CRIMSONX DISCORD BOT

FULL TECHNICAL SPECIFICATION

0xDC143C

Language: Rust

Framework: Serenity + Poise

Platform: Discord

Version 1.0 — February 2026

Table of Contents

Project Overview

CrimsonBot is a custom Discord bot built from scratch in Rust for the CrimsonX streaming community server ("The Crimson Den"). It serves as the sole bot in the server, replacing all third-party bots like MEE6, Carl-bot, and Streamcord. The bot is developed live on stream as an ongoing content series, doubling as both a community tool and a learn-in-public Rust education project.

Project Goals

- Serve as the single, all-purpose bot for the CrimsonX Discord server
- Replace all third-party bot functionality (moderation, roles, notifications, engagement)
- Provide ongoing dev stream content through the "Building a Discord Bot" series
- Learn Rust through practical application in a real project
- Integrate Twitch, GitHub, and StreamElements with the Discord community

Technical Stack

Component	Choice
Language	Rust (learning in public)
Discord Library	Serenity v0.12+
Command Framework	Poise (built on Serenity)
Async Runtime	Tokio
Database	SQLite via sqlx (start simple, migrate to PostgreSQL later if needed)
HTTP Client	reqwest (for Twitch/GitHub API calls)
Configuration	Environment variables via dotenvy + config structs
Logging	tracing + tracing-subscriber
Hosting	Self-hosted on Linux (same machine as streaming PC initially)

Bot Identity

Field	Value
Bot Name	CrimsonBot
Bot Username	crimsonbot or 0xbot
Avatar	Variant of the CrimsonX logo (crimson emblem on black)

Status	Dynamic — shows current stream status or a rotating tagline
Prefix	/ (slash commands primary), ! (legacy text commands optional)

Discord Gateway Intents & Permissions

Required Gateway Intents

These intents must be enabled in the Discord Developer Portal and requested at connection time.

Intent	Reason
GUILDS	Access to guild/channel/role info, channel create/update/delete events
GUILD_MEMBERS (Privileged)	Member join/leave events for welcome messages and auto-role
GUILD_MESSAGES	Message events for auto-mod, logging, and legacy commands
GUILD_MESSAGE_REACTIONS	Reaction role system
MESSAGE_CONTENT (Privileged)	Read message content for auto-mod and text commands
GUILD_VOICE_STATES	Track voice channel activity for engagement features
GUILD_PRESENCES (Privileged)	Optional — for activity-based roles (e.g. currently playing Minecraft)

Required Bot Permissions

The bot needs the following permissions, calculated into an invite link integer:

Permission	Feature
Manage Roles	Auto-role assignment, reaction roles
Manage Channels	Lock/unlock #live-chat when stream goes live/offline
Kick Members	Moderation
Ban Members	Moderation
Manage Messages	Auto-mod message deletion, purge command
Embed Links	Rich embed messages for notifications and info commands
Attach Files	Sending images in embeds
Read Message History	Message logging, context for moderation
Add Reactions	Reaction role setup
Use External Emojis	Cross-server emoji support
Moderate Members	Timeout functionality

Send Messages	Core bot communication
Use Slash Commands	Primary command interface
View Channels	Access to all server channels

Feature Specification

Features are organized into development phases, designed to align with the dev stream content series. Each phase represents roughly one to two stream sessions of work.

Phase 1: Core Foundation

The essential bot skeleton — connect to Discord, respond to commands, and handle new members. This is the first dev stream episode.

1.1 Bot Startup & Connection

Requirement	Details
Token Management	Load bot token from .env file via dotenvy, never hardcode
Intent Registration	Register all required intents at client build time
Ready Event	Log bot name, guild count, and latency on successful connection
Error Handling	Graceful reconnection on disconnect, exponential backoff
Status	Set bot status to a default message on startup (e.g. "Watching the crimson tide")
Shutdown	Handle SIGINT/SIGTERM for clean shutdown, log disconnect

1.2 Welcome System

Event	Action	Channel
Member Join	Send themed welcome embed with member name, avatar, and member count	#welcome
Member Join	Assign 🩸 Viewer role automatically	N/A (role assignment)
Member Join	Log join event with timestamp and account age	#mod-logs
Member Leave	Log leave event with roles held and join duration	#mod-logs

Welcome Embed Design: Crimson-themed embed with the server logo as thumbnail, member's avatar as author icon, and a randomized welcome message from a configurable list. Include a mention to #rules and #roles channels.

1.3 Information Commands

All commands use Discord slash commands via Poise. Legacy ! prefix commands are optional fallbacks.

Command	Description	Response
/socials	Links to all CrimsonX social platforms	Embed with Twitch, YouTube, X, GitHub links
/schedule	Current streaming schedule	Embed with schedule (pulled from config or database)
/server	Server info and stats	Embed with member count, creation date, boost level
/help	List all available commands	Paginated embed grouped by category
/about	Bot info and credits	Embed with bot version, uptime, GitHub repo link
/ping	Bot latency check	Gateway and API latency in milliseconds

Phase 2: Twitch Stream Integration

Connect the bot to Twitch so the server knows when CrimsonX goes live. This is the most important integration for a streaming community.

2.1 Twitch EventSub Integration

Transport: WebSocket (preferred for single-broadcaster bots) or Webhook via a lightweight HTTP server (using axum or actix-web). WebSocket is simpler since it avoids needing a public HTTPS endpoint.

Required Twitch EventSub Subscriptions:

Event Type	Trigger	Usage
stream.online	Stream goes live	Post go-live notification in #go-live
stream.offline	Stream ends	Update/edit go-live embed, lock #live-chat
channel.update	Title/game/category changes	Update the current stream embed in real-time
channel.follow	New follower	Optional: post in #stream-activity or log
channel.subscribe	New subscriber	Optional: post in #stream-activity
channel.raid	Incoming raid	Optional: post raid alert in #go-live

2.2 Go-Live Notification Flow

When a stream.online event is received:

- Fetch stream details from Twitch Helix API (title, game, thumbnail, viewer count)
- Build a crimson-themed embed with stream title, game name, thumbnail, and a direct link to the Twitch channel
- Ping the 🎬 Live notification role
- Post the embed in #go-live
- Store the message ID so it can be edited later when the stream goes offline
- Unlock #live-chat channel (remove send-message deny for @everyone)
- Update bot status to "LIVE: [stream title]"

When a stream.offline event is received:

- Edit the go-live embed to show "STREAM ENDED" with duration and peak viewer count if available
- Lock #live-chat channel (add send-message deny for @everyone)
- Reset bot status to default

2.3 Twitch Authentication

Requirement	Details
Client ID / Secret	Register an application on the Twitch Developer Console
App Access Token	OAuth2 client credentials flow for EventSub webhook transport
User Access Token	OAuth2 authorization code flow for EventSub WebSocket transport
Token Refresh	Automatic token refresh before expiry using the refresh token
Scopes	moderator:read:followers (for follow events), channel:read:subscriptions (for sub events)
Storage	Store tokens in .env or encrypted config file, never in source code

Phase 3: Role Management

Self-assignable roles let members curate their own experience by opting into game-specific and notification channels.

3.1 Reaction Role System

Approach: Post a persistent embed in #roles with emoji reactions mapped to roles. The bot monitors reaction add/remove events and assigns/removes roles accordingly.

Role Categories and Mappings:

Emoji	Role	Access Granted
⛏️	⛏️ Minecraft	#minecraft channel
💀	💀 Soulsborne	#soulsborne channel
🎲	🎲 RPG	#rpg-corner channel
💻	💻 Dev	#dev-chat, #project-showcase, #code-help, #bot-development
🎬	🎬 Live	Pinged when stream goes live
📢	📢 Announcements	Pinged for major announcements
🎮	🎮 Content	Pinged for YouTube uploads

3.2 Button Role System (Alternative)

Discord buttons provide a cleaner UX than reactions. The bot posts an embed with interactive buttons, each mapped to a role toggle. Clicking a button grants the role if the user doesn't have it, or removes it if they do. Buttons survive bot restarts since they are tied to the message, not the bot's memory.

3.3 Role Management Commands

Command	Permission	Description
/role setup	Admin	Post the role selection embed with reactions/buttons in #roles

/role add	Admin	Add a new self-assignable role mapping
/role remove	Admin	Remove a self-assignable role mapping
/role list	Everyone	Show all available self-assignable roles
/role refresh	Admin	Re-post the role selection embed (after config changes)

Phase 4: Moderation

Even with a small community, having moderation tooling ready from day one prevents scrambling later. All moderation actions are logged to #mod-logs.

4.1 Moderation Commands

Command	Permission	Description
/warn @user [reason]	Mod	Issue a warning (stored in database), DM the user
/timeout @user [duration] [reason]	Mod	Timeout a user for specified duration
/kick @user [reason]	Mod	Kick a user from the server
/ban @user [reason] [delete_days]	Mod	Ban a user, optionally delete recent messages
/unban @user	Mod	Unban a user by ID or username
/warnings @user	Mod	View a user's warning history
/clearwarnings @user	Admin	Clear all warnings for a user
/purge [count] [user]	Mod	Bulk delete messages (max 100), optionally filter by user
/slowmode [seconds]	Mod	Set channel slowmode (0 to disable)
/lock [channel]	Mod	Lock a channel (prevent @everyone from sending messages)
/unlock [channel]	Mod	Unlock a previously locked channel

4.2 Auto-Moderation

Automated content filtering that runs on every message in real-time.

Feature	Details
Banned Words Filter	Configurable word/phrase blocklist stored in database. Deletes message, warns user via DM, logs to #mod-logs.
Spam Detection	Detect rapid-fire messages (e.g. 5+ messages in 3 seconds) from the same user. Auto-timeout for repeat offenders.
Link Filter	Optional allowlist/blocklist for URLs. Block Discord invite links from non-mods by default.
Mention Spam	Auto-delete messages with excessive mentions (threshold: 5+ unique mentions).

Duplicate Message Detection	Detect and remove identical messages posted in quick succession.
New Account Filter	Flag or restrict accounts younger than a configurable age (default: 7 days). Log to #mod-logs.

4.3 Message Logging

All moderation-relevant events are logged to #mod-logs with rich embeds.

Event	Logged Data
Message Edit	Before/after content, author, channel, timestamp
Message Delete	Content (if cached), author, channel, timestamp
Member Join	Username, account creation date, account age
Member Leave	Username, roles held, join date, time in server
Role Changes	User, roles added/removed, who made the change
Mod Actions	Action type, moderator, target user, reason, duration (for timeouts)

Phase 5: GitHub Integration

Since the bot itself is built on stream and hosted on GitHub, integrating GitHub activity into the Discord server closes the loop between dev streams and community engagement.

5.1 GitHub Webhook Receiver

Approach: Run a lightweight HTTP server (axum) alongside the Discord bot to receive GitHub webhook payloads. This can share the same process or run as a separate task within the Tokio runtime.

Supported GitHub Events:

Event	Channel	Notification Content
push	#dev-announcements	Commit summary with message, author, changed file count, and link to diff
pull_request (opened/merged)	#dev-announcements	PR title, description preview, author, and link
issues (opened/closed)	#dev-announcements	Issue title, labels, and link
release (published)	#announcements	Release name, changelog summary, and download link
star (created)	#dev-chat	Optional fun notification: "Someone starred the repo!"

5.2 GitHub Commands

Command	Description	Response
/repo	Link to the bot's GitHub repository	Embed with repo URL, stars, open issues, last commit
/issues	List open issues	Paginated embed of open GitHub issues with labels
/commits [count]	Recent commit history	Embed with last N commits (default 5)
/project	Current dev stream project info	Embed with project description, tech stack, progress

Phase 6: Engagement & Economy

Fun features to encourage community participation and reward active members. These are lower priority but make for great stream content to build.

6.1 Custom Currency System

Feature	Details
Currency Name	Crimson Coins (or Shards, Embers — TBD)
Earning Methods	Chat activity (1 coin/message, 5 min cooldown), stream watch time (if trackable), voice channel presence, event participation
Storage	SQLite database with user_id, balance, and transaction history
Commands	/balance — check your balance; /leaderboard — top 10 earners; /give @user [amount] — transfer coins; /daily — daily login bonus
Admin Commands	/coins add @user [amount]; /coins remove @user [amount]; /coins reset @user

6.2 Leaderboards

Embeds showing top community members by various metrics: currency balance, message count, stream watch time, and voice channel hours. Updated on command or posted weekly by a scheduled task.

6.3 Mini-Games

Simple interactive games playable via slash commands, costing and rewarding Crimson Coins:

- Coin flip — double or nothing
- Number guessing — guess a number 1-100 in limited attempts
- Rock-paper-scissors — challenge another member
- Trivia — gaming/coding trivia questions with timed responses

6.4 Leveling System (Optional)

XP-based leveling from chat and voice activity. Level-up announcements in #general with a themed embed. Milestone roles at levels 5, 10, 25, and 50. Can be implemented as an alternative to or alongside the currency system.

Phase 7: Advanced Integrations

Stretch goals that add significant value but require more complex implementation.

7.1 Twitch Chat Bridge

Bridge messages between the Discord #live-chat channel and Twitch chat during live streams. Messages from Discord are relayed to Twitch prefixed with [Discord], and Twitch messages appear in Discord as webhook embeds. Requires a Twitch IRC/TMI connection running alongside the EventSub WebSocket.

7.2 Clip Submission System

Members submit clips via /clip [twitch-clip-url]. Clips are posted in #clips with a voting embed (thumbs up/down reactions). A weekly "Clip of the Week" is selected based on votes and announced in #announcements. Potential YouTube content pipeline.

7.3 StreamElements Integration

Receive StreamElements alert events (followers, subs, cheers, tips) via their WebSocket API or webhook forwarding. Post formatted notifications in a #stream-activity channel. This complements the existing custom StreamElements overlay widgets already built for OBS.

7.4 Custom Embeds Command

Command	Permission	Description
/embed create	Mod	Interactive embed builder with title, description, color, fields, image, footer
/embed edit [message_id]	Mod	Edit an existing bot-posted embed
/embed send [channel]	Mod	Send the built embed to a specific channel

Database Schema

SQLite for initial development. All tables use Discord snowflake IDs (u64) as primary keys where applicable. Timestamps use UTC ISO 8601 format.

Core Tables

Table	Purpose	Key Columns
guild_config	Per-server configuration	guild_id (PK), prefix, welcome_channel_id, log_channel_id, live_channel_id, live_role_id, mod_role_id
members	Member data and stats	guild_id, user_id (composite PK), join_date, message_count, xp, level, currency_balance
warnings	Moderation warnings	id (PK), guild_id, user_id, moderator_id, reason, created_at
mod_actions	Full moderation audit log	id (PK), guild_id, user_id, moderator_id, action_type, reason, duration, created_at
reaction_roles	Reaction-to-role mappings	id (PK), guild_id, message_id, emoji, role_id
auto_mod_config	Auto-moderation settings	guild_id (PK), banned_words (JSON), max_mentions, spam_threshold, link_allowlist (JSON)
stream_sessions	Stream history for stats	id (PK), started_at, ended_at, title, game, peak_viewers, notification_message_id
currency_transactions	Economy audit trail	id (PK), user_id, amount, transaction_type, description, created_at
clips	Submitted clips and votes	id (PK), guild_id, user_id, clip_url, message_id, upvotes, downvotes, submitted_at

Project Structure

Rust workspace organized by feature module. Each module is self-contained with its own commands, event handlers, and database queries.

```
crimsonbot/
├── Cargo.toml
├── .env
├── migrations/                 ← SQL migration files
├── src/
│   ├── main.rs                 ← Entry point, client setup
│   ├── config.rs               ← Configuration structs, env loading
│   ├── db.rs                   ← Database pool setup, migrations
│   ├── error.rs                ← Custom error types
│   ├── commands/               ← Slash command handlers
│   │   ├── mod.rs
│   │   ├── general.rs          ← /ping, /help, /about, /socials
│   │   ├── moderation.rs       ← /warn, /kick, /ban, /purge, etc.
│   │   ├── roles.rs            ← /role setup, add, remove
│   │   ├── economy.rs          ← /balance, /daily, /leaderboard
│   │   ├── github.rs           ← /repo, /issues, /commits
│   │   └── stream.rs           ← /schedule, /uptime
│   ├── events/                 ← Discord event handlers
│   │   ├── mod.rs
│   │   ├── member.rs           ← Join/leave handlers
│   │   ├── message.rs          ← Auto-mod, logging, XP tracking
│   │   └── reaction.rs         ← Reaction role handler
│   ├── integrations/           ← External API clients
│   │   ├── mod.rs
│   │   ├── twitch.rs           ← Twitch EventSub + Helix API
│   │   ├── github.rs           ← GitHub webhook receiver + API
│   │   └── streamelements.rs   ← StreamElements WebSocket client
│   └── utils/                  ← Shared utilities
│       ├── mod.rs
│       ├── embeds.rs           ← Themed embed builder helpers
│       └── permissions.rs      ← Permission check helpers
└── config/
```

```
└── default.toml              ← Default configuration values
```

Cargo Dependencies

Core dependencies for Cargo.toml:

Crate	Version	Purpose
serenity	0.12+	Discord API library
poise	0.6+	Slash command framework built on Serenity
tokio	1 (full features)	Async runtime
sqlx	0.8+ (sqlite, runtime-tokio)	Async database driver with compile-time query checking
reqwest	0.12+ (json feature)	HTTP client for Twitch and GitHub APIs
serde	1 (derive feature)	Serialization/deserialization
serde_json	1	JSON parsing
dotenvy	0.15+	Environment variable loading from .env
tracing	0.1	Structured logging
tracing-subscriber	0.3	Log output formatting
chrono	0.4	Date/time handling
axum	0.7+	HTTP server for GitHub/Twitch webhooks
hmac + sha2	latest	HMAC-SHA256 for webhook signature verification
rand	0.8+	Random number generation for games and welcome messages

Environment Variables

All sensitive configuration is stored in a .env file at the project root. This file must never be committed to Git.

Variable	Required	Description
DISCORD_TOKEN	Yes	Discord bot token from Developer Portal
DATABASE_URL	Yes	SQLite path (e.g. sqlite:crimsonbot.db)
TWITCH_CLIENT_ID	Phase 2	Twitch application client ID
TWITCH_CLIENT_SECRET	Phase 2	Twitch application client secret
TWITCH_CHANNEL_ID	Phase 2	Your Twitch broadcaster user ID
GITHUB_WEBHOOK_SECRET	Phase 5	Secret for verifying GitHub webhook payloads
GITHUB_TOKEN	Phase 5	GitHub personal access token for API calls (optional, increases rate limits)
WEBHOOK_PORT	Phase 2	Port for the HTTP webhook server (default: 8080)
RUST_LOG	No	Log level filter (e.g. crimsonbot=debug,serenity=info)

Embed Theme System

All bot embeds follow a consistent CrimsonX visual identity. A shared embed builder utility enforces this across all features.

Standard Embed Template

Property	Value
Color	#DC143C (crimson red) for standard embeds
Footer	"CrimsonX • 0xDC143C" with bot avatar as footer icon
Timestamp	Always include, set to current UTC time
Author	Context-dependent (e.g. moderator name for mod actions, user name for user commands)
Thumbnail	Server logo for server-related embeds, user avatar for user-related embeds

Color Variants

Context	Color
Standard / Info	#DC143C (crimson)
Success	#00FF7F (green)
Warning	#FFD700 (gold)
Error	#FF4444 (red)
Moderation	#8B0A1E (dark crimson)
Twitch / Live	#9146FF (Twitch purple)
GitHub	#238636 (GitHub green)
Economy	#00CED1 (teal)

Dev Stream Content Roadmap

Each phase maps to one or two dev stream episodes. This gives the content series a natural arc from simple bot setup to complex integrations.

Episode	Title	Features Built
1	"Building My First Discord Bot in Rust"	Project setup, Cargo init, bot connects, /ping, /help
2	"Welcome to the Crimson Den"	Welcome system, auto-role, /socials, /server
3	"Going Live Notifications"	Twitch EventSub, go-live embeds, #live-chat lock/unlock
4	"Let Chat Pick Their Roles"	Reaction roles or button roles, /role commands
5	"Auto-Mod From Scratch"	Banned words, spam detection, message logging
6	"Moderation Commands"	Warn, kick, ban, purge, timeout commands
7	"GitHub Meets Discord"	GitHub webhook receiver, commit/PR notifications
8	"Custom Currency & Economy"	Crimson Coins, /balance, /daily, /leaderboard
9	"Mini-Games & Fun Stuff"	Coinflip, trivia, rock-paper-scissors
10	"Bridging Twitch & Discord"	Twitch-Discord chat bridge
Ongoing	"Bug Fixes & Feature Requests"	Community-requested features, refactoring

Deployment & Operations

Initial Deployment (Self-Hosted)

The bot runs on the same Linux machine used for streaming. For a single-server bot with no uptime SLA requirements, this is the simplest approach.

- Build a release binary: cargo build --release
- Run as a systemd service for automatic restart on crash
- SQLite database file stored alongside the binary
- Logs written to journald via systemd

Future Hosting Options

Option	When to Consider
VPS (DigitalOcean, Hetzner)	When 24/7 uptime matters more than cost ($5-10/month)
Shuttle.rs	Free Rust-native hosting, good for Serenity bots, easy deployment
Docker container	When you want reproducible builds and easy migration between hosts
Raspberry Pi	Low-cost always-on option if you have one available

Monitoring & Health

- Bot status in Discord (online/offline indicator)
- Heartbeat logging (log a health check every 5 minutes)
- /status command showing uptime, memory usage, database size, event counts
- Error alerting: post critical errors to #mod-logs or a dedicated #bot-logs channel

Backup Strategy

- SQLite database: automated daily backup via cron job or systemd timer
- Configuration: stored in Git (minus secrets)
- Bot code: GitHub repository (built on stream)

Security Considerations

- Never commit tokens or secrets to Git — use .env and add it to .gitignore
- Verify all incoming webhook payloads using HMAC-SHA256 signatures (both GitHub and Twitch)
- Rate-limit commands per user to prevent abuse (Poise has built-in cooldown support)
- Validate all user input before database queries — use parameterized queries via sqlx (never string concatenation)
- Principle of least privilege: request only the Discord permissions and Twitch scopes actually needed
- Log all moderation actions with moderator ID for accountability
- Sanitize embed content to prevent mention injection (@everyone, @here) in user-supplied text
- Set appropriate Discord permission checks on all slash commands via Poise's required_permissions

Appendix: Discord Server Channel Map

Quick reference showing which bot features interact with each channel in The Crimson Den.

Channel	Bot Feature	Bot Access
#welcome	Welcome embeds on member join	Send Messages, Embed Links
#rules	No bot interaction (static content)	None
#roles	Reaction/button role embed	Send Messages, Add Reactions, Manage Roles
#announcements	GitHub release notifications	Send Messages, Embed Links
#go-live	Twitch go-live/offline notifications	Send Messages, Embed Links, Mention Roles
#live-chat	Lock/unlock on stream start/end	Manage Channels
#stream-vods	Optional auto-post VOD links	Send Messages
#clips	Clip submission and voting system	Send Messages, Add Reactions, Embed Links
#general	Level-up announcements, mini-games	Send Messages, Embed Links
#bot-commands	All slash commands accepted here	Send Messages, Embed Links
#dev-announcements	GitHub push/PR/issue notifications	Send Messages, Embed Links
#mod-logs	All moderation and audit logging	Send Messages, Embed Links
#bot-testing	Command testing (staff only)	Send Messages, Embed Links