TanukiPanel - GitLab Operations Desktop Application

Project Overview

TanukiPanel is a cross-platform desktop application built with **Avalonia** that serves as a GitLab Operations companion. It provides an intuitive interface for managing GitLab projects, viewing commits, managing issues, and accessing package registries - all from a single desktop application.

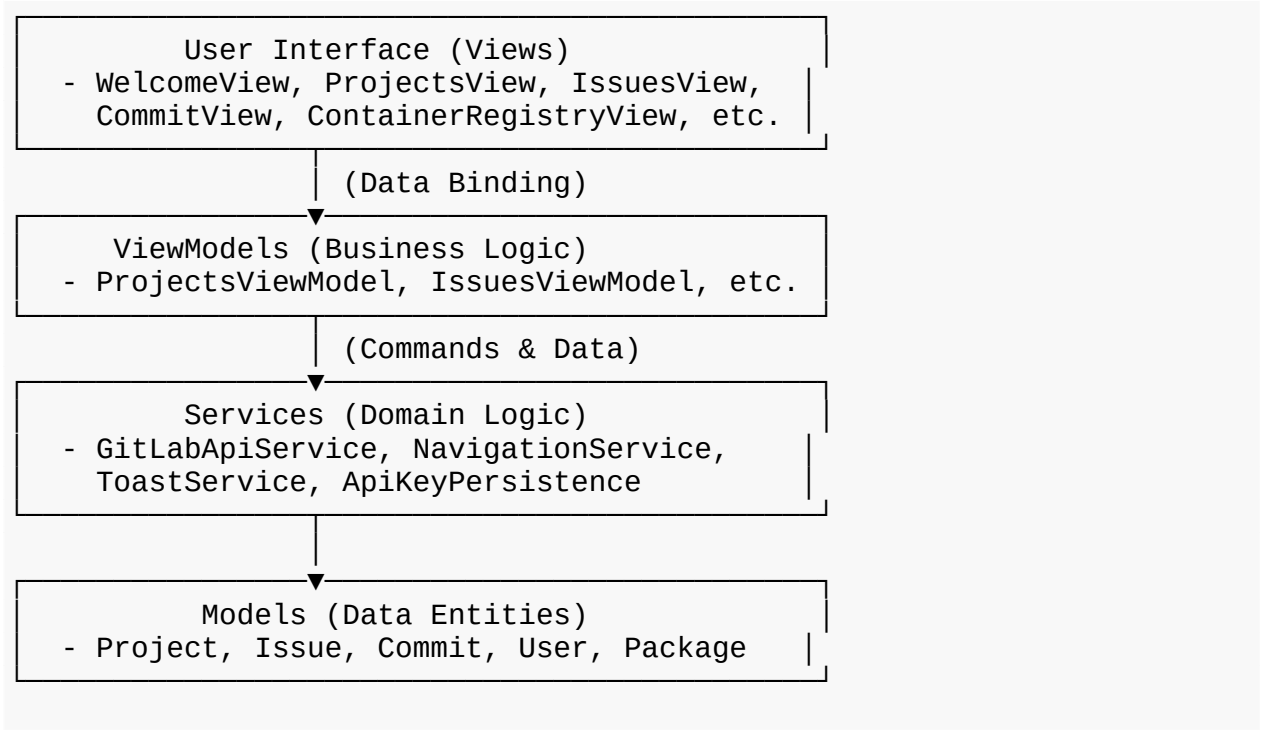
Key Information

- **Framework**: Avalonia 11.3.9 (Cross-platform MVVM UI)
- **Language**: C# (.NET 9.0)
- **Platform**: Windows, Linux, macOS
- **Architecture**: MVVM (Model-View-ViewModel) with Dependency Injection
- **Design Pattern**: Command-based user interactions with Repository pattern for API calls

---

Architecture

High-Level Architecture Diagram



Project Structure

1. Views/ - User Interface Components

Avalonia-based UI components styled with GNOME design system colors.

File	Purpose
`MainWindow.axaml.cs`	Main application window with sidebar navigation
`WelcomeView.cs`	Welcome screen with API key setup guide link
`ApiKeyGuideView.cs`	Interactive carousel guide (3 snapshots) for finding API key
`ApiKeyView.cs`	API key input and configuration screen
`ProjectsView.cs`	Display GitLab projects with search & filter
`CommitView.cs`	View commits with date range filtering & pagination
`IssuesView.cs`	Manage issues (create, read, close/reopen)
`ContainerRegistryView.cs`	View Docker/container registries
`SideBarContentView.cs`	Navigation sidebar with menu buttons
`ToastContainer.cs`	Toast notification UI

2. ViewModels/ - Business Logic Layer

Implements MVVM commands and state management using `CommunityToolkit.Mvvm`.

ViewModel	Responsibilities
`MainWindowViewModel`	Main app state, current user, navigation coordination
`WelcomeViewModel`	Welcome screen logic, API key guide navigation
`ApiKeyViewModel`	API key input validation, storage, authentication
`ApiKeyGuideViewModel`	Guide carousel navigation, back button
`ProjectsViewModel`	Project listing, searching, pagination (15/page)
`CommitViewModel`	Commit history, date filtering, pagination (20/page)
`IssuesViewModel`	Issue CRUD, state management, pagination (15/page)
`ContainerRegistryViewModel`	Registry listing and management
`SideBarContentViewModel`	Navigation menu state
`ViewModelBase`	Base class with INotifyPropertyChanged implementation

3. Services/ - Domain Logic & Dependencies

Handles API communication, persistence, and cross-cutting concerns.

Service	Interface	Purpose
`GitLabApiService`	`IGitLabApiService`	REST client for GitLab API v4
`NavigationService`	`INavigationService`	MVVM navigation between views
`ToastService`	`IToastService`	User notification system (Success/Error/Info/Warning)

`ApiKeyPersistence`	`IApiKeyPersistence`	Persist API key to local JSON file
`FilePickerService`	`IFilePickerService`	File selection dialogs

4. Models/ - Data Entities

Domain models representing GitLab resources.

Model	Fields	Purpose
`User`	Id, Name, Username, AvatarUrl, WebUrl	GitLab user information
`Project`	Id, Name, Description, Owner, Visibility, StarCount	Project metadata
`Commit`	Id, Title, Message, AuthorName, AuthorDate, WebUrl	Repository commit
`Issue`	Id, Title, Description, State, Author, CreatedAt	Project issue
`Package`	Id, Name, Version, Type, PackageType	Registry package
`RegistryRepository`	Id, Name, Location, Status	Container registry
`ApiKeyConfig`	ApiKey, GitLabUrl	API credentials storage

GitLab API Integration

API Communication Layer (`IGitLabApiService`)

The application communicates with GitLab using the **GitLab REST API v4** through the following endpoints:

Authentication

- **Endpoint**: All requests include `PRIVATE-TOKEN` header
- **Base URL**: `https://gitlab.com` (configurable)
- **Authentication**: Private token-based (user's GitLab API key)

API Endpoints Used

Feature	HTTP Method	Endpoint	Purpose
User Info	GET	`/api/v4/user`	Get current authenticated user
Projects	GET	`/api/v4/projects?per_page=15`	List user's projects with pagination
Project Details	GET	`/api/v4/projects/{id}`	Get specific project information
Commits	GET	`/api/v4/projects/{id}/repository/commits`	List repository commits with date filtering

Issues	GET	`/api/v4/projects/{id}/issues?per_page=15`	List project issues with pagination
Create Issue	POST	`/api/v4/projects/{id}/issues`	Create new issue with title & description
Update Issue	PUT	`/api/v4/projects/{id}/issues/{issue_id}`	Change issue state (open/close)
Search Issues	GET	`/api/v4/issues?search={query}`	Global issue search across projects
Container Registries	GET	`/api/v4/projects/{id}/registry/repositories`	List container registries
Packages	GET	`/api/v4/projects/{id}/packages`	List project packages

Request/Response Handling

- **JSON Serialization**: Built-in .NET JSON serialization
- **Error Handling**: HTTP status codes with descriptive error messages
- **Pagination**: Default 15-20 items per page with Next/Previous navigation
- **Rate Limiting**: Respects GitLab API rate limits (600 requests/minute for authenticated users)

Key Features

1. API Key Management

- Secure storage of GitLab API key in local JSON file
- Interactive guide with 3 snapshots showing how to find API key
- Configurable GitLab instance URL support

2. Project Management

- List all projects accessible to user
- Search projects by name
- View project metadata (stars, visibility, description)
- Pagination support (15 projects/page)

3. Commit Viewing

- View repository commits with author information
- Date range filtering (before/after specific dates)
- Pagination with 20 commits per page
- Direct links to GitLab web interface

4. Issue Management

- **Create Issues**: Add new issues with title and description
- **Read Issues**: View project issues with full details
- **Update State**: Close or reopen issues directly from app
- **Search**: Global search across repositories
- **Filter**: View issues by project or search term
- **Pagination**: 15 issues per page

5. Toast Notifications

- Real-time user feedback for actions
- Notification types: Success, Error, Info, Warning
- Auto-dismiss after 3 seconds

6. User Interface

- GNOME-inspired design system colors
- Circular sidebar navigation buttons
- Smooth transitions and animations
- Responsive grid-based layouts
- Scrollable list views with pagination

Dependency Injection Setup

The application uses **Microsoft.Extensions.DependencyInjection** for loose coupling and testability.

Service Registration (App.axaml.cs)

```
services.AddSingleton<IApiKeyPersistence, ApiKeyPersistence>();
services.AddSingleton<IToastService, ToastService>();
services.AddSingleton<MainWindowViewModel>(sp =>
{
    var persistence = sp.GetRequiredService<IApiKeyPersistence>();
    var toastService = sp.GetRequiredService<IToastService>();
    return new MainWindowViewModel(persistence, toastService);
});
services.AddSingleton<INavigationService>(sp =>
{
    var mainVM = sp.GetRequiredService<MainWindowViewModel>();
    return new NavigationService(mainVM);
});
services.AddSingleton<IGitLabApiService>(sp =>
{
    var persistence = sp.GetRequiredService<IApiKeyPersistence>();
    var config = persistence.LoadApiKey();
    // ... initialize with API key
});
```

Dependency Injection Chain

```
App (registers services)
  ↓
MainWindowViewModel (receives IToastService)
  ↓
WelcomeViewModel (receives INavigationService, IApiKeyPersistence,
IToastService)
  ↓
ApiKeyViewModel (same dependencies)
  ↓
SideBarContentViewModel (same dependencies)
  ↓
ProjectsViewModel, IssuesViewModel, CommitViewModel (receive
IGitLabApiService + IToastService)
```

MVVM Command Pattern

All user interactions are handled through `ICommand` implementations from `CommunityToolkit.Mvvm`.

Example: Creating an Issue

ViewModel (IssuesViewModel.cs):

```
[RelayCommand]
public async Task CreateNewIssueAsync()
{
    // Validate inputs
    if (string.IsNullOrWhiteSpace(IssueTitle))
    {
        _toastService?.ShowError("Title is required");
        return;
    }

    // Call API
    var result = await _apiService.CreateIssueAsync(CurrentProject.Id,
IssueTitle, IssueDescription);

    // Show feedback
    _toastService?.ShowSuccess("Issue created successfully");

    // Refresh list
    await LoadProjectIssuesAsync();
}
```

View (IssuesView.cs):

```
createButton.Click += (s, e) =>
{
    if (DataContext is IssuesViewModel vm)
    {
        vm.CreateNewIssueCommand.Execute(null);
    }
};
```

Data Persistence

API Key Storage

- **Location**: `tanuki_api_key.json` in project root
- **Format**: JSON with ApiKey and GitLabUrl fields
- **Access**: `IApiKeyPersistence.LoadApiKey()` and `SaveApiKey()`

```
{
  "ApiKey": "glpat_1234567890abcdef",
  "GitLabUrl": "https://gitlab.com"
}
```

UI Design Principles

Color Scheme (GNOME Inspired)

- **Background**: `#F6F5F4` (Light Gray)
- **Text**: `#2E2E2E` (Dark Gray)
- **Primary**: `#3584E4` (Blue)
- **Border**: `#D0CFCC` (Medium Gray)
- **Error**: Red variants
- **Success**: Green variants

Layout Patterns

Scrollable Lists Pattern

```
Grid (3 rows):
  Row 0 (Auto):    Controls/Filters
  Row 1 (Star):    ListBox (scrollable content)
```

```
  Row 2 (Auto):    Pagination buttons
```

Modal Dialogs

- Issue creation dialog with title/description inputs
- Confirmation dialogs for destructive actions
- Toast notifications for feedback

Development Workflow

Building

```
cd TanukiPanel
dotnet build
```

Running

```
dotnet run
```

Project File (TanukiPanel.csproj)

- **SDK**: Microsoft.NET.Sdk
- **Target Framework**: net9.0
- **Output Type**: WinExe (Windows executable)
- **Nullable**: Enabled
- **Compiled Bindings**: Enabled for performance

Dependencies

- `Avalonia 11.3.9` - UI Framework
- `Avalonia.Desktop 11.3.9` - Desktop integration
- `Avalonia.Themes.Fluent 11.3.9` - Modern theme
- `CommunityToolkit.Mvvm 8.2.1` - MVVM utilities
- `Microsoft.Extensions.DependencyInjection 9.0.0` - DI container

Error Handling & User Feedback

Toast Notifications

- **Success**: Issue created, state changed
- **Error**: API failures, validation errors

- **Info**: General information messages
- **Warning**: Confirmation prompts

HTTP Error Handling

- Network timeouts
- Authentication failures (invalid API key)
- Not Found (404) errors
- Server errors (5xx)

Future Enhancement Opportunities

1. **Merge Requests**: Add MR viewing and filtering
2. **Pipelines**: Display CI/CD pipeline status and logs
3. **Wiki**: View and edit project wikis
4. **Board View**: Kanban-style issue boards
5. **Webhook Integration**: Real-time notifications
6. **Dark Mode**: Additional theme support
7. **Caching**: Cache API responses locally
8. **Offline Support**: Work with cached data offline

Technical Achievements

Cross-platform desktop application

MVVM pattern with proper separation of concerns

Dependency injection for testability

RESTful API integration with proper error handling

Secure credential storage

Toast notification system

Pagination for large datasets

Complex date filtering

Interactive carousel UI component

Navigation history with back button

Conclusion

TanukiPanel demonstrates a professional desktop application architecture following MVVM principles, clean code practices, and GitLab API best practices. It provides a user-friendly alternative to web-based GitLab management with offline-capable features and responsive UI interactions.

The project showcases:

- Object-oriented design principles
- API integration patterns
- Modern C# practices
- Cross-platform UI development
- Proper service layer abstraction
- Command-based user interactions