TaskQuest - Technical Specification

Gamified RPG System for Taskwarrior

Version: 1.0
Language: **Rust** (chosen for speed, memory safety, cross-platform support)
Target Platforms: Linux (Fedora Bazzite), Android (Termux)

Executive Summary

TaskQuest transforms Taskwarrior into an epic RPG adventure where completing tasks earns XP, gold, loot, and achievements. The system uses Taskwarrior's hook system for **fully automatic** integration - no manual commands needed beyond normal `task` operations.

Key Design Principles (Based on Research)

1. **No Punishment System** - Only rewards for completion (avoiding Habitica's main pitfall)
2. **Grace Period** - Late tasks still earn reduced XP (no harsh penalties during busy periods)
3. **Organic Progression** - Stats tied to actual task completion, not arbitrary choices
4. **Non-Toxic Achievements** - No streak requirements or completion quotas per day
5. **Sustainable Motivation** - XP curve challenging but not demotivating

---

System Architecture

```
TaskQuest/
├── src/
│   ├── main.rs              # CLI/TUI entry point
│   ├── hooks/
│   │   ├── on_add.rs        # Hook: task creation
│   │   ├── on_modify.rs     # Hook: task completion/modification
│   │   └── on_exit.rs       # Hook: post-command processing
│   ├── character/
│   │   ├── mod.rs           # Character system
│   │   ├── stats.rs         # D&D 5e stats (STR, DEX, CON, INT, WIS, CHA)
│   │   ├── class.rs         # Classes (Rogue, Ranger, Warrior, Paladin, Monk)
│   │   ├── level.rs         # XP curve & leveling
│   │   └── avatar.rs        # ASCII art avatars
│   ├── progression/
│   │   ├── xp.rs            # XP calculation engine
│   │   ├── gold.rs          # Gold economy
│   │   └── loot.rs          # Loot table & drops
│   ├── achievements/
│   │   ├── mod.rs           # Achievement system
│   │   ├── definitions.rs   # Built-in achievements
│   │   └── tracker.rs       # Progress tracking
│   ├── shop/
│   │   ├── mod.rs           # Reward store
│   │   └── rewards.rs       # Reward definitions
│   ├── display/
│   │   ├── cli.rs           # Command-line interface
│   │   ├── tui.rs           # Terminal UI (using ratatui)
│   │   └── formatter.rs     # Output formatting
│   ├── taskwarrior/
│   │   ├── integration.rs   # TW interface
│   │   ├── uda.rs           # UDA management
│   │   └── parser.rs        # JSON parsing
│   └── storage/
│       ├── mod.rs           # Data persistence
│       └── git_sync.rs      # Git-friendly storage
├── data/                    # Git-tracked (mirrors ~/.task/)
│   ├── character.json       # Player stats, class, XP, level
│   ├── achievements.json    # Achievement progress
│   ├── shop.json            # Custom rewards
│   ├── loot_history.json    # Drop history (for RNG seed)
│   └── settings.json        # User preferences
└── hooks/                   # Symlinked to ~/.task/hooks/
    ├── on-add-taskquest
    ├── on-modify-taskquest
    └── on-exit-taskquest
```

Data Storage Philosophy

Following Taskwarrior's methodology:

- **Location**: `~/.taskquest/` (configurable via `TASKQUEST_DATA`)
- **Format**: JSON files (one object per line for some files)
- **Git-friendly**: Designed for version control
- **Atomic writes**: Prevent corruption during crashes
- **File locking**: Prevent concurrent modification
- **Backward compatible**: Preserve unknown fields

Taskwarrior Integration

User Defined Attributes (UDAs)

Configure in `~/.taskrc`:

ini

```ini
# TaskQuest UDAs
uda.challenge.type=numeric
uda.challenge.label=Challenge
uda.challenge.values=1,2,3,4,5,6,7,8,9,10

# Character setup (set once)
uda.tq_name.type=string
uda.tq_name.label=Hero Name

uda.tq_class.type=string
uda.tq_class.label=Class
uda.tq_class.values=Rogue,Ranger,Warrior,Paladin,Monk

# Primary/Secondary stats
uda.tq_statpri.type=string
uda.tq_statpri.label=Primary Stat
uda.tq_statpri.values=STR,DEX,CON,INT,WIS,CHA

uda.tq_statsec.type=string
uda.tq_statsec.label=Secondary Stat
uda.tq_statsec.values=STR,DEX,CON,INT,WIS,CHA

# Optional: Color coding for challenge levels
color.uda.challenge.1=color246   # Trivial - gray
color.uda.challenge.2=color246
color.uda.challenge.3=color250   # Easy - lighter gray
color.uda.challenge.4=color250
color.uda.challenge.5=color255   # Medium - white
color.uda.challenge.6=color255
color.uda.challenge.7=color226   # Hard - yellow
color.uda.challenge.8=color208   # Deadly - orange
color.uda.challenge.9=color196   # Legendary - red
color.uda.challenge.10=color201  # Epic - magenta
```

Hook System (Fully Automatic)

Hook: on-add-taskquest

- Triggers: When task is created
- Actions:
  - Validate UDA values
  - Set defaults if missing (challenge:5 if not specified)
  - Log task creation

Hook: on-modify-taskquest

- Triggers: When task is modified
- Actions:
  - Detect completion (status: pending → completed)
  - Calculate XP and gold rewards
  - Award loot (probabilistic)
  - Update character stats
  - Check for new achievements
  - Detect late completion (grace period handling)

Hook: on-exit-taskquest

- Triggers: After any task command
- Actions:
  - Save character state
  - Commit to git (if enabled)
  - Display level-up notification (if applicable)

Character System

D&D 5e Stats

```rust
pub struct Stats {
    pub strength: u16,     // Physical power
    pub dexterity: u16,    // Agility, reflexes
    pub constitution: u16, // Endurance, health
    pub intelligence: u16, // Logic, memory
    pub wisdom: u16,       // Perception, insight
    pub charisma: u16,     // Force of personality
}

impl Stats {
    pub fn new() -> Self {
        Self {
            strength: 10,
            dexterity: 10,
            constitution: 10,
            intelligence: 10,
            wisdom: 10,
            charisma: 10,
        }
    }
}
```

Stat Improvement System:

- **Primary Stat**: +2 per task completion (66% contribution)
- **Secondary Stat**: +1 per task completion (33% contribution)
- **Additional**: +1 to random stat on level-up
- **Scaling**: Difficulty multiplier applies

Example: Challenge 8 task with statpri:INT, statsec:WIS

- INT: +16 points
- WIS: +8 points
- Random stat on level-up: +1

Classes

```rust
pub enum Class {
    Rogue,   // Cunning, agile
    Ranger,  // Skilled, versatile
    Warrior, // Strong, durable
    Paladin, // Righteous, balanced
    Monk,    // Disciplined, wise
}
```

Class Characteristics:

- **Visual Only**: Different ASCII avatars per class
- **Suggested Stat Combos** (not enforced):
  - Rogue: DEX (pri), CHA (sec)
  - Ranger: DEX (pri), WIS (sec)
  - Warrior: STR (pri), CON (sec)
  - Paladin: STR (pri), CHA (sec)
  - Monk: WIS (pri), DEX (sec)

Progression System

XP Calculation Engine

```rust
pub struct XPCalculator;

impl XPCalculator {
    pub fn calculate(
        challenge: u8,        // 1-10
        urgency: f64,         // Taskwarrior's urgency score
        timing: TaskTiming,   // On time, early, late
    ) -> u32 {
        // Base XP scales with challenge
        let base_xp = challenge as u32 * 10;

        // Urgency modifier (1.0 to 1.5x)
        let urgency_multiplier = 1.0 + (urgency * 0.5).min(0.5);

        // Timing bonus/penalty
        let timing_multiplier = match timing {
            TaskTiming::Early => 1.3,       // >24hrs before due
            TaskTiming::OnTime => 1.0,      // Day of due date
            TaskTiming::GracePeriod => 0.8, // <24hrs late
            TaskTiming::Late => 0.5,        // >24hrs late
            TaskTiming::NoDueDate => 1.0,   // No penalty
        };

        ((base_xp as f64) * urgency_multiplier * timing_multiplier) as u32
    }
}

pub enum TaskTiming {
    Early,
    OnTime,
    GracePeriod,
    Late,
    NoDueDate,
}
```

XP Examples:

- Challenge 5, medium urgency, on time: ~75 XP
- Challenge 10, high urgency, early: ~195 XP
- Challenge 3, low urgency, late: ~15 XP

Level Curve (WoW-inspired)

```rust
pub struct LevelSystem;

impl LevelSystem {
    // XP required for level N: 100 * (N ^ 2.1)
    pub fn xp_for_level(level: u32) -> u32 {
        if level == 1 {
            return 0;
        }
        (100.0 * (level as f64).powf(2.1)) as u32
    }

    pub fn level_from_xp(total_xp: u32) -> u32 {
        let mut level = 1;
        while total_xp >= Self::xp_for_level(level + 1) {
            level += 1;
        }
        level
    }
}
```

Level Progression Table:

Level	Total XP Required	XP for Next Level
1	0	121
2	121	362
3	483	726
5	2,023	1,751
10	15,849	6,135
20	105,737	21,438
30	296,059	43,714
50	1,284,025	105,128

Balancing Notes:

- Challenge 5 task (avg): ~75 XP
- Level 1→2: ~2 medium tasks
- Level 10→11: ~82 medium tasks
- Level 30→31: ~583 medium tasks
- Progressive but not overwhelming

Gold Economy

Gold Calculation

```rust
pub struct GoldCalculator;

impl GoldCalculator {
    pub fn calculate(challenge: u8) -> u32 {
        // Base: challenge * 5
        let base = challenge as u32 * 5;

        // Random variance: ±20%
        let variance = (base as f64 * 0.2) as u32;
        let min = base.saturating_sub(variance);
        let max = base + variance;

        rand::thread_rng().gen_range(min..=max)
    }
}
```

Gold Examples:

- Challenge 1: 4-6 gold
- Challenge 5: 20-30 gold
- Challenge 10: 40-60 gold

Shop System

Pre-populated Rewards:

json


```
{
 "rewards": [
  {
   "id": 1,
   "name": "Coffee Break",
   "description": "Enjoy a 15-minute coffee break",
   "cost": 50,
   "tier": "normal",
   "cooldown_hours": 0
  },
  {
   "id": 2,
   "name": "Gaming Session",
   "description": "30 minutes of guilt-free gaming",
   "cost": 100,
   "tier": "normal",
   "cooldown_hours": 0
  },
  {
   "id": 3,
   "name": "Movie Night",
   "description": "Watch a movie of your choice",
   "cost": 150,
   "tier": "heroic",
   "cooldown_hours": 0
  },
  {
   "id": 4,
   "name": "Treat Meal",
   "description": "Order your favorite takeout",
   "cost": 200,
   "tier": "heroic",
   "cooldown_hours": 0
  },
  {
   "id": 5,
   "name": "New Book",
   "description": "Buy that book you've been eyeing",
   "cost": 300,
   "tier": "heroic",
   "cooldown_hours": 0
  },
  {
   "id": 6,
   "name": "Day Off",
   "description": "Take a guilt-free rest day",
   "cost": 500,
```

```
    "tier": "epic",
    "cooldown_hours": 168
  },
  {
    "id": 7,
    "name": "Hobby Supplies",
    "description": "Buy supplies for your hobby",
    "cost": 400,
    "tier": "epic",
    "cooldown_hours": 0
  },
  {
    "id": 8,
    "name": "Weekend Adventure",
    "description": "Plan a day trip or adventure",
    "cost": 750,
    "tier": "epic",
    "cooldown_hours": 336
  },
  {
    "id": 9,
    "name": "Major Purchase",
    "description": "Buy that expensive item you want",
    "cost": 1500,
    "tier": "legendary",
    "cooldown_hours": 0
  },
  {
    "id": 10,
    "name": "Epic Reward",
    "description": "Your ultimate reward - define it yourself!",
    "cost": 3000,
    "tier": "legendary",
    "cooldown_hours": 720
  }
 ]
}
```


Custom Rewards: Users can add their own via CLI:

```bash
taskquest shop add "Massage" 250 "60-minute massage session"
```

Loot System

Loot Table (Weighted Random)

rust

```rust
pub struct LootSystem;

impl LootSystem {
    pub fn roll_for_loot(challenge: u8) -> Option<LootDrop> {
        let mut rng = rand::thread_rng();

        // Base drop chance: 30% + (challenge * 2%)
        let drop_chance = 0.30 + (challenge as f64 * 0.02);

        if rng.gen_bool(drop_chance) {
            return Some(Self::determine_loot_type(&mut rng));
        }
        None
    }

    fn determine_loot_type(rng: &mut impl Rng) -> LootDrop {
        let roll: f64 = rng.gen();

        match roll {
            r if r < 0.70 => {
                // 70% - Gold drop
                let amount = rng.gen_range(10..=50);
                LootDrop::Gold(amount)
            },
            r if r < 0.90 => {
                // 20% - Normal tier reward
                LootDrop::Reward {
                    tier: RewardTier::Normal,
                    from_shop: Self::random_reward_by_tier(RewardTier::Normal),
                }
            },
            r if r < 0.98 => {
                // 8% - Heroic tier reward
                LootDrop::Reward {
                    tier: RewardTier::Heroic,
                    from_shop: Self::random_reward_by_tier(RewardTier::Heroic),
                }
            },
            _ => {
                // 2% - Epic tier reward
                LootDrop::Reward {
                    tier: RewardTier::Epic,
                    from_shop: Self::random_reward_by_tier(RewardTier::Epic),
                }
            }
        }
    }
}
```

```
}

pub enum LootDrop {
    Gold(u32),
    Reward {
        tier: RewardTier,
        from_shop: Reward,
    },
}

pub enum RewardTier {
    Normal,   // Color: white
    Heroic,   // Color: blue
    Epic,     // Color: purple
    Legendary,// Color: orange (not droppable)
}
```

Loot Drop Colors (Terminal):

- **Gold**: Yellow/gold
- **Normal**: White
- **Heroic**: Bright blue
- **Epic**: Magenta/purple

Achievement System

Non-Toxic Achievement Philosophy

Achievements Focus On:

- ✅ Milestones (first task, 100 tasks, etc.)
- ✅ Variety (using different difficulty levels)
- ✅ Long-term progress (doesn't require consecutive days)
- ✅ Comeback moments (returning after breaks)
- ✅ Exploration (trying different features)

Achievements Avoid:

- ❌ Daily streaks (creates anxiety on rest days)
- ❌ Excessive quotas ("100 tasks in a day")
- ❌ Competitive metrics
- ❌ Punishment for taking breaks

Achievement Definitions

rust

```rust
pub struct Achievement {
    pub id: String,
    pub title: String,
    pub description: String,
    pub tier: AchievementTier,
    pub icon: &'static str,
    pub check: fn(&CharacterData, &[TaskHistory]) -> bool,
}

pub enum AchievementTier {
    Common,
    Uncommon,
    Rare,
    Epic,
    Legendary,
}
```

Built-in Achievements:

Title	Description	Tier	Unlocks
First Steps	Complete your first quest	Common	After first task
The Journey Begins	Reach level 5	Common	Level 5
Seasoned Adventurer	Complete 100 quests	Uncommon	100 tasks
Veteran Hero	Complete 500 quests	Rare	500 tasks
Master of Balance	Complete at least one task of each difficulty (1-10)	Uncommon	All difficulties used
The Undaunted	Complete 10 difficulty-10 quests	Rare	10 max-difficulty tasks
Legendary Warrior	Reach level 30	Epic	Level 30
Consistent Hero	Be active for 30 different days (non-consecutive)	Uncommon	30 active days
Renaissance Soul	Complete tasks in 10 different projects	Uncommon	10 unique projects
Phoenix Rising	Complete 5 quests after a 30+ day break	Rare	Return after break
Epic Collector	Receive an Epic-tier loot drop	Rare	Get epic loot
Gold Hoarder	Accumulate 5000 gold	Rare	5k gold total
Wise Spender	Purchase 10 rewards from the shop	Uncommon	10 purchases
Completionist	Complete 1000 quests	Epic	1000 tasks
Living Legend	Reach level 50	Legendary	Level 50
Jack of All Trades	Complete at least 20 tasks of each difficulty	Epic	Variety mastery
Marathon Hero	Be active for 100 different days	Epic	100 active days
Early Riser	Complete 50 tasks early (>24hrs before due)	Uncommon	50 early completions
Pressure Handler	Complete 25 tasks within grace period	Uncommon	25 late-but-grace
Time Master	Complete 100 tasks with a due date set	Uncommon	100 scheduled tasks

Title System

Titles are earned from achievements:

- Achievement unlocked = Title available
- User selects active title to display
- Format: `[Name], [Title] [Class], Level [N]`
- Example: `Aria, The Undaunted Rogue, Level 23`

ASCII Avatar System

Avatar Structure

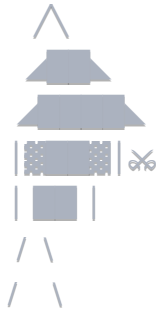
Dimensions: 10-20 lines tall, varying width **Complexity:** 50+ characters of detail per avatar **Evolution:** 5 stages (1-10, 11-20, 21-30, 31-40, 41-50)

Class-Specific Avatars

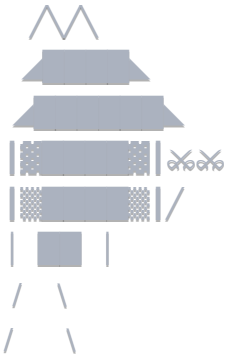
Rogue (Agile, Cunning)

Stage 1 (Levels 1-10):

Stage 3 (Levels 21-30):

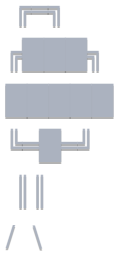


Stage 5 (Levels 41-50):

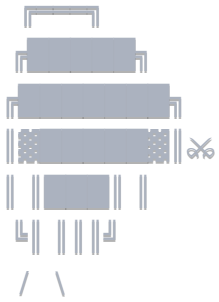


Warrior (Strong, Armored)

Stage 1 (Levels 1-10):



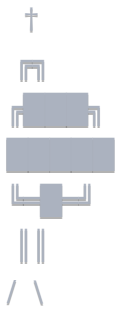
Stage 3 (Levels 21-30):



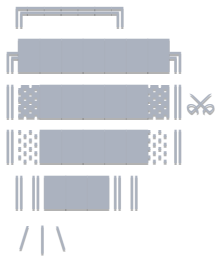
Stage 5 (Levels 41-50):

Paladin (Righteous, Holy)

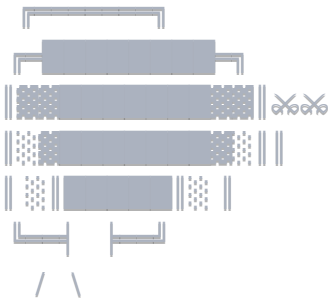
Stage 1 (Levels 1-10):



Stage 3 (Levels 21-30):

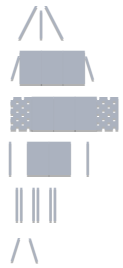


Stage 5 (Levels 41-50):

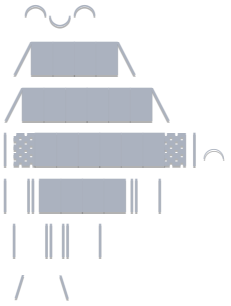


Ranger (Skilled, Swift)

Stage 1 (Levels 1-10):



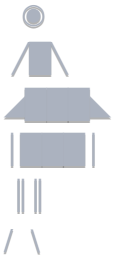
Stage 3 (Levels 21-30):



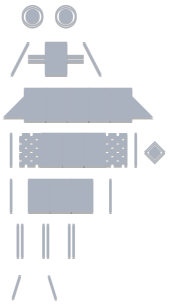
Stage 5 (Levels 41-50):

Monk (Disciplined, Balanced)

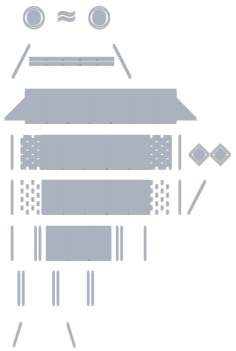
Stage 1 (Levels 1-10):



Stage 3 (Levels 21-30):



Stage 5 (Levels 41-50):



CLI Commands

Character Management

bash

```
# View character sheet (stat page)
taskquest status
taskquest stat
taskquest char

# Create/modify character
taskquest name "Aria Shadowstep"
taskquest class Rogue
taskquest statpri DEX
taskquest statsec CHA

# View avatar
taskquest avatar
```

Shop & Rewards

```bash
# View shop
taskquest shop
taskquest shop list

# Purchase reward
taskquest shop buy 3
taskquest buy "Movie Night"

# Add custom reward
taskquest shop add "Massage" 250 "60-minute massage"

# Remove reward
taskquest shop remove 11
```

Achievements

```bash
# View achievements
taskquest achievements
taskquest achieve

# View achievement progress
taskquest achieve progress

# Set active title
taskquest title "The Undaunted"
taskquest title list
```

Progression

```bash
# View XP breakdown
taskquest xp

# View level progression
taskquest levels

# View gold history
taskquest gold

# View loot history
taskquest loot
```

Statistics

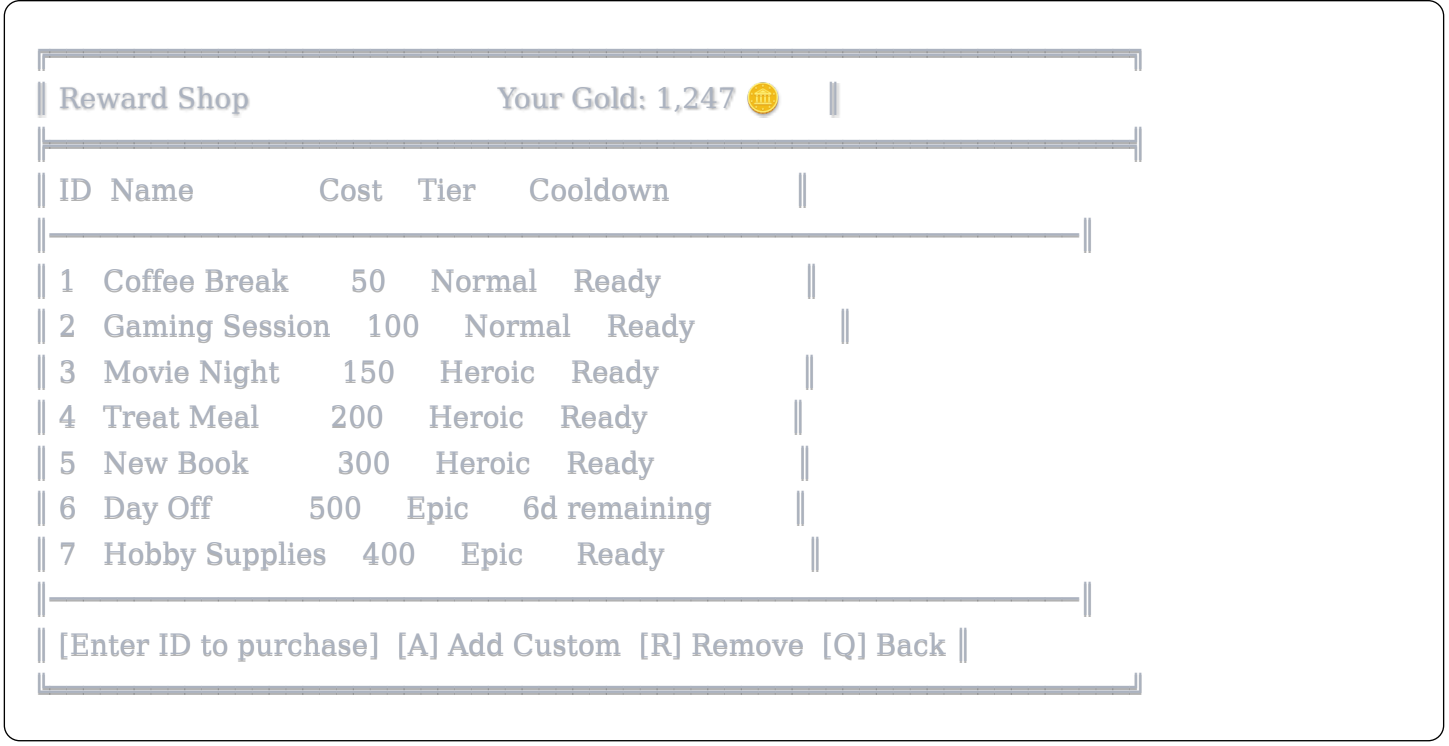
```bash
# View detailed stats
taskquest stats

# Task completion analytics
taskquest analytics

# Gold earnings over time
taskquest gold history
```

TUI (Terminal User Interface)

Main Screen Layout

```
╔═══════════════════════════════════════════════════════════════╗
║ TaskQuest - Aria Shadowstep                    [Press ? Help] ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║  Avatar:        Stats:              Progression:              ║
║    /|\          ──────────          Level: 12                 ║
║   ▄▄▄           STR: 15             XP: 15,240/16,932         ║
║   ▓▓▓           DEX: 28 ⭐          [████████████████░] 90%   ║
║  /▓▓▓\          CON: 18             Gold: 1,247 🪙            ║
║   ‖‖            INT: 22 ★                                     ║
║  / \            WIS: 16             Next Level: 1,692 XP      ║
║                 CHA: 20                                       ║
║  Rogue                              Tasks: 342 ✓              ║
║  "The Undaunted"                                              ║
╠═══════════════════════════════════════════════════════════════╣
║ Recent Activity:                                              ║
║  • +85 XP, +45 gold - Completed: Fix critical bug (Chal: 8)   ║
║  • ✨ LOOT DROP: "Coffee Break" (Normal)                      ║
║  • 🏆 Achievement Unlocked: "Master of Balance"               ║
╠═══════════════════════════════════════════════════════════════╣
║ [1] Status  [2] Shop  [3] Achievements  [4] Stats  [Q] Quit   ║
╚═══════════════════════════════════════════════════════════════╝
```

Shop Screen



Installation & Setup

Installation Methods

1. From Source (Recommended for Development):

bash

```
# Clone repository
git clone https://github.com/yourusername/taskquest
cd taskquest

# Build
cargo build --release

# Install
cargo install --path .
```

2. Via Cargo:

bash

```
cargo install taskquest
```

3. Pre-built Binaries: Download from GitHub releases for your platform.

First-Time Setup

```bash
# Run setup wizard
taskquest init

# Wizard prompts:
# 1. "What is your hero's name?"
# 2. "Choose your class: [1] Rogue, [2] Ranger, [3] Warrior, [4] Paladin, [5] Monk"
# 3. "Primary stat (66% growth): [STR/DEX/CON/INT/WIS/CHA]"
# 4. "Secondary stat (33% growth): [STR/DEX/CON/INT/WIS/CHA]"
# 5. "Enable git auto-commit? [y/n]"

# Setup creates:
# - ~/.taskquest/ directory
# - ~/.task/hooks/on-*-taskquest scripts
# - ~/.taskrc entries for UDAs
# - data/character.json with initial state
```

Taskwarrior Configuration

TaskQuest automatically adds UDA configurations to `~/.taskrc`. Manual verification:

```bash
task show | grep uda.
```

Should show:

```
uda.challenge.type=numeric
uda.challenge.label=Challenge
uda.tq_name.type=string
...
```

Cross-Platform Considerations

Linux (Fedora Bazzite)

- Standard installation works
- Uses XDG directories: `~/.config/taskquest/` for config
- Hook scripts: `~/.task/hooks/`

Android (Termux)

- Install Rust: `pkg install rust`
- Install Taskwarrior: `pkg install task`
- Build from source (cargo install takes too long)
- Hooks location: `~/data/data/com.termux/files/home/.task/hooks/`
- Use `termux-setup-storage` for syncing with device storage

Git Sync Strategy

Recommended Workflow:

1. Store TaskQuest data in git repo
2. Use branches for different devices (optional)
3. Sync via push/pull

```bash
# Initialize git in TaskQuest data directory
cd ~/.taskquest/
git init
git add .
git commit -m "Initial TaskQuest data"
git remote add origin <your-repo-url>
git push -u origin main

# On second device (after cloning)
git clone <your-repo-url> ~/.taskquest
taskquest init --skip-wizard  # Uses existing data
```

Conflict Resolution:

- Character data: Last-write-wins for stats (additive)
- Achievements: Merge (union of unlocked)
- Shop: Custom rewards merge by ID
- Loot history: Append-only

Performance Requirements

Speed Targets

- Hook execution: <50ms (imperceptible)
- CLI commands: <100ms to first output
- TUI launch: <200ms
- JSON parsing: Minimal overhead

Memory Requirements

- Runtime: <10MB RAM
- Data files: <1MB total (even after years of use)

Binary Size

- Target: <5MB static binary
- Strip debug symbols for release

Testing Strategy

Unit Tests

```bash
cargo test
```

Integration Tests

```bash
# Test hook integration
./tests/integration/test_hooks.sh

# Test XP calculation edge cases
./tests/integration/test_xp_calculator.sh

# Test loot RNG distribution
./tests/integration/test_loot_distribution.sh
```

Manual Testing Checklist

- [ ] Create character
- [ ] Complete task (various difficulties)
- [ ] Level up
- [ ] Earn achievement
- [ ] Get loot drop
- [ ] Purchase reward
- [ ] Sync between devices
- [ ] Handle corrupted data gracefully

Development Phases

Phase 1: Core System (MVP)

- [ ] Character data structure
- [ ] Basic hooks (on-modify for XP/gold)
- [ ] XP & level calculation
- [ ] Simple CLI (status, stats)
- [ ] JSON storage

Phase 2: Progression

- [ ] Loot system
- [ ] Shop & rewards
- [ ] Achievement tracking
- [ ] ASCII avatars (all classes, all stages)

Phase 3: Polish

- [ ] TUI interface (ratatui)
- [ ] Colors & formatting
- [ ] Git sync automation
- [ ] Error handling & recovery

Phase 4: Advanced Features

- [ ] Custom achievement editor
- [ ] Analytics & graphs
- [ ] Export/import character
- [ ] Achievement sharing

Dependencies (Rust Crates)

toml

```toml
[dependencies]
serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0"
rand = "0.8"
chrono = "0.4"
clap = { version = "4.5", features = ["derive"] }
ratatui = "0.26"  # For TUI
crossterm = "0.27"  # Terminal manipulation
colored = "2.1"  # Colored output
thiserror = "1.0"  # Error handling
anyhow = "1.0"  # Error context
```

Safety & Data Integrity

Preventing Data Loss

1. **Atomic writes**: Write to `.tmp` file, then rename
2. **Backup on modification**: Keep `.bak` copy
3. **Validation**: JSON schema validation before save
4. **File locking**: Prevent concurrent modifications

Error Recovery

```rust
pub fn safe_write<T: Serialize>(path: &Path, data: &T) -> Result<()> {
    let tmp_path = path.with_extension("tmp");
    let bak_path = path.with_extension("bak");

    // Write to temp file
    let file = File::create(&tmp_path)?;
    serde_json::to_writer_pretty(file, data)?;

    // Backup existing file
    if path.exists() {
        fs::copy(path, &bak_path)?;
    }

    // Atomic rename
    fs::rename(&tmp_path, path)?;

    Ok(())
}
```

Future Enhancements

Potential Features

- **Party System**: Collaborate with friends (shared achievements)
- **Seasonal Events**: Limited-time achievements
- **Quest Chains**: Multi-task storylines
- **Boss Battles**: Extra-hard tasks with bonus rewards
- **Prestige System**: Reset level for permanent bonuses
- **Character Export**: Share your hero with others
- **Themes**: Customize colors & ASCII art style
- **Mobile App**: Native Android/iOS companion (view-only)

Contributing Guidelines

1. **Code Style**: Follow `rustfmt` conventions
2. **Testing**: All PRs must include tests
3. **Documentation**: Update docs for new features
4. **Performance**: Profile before/after for hook changes
5. **Compatibility**: Test on Linux + Termux

License

MIT License - See LICENSE file for details

Credits & Acknowledgments

- **Taskwarrior**: The foundation of this system
- **Habitica**: Inspiration (and lessons learned from pitfalls)
- **Research**: Based on academic studies of gamification effectiveness
- **D&D 5e**: Stat system inspiration
- **World of Warcraft**: Level curve & loot system inspiration

---

Contact & Support

- **GitHub Issues**: Bug reports & feature requests
- **Discussions**: General questions & ideas
- **Discord**: Community chat (optional)

End of Technical Specification

This document is living and will be updated as development progresses.