148 { dialCode: '+379', name: 'Vatican City' },
149 { dialCode: '+423', name: 'Liechtenstein' },
150 { dialCode: '+356', name: 'Malta' },
151 { dialCode: '+298', name: 'Faroe Islands' },
152 { dialCode: '+47', name: 'Svalbard and Jan Mayen' },
153 // Africa
154 { dialCode: '+20', name: 'Egypt' },
155 { dialCode: '+212', name: 'Morocco' },
156 { dialCode: '+213', name: 'Algeria' },
157 { dialCode: '+216', name: 'Tunisia' },
158 { dialCode: '+218', name: 'Libya' },
159 { dialCode: '+221', name: 'Senegal' },
160 { dialCode: '+225', name: 'Ivory Coast' },
161 { dialCode: '+229', name: 'Benin' },
162 { dialCode: '+233', name: 'Ghana' },
163 { dialCode: '+234', name: 'Nigeria' },
164 { dialCode: '+237', name: 'Cameroon' },
165 { dialCode: '+240', name: 'Equatorial Guinea' },
166 { dialCode: '+241', name: 'Gabon' },
167 { dialCode: '+242', name: 'Republic of the Congo' },
168 { dialCode: '+243', name: 'Democratic Republic of the Congo' },
169 { dialCode: '+244', name: 'Angola' },
170 { dialCode: '+248', name: 'Seychelles' },
171 { dialCode: '+249', name: 'Sudan' },
172 { dialCode: '+250', name: 'Rwanda' },
173 { dialCode: '+251', name: 'Ethiopia' },
174 { dialCode: '+252', name: 'Somalia' },
175 { dialCode: '+253', name: 'Djibouti' },
176 { dialCode: '+254', name: 'Kenya' },
177 { dialCode: '+255', name: 'Tanzania' },
178 { dialCode: '+256', name: 'Uganda' },
179 { dialCode: '+257', name: 'Burundi' },
180 { dialCode: '+258', name: 'Mozambique' },
181 { dialCode: '+260', name: 'Zambia' },
182 { dialCode: '+261', name: 'Madagascar' },
183 { dialCode: '+263', name: 'Zimbabwe' },
184 { dialCode: '+264', name: 'Namibia' },
185 { dialCode: '+265', name: 'Malawi' },
186 { dialCode: '+266', name: 'Lesotho' },
187 { dialCode: '+267', name: 'Botswana' },
188 { dialCode: '+268', name: 'Eswatini' },
189 { dialCode: '+269', name: 'Comoros' },
190 { dialCode: '+27', name: 'South Africa' },
191 { dialCode: '+290', name: 'Saint Helena' },
192 { dialCode: '+291', name: 'Eritrea' },
193 { dialCode: '+220', name: 'Gambia' },
194 { dialCode: '+223', name: 'Mali' },
195 { dialCode: '+226', name: 'Burkina Faso' },
196 { dialCode: '+ Niger', name: 'Niger' },
197 { dialCode: '+228', name: 'Togo' },
198 { dialCode: '+232', name: 'Sierra Leone' },
199 { dialCode: '+231', name: 'Liberia' },
200 { dialCode: '+235', name: 'Chad' },
201 { dialCode: '+236', name: 'Central African Republic' },
202 { dialCode: '+237', name: 'Cameroon' },
203 { dialCode: '+238', name: 'Cape Verde' },
204 { dialCode: '+239', name: 'São Tomé and Príncipe' },
205 { dialCode: '+248', name: 'Seychelles' },
206 { dialCode: '+ Mauritania', name: 'Mauritania' },
207 { dialCode: '+222', name: 'Mauritania' },
208 { dialCode: '+230', name: 'Mauritius' },
209 // Middle East / Asia
210 { dialCode: '+90', name: 'Turkey' },
211 { dialCode: '+30', name: 'Greece' },
212 { dialCode: '+972', name: 'Israel' },

```

```

213     { dialCode: '+970', name: 'Palestine' },
214     { dialCode: '+961', name: 'Lebanon' },
215     { dialCode: '+962', name: 'Jordan' },
216     { dialCode: '+963', name: 'Syria' },
217     { dialCode: '+964', name: 'Iraq' },
218     { dialCode: '+965', name: 'Kuwait' },
219     { dialCode: '+966', name: 'Saudi Arabia' },
220     { dialCode: '+971', name: 'United Arab Emirates' },
221     { dialCode: '+973', name: 'Bahrain' },
222     { dialCode: '+974', name: 'Qatar' },
223     { dialCode: '+968', name: 'Oman' },
224     { dialCode: '+98', name: 'Iran' },
225     { dialCode: '+92', name: 'Pakistan' },
226     { dialCode: '+93', name: 'Afghanistan' },
227     { dialCode: '+94', name: 'Sri Lanka' },
228     { dialCode: '+95', name: 'Myanmar' },
229     { dialCode: '+855', name: 'Cambodia' },
230     { dialCode: '+856', name: 'Laos' },
231     { dialCode: '+66', name: 'Thailand' },
232     { dialCode: '+84', name: 'Vietnam' },
233     { dialCode: '+60', name: 'Malaysia' },
234     { dialCode: '+62', name: 'Indonesia' },
235     { dialCode: '+63', name: 'Philippines' },
236     { dialCode: '+65', name: 'Singapore' },
237     { dialCode: '+81', name: 'Japan' },
238     { dialCode: '+82', name: 'South Korea' },
239     { dialCode: '+86', name: 'China' },
240     { dialCode: '+852', name: 'Hong Kong' },
241     { dialCode: '+853', name: 'Macau' },
242     { dialCode: '+886', name: 'Taiwan' },
243     { dialCode: '+91', name: 'India' },
244     { dialCode: '+880', name: 'Bangladesh' },
245     { dialCode: '+977', name: 'Nepal' },
246     { dialCode: '+975', name: 'Bhutan' },
247     { dialCode: '+960', name: 'Maldives' },
248     { dialCode: '+7', name: 'Kazakhstan' },
249     { dialCode: '+998', name: 'Uzbekistan' },
250     { dialCode: '+996', name: 'Kyrgyzstan' },
251     { dialCode: '+992', name: 'Tajikistan' },
252     { dialCode: '+993', name: 'Turkmenistan' },
253     { dialCode: '+373', name: 'Moldova' },
254     // Oceania
255     { dialCode: '+61', name: 'Australia' },
256     { dialCode: '+64', name: 'New Zealand' },
257     { dialCode: '+679', name: 'Fiji' },
258     { dialCode: '+682', name: 'Cook Islands' },
259     { dialCode: '+683', name: 'Niue' },
260     { dialCode: '+685', name: 'Samoa' },
261     { dialCode: '+676', name: 'Tonga' },
262     { dialCode: '+678', name: 'Vanuatu' },
263     { dialCode: '+675', name: 'Papua New Guinea' },
264     { dialCode: '+674', name: 'Nauru' },
265     { dialCode: '+686', name: 'Kiribati' },
266     { dialCode: '+688', name: 'Tuvalu' },
267     { dialCode: '+690', name: 'Tokelau' },
268     { dialCode: '+689', name: 'French Polynesia' },
269     { dialCode: '+687', name: 'New Caledonia' },
270     { dialCode: '+681', name: 'Wallis and Futuna' },
271   ];
272
273
274   private normalizePhone(input?: string): string | null {
275     if (!input) return null;
276     const raw = input
277       .toString()
278       .trim()
279       .replace(/[\s\-\(\)]/g, '');
280     if (!raw) return null;
281     if (raw.startsWith('+')) return raw;
282     if (raw.startsWith('00')) return `+${raw.slice(2)}`;

```

```

283     // Sin prefijo internacional, asumimos código por defecto para detección
284     return `${this.DEFAULT_DIAL_CODE}${raw}`;
285   }
286
287   private resolveCountryFromPhone(phone?: string): string {
288     const normalized = this.normalizePhone(phone);
289     if (!normalized) return 'Unknown';
290     const match = [...this.fallbackCountries]
291       .sort((a, b) => b.dialCode.length - a.dialCode.length)
292       .find((c) => normalized.startsWith(c.dialCode));
293     return match ? match.name : 'Unknown';
294   }
295
296   get countryFromPhone(): string {
297     return this.resolveCountryFromPhone(this.user?.phoneNumber as unknown as string);
298   }
299
300   get aov(): string {
301     return this.user.total_spend && this.user.number_trades
302       ? (this.user.total_spend / this.user.number_trades).toFixed(2)
303       : '0.00';
304   }
305
306   onlyNameInitials(user: User) {
307     return user.firstName.charAt(0) + user.lastName.charAt(0);
308   }
309   getUserDate(date: number): Date {
310     return new Date(date);
311   }
312 }
313

```

Ø=ÜÁ features\users-details\components\users-table

Ø=ÜÁ features\users-details\components\users-table\users-table.component.ts

```

1 import { Component, Input, Output } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 import { FormsModule } from '@angular/forms';
5 import { User } from '../../../../../overview/models/overview';
6 import { EventEmitter } from '@angular/core';
7 import { Timestamp } from 'firebase/firestore';
8 import { CreateUserRolePopupComponent } from '../create-user-role-popup/create-user-role-
9 popup.component';
10 /**
11  * Component for displaying a filterable and sortable table of users.
12  *
13  * This component displays users in a paginated table with search, filtering,
14  * and sorting capabilities. It also includes functionality to create new users
15  * (both regular users and admins) through a popup component.
16  *
17  * Features:
18  * - Search users by name (first name + last name)
19  * - Filter by strategy followed percentage range
20  * - Sort by first name or last name (ascending/descending)
21  * - Pagination with configurable items per page
22  * - Status classification (banned, created, active)
23  * - Create new user popup (user or admin role)
24  * - User selection for detailed view
25  *
26  * Status Classification:
27  * - "banned": User status is explicitly banned

```

```

28 * - "created": All user metrics are zero (newly created account)
29 * - "active": User has activity (non-zero metrics)
30 *
31 * Relations:
32 * - CreateUserRolePopupComponent: Modal for creating new users
33 * - UsersDetails: Parent component that receives selected users
34 *
35 * @component
36 * @selector app-users-table
37 * @standalone true
38 */
39 @Component({
40   selector: 'app-users-table',
41   standalone: true,
42   imports: [CommonModule, FormsModule, CreateUserRolePopupComponent],
43   templateUrl: './users-table.component.html',
44   styleUrls: ['./users-table.component.scss'],
45 })
46 export class UsersTableComponent {
47   @Input() users: User[] = [];
48   @Output() userSelected = new EventEmitter<User>();
49   @Output() userCreated = new EventEmitter<void>();
50
51   initialMinStrat = 0;
52   initialMaxStrat = 100;
53   showFilter = false;
54   currentPage: number = 1;
55   itemsPerPage: number = 10;
56   sortField: 'firstName' | 'lastName' = 'firstName';
57   sortAsc: boolean = true;
58
59   showCreateUserPopup = false;
60
61   private _searchTerm = '';
62   get searchTerm(): string {
63     return this._searchTerm;
64   }
65   set searchTerm(val: string) {
66     this._searchTerm = val;
67     this.goToPage(1);
68   }
69
70   get filteredUsers(): User[] {
71     const lower = this._searchTerm.trim().toLowerCase();
72
73     let result = this.users.filter((user) => {
74       const matchesSearch = `${user.firstName} ${user.lastName}`
75         .toLowerCase()
76         .includes(lower);
77
78     let matchesMinStrat = user.strategy_followed >= this.initialMinStrat;
79     let matchesMaxStrat = user.strategy_followed <= this.initialMaxStrat;
80
81     if (user.strategy_followed === undefined) {
82       matchesMinStrat = true;
83       matchesMaxStrat = true;
84     }
85
86     return matchesSearch && matchesMinStrat && matchesMaxStrat;
87   });
88
89   result = result.sort((a, b) => {
90     const fieldA = a[this.sortField].toLowerCase();
91     const fieldB = b[this.sortField].toLowerCase();
92     if (fieldA < fieldB) return this.sortAsc ? -1 : 1;
93     if (fieldA > fieldB) return this.sortAsc ? 1 : -1;
94     return 0;
95   });
96
97   return result;

```

```

98 }
99
100 get paginatedUsers(): User[] {
101   const start = (this.currentPage - 1) * this.itemsPerPage;
102   const end = start + this.itemsPerPage;
103   return this.filteredUsers.slice(start, end);
104 }
105
106 get totalPages(): number {
107   return Math.ceil(this.filteredUsers.length / this.itemsPerPage);
108 }
109
110 statusClass(user: User): string {
111   // Si el status es banned, retornar banned
112   if (String(user.status) === 'banned') {
113     return 'banned';
114   }
115
116   // Verificar si todos los valores están en 0
117   const allValuesZero =
118     (user.trading_accounts ?? 0) === 0 &&
119     (user.strategies ?? 0) === 0 &&
120     (user.strategy_followed ?? 0) === 0 &&
121     (user.netPnl ?? 0) === 0 &&
122     (user.profit ?? 0) === 0 &&
123     (user.number_trades ?? 0) === 0 &&
124     (user.total_spend ?? 0) === 0;
125
126   // Si todos los valores están en 0, retornar created
127   if (allValuesZero) {
128     return 'created';
129   }
130
131   // Si no todos están en 0, retornar active
132   return 'active';
133 }
134
135 getDisplayStatus(user: User): string {
136   const status = this.statusClass(user);
137   return status.charAt(0).toUpperCase() + status.slice(1);
138 }
139
140 returnClass(returnValue: number) {
141   return returnValue >= 0 ? 'green' : 'red';
142 }
143
144 openFilter() {
145   this.showFilter = !this.showFilter;
146 }
147
148 closeFilter() {
149   this.showFilter = false;
150 }
151
152 apply() {
153   this.showFilter = false;
154
155   this.applyFilters();
156 }
157
158 applyFilters() {
159   this.goToPage(1);
160 }
161
162 resetFilters() {
163   this.initialMinStrat = 0;
164   this.initialMaxStrat = 100;
165   this.applyFilters();
166 }
167

```

```

168  goToPage(page: number) {
169    if (page < 1) page = 1;
170    if (page > this.totalPages) page = this.totalPages;
171    this.currentPage = page;
172  }
173
174  prevPage() {
175    this.goToPage(this.currentPage - 1);
176  }
177
178  nextPage() {
179    this.goToPage(this.currentPage + 1);
180  }
181
182  toggleSort() {
183    this.sortAsc = !this.sortAsc;
184  }
185
186  getUserDate(date: number): Date {
187    return new Date(date);
188  }
189
190  emitUser(user: User) {
191    this.userSelected.emit(user);
192  }
193
194  onOpenCreateUser() {
195    this.showCreateUserPopup = true;
196  }
197
198  onCloseCreateUser() {
199    this.showCreateUserPopup = false;
200  }
201
202  onSelectRole(role: 'user' | 'admin') {
203    // No cerrar el popup aquí; el componente interno cambia a step 'form'.
204  }
205
206  onPopupCreated() {
207    this.userCreated.emit();
208  }
209}

```