詳細設計書

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	DD-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. 認証・認可機能

1.1 Firebase初期化（plugins/firebase.client.ts）

初期化処理フロー

```
アプリ起動
  │
  ├── Firebase App 初期化
  ├── Firestore 初期化 (ignoreUndefinedProperties: true)
  ├── Firebase Auth 初期化
  ├── Cloud Storage 初期化
  ├── Cloud Functions 初期化 (asia-northeast1)
  ├── Firebase Analytics 初期化
  └── Vertex AI 初期化 (Gemini 2.0 Flash)
  │
  └── onAuthStateChanged リスナー設定
       │
       ├── ユーザーなし → user = null, userProfile = null
       │
       └── ユーザーあり
            ├── IDトークン取得 → admin claim チェック → isAdmin 設定
            ├── Firestoreからユーザープロフィール取得
            ├── status === 'inactive' → 強制ログアウト
            └── useState にユーザー情報を保存
```

グローバルステート

ステート名	型	初期値	説明
firebaseApp	FirebaseApp	-	Firebaseアプリインスタンス
auth	Auth	-	Firebase Authインスタンス
db	Firestore	-	Firestoreインスタンス
storage	FirebaseStorage	-	Storageインスタンス
functions	Functions	-	Cloud Functionsインスタンス
user	User | null	null	認証済みユーザー
userProfile	ExtendedUserProfile | null	null	ユーザープロフィール
analytics	Analytics	-	Analyticsインスタンス
isAdmin	boolean	false	管理者フラグ

1.2 ルートミドルウェア（middleware/router.global.ts）

```
ページ遷移発生
  │
  ├── useState('auth') からAuth状態取得
  │
  ├── 未認証 AND 遷移先が /signin 以外
  │     └── /signin にリダイレクト
  │
  ├── 認証済み AND 遷移先が /signin
  │     └── / にリダイレクト
  │
  └── その他 → そのまま遷移を許可
```

1.3 認証メソッド詳細（composables/firebase/useAuth.ts）

メソッド	引数	戻り値	処理概要
createUserWithEmailAndPasswordAsync	email, password, verify?	void	アカウント作成（verify=trueで確認メール送信）
loginWithEmailAndPasswordAsync	email, password	void	メール/パスワードログイン
logoutAsync	-	void	ログアウト → /signin へリダイレクト
sendPasswordResetEmailAsync	email	void	パスワードリセットメール送信

メソッド	引数	戻り値	処理概要
updatePasswordAsync	newPassword	void	パスワード変更
updateEmailWithVerification	newEmail	void	メール変更（確認メール付き）
reauthenticateWithCredentialAsync	email, password	void	再認証（セキュリティ操作前に必要）

---

2. カレンダー機能

2.1 useCalendar コンポーザブル

ステート定義

ステート	型	初期値	説明
currentDate	Ref<Date>	new Date()	現在のカレンダー日付
selectedDate	Ref<Date>	new Date()	選択された日付
currentView	Ref<CalendarView>	localStorage or 'weekly'	カレンダービュー種別
users	Ref<ExtendedUserProfile[]>	[]	ユーザーマスター
facilities	Ref<Facility[]>	[]	施設マスター
equipments	Ref<Equipment[]>	[]	設備マスター
holidays	Ref<Holiday[]>	[]	祝日マスター
events	Ref<EventDisplay[]>	[]	表示用イベントリスト
isLoading	Ref<boolean>	false	ローディング状態

マスターデータキャッシュ機構

```
// キャッシュ構造
interface CacheEntry<T> {
  data: T
  timestamp: number  // キャッシュ取得時のタイムスタンプ
}

// TTL設定
const MASTER_DATA_TTL = 10 * 60 * 60 * 1000  // 10時間
const DAILY_OPTIONS_TTL = 60 * 60 * 1000       // 1時間
```

キャッシュ処理フロー:

```
getMasterDataCacheAsync(key, forceRefresh)
  │
  ├── forceRefresh = true → Firestoreから取得 → キャッシュ更新
  │
  ├── キャッシュが存在し、TTL内
  │    └── キャッシュを返却
  │
  ├── 同一キーのリクエストが進行中（in-flight）
  │    └── 既存のPromiseを返却（重複防止）
  │
  └── キャッシュが無い or TTL超過
       ├── Firestoreから取得（成功）→ キャッシュ更新 → データ返却
       └── Firestoreから取得（失敗）→ 期限切れキャッシュがあれば返却（フォールバック）
```

データ読み込み（loadData）

```
loadData(forceRefresh?)
  │
  ├── isLoading = true の場合 → 何もしない（重複防止）
  │
  └── isLoading = true に設定
       │
       ├── [並列] マスターデータ取得
       │    ├── getMasterDataCacheAsync('users')
       │    ├── getMasterDataCacheAsync('holidays')
       │    ├── getMasterDataCacheAsync('facilities')
       │    └── getMasterDataCacheAsync('equipments')
       │
       ├── イベントデータ取得（ビュー別）
       │    ├── weekly → getEventsInRange(startDate, endDate)
       │    └── monthly/daily → getEventsByParticipantInRange(uid, start, end)
       │
       ├── プライベートイベントフィルタリング
       │
       └── isLoading = false
```

カレンダー日付計算

メソッド	処理
generateWeekDays	currentDateを基準に月〜日の7日間を生成
generateCalendarDays	currentDateの月を基準に6週分（42日）を生成
timeSlots	9:00〜18:30の30分刻みスロットを生成
timeToPixels(timeStr)	"HH:mm" → ピクセル値（1時間=60px基準）

位置保持機構

```
saveCalendarPosition()
  └── localStorage.setItem('calendar-position', JSON.stringify({
        date: currentDate,
        view: currentView
      }))

loadCalendarPosition()
  └── localStorage.getItem('calendar-position')
       └── 存在すれば currentDate と currentView を復元
```

2.2 useEventForm コンポーザブル

フォームデータ構造（EventFormData）

フィールド	型	必須	説明
title	string	はい	イベントタイトル
description	string	いいえ	詳細説明
dateType	DateType	はい	日付種別（single/range/recurring）
date	string	条件付き	単日予定の日付（YYYY-MM-DD）
startDate	string	条件付き	範囲開始日
endDate	string	条件付き	範囲終了日
startTime	string	はい	開始時刻（HH:mm）
endTime	string	はい	終了時刻（HH:mm）
eventType	EventType	はい	イベント種別
participantIds	string[]	いいえ	参加者UID配列
facilityIds	string[]	いいえ	施設ID配列
equipmentIds	string[]	いいえ	設備ID配列
recurringPattern	RecurringPattern	条件付き	繰り返しパターン
recurringEndType	RecurringEndType	条件付き	繰り返し終了条件
recurringEndDate	string	条件付き	繰り返し終了日
recurringCount	number	条件付き	繰り返し回数
selectedWeekdays	number[]	条件付き	選択曜日（0-6）

フィールド	型	必須	説明
monthlyType	MonthlyType	条件付き	月次タイプ（date/weekday）
isPrivate	boolean	いいえ	プライベートフラグ

バリデーションルール

フィールド	ルール	エラーメッセージ
title	空文字不可	タイトルは必須です
date	dateType='single' の場合必須	日付は必須です
startDate	dateType='range' の場合必須	開始日は必須です
endDate	dateType='range' の場合必須	終了日は必須です
startTime	endTime以前であること	開始時刻は終了時刻より前にしてください
selectedWeekdays	recurringPattern='weekly' の場合1つ以上	曜日を選択してください

コンフリクト検出フロー

```
checkConflicts()
  │
  ├── getDateRangeForConflictCheck()
  │    ├── single → [date, date]
  │    ├── range → [startDate, endDate]
  │    └── recurring → [startDate, startDate + 3ヶ月]
  │
  ├── generateTimeSlots(formData)
  │    ├── single → [{date, startTime, endTime}]
  │    ├── range → 日ごとのスロット配列
  │    └── recurring → generateRecurringTimeSlots()
  │         └── 3ヶ月分のスロットを生成
  │              ├── daily: 毎日
  │              ├── weekdays: 平日のみ
  │              ├── weekly: 指定曜日
  │              ├── monthly(date): 毎月同日
  │              ├── monthly(weekday): 毎月第N曜日
  │              └── yearly: 毎年同月同日
  │
  └── [各スロット] checkConflictsForTimeSlot()
       ├── 参加者の既存予定と時間重複チェック
       ├── 施設の既存予定と時間重複チェック
       └── 設備の既存予定と時間重複チェック
```

---

3. イベントサービス（services/eventService.ts）

3.1 イベント作成処理

```
createEvent(formData)
  │
  ├── masterRefForId = UUID生成
  │
  ├── dateType === 'single'
  │    └── 1件のイベントドキュメントを作成
  │         └── addWithBatch([{reference, entity}])
  │
  ├── dateType === 'range'
  │    ├── startDate〜endDate の各日に対して
  │    │    └── イベントドキュメントを生成（masterId = masterRefForId）
  │    └── addWithBatch(全ドキュメント)
  │
  └── dateType === 'recurring'
       ├── generateRecurringDates() で展開日リストを取得
       │    ├── 最大展開期間: 1年（デフォルト）
       │    ├── recurringEndType='date' → endDateまで
       │    ├── recurringEndType='count' → N回分
       │    └── recurringEndType='never' → 1年分
       │
       ├── 各展開日に対して
       │    └── イベントドキュメントを生成
       │         ├── masterId = masterRefForId
       │         └── recurrenceRule = パターン情報
       │
       └── addWithBatch(全ドキュメント)
```

3.2 イベント削除処理

```
deleteEvent(eventId, option, targetDate)
  │
  ├── option === 'single'（デフォルト）
  │    └── 指定eventIdの1件を削除
  │
  ├── option === 'all'
  │    ├── 対象イベントのmasterIdを取得
  │    └── 同一masterIdの全イベントを削除
  │
  ├── option === 'after'
  │    ├── 対象イベントのmasterIdを取得
  │    └── targetDate以降の同一masterIdイベントを削除
  │
  └── option === 'before'
       ├── 対象イベントのmasterIdを取得
       └── targetDate以前の同一masterIdイベントを削除
```

3.3 キャッシュからのイベント取得

```
getEventsFromCacheAsync(cacheKey, forceNoCache)
  │
  ├── cacheKey = getCacheKeyForDate(date)
  │    └── "{isoWeekYear}-{isoWeek}" 形式
  │
  ├── Firebase Storage URL構築
  │    └── calendar-cache/{cacheKey}-cache.json
  │
  ├── fetch(url, { cache: 'no-store' })
  │    ├── 成功 → JSONパース → EventDisplay[] を返却
  │    └── 失敗 → 空配列を返却
  │
  └── forceNoCache = true
       └── URL末尾に ?v={timestamp} を付与
```

3.4 繰り返し日付展開ロジック

```
generateRecurringDates(formData, viewStart, viewEnd, interval)
  │
  ├── パターン別日付生成
  │
  ├── 'daily'
  │    └── startDate から毎日（interval日間隔）
  │
  ├── 'weekdays'
  │    └── startDate から平日のみ
  │
  ├── 'weekly'
  │    └── startDate から指定曜日のみ（interval週間隔）
  │
  ├── 'monthly' (date)
  │    └── 毎月同日（例: 毎月15日）
  │
  ├── 'monthly' (weekday)
  │    └── 毎月第N曜日（例: 毎月第3月曜日）
  │
  └── 'yearly'
       └── 毎年同月同日
  │
  ├── 終了条件チェック
  │    ├── 'never' → viewEnd or 1年後まで
  │    ├── 'date' → recurringEndDate まで
  │    └── 'count' → N件まで
  │
  └── ハードリミット: 20年
```

---

4. Wiki機能

4.1 Wikiサービス（services/wikiService.ts）

操作	コレクション	処理
記事作成	wikiArticles	addDocAsync + version=1
記事更新	wikiArticles	updateDocAsync + version++
記事取得	wikiArticles	getDocAsync
記事一覧	wikiArticles	getCollectionAsync
履歴保存	wikiArticles/{id}/histories	addDocAsync(snapshot)

4.2 Markdownレンダリング

```
入力テキスト（Markdown）
  │
  ├── Marked ライブラリでHTML変換
  │
  ├── DOMPurify でサニタイズ（XSS対策）
  │
  └── MarkdownRenderer.vue で描画
```

---

5. 日別オプション機能

5.1 dailyOptionService

メソッド	処理	Firestoreクエリ
getDailyOption(uid, date)	1ユーザー1日分取得	where('uid', '==') AND where('date', '==')
getDailyOptionsInRange(start, end)	全ユーザー期間取得	where('date', '>=') AND where('date', '<=')
getDailyUserOptionsInRange(uid, start, end)	1ユーザー期間取得	where('uid', '==') AND where('date', '>=') AND where('date', '<=')
setDailyOption(data)	Upsert	既存チェック → addDoc or updateDoc
deleteDailyOption(uid, date)	削除	uid+date で検索 → deleteDoc

---

6. ファイルストレージ機能

6.1 useStorage

メソッド	処理
uploadFile(key, file, contentType)	ファイルをCloud Storageにアップロード
uploadFileWithUrlResponseAsync(key, file, contentType)	アップロード後にダウンロードURLを返却
getUrl(path)	ファイルのダウンロードURL取得
deleteFile(path)	ファイル削除

6.2 ストレージパス設計

リソース	パス	例
サンプル	`sample/{id}`	`sample/abc123`
AIファイル	`users/{uid}/ai/{aid}/files/{fid}`	`users/user1/ai/session1/files/file1`
カレンダーキャッシュ	`calendar-cache/{key}-cache.json`	`calendar-cache/2026-09-cache.json`

---

7. リアルタイム同期機能

7.1 useFirestoreSync

```
// コレクション同期
syncCollection<T>(collectionPath): {
  data: Ref<T[]>,      // リアクティブなデータ配列
  pending: Ref<boolean>, // ローディング状態
  error: Ref<Error|null> // エラー状態
}

// ドキュメント同期
syncDocument<T>(documentPath): {
  data: Ref<T|null>,
  pending: Ref<boolean>,
  error: Ref<Error|null>
}
```

内部処理:

- Firestoreの `onSnapshot` を使用してリアルタイムリスナーを設定
- データ変更時に自動的にリアクティブ変数を更新
- エラー発生時はコンソールログに記録

---

8. CSV入出力機能

8.1 useCsvFirestore

機能	処理
CSVエクスポート	Firestoreコレクション → PapaParse → CSVファイルダウンロード
CSVインポート	CSVファイル → PapaParse → Firestoreバッチ書き込み

---

9. ランチシャッフル機能

9.1 useLunchShuffle

ステート	型	説明
participants	string[20]	参加者リスト（固定20要素）
teams	string[][]	チーム分け結果
resultMessage	string	結果メッセージ
lastShuffledAt	string | null	最終シャッフル日時
lastShuffledBy	string | null	最終シャッフル実行者

処理フロー:

1. 参加者リストから空でない要素を抽出
2. Fisher-Yatesアルゴリズム等でランダムシャッフル
3. 指定チーム数で分割
4. 結果をFirestoreに保存