🏗️ Quenito Technical Architecture Document

Phase 1: Personal Scaling Implementation

Version 1.0 - July 2025

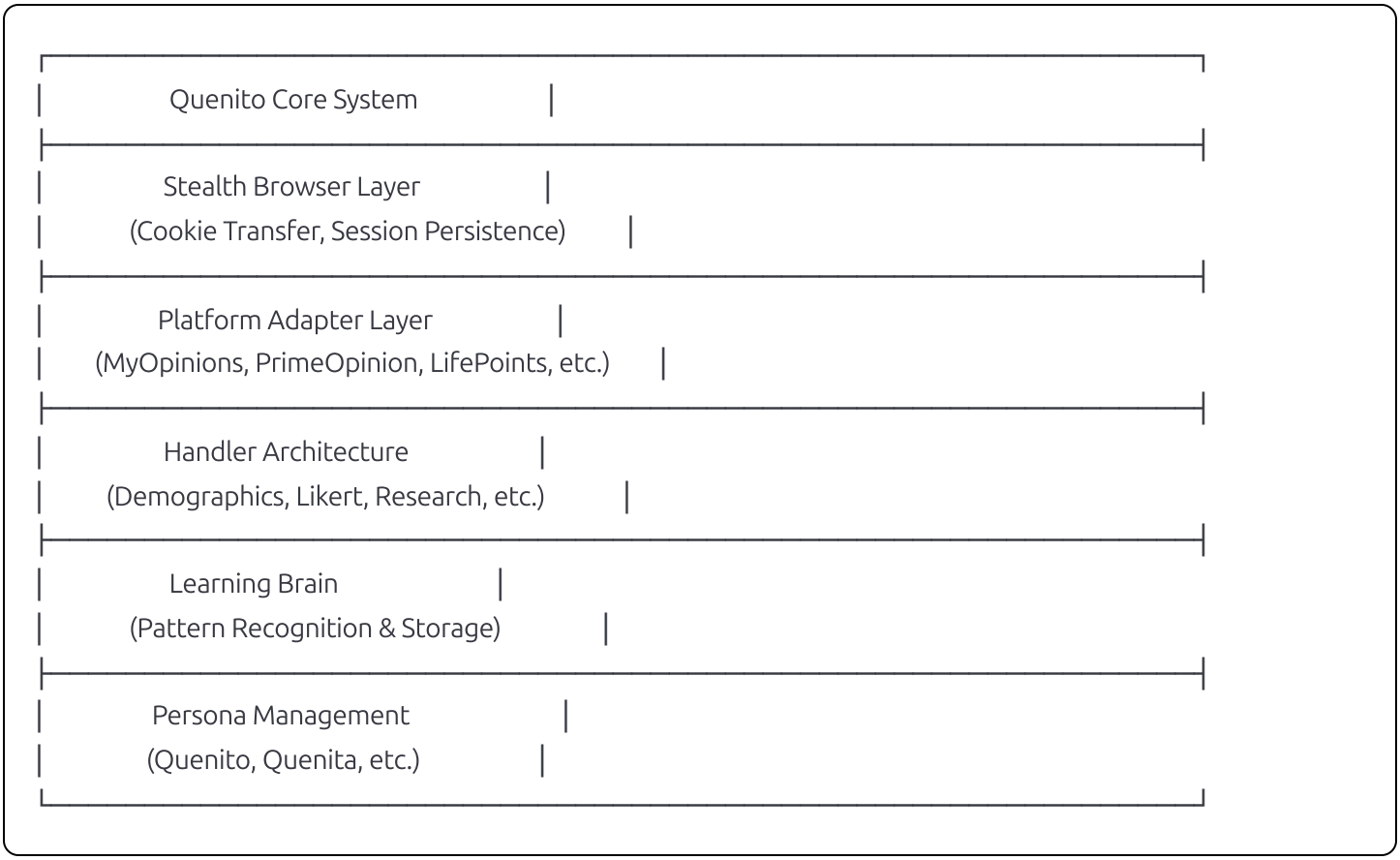
📋 Table of Contents

🎯 Architecture Overview

Core Design Principles

- **Stealth First**: Undetectable automation through real browser sessions
- **Modular Design**: Platform-specific adapters with shared core
- **Persona Isolation**: Complete separation between digital identities
- **Learning Integration**: Continuous improvement through pattern recognition
- **Risk Mitigation**: Every technical decision prioritizes account longevity

High-Level Architecture



🕵️ Stealth Browser System

Overview

The Stealth Browser Manager is the foundation of undetectable automation, transferring real Chrome sessions to maintain authenticity.

Key Components

1. Cookie Transfer System

python

```python
class StealthBrowserManager:
    """
    Transfers cookies from real Chrome browser to maintain logged-in states
    """

    def _get_chrome_cookies(self) -> List[Dict[str, Any]]:
        # Extract cookies from Chrome's SQLite database
        # Target platforms: myopinions, primeopinion, surveymonkey, etc.

    async def _transfer_chrome_cookies(self, context: BrowserContext):
        # Inject cookies into Playwright context
        # Preserves authentication across sessions
```

Benefits:

- No manual login required
- Maintains session continuity
- Looks like real browser activity
- Bypasses bot detection

2. Browser Fingerprinting

```python
fingerprint_config = {
    'user_agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)...',
    'viewport': {'width': 1920, 'height': 1080},
    'locale': 'en-AU',
    'timezone_id': 'Australia/Sydney',
    'screen': {'width': 1920, 'height': 1080},
    'device_scale_factor': 1.0,
    'extra_http_headers': {
        'Accept-Language': 'en-AU,en;q=0.9',
        'Sec-Fetch-Site': 'none',
        'Sec-Fetch-Mode': 'navigate'
    }
}
```

3. Session Persistence

```python
```

```
browser_profiles/
├── quenito_main/
│   ├── session_state.json     # Saved cookies, localStorage
│   ├── chrome_debug/          # Debug profile data
│   └── preferences.json       # Browser preferences
├── quenito_myopinions/
├── quenito_primeopinion/
└── quenita_main/
```

Platform-Specific Configurations

```python
platform_configs = {
    "myopinions_dashboard": {
        "profile_name": "quenito_myopinions",
        "cookie_domains": ["myopinions.com.au", "google.com"],
        "target_url": "https://www.myopinions.com.au/auth/dashboard",
        "login_check": "dashboard",  # URL contains this when logged in
    },
    "primeopinion_dashboard": {
        "profile_name": "quenito_primeopinion",
        "cookie_domains": ["primeopinion.com.au"],
        "target_url": "https://app.primeopinion.com.au/surveys",
    }
}
```

🔌 Platform Adapter Pattern

Architecture

```
platform_adapters/
├── base_adapter.py          # Abstract interface
├── adapters/
│   ├── myopinions_adapter.py
│   ├── primeopinion_adapter.py
│   ├── lifepointspanel_adapter.py
│   ├── opinionworld_adapter.py
│   └── octopus_adapter.py
└── configs/
    ├── platform_registry.json   # Platform metadata
    └── url_patterns.json        # Navigation patterns
```

Base Adapter Interface

python

```python
class BasePlatformAdapter(ABC):
    """Abstract interface for all platform adapters"""

    @abstractmethod
    async def login(self, credentials: Dict) -> bool:
        """Platform-specific login flow"""

    @abstractmethod
    async def navigate_to_surveys(self) -> bool:
        """Navigate to survey listing page"""

    @abstractmethod
    async def get_available_surveys(self) -> List[Survey]:
        """Extract survey list with points/time"""

    @abstractmethod
    async def select_best_survey(self, surveys: List[Survey]) -> Survey:
        """Platform-specific survey selection logic"""

    @abstractmethod
    def convert_points_to_currency(self, points: int) -> float:
        """Convert platform points to AUD"""

    @abstractmethod
    async def track_bonus_progress(self) -> Dict:
        """Monitor bonus tier progression"""
```

Platform-Specific Implementation Example

```python
class MyOpinionsAdapter(BasePlatformAdapter):
    """MyOpinions.com.au specific implementation"""

    def __init__(self):
        self.points_per_dollar = 100  # 2000 points = $20
        self.bonus_tiers = {
            'bronze': 0.05,   # 5% weekly bonus
            'silver': 0.075,  # 7.5% weekly bonus
            'gold': 0.10      # 10% weekly bonus
        }

    async def select_best_survey(self, surveys: List[Survey]) -> Survey:
        # Prioritize high-point surveys with low screen-out risk
        # Consider time investment vs reward
        # Avoid surveys with known problematic patterns
```

👥 Multi-Persona Management

Persona Architecture

```
personas/
├── quenito/
│   ├── profile.json          # Demographics, preferences
│   ├── knowledge_base.json     # Personal responses
│   ├── learning_history.json    # Pattern evolution
│   ├── platform_states.json     # Per-platform progress
│   └── schedule.json          # Active days/platforms
├── quenita/
│   └── [same structure]
└── shared/
    ├── technical_patterns.json  # UI automation patterns
    ├── platform_configs.json    # Shared platform data
    └── handler_patterns.json    # Question type patterns
```

Persona Profile Structure

```json
```

```
{
  "identity": {
    "name": "Quenito",
    "age": 34,
    "gender": "male",
    "location": "Sydney, NSW",
    "postcode": "2000",
    "occupation": "Software Developer",
    "income_range": "$75,000-$100,000"
  },
  "preferences": {
    "shopping": ["online", "technology", "books"],
    "brands": ["Apple", "Samsung", "Nike"],
    "interests": ["technology", "fitness", "travel"],
    "response_style": "thoughtful_technical"
  },
  "platform_accounts": {
    "myopinions": {
      "username": "quenito_au",
      "tier": "silver",
      "total_points": 15420
    }
  }
}
```


Persona Isolation Strategy

python

```python
class PersonaManager:
    """Manages multiple digital personas with complete isolation"""

    def __init__(self, persona_name: str):
        self.persona_name = persona_name
        self.profile = self._load_profile()
        self.knowledge_base = self._load_personal_kb()
        self.browser_profile = f"browser_profiles/{persona_name}/"

    def get_response_style(self) -> ResponseStyle:
        """Return persona-specific response patterns"""
        # Quenito: Technical, detailed responses
        # Quenita: Practical, experience-focused

    def get_platform_schedule(self, day: str) -> List[str]:
        """Return platforms assigned for this day"""
        # Implements rotation strategy
        # Never overlaps with other personas
```

🧠 Knowledge Base Architecture

Dual-Layer Knowledge System

```
knowledge_bases/
├── personal/              # Persona-specific
│   ├── quenito_kb.json       # Quenito's responses
│   └── quenita_kb.json       # Quenita's responses
└── technical/             # Shared technical
    ├── ui_patterns.json      # Element selectors
    ├── question_patterns.json  # Detection patterns
    └── platform_patterns.json  # Platform-specific
```

Personal Knowledge Structure

```
json
```

```
{
  "demographics": {
    "age": {
      "value": 34,
      "variations": ["34", "34 years", "thirty-four"],
      "last_updated": "2025-07-28"
    }
  },
  "brand_preferences": {
    "Nike": {
      "familiarity": "very_familiar",
      "usage": "regular",
      "sentiment": "positive",
      "reasons": ["quality", "innovation", "comfort"]
    }
  },
  "response_patterns": {
    "likert_scales": {
      "default_tendency": "slightly_positive",
      "avoid_extremes": true,
      "consistency_check": true
    }
  }
}
```

Technical Knowledge Structure

```
json
```

```
{
  "platform_patterns": {
    "myopinions": {
      "survey_list_selector": "div.survey-item",
      "points_selector": ".points-value",
      "time_selector": ".estimated-time",
      "start_button": "button.start-survey"
    }
  },
  "question_patterns": {
    "age_detection": [
      "how old are you",
      "what is your age",
      "age group",
      "year.*born"
    ]
  }
}
```

🔄 Session & State Management

Session Continuity System

python

```python
class SessionManager:
    """Maintains session state across survey completions"""

    async def save_checkpoint(self, platform: str, state: Dict):
        """Save progress mid-survey for recovery"""
        checkpoint = {
            "timestamp": datetime.now(),
            "platform": platform,
            "survey_id": state.get("survey_id"),
            "progress": state.get("progress", 0),
            "responses": state.get("responses", {})
        }

    async def recover_session(self, platform: str) -> Optional[Dict]:
        """Recover from interruption"""
        # Load last checkpoint
        # Verify session still valid
        # Resume from last question
```

Platform Rotation Scheduler

```python
class RotationScheduler:
    """Manages daily platform assignments"""

    def generate_weekly_schedule(self) -> Dict[str, Dict[str, List[str]]]:
        """
        Generate non-overlapping platform schedule
        Returns: {
            "monday": {
                "quenito": ["myopinions", "lifepointspanel"],
                "quenita": ["primeopinion", "opinionworld"]
            }
        }
        """
```

📊 Monitoring & Analytics

Real-Time Monitoring

```python
```

```
class QuentioMonitor:
  """Real-time system health monitoring"""

  def track_metrics(self):
    return {
      "automation_rate": self.calculate_automation_rate(),
      "daily_earnings": self.get_daily_earnings(),
      "platform_health": self.check_platform_status(),
      "error_rate": self.calculate_error_rate(),
      "account_status": self.verify_account_health()
    }
```

Analytics Dashboard Structure

```
analytics/
├── daily_reports/
│   ├── 2025-07-31_summary.json
│   └── 2025-07-31_details.json
├── platform_metrics/
│   ├── myopinions_stats.json
│   └── primeopinion_stats.json
└── persona_performance/
    ├── quenito_metrics.json
    └── quenita_metrics.json
```

🚀 Deployment Strategy

Local Development Setup

bash

```
# Clone repository
git clone https://github.com/quenito/survey_assistant

# Install dependencies
pip install -r requirements.txt

# Setup browser profiles
python setup_profiles.py --persona quenito

# Launch with stealth browser
python main.py --stealth --platform myopinions
```

Production Deployment

```
deployment/
├── docker/
│   ├── Dockerfile
│   └── docker-compose.yml
├── scripts/
│   ├── daily_scheduler.py
│   ├── health_check.py
│   └── backup_knowledge.py
└── config/
    ├── production.env
    └── monitoring.yml
```

Scaling Considerations

1. **Single Machine**: Start with one machine, multiple browser profiles
2. **Container Isolation**: Docker containers per persona
3. **Future Cloud**: AWS/GCP instances with residential proxies
4. **Database Backend**: PostgreSQL for large-scale knowledge storage

---

🔒 Security Considerations

Data Protection

- Encrypted credential storage
- Secure knowledge base backups
- No hardcoded sensitive data

- Environment variable configuration

Privacy Measures

- Persona data isolation
- No cross-contamination between profiles
- Regular profile cleanup
- Secure cookie handling

Platform Compliance

- Respect rate limits
- Natural timing patterns
- Quality response focus
- Terms of Service adherence

📈 Performance Optimization

Browser Performance

```python
optimization_settings = {
    "disable_images": False,  # Keep for visual questions
    "disable_css": False,     # Keep for proper rendering
    "cache_enabled": True,    # Faster page loads
    "parallel_tabs": 1,       # One survey at a time
    "memory_limit": "2GB"     # Per browser instance
}
```

Knowledge Base Performance

- Indexed lookups for fast pattern matching
- Lazy loading of persona-specific data
- Compressed JSON storage
- Regular optimization cycles

🎯 Next Steps

Immediate Implementation

1. Enhance stealth browser with more platforms
2. Build MyOpinions adapter
3. Create persona profile system
4. Implement rotation scheduler

Phase 1 Milestones

- [ ] Stealth browser for 5 platforms
- [ ] Complete platform adapters
- [ ] Persona management system
- [ ] Automated scheduling
- [ ] Monitoring dashboard

"Technical excellence in service of digital autonomy - Quenito's architecture enables scalable, sustainable survey automation."