

PG設計書（プログラム設計書）

Tascal（タスカル） - 社員タスク管理・カレンダーアプリケーション

項目	内容
ドキュメントID	PG-001
バージョン	1.0
作成日	2026-02-26
プロジェクト名	SVN Dashboard App (Tascal)
ステータス	初版

1. プログラム一覧

1.1 Composables（ビジネスロジック）

No	ファイル	モジュール名	概要	行数
1	composables/useCalendar.ts	useCalendar	カレンダーメインロジック	~700
2	composables/useEventForm.ts	useEventForm	イベントフォームロジック	~500
3	composables/useUserProfile.ts	useUserProfile	ユーザープロフィール管理	~300
4	composables/useCsvFirestore.ts	useCsvFirestore	CSV入出力	~350
5	composables/useDailyOptions.ts	useDailyOptions	日別オプション管理	~150
6	composables/useLunchShuffle.ts	useLunchShuffle	ランチシャッフル	~250
7	composables/useWiki.ts	useWiki	Wiki管理	-
8	composables/firebase/useAuth.ts	useAuth	Firebase認証操作	-
9	composables/firebase/useFirestore.ts	useFirestore	Firestore CRUD操作	-
10	composables/firebase/useStorage.ts	useStorage	Cloud Storage操作	-
11	composables/firebase/useStoragePath.ts	useStoragePath	Storageパス管理	-
12	composables/firebase/useDocumentRoot.ts	useDocumentRoot	ドキュメントパス管理	-

No	ファイル	モジュール名	概要	行数
13	composables/firebase/useFirestoreSync.ts	useFirestoreSync	リアルタイム同期	-

1.2 Services（ドメインサービス）

No	ファイル	モジュール名	概要
1	services/eventService.ts	eventService	イベントCRUD・キャッシュ
2	services/wikiService.ts	wikiService	Wiki記事サンプルデータ
3	services/dailyOptionService.ts	dailyOptionService	日別オプションCRUD

1.3 Plugins（初期化プラグイン）

No	ファイル	概要
1	plugins/firebase.client.ts	Firebase SDK初期化・認証状態管理
2	plugins/vuetify.ts	Vuetify初期化
3	plugins/chart.ts	Chart.js グローバル登録
4	plugins/initState.ts	アプリ 初期状態設定

1.4 Middleware（ミドルウェア）

No	ファイル	概要
1	middleware/router.global.ts	グローバル認証チェック

1.5 型定義ファイル

No	ファイル	定義内容
1	types/EventForm.d.ts	イベント関連型（EventFormData, EventData, EventDisplay, RecurrenceRule等）
2	types/UserProfile.d.ts	ユーザー関連型（UserProfile, ExtendedUserProfile, UserFormData等）
3	types/DailyOption.d.ts	日別オプション型（DailyUserOption, WorkStyle, ParticipationStatus）
4	types/Facility.d.ts	施設型（Facility, FacilityFormData）
5	types/Equipment.d.ts	設備型（Equipment, EquipmentFormData）
6	types/Team.d.ts	チーム型（Team, TeamFormData）

No	ファイル	定義内容
7	types/Section.d.ts	部署型 (Section, SectionFormData)
8	types/WikiArticle.d.ts	Wiki型 (WikiArticle, WikiArticleForm, WikiArticleHistory)
9	types/Tag.d.ts	タグ型 (Tag)
10	types/Menu.d.ts	メニュー型 (Menu)
11	types/GroupMember.d.ts	グループメンバー型 (GroupMember)
12	types/LunchShuffle.d.ts	ランチシャッフル型 (LunchShuffle, ShuffleResult)
13	types/AppState.d.ts	アプリ状態型 (AppState)
14	types/FieldDefinition.d.ts	フォームフィールド定義型
15	types/FirebaseCommon.d.ts	Firebaseバッチ操作型 (BatchAction)
16	types/DocReference.d.ts	ドキュメント参照型
17	types/CloudStorageCommon.d.ts	Storage型 (CloudStorageFile, CloudStorageFileUploader)
18	types/OwnCompany.d.ts	自社情報型 (OwnCompany)

2. 主要プログラム詳細設計

2.1 useCalendar.ts

2.1.1 公開API

```
function useCalendar(): {  
  // ステート  
  currentDate: Ref<Date>  
  selectedDate: Ref<Date>  
  currentView: Ref<CalendarView>  
  users: Ref<ExtendedUserProfile[]>  
  facilities: Ref<Facility[]>  
  equipments: Ref<Equipment[]>  
  holidays: Ref<Holiday[]>  
  events: Ref<EventDisplay[]>  
  isLoading: Ref<boolean>  
  
  // イベント CRUD  
  createEventAndRefresh(formData: EventFormData): Promise<void>  
  updateEventAndRefresh(initialData: EventFormData, formData: EventFormData): Promise<void>  
  deleteEventAndRefresh(eventId: string, option?: string, targetDate?: string): Promise<void>  
  refreshEvents(): Promise<void>  
  loadData(forceRefresh?: boolean): Promise<void>  
  
  // カレンダーナビゲーション  
  setView(view: CalendarView): void  
  selectDay(date: Date): void  
  moveDate(unit: string, amount: number): void  
  previousDay(): void  
  nextDay(): void  
  previousWeek(): void  
  nextWeek(): void  
  previousMonth(): void  
  nextMonth(): void  
  goToToday(): void  
  goToSelectDate(date: Date): void  
  
  // データ取得  
  getSchedulesForDay(date: Date, userId?: string): EventDisplay[]  
  getUserSchedulesForDay(userId: string, date: Date): EventDisplay[]  
  isHoliday(date: Date): boolean  
  getHolidayName(date: Date): string  
  toggleUserVisibility(userId: string): void  
  
  // ユーティリティ  
  formatDate(date: Date): string // YYYY/MM/DD  
  formatDatetime(date: Date): string // YYYY/MM/DD HH:mm:ss  
  formatShortDate(date: Date): string // MM/DD  
  formatDateForDb(date: Date): string // YYYY-MM-DD  
  getDayOfWeek(date: Date): string // 日本語曜日名
```

```

timeToPixels(timeStr: string): number
timeToPixelsForHorizontal(timeStr: string): number

// カレンダーデータ
generateWeekDays: ComputedRef<CalendarDay[]>
generateCalendarDays: ComputedRef<CalendarDay[]>
timeSlots: ComputedRef<string[]>

// 位置保持
saveCalendarPosition(): void
loadCalendarPosition(): void
clearCalendarPosition(): void
}

```

2.1.2 プライベートイベントフィルタリング

```

function filterPrivateEvents(
  events: EventDisplay[],
  currentUserId: string
): EventDisplay[]

```

処理:

1. 各イベントの `isPrivate` フラグを確認
2. `isPrivate === true` かつ `participantIds` に `currentUserId` が含まれない場合
3. タイトルを「予定あり」に、説明を空に置換

2.2 useEventForm.ts

2.2.1 公開API

```
function useEventForm(initialData?: EventFormData): {  
  // ステート  
  formData: Reactive<EventFormData>  
  errors: Reactive<Record<string, string>>  
  conflicts: Ref<ConflictInfo[]>  
  notification: Ref<{message: string, type: string} | null>  
  isLoading: Ref<boolean>  
  isCheckingConflicts: Ref<boolean>  
  
  // 算出プロパティ  
  isValid: ComputedRef<boolean>  
  hasConflicts: ComputedRef<boolean>  
  
  // メソッド  
  validateForm(): boolean  
  validateField(fieldName: string): void  
  clearError(fieldName: string): void  
  checkConflicts(): Promise<void>  
  clearConflicts(): void  
  saveEvent(): Promise<void>  
  showNotification(message: string, type: string): void  
  resetForm(): void  
  setDefaultValues(data: Partial<EventFormData>): void  
}
```

2.2.2 コンフリクト検出ロジック

```
// 時間重複判定
function isTimeOverlapping(
  start1: string, // HH:mm
  end1: string,   // HH:mm
  start2: string, // HH:mm
  end2: string    // HH:mm
): boolean {
  const s1 = timeToMinutes(start1)
  const e1 = timeToMinutes(end1)
  const s2 = timeToMinutes(start2)
  const e2 = timeToMinutes(end2)
  return s1 < e2 && s2 < e1
}

// 時刻を分に変換
function timeToMinutes(timeStr: string): number {
  const [hours, minutes] = timeStr.split(':').map(Number)
  return hours * 60 + minutes
}

// 第N週目の曜日判定
function isNthWeekdayOfMonth(
  date: Date,
  weekday: number,
  n: number
): boolean
```

2.3 eventService.ts

2.3.1 公開API

```
function useEventService(): {  
  createEvent(formData: EventFormData): Promise<string[]>  
  updateEvent(  
    initialData: EventFormData,  
    formData: EventFormData  
  ): Promise<void>  
  deleteEvent(  
    eventId: string,  
    option?: 'single' | 'all' | 'after' | 'before',  
    targetDate?: string  
  ): Promise<void>  
  getEventsByParticipantInRange(  
    uid: string,  
    startDate: string,  
    endDate: string,  
    forceNoCache?: boolean  
  ): Promise<EventDisplay[]>  
  getEventsByFacilityInRange(  
    facilityId: string,  
    startDate: string,  
    endDate: string,  
    forceNoCache?: boolean  
  ): Promise<EventDisplay[]>  
  getEventsByEquipmentInRange(  
    equipmentId: string,  
    startDate: string,  
    endDate: string,  
    forceNoCache?: boolean  
  ): Promise<EventDisplay[]>  
  getEventsFromCacheAsync(  
    cacheKey: string,  
    forceNoCache?: boolean  
  ): Promise<EventDisplay[]>  
  getCacheKeyForDate(dateStr: string): string  
}
```

2.3.2 イベントエンティティ構造

```
// Firestoreに保存されるイベントデータ
interface EventEntity {
  title: string
  description: string
  dateType: DateType
  date: string // YYYY-MM-DD
  startDate: string // YYYY-MM-DD
  endDate: string // YYYY-MM-DD
  startTime: string // HH:mm
  endTime: string // HH:mm
  eventType: EventType
  participantIds: string[]
  facilityIds: string[]
  equipmentIds: string[]
  isPrivate: boolean
  masterId: string | null // 範囲/繰り返しのグループID
  recurrenceRule: RecurrenceRule | null
  createdAt: Timestamp // 自動設定
  updatedAt: Timestamp // 自動設定
  createdBy: string // 自動設定
  updatedBy: string // 自動設定
}
```

2.4 useFirestore.ts

2.4.1 公開API

```
function useFirestore(): {  
  // 読み取り  
  getCollectionAsync<T>(  
    collection: string,  
    ...constraints: QueryConstraint[]  
  ): Promise<T[]>  
  getCollectionWithCollectionGroupAsync<T>(  
    collection: string,  
    ...constraints: QueryConstraint[]  
  ): Promise<T[]>  
  countCollectionAsync(  
    collection: string,  
    ...constraints: QueryConstraint[]  
  ): Promise<number>  
  getDocAsync<T>(collection: string, id: string): Promise<T | null>  
  getDocWithRefAsync<T>(reference: DocumentReference): Promise<T | null>  
  existsDocWithRefAsync(reference: DocumentReference): Promise<boolean>  
  
  // 書き込み  
  addDocAsync(collection: string, data: any): Promise<string>  
  addDocWithRefAsync(reference: DocumentReference, data: any): Promise<void>  
  setDocWithRefAsync(reference: DocumentReference, data: any): Promise<void>  
  updateDocAsync(collection: string, id: string, data: any): Promise<void>  
  updateDocWithRefAsync(reference: DocumentReference, data: any): Promise<void>  
  deleteDocAsync(collection: string, id: string): Promise<void>  
  deleteDocWithRefAsync(reference: DocumentReference): Promise<void>  
  
  // バッチ  
  addWithBatch(batchActions: BatchAction[]): Promise<void>  
  updateWithBatch(batchActions: BatchAction[]): Promise<void>  
  deleteWithBatch(batchActions: BatchAction[]): Promise<void>  
  
  // ユーティリティ  
  getCollection(name: string): CollectionReference  
  getCollectionGroup(name: string): Query  
  getCollectionRef(name: string): CollectionReference  
  getDocRef(path: string): DocumentReference  
  generateSequenceNumberAsync(  
    manageCollection: string,  
    targetCollection: string,  
    field: string  
  ): Promise<number>  
  queryByIdsInChunks<T>(opts: {  
    collectionName: string  
    fieldPath: string  
    ids: string[]  
    chunkSize?: number  
  }): Promise<T[]>  
}
```

```
// プロファイリング
recordQuery(collectionName: string, startTime: number): void
printFirestoreDebugSummary(force?: boolean): void
resetFirestoreProfiler(): void
getFirestoreProfilerStats(): object
}
```

2.4.2 メタデータ自動付与

```
// 書き込み時に自動付与されるフィールド
{
  createdAt: serverTimestamp(), // addDoc時のみ
  updatedAt: serverTimestamp(), // 全書き込み時
  createdBy: user.value?.uid,    // addDoc時のみ
  updatedBy: user.value?.uid     // 全書き込み時
}
```

2.4.3 チャンク分割クエリ

```
// Firestoreの IN 句制限 (30件) を超えるIDリストを分割
async function queryByIdsInChunks<T>(opts: {
  collectionName: string // コレクション名
  fieldPath: string      // 検索フィールドパス
  ids: string[]           // IDリスト
  chunkSize?: number     // チャンクサイズ (デフォルト30)
}): Promise<T[]>
```

処理:

1. IDリストを chunkSize (30) 件ごとに分割
2. 各チャンクで `where(fieldPath, 'in', chunk)` クエリを実行
3. 結果を結合して返却

2.5 useAuth.ts

2.5.1 エラーハンドリング

```
// Firebase Authエラーコードと対応
const AUTH_ERROR_MAP = {
  'auth/user-not-found': 'ユーザーが見つかりません',
  'auth/wrong-password': 'パスワードが間違っています',
  'auth/email-already-in-use': 'このメールアドレスは既に使用されています',
  'auth/requires-recent-login': '再認証が必要です',
  'auth/weak-password': 'パスワードが弱すぎます',
  'auth/invalid-email': 'メールアドレスが無効です',
}
```

2.6 dailyOptionService.ts

2.6.1 公開API

```
function useDailyOptionService(): {
  getDailyOption(
    uid: string,
    date: string // YYYY-MM-DD
  ): Promise<DailyUserOption | null>

  getDailyOptionsInRange(
    startDate: string, // YYYY-MM-DD
    endDate: string // YYYY-MM-DD
  ): Promise<DailyUserOption[]>

  getDailyUserOptionsInRange(
    uid: string,
    startDate: string,
    endDate: string
  ): Promise<DailyUserOption[]>

  setDailyOption(
    optionData: DailyUserOption
  ): Promise<string> // ドキュメントID

  deleteDailyOption(
    uid: string,
    date: string
  ): Promise<void>
}
```

3. コンポーネント一覧

3.1 共通コンポーネント

No	ファイル	概要
1	components/AppHeader.vue	アプリケーションヘッダー
2	components/SideBar.vue	サイドバーメニュー
3	components/MarkdownRenderer.vue	Markdownレンダラー

3.2 カレンダーコンポーネント

No	ファイル	概要
1	components/Calendar/DailyView/*	日別ビューコンポーネント群
2	components/Calendar/WeeklyView/*	週別ビューコンポーネント群
3	components/Calendar/MonthlyView/*	月別ビューコンポーネント群

4. ページコンポーネント設計

4.1 カレンダーページ (pages/calendar/index.vue)

```
<template>
  | ビュー切替ボタン [日][週][月]
  | ナビゲーション [◀][今日][▶]
  |
  |
  |   currentView === 'daily'
  |   → <DailyView />
  |
  |   currentView === 'weekly'
  |   → <WeeklyView />
  |
  |   currentView === 'monthly'
  |   → <MonthlyView />
  |
</template>

<script setup>
  const {
    currentView, setView,
    previousDay, nextDay,
    previousWeek, nextWeek,
    previousMonth, nextMonth,
    goToToday, loadData, ...
  } = useCalendar()

  onMounted(() => loadData())
</script>
```

4.2 予定登録ページ (pages/calendar/register/index.vue)

```
<template>
  | 予定登録フォーム（フルスクリーンモーダル） |
  |-----|
  | タイトル: [ _____ ] |
  | 種別:      [▼ single/range/recurring] |
  | 日付:      [YYYY-MM-DD] |
  | 時間:      [HH:mm] ~ [HH:mm] |
  | タイプ:    [▼ meeting/focus/away/...] |
  | 参加者:    [選択モーダル] |
  | 施設:      [選択モーダル] |
  | 設備:      [選択モーダル] |
  | 説明:      [ _____ ] |
  | プライベート: [✓] |
  |-----|
  | [コンフリクト確認] [キャンセル] [登録] |
  |-----|
</template>

<script setup>
  const { formData, errors, validateForm,
          checkConflicts, saveEvent } = useEventForm()
  const { createEventAndRefresh,
          saveCalendarPosition } = useCalendar()
</script>
```

4.3 CRUD共通ページパターン

各マスター管理は以下のパターンに従う。

一覧ページ (index.vue)

- └─ v-data-table による一覧表示
- └─ 検索・フィルター機能
- └─ [新規作成]ボタン → new.vue へ遷移

新規登録ページ (new.vue)

- └─ フォーム入力
- └─ バリデーション
- └─ Firestore addDoc → 一覧へ遷移

詳細ページ ([id]/index.vue)

- └─ Firestore getDoc でデータ取得
- └─ 詳細情報表示
- └─ [編集]ボタン → edit.vue へ遷移

編集ページ ([id]/edit.vue)

- └─ Firestore getDoc でデータ取得
- └─ フォーム編集
- └─ バリデーション
- └─ Firestore updateDoc → 詳細へ遷移

5. コーディング規約

5.1 命名規則

対象	規則	例
変数・関数	camelCase	currentDate, loadData
コンポーネント	PascalCase	AppHeader, DailyView
型・インターフェース	PascalCase	EventFormData, UserProfile
ファイル (コンポーネント)	PascalCase	AppHeader.vue
ファイル (composables)	camelCase (use接頭辞)	useCalendar.ts
ファイル (services)	camelCase (Service接尾辞)	eventService.ts
ファイル (型定義)	PascalCase	EventForm.d.ts
定数	UPPER_SNAKE_CASE	MASTER_DATA_TTL
Firestoreコレクション	snake_case	daily_user_options

5.2 Vue/Nuxt 規約

- Composition API (`<script setup>`) を使用する

- リアクティブ変数は `ref()` / `reactive()` を使用する
- 共有ロジックは `composable` に抽出する
- グローバルステートは Nuxt の `useState()` を使用する
- 型定義は `types/` ディレクトリに集約する

5.3 Firestore 規約

- 書き込み時は `createdAt` / `updatedAt` / `createdBy` / `updatedBy` を自動付与する
- バッチ書き込みで複数ドキュメントの整合性を担保する
- IN句は30件制限のため `queryByIdsInChunks` を使用する
- クエリプロファイリングで性能監視を行う

5.4 エラーハンドリング規約

- 致命的エラー: `try-catch` + ユーザーへの通知
- 非致命的エラー: `try-catch` + `console.warn` + フォールバック
- Firebaseエラー: エラーコード別のメッセージマッピング
- バリデーションエラー: リアクティブな`errors`オブジェクトで管理