⏺ ***📋 ZipPicks Launch System PRD (v2.0)***

**🎯 Project Vision**

Build a simple, scalable system to generate and publish AI-powered Top 10 restaurant lists with full SEO optimization, starting manual but designed for easy

automation.

**🏃‍♂️ Launch Goals**

- **Week 1**: Generate first 50 Top 10 lists manually with schema markup

- **Week 2**: Publish to WordPress with one-click workflow and SEO optimization

- **Month 1**: 500+ lists across major cities with validated quality

- **Month 2**: Semi-automated pipeline with version tracking

---

**🏗️ System Architecture**

Phase 1: Manual Excellence (Week 1-2)

[Restaurant Data] → [Python Scripts] → [Claude Manual] → [Validated Output] → [WordPress + Schema]

Phase 2: Semi-Automated (Week 3-4)

[Restaurant Data] → [Python Scripts] → [Claude API/Browser] → [Auto-Validation] → [WordPress Auto-Publish]

---

**📁 Project Structure**

/zippicks-generator/

├── generate.py # Main CLI - Run everything from here

├── data*\_manager.py # Load and prep restaurant data*

*├── prompt\_*engine.py # Format prompts for Claude (with versioning)

├── response*\_validator.py # Validate Claude responses*

*├── publisher.py # Push to WordPress with SEO*

*├── schema\_*builder.py # Generate JSON-LD structured data

├── monitor.py # Track what's been generated

│

├── /config/

│ ├── cities.yaml # Cities to process

│ ├── vibes.yaml # Vibe definitions

│ ├── wp*\_config.yaml # WordPress settings*

*│ └── prompts/ # Versioned prompt templates*

*│ ├── v1.0/*

*│ │ └── top10\_*prompt.txt

│ └── v1.1/

│ └── top10*\_prompt.txt*

*│*

*├── /data/*

*│ ├── restaurants.csv # Your restaurant data*

*│ └── /cache/ # Processed data cache*

*│*

*├── /output/*

*│ ├── /drafts/ # Claude responses with metadata*

*│ ├── /validated/ # Validated and cleaned responses*

*│ ├── /published/ # Final formatted content*

*│ └── /logs/ # Generation history*

*│*

*└── /templates/*

*├── top10\_*prompt.txt # Active prompt template

└── wp\_post.html # WordPress HTML template

---

**🚀 Core Workflow**

1. Daily Generation Flow

# Morning routine (30 minutes human time)

python generate.py --mode "daily" --prompt-version "1.1"

# What happens:

1. Shows you which city/vibe combos to generate today

2. Prepares formatted prompts with version tracking

3. You paste into Claude, get responses

4. System validates responses automatically

5. One command publishes all to WordPress with schema

2. The Main Script (Updated)

# generate.py - Your single command center

"""

Usage:

python generate.py --mode daily --prompt-version 1.1 # Run daily batch

python generate.py --city "new-york" --vibe "date-night" # Single list

python generate.py --publish-all # Push validated drafts to WordPress

python generate.py --status # See what's been done

python generate.py --validate-pending # Re-validate drafts

"""

import click

import yaml

from datetime import datetime

from pathlib import Path

from response\_validator import ResponseValidator

from prompt\_engine import PromptEngine

@click.command()

@click.option('--mode', help='daily, single, publish-all, status, validate-pending')

@click.option('--city', help='City slug (e.g., new-york)')

@click.option('--vibe', help='Vibe slug (e.g., date-night)')

@click.option('--prompt-version', default='1.0', help='Prompt version to use')

def main(mode, city, vibe, prompt\_version):

if mode == 'daily':

run\_daily\_batch(prompt\_version)

elif mode == 'single':

generate\_single(city, vibe, prompt\_version)

elif mode == 'publish-all':

publish\_validated\_drafts()

elif mode == 'status':

show\_generation\_status()

elif mode == 'validate-pending':

validate\_all\_pending()

def run\_daily\_batch(prompt\_version):

"""Your daily workflow in one command"""

# Initialize engines

prompt\_engine = PromptEngine(version=prompt\_version)

validator = ResponseValidator()

# 1. Load today's targets

targets = get\_todays\_targets()

print(f"\n📋 Today's Generation Queue: {len(targets)} lists")

print(f"📝 Using Prompt Version: {prompt\_version}\n")

for city, vibe in targets:

print(f"\n{'='\*60}")

print(f"🏙️ {city.upper()} × {vibe.upper()}")

print('='\*60)

# 2. Prepare data

restaurants = load\_city\_restaurants(city)

print(f"✓ Loaded {len(restaurants)} restaurants")

# 3. Generate versioned prompt

prompt\_data = prompt\_engine.create\_prompt(restaurants, city, vibe)

# 4. Save prompt with metadata

prompt\_file = save\_prompt\_with\_metadata(city, vibe, prompt\_data)

print(f"✓ Prompt saved: {prompt\_file}")

print(f" Version: {prompt\_data['version']}")

print(f" Restaurants: {prompt\_data['restaurant\_count']}")

# 5. Display prompt for copying

print("\n" + "="\*60)

print("📋 COPY THIS PROMPT TO CLAUDE:")

print("="\*60)

print(prompt\_data['prompt'])

print("="\*60)

# 6. Wait for Claude response

print("\n📝 Paste Claude's response below (end with '###' on new line):\n")

response = get\_multiline\_input()

# 7. Validate response

validation\_result = validator.validate\_response(response, city, vibe)

if validation\_result['valid']:

print("✅ Response validated successfully!")

# Save validated draft with metadata

save\_validated\_draft(city, vibe, validation\_result['parsed\_data'], prompt\_version)

# Show validation summary

print(f" ✓ Restaurant count: {validation\_result['restaurant\_count']}")

print(f" ✓ All required fields present")

print(f" ✓ No duplicates found")

else:

print("❌ Validation failed:")

for error in validation\_result['errors']:

print(f" - {error}")

# Save to failed directory for manual review

save\_failed\_draft(city, vibe, response, validation\_result['errors'])

if click.confirm("Try again with this city/vibe?"):

# Retry logic here

continue

print(f"\n{'='\*40}\n")

# Quick break between generations

if not click.confirm("Continue with next city/vibe?", default=True):

break

print(f"\n🎉 Daily batch complete!")

print(f"Run 'python generate.py --publish-all' to push validated drafts to WordPress")

def save\_prompt\_with\_metadata(city, vibe, prompt\_data):

"""Save prompt with YAML frontmatter"""

# Create metadata

metadata = {

'city': city,

'vibe': vibe,

'date': datetime.now().isoformat(),

'prompt\_version': prompt\_data['version'],

'restaurant\_count': prompt\_data['restaurant\_count'],

'min\_rating': 4.3,

'template': prompt\_data['template\_name']

}

# Format with frontmatter

content = "---\n"

content += yaml.dump(metadata, default\_flow\_style=False)

content += "---\n\n"

content += prompt\_data['prompt']

# Save to file

prompt\_dir = Path(f"output/drafts/{city}/{vibe}")

prompt\_dir.mkdir(parents=True, exist\_ok=True)

prompt\_file = prompt\_dir / f"prompt\_v{prompt\_data['version']}.txt"

with open(prompt\_file, 'w') as f:

f.write(content)

return prompt\_file

3. Response Validator (New Component)

# response\_validator.py - Validate Claude responses

import re

import json

from typing import Dict, List, Tuple

from datetime import datetime

class ResponseValidator:

"""Validates Claude responses for structure and quality"""

def \_\_init\_\_(self):

self.required\_fields = ['name', 'why\_perfect', 'must\_try', 'address', 'price\_range']

self.price\_ranges = ['$', '$$', '$$$', '$$$$']

def validate\_response(self, response: str, city: str, vibe: str) -> Dict:

"""

Validate a Claude response for correctness

Returns:

Dict with keys:

- valid: bool

- errors: List[str]

- parsed\_data: Dict (if valid)

- restaurant\_count: int

"""

result = {

'valid': True,

'errors': [],

'parsed\_data': None,

'restaurant\_count': 0

}

try:

# 1. Parse the response

parsed = self.parse\_claude\_response(response)

# 2. Check basic structure

if 'restaurants' not in parsed or not isinstance(parsed['restaurants'], list):

result['errors'].append("Missing or invalid restaurants list")

result['valid'] = False

return result

restaurants = parsed['restaurants']

result['restaurant\_count'] = len(restaurants)

# 3. Validate restaurant count

if len(restaurants) != 10:

result['errors'].append(f"Expected 10 restaurants, got {len(restaurants)}")

result['valid'] = False

# 4. Check for duplicates

names = [r.get('name', '').lower() for r in restaurants]

if len(names) != len(set(names)):

result['errors'].append("Duplicate restaurant names found")

result['valid'] = False

# 5. Validate each restaurant

for i, restaurant in enumerate(restaurants, 1):

errors = self.validate\_restaurant(restaurant, i)

if errors:

result['errors'].extend(errors)

result['valid'] = False

# 6. Validate ranking sequence

ranks = [r.get('rank', 0) for r in restaurants]

if ranks != list(range(1, len(ranks) + 1)):

result['errors'].append("Invalid ranking sequence")

result['valid'] = False

# 7. Check vibe relevance (basic check)

if not self.validate\_vibe\_relevance(restaurants, vibe):

result['errors'].append(f"Some restaurants don't match the '{vibe}' vibe")

# This is a warning, not a failure

# If valid, add metadata

if result['valid']:

parsed['city'] = city

parsed['vibe'] = vibe

parsed['validated\_at'] = datetime.now().isoformat()

parsed['city\_title'] = city.replace('-', ' ').title()

parsed['vibe\_title'] = vibe.replace('-', ' ').title()

result['parsed\_data'] = parsed

except Exception as e:

result['errors'].append(f"Parse error: {str(e)}")

result['valid'] = False

return result

def parse\_claude\_response(self, response: str) -> Dict:

"""Parse Claude's response into structured data"""

restaurants = []

# Updated regex patterns for common Claude formats

patterns = [

# Format: \*\*1. Restaurant Name\*\*

r'\\*\\*(\d+)\.\s+([^\*\n]+)\\*\\*\s\*\n\s\*-?\s\*Why:?\s\*([^\n]+)\n\s\*-?\s\*Must-try:?\s\*([^\n]+)\n\s\*-?\s\*Address:?\s\*([^\n]+)\n\s\*-?\s\*Price:?\s\*([^\n]+)',

# Format: 1. Restaurant Name (without bold)

r'(\d+)\.\s+([^\n]+)\n\s\*-?\s\*Why:?\s\*([^\n]+)\n\s\*-?\s\*Must-try:?\s\*([^\n]+)\n\s\*-?\s\*Address:?\s\*([^\n]+)\n\s\*-?\s\*Price:?\s\*([^\n]+)',

# Format with headers

r'(?:###?\s\*)?(\d+)\.\s+([^\n]+)\n\s\*(?:Why[^\n]\*:)?\s\*([^\n]+)\n\s\*(?:Must-try[^\n]\*:)?\s\*([^\n]+)\n\s\*(?:Address[^\n]\*:)?\s\*([^\n]+)\n\s\*(?:Price[^\n

]\*:)?\s\*([^\n]+)'

]

# Try each pattern

for pattern in patterns:

matches = re.finditer(pattern, response, re.MULTILINE | re.IGNORECASE)

for match in matches:

restaurant = {

'rank': int(match.group(1)),

'name': match.group(2).strip(),

'why\_perfect': match.group(3).strip(),

'must\_try': match.group(4).strip(),

'address': match.group(5).strip(),

'price\_range': self.normalize\_price(match.group(6).strip())

}

restaurants.append(restaurant)

if restaurants:

break

return {'restaurants': restaurants}

def validate\_restaurant(self, restaurant: Dict, index: int) -> List[str]:

"""Validate individual restaurant entry"""

errors = []

# Check required fields

for field in self.required\_fields:

if field not in restaurant or not restaurant[field]:

errors.append(f"Restaurant #{index}: Missing {field}")

elif field == 'price\_range' and restaurant[field] not in self.price\_ranges:

errors.append(f"Restaurant #{index}: Invalid price range '{restaurant[field]}'")

# Validate field lengths

if restaurant.get('why\_perfect') and len(restaurant['why\_perfect']) < 20:

errors.append(f"Restaurant #{index}: 'Why perfect' too short")

if restaurant.get('must\_try') and len(restaurant['must\_try']) < 5:

errors.append(f"Restaurant #{index}: 'Must try' too short")

# Validate address format (basic check)

if restaurant.get('address') and not re.search(r'\d+.\*,', restaurant['address']):

errors.append(f"Restaurant #{index}: Address may be incomplete")

return errors

def normalize\_price(self, price: str) -> str:

"""Normalize price range to standard format"""

# Remove extra characters

price = re.sub(r'[^$]', '', price)

# Ensure it's valid

if price in self.price\_ranges:

return price

elif len(price) > 0 and len(price) <= 4:

return '$' \* len(price)

else:

return '$$' # Default

def validate\_vibe\_relevance(self, restaurants: List[Dict], vibe: str) -> bool:

"""Basic check that restaurants match the vibe"""

vibe\_keywords = {

'date-night': ['romantic', 'intimate', 'cozy', 'candlelit', 'special'],

'family-friendly': ['kids', 'family', 'casual', 'spacious', 'friendly'],

'quick-lunch': ['fast', 'quick', 'lunch', 'convenient', 'efficient'],

'trendy-vibes': ['trendy', 'hip', 'modern', 'instagram', 'stylish'],

'late-night': ['late', 'night', 'open late', '24', 'midnight']

}

keywords = vibe\_keywords.get(vibe, [])

if not keywords:

return True # Can't validate unknown vibes

# Check if at least 50% of restaurants mention vibe keywords

matches = 0

for restaurant in restaurants:

why\_perfect = restaurant.get('why\_perfect', '').lower()

if any(keyword in why\_perfect for keyword in keywords):

matches += 1

return matches >= len(restaurants) \* 0.5

def generate\_validation\_report(self, validation\_results: List[Dict]) -> str:

"""Generate a summary report of validation results"""

total = len(validation\_results)

valid = sum(1 for r in validation\_results if r['valid'])

report = f"""

VALIDATION REPORT

================

Total Processed: {total}

Valid: {valid} ({valid/total\*100:.1f}%)

Failed: {total - valid}

Common Errors:

"""

# Aggregate errors

error\_counts = {}

for result in validation\_results:

for error in result.get('errors', []):

error\_type = error.split(':')[0]

error\_counts[error\_type] = error\_counts.get(error\_type, 0) + 1

for error\_type, count in sorted(error\_counts.items(), key=lambda x: x[1], reverse=True):

report += f"- {error\_type}: {count} occurrences\n"

return report

# Helper function for CLI

def validate\_draft\_file(file\_path: str) -> Dict:

"""Validate a saved draft file"""

validator = ResponseValidator()

with open(file\_path, 'r') as f:

content = f.read()

# Extract metadata if present

if content.startswith('---'):

# Has frontmatter

parts = content.split('---', 2)

metadata = yaml.safe\_load(parts[1])

response = parts[2].strip()

else:

metadata = {}

response = content

return validator.validate\_response(

response,

metadata.get('city', 'unknown'),

metadata.get('vibe', 'unknown')

)

4. Prompt Engine with Versioning (New Component)

# prompt\_engine.py - Versioned prompt management

import yaml

from pathlib import Path

from datetime import datetime

from typing import Dict, List

class PromptEngine:

"""Manages prompt templates with version tracking"""

def \_\_init\_\_(self, version: str = "1.0"):

self.version = version

self.template\_dir = Path(f"config/prompts/v{version}")

self.template = self.\_load\_template()

def \_load\_template(self) -> str:

"""Load prompt template for current version"""

template\_file = self.template\_dir / "top10\_prompt.txt"

if not template\_file.exists():

# Fall back to latest version

print(f"⚠️ Version {self.version} not found, using latest")

latest = self.\_get\_latest\_version()

template\_file = Path(f"config/prompts/v{latest}/top10\_prompt.txt")

with open(template\_file, 'r') as f:

return f.read()

def \_get\_latest\_version(self) -> str:

"""Find the latest prompt version"""

prompt\_dir = Path("config/prompts")

versions = [d.name.replace('v', '') for d in prompt\_dir.iterdir() if d.is\_dir()]

return sorted(versions, key=lambda x: float(x))[-1]

def create\_prompt(self, restaurants\_df, city: str, vibe: str) -> Dict:

"""Create a versioned prompt with metadata"""

# Load vibe description

vibes = self.\_load\_vibes()

vibe\_info = vibes.get(vibe, {})

# Format restaurant data

restaurants\_text = self.\_format\_restaurants(restaurants\_df)

# Fill template

prompt = self.template.format(

city=city.replace('-', ' ').title(),

vibe=vibe.replace('-', ' '),

vibe\_description=vibe\_info.get('description', ''),

restaurants\_list=restaurants\_text,

date=datetime.now().strftime('%B %Y')

)

return {

'prompt': prompt,

'version': self.version,

'template\_name': 'top10\_prompt.txt',

'restaurant\_count': len(restaurants\_df),

'city': city,

'vibe': vibe,

'generated\_at': datetime.now().isoformat()

}

def \_format\_restaurants(self, df) -> str:

"""Format restaurant data for prompt"""

restaurants = []

for \_, row in df.iterrows():

# Parse vibe attributes

vibe\_attrs = []

if isinstance(row.get('vibe\_attributes'), dict):

vibe\_attrs = [k for k, v in row['vibe\_attributes'].items() if v]

# Format entry

entry = f"""

Name: {row['name']}

Rating: {row['yelp\_rating']} ({row.get('yelp\_review\_count', 'N/A')} reviews)

Price: {row.get('price\_range', 'N/A')}

Cuisine: {row.get('cuisine\_type', 'N/A')}

Address: {row['address']}

Vibe: {', '.join(vibe\_attrs) if vibe\_attrs else 'N/A'}

"""

# Add review excerpt if available

if row.get('review\_excerpts'):

try:

reviews = json.loads(row['review\_excerpts'])

if reviews and len(reviews) > 0:

entry += f"Recent Review: \"{reviews[0].get('text', '')[:150]}...\"\n"

except:

pass

restaurants.append(entry.strip())

return '\n\n'.join(restaurants)

def \_load\_vibes(self) -> Dict:

"""Load vibe definitions"""

with open('config/vibes.yaml', 'r') as f:

return yaml.safe\_load(f).get('vibes', {})

def get\_version\_info(self) -> Dict:

"""Get information about this prompt version"""

changelog\_file = self.template\_dir / "changelog.txt"

changelog = ""

if changelog\_file.exists():

with open(changelog\_file, 'r') as f:

changelog = f.read()

return {

'version': self.version,

'template\_path': str(self.template\_dir),

'changelog': changelog,

'created\_at': datetime.fromtimestamp(self.template\_dir.stat().st\_mtime).isoformat()

}

# Version management helpers

def create\_new\_prompt\_version(base\_version: str, changes: str) -> str:

"""Create a new prompt version based on existing one"""

# Parse version

major, minor = base\_version.split('.')

new\_version = f"{major}.{int(minor) + 1}"

# Create new version directory

old\_dir = Path(f"config/prompts/v{base\_version}")

new\_dir = Path(f"config/prompts/v{new\_version}")

new\_dir.mkdir(parents=True, exist\_ok=True)

# Copy template

import shutil

shutil.copy(old\_dir / "top10\_prompt.txt", new\_dir / "top10\_prompt.txt")

# Create changelog

with open(new\_dir / "changelog.txt", 'w') as f:

f.write(f"Version {new\_version} - {datetime.now().strftime('%Y-%m-%d')}\n")

f.write(f"Based on: v{base\_version}\n\n")

f.write(f"Changes:\n{changes}\n")

print(f"✅ Created new prompt version: {new\_version}")

return new\_version

5. Updated Publisher with Schema Integration

The publisher.py now includes:

- Full JSON-LD schema generation for SEO

- Automatic keyword mapping based on vibes

- WordPress meta field integration

- Schema validation before publishing

---

**📈 Scaling Path**

Week 1-2: Manual Excellence with Validation

- Generate 5-10 lists per day manually

- All responses validated before saving

- Track prompt performance with versioning

Week 3-4: Automation Ready

- Browser automation with built-in validation

- A/B test different prompt versions

- Schema markup ensures rich snippets

Month 2: Full Pipeline

- API integration when available

- Automated validation and retry logic

- Prompt optimization based on validation metrics

---

**🎯 Success Metrics**

Launch Week Goals

- 50 Top 10 lists generated and validated

- 100% validation pass rate

- Schema markup on all published posts

- Track which prompt versions perform best

Month 1 Goals

- 500+ validated lists

- 20 cities with complete coverage

- Rich snippets appearing in Google

- Prompt version 2.0 based on learnings

---

**💻 Quick Start Commands**

# Initial setup

git clone [repo]

cd zippicks-generator

pip install -r requirements.txt

# Test validation

python generate.py --city "san-francisco" --vibe "date-night" --prompt-version "1.0"

# Run daily batch with validation

python generate.py --mode daily --prompt-version "1.1"

# Validate existing drafts

python generate.py --validate-pending

# Publish validated content with schema

python generate.py --publish-all

# Check generation status

python generate.py --status

---

**🚀 Why This Approach Wins**

1. **Quality Assured**: Validation ensures consistent, high-quality output

2. **SEO Optimized**: Schema markup maximizes search visibility

3. **Version Controlled**: Track what prompts work best

4. **Scale Ready**: Every component built for automation

5. **Data Driven**: Learn from validation metrics to improve

The system now ensures every published list meets quality standards and is optimized for search engines from day one!