Project Structure

This document provides a comprehensive overview of the APBeeper Bot project structure and organization.

📁 Directory Overview

```
apbeeper_bot/
├── .github/                 # GitHub workflows and templates
│   └── workflows/
│       └── ci.yml           # Continuous integration
├── docs/                    # Documentation files
│   ├── railway-deployment.md
│   ├── troubleshooting.md
│   └── project-structure.md
├── scripts/                 # Utility and maintenance scripts
│   └── migrations/          # Database migration scripts
├── src/                     # Source code
│   ├── commands/            # Discord slash commands
│   ├── services/            # Core bot services
│   ├── utils/               # Utility functions and helpers
│   ├── database.js          # Database configuration and setup
│   ├── envCheck.js          # Environment validation
│   └── index.js             # Main application entry point
├── data/                    # Local data storage (gitignored)
├── logs/                    # Log files (gitignored)
├── .env.example             # Environment variable template
├── .gitignore               # Git ignore rules
├── CONTRIBUTING.md          # Contribution guidelines
├── LICENSE                  # MIT license
├── README.md                # Main project documentation
├── package.json             # Node.js dependencies and scripts
└── railway.json             # Railway deployment configuration
```

🎯 Core Components

Entry Point (`src/index.js`)

The main application file that:
- Initializes the Discord client
- Loads commands and event handlers
- Sets up scheduled tasks (cron jobs)
- Manages bot lifecycle

Key Features:
- Command loading and registration
- Event handler setup
- Cron job scheduling
- Error handling and logging

Database Layer (`src/database.js`)

Handles all database operations with support for both SQLite (local) and PostgreSQL (production):

Features:
- Automatic database detection (SQLite vs PostgreSQL)
- Connection management and pooling
- Schema creation and migration
- Promise-based query interface

Tables:
- `guilds` - Discord server configurations
- `channels` - Channel-specific settings
- `twitch_streamers` - Twitch integration data
- `population_history` - Historical server data

Commands (`src/commands/`)

Discord slash commands organized by functionality:

`apbpop.js`

- Displays current APB server population
- Shows server status and player counts
- Provides historical data trends

`players.js`

- Detailed player statistics
- Server-specific player information
- Population analytics

`setchannel.js`

- Configure notification channels
- Set up automated announcements
- Channel permission management

`setgame.js`

- Game-specific configuration
- Server selection and preferences
- Game mode settings

`setclangroup.js`

- Clan and group management
- Role assignments
- Group-specific notifications

`twitch.js`

- Twitch streamer management
- Stream notification setup
- Integration configuration

Services (`src/services/`)

Core bot functionality and external integrations:

`apbPopulation.js`

- APB server monitoring
- Population data collection

- Server status tracking
- Data caching and optimization

`discordStatus.js`

- Discord bot status management
- Presence updates
- Activity tracking

`twitchNotify.js`

- Twitch API integration
- Stream status monitoring
- Notification delivery
- Rate limit handling

Utilities (`src/utils/`)

Helper functions and shared utilities:

`embedStyles.js`

- Discord embed templates
- Consistent styling
- Color schemes and formatting
- Rich embed builders

`logger.js`

- Structured logging with Winston
- Log levels and formatting
- File and console output
- Production-ready logging

🔧 Configuration Files

`package.json`

Node.js project configuration:
- Dependencies and versions
- NPM scripts for development and production
- Engine requirements
- Project metadata

`railway.json`

Railway deployment configuration:
- Build and deploy settings
- Health check configuration
- Environment-specific settings
- Restart policies

`.env.example`

Environment variable template:
- Required configuration variables
- Optional settings with defaults

- Documentation for each variable
- Development and production examples

`.gitignore`

Git ignore rules:
- Node.js specific ignores
- Environment files
- Log files and databases
- IDE and OS specific files

📊 Data Flow

Command Processing

```
Discord User → Slash Command → Command Handler → Service Layer → Database → Response
```

Scheduled Tasks

```
Cron Scheduler → Service Function → External API → Database Update → Discord Notifica-
tion
```

Health Monitoring

```
Railway Health Check → Express Server → Bot Status → Health Response
```

🔄 Development Workflow

Local Development

1. **Environment Setup**: Copy `.env.example` to `.env`
2. **Database**: SQLite database in `data/` directory
3. **Development Server**: `npm run dev` with auto-reload
4. **Testing**: Manual testing in development Discord server

Production Deployment

1. **Environment**: Railway with PostgreSQL database
2. **Build Process**: `npm ci` for clean dependency install
3. **Health Checks**: Express server for Railway monitoring
4. **Logging**: Structured logs for debugging

🛡️ Security Considerations

Environment Variables

- Sensitive data stored in environment variables
- No secrets committed to repository
- Separate development and production configurations

Database Security

- Parameterized queries to prevent SQL injection
- Connection encryption in production
- Regular backups and monitoring

API Security

- Rate limiting for external APIs
- Token rotation and validation
- Error handling without exposing sensitive data

📈 Scalability

Database Design

- Indexed queries for performance
- Connection pooling for concurrent access
- Migration scripts for schema updates

Service Architecture

- Modular service design
- Async/await for non-blocking operations
- Caching for frequently accessed data

Monitoring

- Health check endpoints
- Structured logging for debugging
- Performance metrics collection

🔮 Future Enhancements

Planned Features

- Web dashboard for configuration
- Advanced analytics and reporting
- Multi-language support
- Plugin system for extensions

Technical Improvements

- Automated testing framework
- CI/CD pipeline enhancements
- Performance optimization
- Enhanced error handling

This structure provides a solid foundation for a scalable, maintainable Discord bot with proper separation of concerns and production-ready deployment capabilities.