# ProfitPilot: AI Agent for Autonomous E-Commerce

## Product Requirements Document for 12-Hour Hackathon

---

# Executive Summary

**Product Name:** ProfitPilot - Autonomous Selling Agent

**Hackathon Track:** The Reapers (Agents that Earn)

**Duration:** 12 Hours

**Primary Prize Targets:** AgentMail ($2,000) + Hyperspell ($2,000) + Browser-Use ($1,000)

**Expected Demo Revenue:** $100-500 in real transactions

## Key Value Proposition

An AI agent that autonomously sources, lists, negotiates, and sells products across multiple platforms while maximizing profit margins through intelligent market analysis and customer engagement.

---

# Problem Statement

Online sellers lose significant revenue due to:

- **67%** of buyers expect responses within 1 hour
- **30-40%** profit left on table due to poor negotiation
- **70%** of deals lost to response lag
- **24/7** monitoring impossible for individuals
- **Multi-platform** management overwhelming

## Market Opportunity

- $400B+ annual e-commerce marketplace transactions
- 2M+ individual sellers on eBay alone
- Average seller manages 50-100 listings
- 15-20 inquiries per day per active seller

---

# Solution Architecture

## Core Components

### 1. Email Intelligence (AgentMail - PRIMARY)

- Automated buyer communication

- Context-aware negotiation
- Multi-thread conversation management
- Proactive outreach campaigns
- Deal closing automation

### 2. Web Automation (Browser-Use)

- Multi-platform listing creation
- Real-time price updates
- Inventory monitoring
- Automated checkout for arbitrage

### 3. Memory System (Hyperspell)

- Buyer preference tracking
- Negotiation history
- Optimal strategy learning
- Context-aware responses

### 4. Market Intelligence (Perplexity)

- Real-time pricing analysis
- Trend identification
- Competitor monitoring
- Demand forecasting

### 5. Data Management (Convex)

- Transaction tracking
- Performance metrics
- Real-time dashboard
- Analytics

---

# Technical Implementation (TypeScript)

## Project Structure

```
/profitpilot-ts
  ├── /src
```

```
│   ├── /agents
│   │   ├── emailAgent.ts       # AgentMail handler
│   │   ├── browserAgent.ts     # Browser automation
│   │   └── marketAgent.ts      # Market analysis
│   ├── /memory
│   │   └── contextStore.ts     # Hyperspell integration
│   ├── /database
│   │   ├── models.ts           # Convex schemas
│   │   └── client.ts           # Database client
│   ├── /workflows
│   │   └── orchestrator.ts     # Main logic
│   ├── /demo
│   │   ├── runner.ts           # Demo execution
│   │   └── scenarios.ts        # Test scenarios
│   └── /ui
│       └── dashboard.tsx       # React dashboard
├── package.json
├── tsconfig.json
└── .env
```

---

# 12-Hour Development Timeline

## Phase 1: Core Setup (Hours 0-1)

**9:00 AM - 10:00 AM**

### Team Roles

- **Developer 1:** Core agent + AgentMail integration
- **Developer 2:** Browser-Use + web automation
- **Developer 3:** Hyperspell + Convex setup
- **Developer 4:** UI/Demo prep + Perplexity integration

### Deliverables

- Replit project initialized
- Git repository created
- API keys configured
- Basic folder structure
- TypeScript configuration

---

## Phase 2: Email Agent (Hours 1-3)

**10:00 AM - 12:00 PM**

## Core Email Agent Implementation

typescript

```
import { AgentMail } from '@agentmail/sdk';
import OpenAI from 'openai';
import { ContextStore } from '../memory/contextStore';

export class EmailAgent {
  private agentmail: AgentMail;
  private openai: OpenAI;
  private contextStore: ContextStore;

  constructor() {
    this.agentmail = new AgentMail({
      apiKey: process.env.AGENTMAIL_API_KEY!
    });
    this.openai = new OpenAI({
      apiKey: process.env.OPENAI_API_KEY!
    });
    this.contextStore = new ContextStore();
  }

  async startMonitoring(): Promise<void> {
    setInterval(async () => {
      const messages = await this.agentmail.getUnread();
      for (const message of messages) {
        await this.processMessage(message);
      }
    }, 30000);
  }

  private async processMessage(message: EmailMessage): Promise<void> {
    const analysis = await this.analyzeEmail(message);
    const context = await this.contextStore.getBuyerProfile(message.from);
    const strategy = this.calculateStrategy(analysis, context);
    const response = await this.generateResponse(message, analysis, strategy);
    await this.sendResponse(message, response);
    await this.contextStore.recordInteraction({
      buyer: message.from,
      intent: analysis.intent,
      product: analysis.product,
      timestamp: new Date()
    });
  }
}
```

## Email Templates

- Initial inquiry response

- Negotiation counter-offer
- Deal closing confirmation
- Follow-up sequences

---

## Phase 3: Browser Automation (Hours 3-5)

### 12:00 PM - 2:00 PM

### Multi-Platform Listing Automation

typescript

```typescript
import { BrowserUse } from '@browser-use/sdk';

export class BrowserAgent {
  private browser: BrowserUse;
  private platforms = ['craigslist', 'facebook', 'ebay'];

  async createListings(product: Product): Promise<ListingResults> {
    const results: ListingResults = {
      success: [],
      failed: [],
      urls: {}
    };

    for (const platform of this.platforms) {
      try {
        const url = await this.createListing(platform, product);
        results.success.push(platform);
        results.urls[platform] = url;
      } catch (error) {
        results.failed.push(platform);
      }
    }
    return results;
  }

  private async createCraigslistListing(product: Product): Promise<string> {
    const session = await this.browser.newSession();
    await session.navigate('https://craigslist.org');
    await session.click('post to classifieds');
    await session.fill('#PostingTitle', product.title);
    await session.fill('#PostingBody', product.description);
    await session.fill('#Ask', product.price.toString());
    await session.click('button[type="submit"]');
    return await session.getCurrentUrl();
  }
}
```

## Supported Platforms

- Craigslist
- Facebook Marketplace
- eBay
- Mercari (stretch goal)
- OfferUp (stretch goal)

## Phase 4: Memory & Intelligence (Hours 5-7)

**2:00 PM - 4:00 PM**

### Context Management with Hyperspell

typescript

```typescript
export class ContextStore {
  private hyperspell: Hyperspell;

  async getBuyerProfile(email: string): Promise<BuyerProfile> {
    const history = await this.hyperspell.search({
      query: `buyer:${email}`,
      limit: 50
    });

    return this.buildProfile(email, history);
  }

  async getOptimalStrategy(buyer: string, product: string): Promise<Strategy> {
    const profile = await this.getBuyerProfile(buyer);
    const productHistory = await this.hyperspell.search({
      query: `product:${product}`
    });

    return {
      initialPrice: this.calculateInitialPrice(productHistory, profile),
      minAcceptable: this.calculateMinPrice(productHistory, profile),
      negotiationRounds: profile.negotiationStyle === 'aggressive' ? 3 : 2,
      tactics: this.selectTactics(profile),
      closingIncentives: this.selectIncentives(profile)
    };
  }
}
```

### Buyer Profiling

- Purchase history tracking
- Price sensitivity analysis
- Communication preferences
- Negotiation style detection
- Conversion probability prediction

## Phase 5: Orchestration (Hours 7-9)

**4:00 PM - 6:00 PM**

### Main Orchestrator

typescript

```typescript
export class ProfitPilotOrchestrator {
  private emailAgent: EmailAgent;
  private browserAgent: BrowserAgent;
  private marketAgent: MarketAgent;
  private contextStore: ContextStore;
  private db: DatabaseClient;

  async start(): Promise<void> {
    await Promise.all([
      this.emailAgent.startMonitoring(),
      this.startMarketMonitoring(),
      this.startMetricsUpdater()
    ]);
  }

  private async handleIncomingEmail(emailData: any): Promise<void> {
    const marketData = await this.marketAgent.analyzeProduct(emailData.product);
    const buyerContext = await this.contextStore.getBuyerProfile(emailData.from);
    const strategy = await this.contextStore.getOptimalStrategy(
      emailData.from,
      emailData.product
    );

    await this.emailAgent.generateResponse(emailData, strategy, marketData);
    await this.db.createTransaction({
      buyerEmail: emailData.from,
      product: emailData.product,
      status: 'negotiating',
      initialPrice: strategy.initialPrice
    });
  }
}
```

---

## Phase 6: Demo & Testing (Hours 9-12)

**6:00 PM - 9:00 PM**

### Demo Scenarios

1. **Load Inventory** - Show 3 products ready to sell
2. **Create Listings** - Live browser automation
3. **Process Inquiry** - Email arrives, AI responds
4. **Handle Negotiation** - Multi-round back-and-forth
5. **Close Deal** - Complete transaction
6. **Show Metrics** - Real-time dashboard

### Live Dashboard

- Real-time metrics updates via WebSocket
- Activity feed showing all actions
- Profit counter animation
- Conversion rate tracking
- Response time monitoring

---

# Demo Execution Guide

## Pre-Demo Setup (5 minutes)

```bash
# Install dependencies
npm install

# Configure environment
cp .env.example .env
# Add all API keys

# Start demo server
npm run demo

# Open dashboard
open http://localhost:3000
```

## Demo Script (5 minutes)

### Opening (30 seconds)

"Every minute, thousands of dollars die in email inboxes. We built ProfitPilot - an AI agent that makes money while you sleep."

## Live Demo (3 minutes)

### Step 1: Dashboard

- Show live metrics dashboard
- Click "Run Full Demo"

### Step 2: Inventory

- Load 3 demo products
- Show market prices

### Step 3: Listings

- Browser creates real listings
- Show multi-platform URLs

### Step 4: Email Processing

- Buyer inquiry arrives
- AI analyzes and responds in <10 seconds
- Show email thread

### Step 5: Negotiation

- Buyer counters at $700 (asked $799)
- AI offers $750 with urgency bonus
- Deal closes

### Step 6: Scale

- Process 5 more inquiries
- Show 3 deals closed
- $2,100 profit in 3 minutes

## Architecture (30 seconds)

- AgentMail: Email automation
- Hyperspell: Buyer memory
- Browser-Use: Listing automation
- Perplexity: Market intelligence

## Results (1 minute)

- 50+ emails processed

- 10 deals closed
- $500 actual profit
- Projected: $10,000/month

---

# Success Metrics

## Minimum Viable Product (6 hours)

- ✅ Email sending/receiving via AgentMail
- ✅ One platform listing automation
- ✅ Basic negotiation logic
- ✅ Simple dashboard

## Target Success (9 hours)

- ✅ Full email automation
- ✅ Multi-platform listings
- ✅ Context-aware negotiations
- ✅ Real transaction completed
- ✅ Metrics dashboard

## Stretch Goals (12 hours)

- ✅ All tools integrated
- ✅ Voice negotiation (LiveKit)
- ✅ Advanced analytics
- ✅ $500+ demonstrated profit

---

# Key Differentiators

1. **Real Money Generation** - Not a concept, actual transactions
2. **Full Autonomy** - Runs without human intervention
3. **Multi-Tool Synergy** - Uses 8+ sponsor tools effectively
4. **Measurable ROI** - Clear profit metrics
5. **Instant Scale** - Handle unlimited products/buyers

---

# Risk Mitigation

## Technical Risks

- **API Failures:** Implement retry logic with exponential backoff
- **Platform Bans:** Rate limiting, human-like behavior patterns
- **Email Deliverability:** Use verified domains, warm up sending

## Demo Risks

- **Live Demo Failure:** Pre-recorded backup video
- **Network Issues:** Local demo mode with cached data
- **Time Constraints:** Focus on core features first

---

# Post-Hackathon Potential

## Immediate Monetization

- **SaaS Model:** $99/month per seller
- **Transaction Fee:** 5% of profit generated
- **Enterprise:** Custom pricing for businesses

## Growth Projections

- **Week 1:** 10 beta users
- **Month 1:** 100 paying customers
- **Month 3:** $50K MRR
- **Year 1:** $1M ARR

## Exit Opportunities

- Acquisition by e-commerce platforms (eBay, Mercari)
- Integration with existing CRM/sales tools
- White-label for enterprise retailers

---

# Appendix: Code Snippets

## Email Response Generation

typescript

```
private async generateResponse(
  message: EmailMessage,
  analysis: EmailAnalysis,
  strategy: ResponseStrategy
): Promise<string> {
  const response = await this.openai.chat.completions.create({
    model: "gpt-4",
    messages: [
      {
        role: "system",
        content: `Generate a ${strategy.tone} response that:
          - Offers $${strategy.pricePoint}
          - Shows ${strategy.flexibility * 100}% flexibility
          - Includes: ${strategy.incentives.join(', ')}
          ${strategy.closeAttempt ? '- Attempts to close NOW' : ''}`
      },
      {
        role: "user",
        content: `Email: ${message.body}\nProduct: ${analysis.product}`
      }
    ]
  });
  return response.choices[0].message.content!;
}
```

## Market Analysis

typescript

```
async analyzeProductValue(productName: string): Promise<MarketData> {
  const query = `current selling price ${productName} marketplace 2024`;
  const response = await this.perplexity.search(query);
  const prices = this.extractPrices(response);

  return {
    average: statistics.mean(prices),
    median: statistics.median(prices),
    optimal: this.calculateOptimalPrice(prices),
    demand: this.calculateDemand(response)
  };
}
```

## Deal Closing Logic

typescript

```
async closeDeal(negotiation: NegotiationState): Promise<Transaction> {
  // Send confirmation email
  await this.emailAgent.sendConfirmation(negotiation);

  // Update listing status
  await this.browserAgent.markAsSold(negotiation.listingUrls);

  // Record transaction
  const transaction = await this.db.createTransaction({
    buyer: negotiation.buyerEmail,
    product: negotiation.product,
    finalPrice: negotiation.agreedPrice,
    profit: negotiation.agreedPrice - negotiation.cost,
    status: 'completed'
  });

  // Update metrics
  this.metrics.dealsCompleted++;
  this.metrics.totalProfit += transaction.profit;

  return transaction;
}
```

---

## Contact & Resources

**Team:** ProfitPilot

**GitHub:** github.com/profitpilot/hackathon

**Demo:** profitpilot.demo.app

**Email:** team@profitpilot.ai

---

*Built with ❤️ for the AI Agent Hackathon*