APIClient

September 4, 2026

APIClient

Kind: Interface

Source: `atloria-monorepo/packages/ui-react/src/types/api-client.ts`

API Client interface for making backend requests
This follows the Repository pattern to abstract data access

Properties

Property	Type
`audiences`	`{

```
list(): Promise<AudienceModel[]>;
get(id: string): Promise<AudienceModel>;
create(data: {
  name: string;
  slug: string;
  color: string;
  icon: string;
  description?: string;
}): Promise<AudienceModel>;
update(
  id: string,
  data: {
    name?: string;
    slug?: string;
    color?: string;
    icon?: string;
    description?: string;
  },
): Promise<AudienceModel>;
delete(id: string): Promise<void>;
```

} | | documents | {
get(id: string): Promise;

```
list(params?: { projectId?: string }): Promise;
create(data: Partial): Promise;
update(id: string, data: Partial): Promise;
delete(id: string): Promise;
publish(id: string, isPublic?: boolean): Promise;
unpublish(id: string): Promise;
getRevisions(id: string, limit?: number, offset?: number):
Promise<DocumentRevision[]>;
getRevision(id: string, revisionId: string): Promise;
restoreRevision(id: string, revisionId: string): Promise;
compareDocuments(
id: string,
from: string,
to: string,
type?: 'version' | 'revision',
): Promise;
} || projects | {
get(id: string): Promise;
list(params?: { organizationId?: string }): Promise<Project[]>;
create(data: Partial): Promise;
update(id: string, data: Partial): Promise;
delete(id: string): Promise;
publish(id: string): Promise<{
id: string;
name: string;
slug: string;
urlId: string;
publishedSlug: string;
publicUrl: string;
publishedAt: string;
isPublic: boolean;
}>;
unpublish(id: string): Promise;
} || comments | {
list(documentId: string, params?: any): Promise<CommentWithUser[]>;
get(documentId: string, commentId: string): Promise;
create(documentId: string, dto: CreateCommentDto): Promise;
update(
```

documentId: string,
commentId: string,
dto: UpdateCommentDto,
): Promise;
delete(documentId: string, commentId: string): Promise;
resolve(documentId: string, commentId: string): Promise;
unresolve(documentId: string, commentId: string): Promise;
reply(
documentId: string,
parentCommentId: string,
dto: Omit<CreateCommentDto, 'parentCommentId'>,
): Promise;
getReplies(parentCommentId: string): Promise<CommentWithUser[]>;
} | | suggestions | {
list(documentId: string, params?: any): Promise<SuggestionWithUser[]>;
get(suggestionId: string): Promise;
create(documentId: string, dto: CreateSuggestionDto): Promise;
update(suggestionId: string, dto: UpdateSuggestionDto): Promise;
delete(suggestionId: string): Promise;
accept(suggestionId: string, reviewComment?: string): Promise;
reject(suggestionId: string, reviewComment?: string): Promise;
} | | auth | {
login(email: string, password: string): Promise<{ user: User; token: string }>;
logout(): Promise;
getCurrentUser(): Promise<User | null>;
register(data: {
email: string;
password: string;
name: string;
}): Promise<{ user: User; token: string }>;
} | | parser | {
parse(code: string, language: string, filename?: string): Promise;
parseDocument(documentId: string): Promise;
getSupportedLanguages(): Promise;
} | | templates | {
list(filters?: TemplateFiltersDto): Promise<DocumentTemplate[]>;
get(id: string): Promise;
create(dto: CreateTemplateDto): Promise;

update(id: string, dto: Partial): Promise;
delete(id: string): Promise;
useTemplate(id: string, dto: CreateFromTemplateDto): Promise;
saveAsTemplate(documentId: string, dto: SaveAsTemplateDto): Promise;
} | | branding | {
get(projectId: string): Promise;
update(projectId: string, dto: any): Promise;
} | | snippets | {
list(projectId: string): Promise<any[]>;
get(projectId: string, id: string): Promise;
references(projectId: string, id: string): Promise<any[]>;
create(projectId: string, data: { name: string; title?: string; content?: string }): Promise;
update(
projectId: string,
id: string,
data: { name?: string; title?: string; content?: string },
): Promise;
remove(projectId: string, id: string): Promise<{ deleted: boolean }>;
} | | issues | {
list(projectId: string, params?: Record<string, any>): Promise;
getStats(projectId: string): Promise;
get(projectId: string, issueId: string): Promise;
update(projectId: string, issueId: string, dto: any): Promise;
bulkUpdate(projectId: string, dto: any): Promise;
addComment(projectId: string, issueId: string, dto: any): Promise;
getDuplicates(projectId: string, issueId: string): Promise;
getBoardsForIssue(projectId: string, issueId: string): Promise;
} | | boards | {
list(projectId: string): Promise;
create(projectId: string, dto: any): Promise;
get(projectId: string, boardId: string, includeCards?: boolean): Promise;
update(projectId: string, boardId: string, dto: any): Promise;
deleteBoard(projectId: string, boardId: string): Promise;
restore(projectId: string, boardId: string): Promise;
addColumn(projectId: string, boardId: string, dto: any): Promise;
updateColumn(projectId: string, boardId: string, columnId: string, dto: any): Promise;
reorderColumns(projectId: string, boardId: string, columnIds: string[]): Promise;
deleteColumn(projectId: string, boardId: string, columnId: string, moveCardsTo?:

string): Promise;
listCards(projectId: string, boardId: string, since?: string): Promise;
addCards(projectId: string, boardId: string, dto: { issueIds: string[]; columnId?: string }): Promise;
moveCard(projectId: string, boardId: string, cardId: string, dto: { columnId: string; position: number }): Promise;
removeCard(projectId: string, boardId: string, cardId: string): Promise;
} | | audit | {
query(
projectId: string,
opts?: {
action?: string;
actorId?: string;
entityType?: string;
startDate?: string;
endDate?: string;
limit?: number;
offset?: number;
},
): Promise;
queryOrg(opts?: {
action?: string;
actorId?: string;
projectId?: string;
entityType?: string;
startDate?: string;
endDate?: string;
limit?: number;
offset?: number;
}): Promise;
facets(): Promise<{ actions: string[]; actors: { actorId: string; actorEmail: string }[] }>;
exportCSV(projectId: string): Promise;
exportJSON(projectId: string): Promise;
exportOrgCSV(): Promise;
exportOrgJSON(): Promise;
} | | analytics | {
getOverview(projectId: string, days: number): Promise;
getDailyTraffic(projectId: string, days: number): Promise;

```
getTopDocuments(projectId: string, days: number): Promise;
getTopSearchQueries(projectId: string, days: number): Promise;
getVersionTraffic(projectId: string, days: number): Promise;
getDeviceBreakdown(projectId: string, days: number): Promise;
getSourceBreakdown(projectId: string, days: number): Promise;
getGeoBreakdown(projectId: string, days: number): Promise;
getZeroResultSearches(projectId: string, days: number): Promise;
getGhostDocs(projectId: string, days: number): Promise;
getTrendingDocs(projectId: string, days: number): Promise;
getScrollHistogram(projectId: string, days: number): Promise;
getFeedbackSummary(projectId: string, days: number): Promise;
submitFeedback(data: any): Promise;
exportNow(projectId: string, data: any): Promise;
listScheduledReports(projectId: string): Promise;
createScheduledReport(projectId: string, data: any): Promise;
updateScheduledReport(projectId: string, id: string, data: any): Promise;
deleteScheduledReport(projectId: string, id: string): Promise;
runScheduledReport(projectId: string, id: string): Promise;
listExportHistory(projectId: string): Promise;
}` |
```