# **Caprae LeadGen Tool Documentation**

**Project Name**: Caprae LeadGen Tool

**Abstract**:

The Caprae LeadGen Tool is a Flask-based web application designed for Caprae Capital Partners to streamline lead generation for acquisition entrepreneurship. Built with a single `app.py` file and one `index.html` template (with embedded styling), it uses SQLite3 to manage leads, offering features like filtering, searching, sorting, adding, editing, deleting, importing, and exporting leads with scoring and pagination. Developed within a 5-hour constraint, it aligns with Caprae’s MaaS model, targeting businesses with $5–$25M revenue, and is optimized for Pydroid 3 on Android.

**Problem Statement**:

Caprae Capital requires an efficient lead generation tool to identify and manage high-value acquisition targets. Existing tools like SaaSquatch Leads lack advanced features such as lead scoring, editing, and robust filtering, limiting their ability to prioritize actionable leads. The challenge is to build a minimal, user-friendly tool within 5 hours that enhances lead management and integrates with sales workflows.

**Objectives**

- Develop a lightweight tool using Flask and SQLite3 to avoid Django-related errors.

- Implement CRUD operations (Create, Read, Update, Delete) for lead management.

- Enable filtering by industry/revenue, searching by company name, and sorting by name/revenue/score.

- Support CSV import/export with deduplication and validation.

- Add lead scoring to prioritize high-value targets.

- Ensure responsive, modern UI for mobile use (Pydroid 3).

- Align with Caprae’s MaaS model for actionable lead generation.

**Implementation**:

- Tech Stack: Flask 2.3.3, SQLite3, Jinja2, HTML/CSS.

- Structure:

- `app.py`: Handles routing, database operations, and logic (filtering, scoring, CRUD, import/export).

- `index.html`: Single template with embedded CSS for lead list, detail/edit view, and forms.

- `sample\_leads.csv`: Dataset with 15 leads for simulated scraping.

- Features:

- CRUD: Add, edit, view, delete leads with validation (email format, positive revenue).

-Search/Filter/Sort: Filter by industry/revenue, search by company name, sort by name/revenue/score.

- Import/Export: Bulk import from CSV; export filtered/all leads with timestamped filenames.

- Scoring: Assigns scores (1–5) based on revenue thresholds.

- Pagination: Displays 5 leads per page.

- Deduplication: Ensures unique company names.

- Database: SQLite3 (`leads.db`) with schema migration to handle `score` column addition.

Framework Code:

App.py

From flask import Flask, render\_template, request, Response, redirect, url\_for

Import sqlite3

Import csv

From io import StringIO

Import os

Import re

From datetime import datetime

App = Flask(\_\_name\_\_)

# Initialize or migrate SQLite3 database

Def init\_db():

Conn = sqlite3.connect(‘leads.db’)

Cursor = conn.cursor()

# Create table if not exists

Cursor.execute(‘’’

CREATE TABLE IF NOT EXISTS leads (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

Company\_name TEXT NOT NULL UNIQUE,

Website TEXT,

Contact\_email TEXT,

Industry TEXT NOT NULL,

Revenue REAL NOT NULL,

Score INTEGER,

Created\_at TEXT NOT NULL

)

‘’’)

# Check if score column exists; if not, add it

Cursor.execute(“PRAGMA table\_info(leads)”)

Columns = [col[1] for col in cursor.fetchall()]

If ‘score’ not in columns:

Cursor.execute(‘ALTER TABLE leads ADD COLUMN score INTEGER’)

# Update existing rows with scores

Cursor.execute(‘SELECT id, revenue FROM leads’)

For row in cursor.fetchall():

Score = calculate\_score(row[1])

Cursor.execute(‘UPDATE leads SET score = ? WHERE id = ?’, (score, row[0]))

Conn.commit()

Conn.close()

# Validate email format

Def is\_valid\_email(email):

Pattern = r’^[\w\.-][+@[\w\.-]+\.\w+$](mailto:+@[\w\.-%5d+\.\w+$)’

Return bool(re.match(pattern, email)) if email else True

# Calculate lead score based on revenue

Def calculate\_score(revenue):

If revenue >= 15000000:

Return 5

Elif revenue >= 10000000:

Return 4

Elif revenue >= 5000000:

Return 3

Elif revenue >= 3000000:

Return 2

Return 1

# Populate database from CSV

Def populate\_leads():

Conn = sqlite3.connect(‘leads.db’)

Cursor = conn.cursor()

Cursor.execute(‘DELETE FROM leads’) # Clear existing data

Seen\_names = set()

With open(‘sample\_leads.csv’, ‘r’) as file:

Reader = csv.DictReader(file)

For row in reader:

Company\_name = row[‘company\_name’].strip()

If company\_name in seen\_names or not is\_valid\_email(row[‘contact\_email’]):

Continue

Try:

Revenue = float(row[‘revenue’])

If revenue < 0:

Continue

Score = calculate\_score(revenue)

Cursor.execute(‘’’

INSERT OR IGNORE INTO leads (company\_name, website, contact\_email, industry, revenue, score, created\_at)

VALUES (?, ?, ?, ?, ?, ?, ?)

‘’’, (

Company\_name,

Row[‘website’].strip(),

Row[‘contact\_email’].strip(),

Row[‘industry’].strip(),

Revenue,

Score,

Datetime.now().strftime(‘%Y-%m-%d %H:%M:%S’)

))

Seen\_names.add(company\_name)

Except ValueError:

Continue

Conn.commit()

Conn.close()

# Routes

@app.route(‘/’, methods=[‘GET’, ‘POST’])

Def lead\_list():

Page = int(request.args.get(‘page’, 1))

Per\_page = 5

Industry = request.args.get(‘industry’, ‘’)

Min\_revenue = request.args.get(‘min\_revenue’, ‘’)

Search = request.args.get(‘search’, ‘’)

Sort\_by = request.args.get(‘sort\_by’, ‘company\_name’)

Lead\_id = request.args.get(‘lead\_id’, ‘’)

Action = request.args.get(‘action’, ‘’)

Conn = sqlite3.connect(‘leads.db’)

Cursor = conn.cursor()

# Handle add new company

If request.method == ‘POST’ and action == ‘add’:

Company\_name = request.form.get(‘company\_name’, ‘’).strip()

Website = request.form.get(‘website’, ‘’).strip()

Contact\_email = request.form.get(‘contact\_email’, ‘’).strip()

Industry\_form = request.form.get(‘industry’, ‘’).strip()

Revenue = request.form.get(‘revenue’, ‘’)

If company\_name and industry\_form and revenue:

Try:

Revenue = float(revenue)

If revenue < 0 or not is\_valid\_email(contact\_email):

Return render\_template(‘index.html’, error=”Invalid email or revenue”, leads=None, lead=None)

Score = calculate\_score(revenue)

Cursor.execute(‘’’

INSERT OR IGNORE INTO leads (company\_name, website, contact\_email, industry, revenue, score, created\_at)

VALUES (?, ?, ?, ?, ?, ?, ?)

‘’’, (company\_name, website, contact\_email, industry\_form, revenue, score, datetime.now().strftime(‘%Y-%m-%d %H:%M:%S’)))

Conn.commit()

Except ValueError:

Return render\_template(‘index.html’, error=”Invalid revenue”, leads=None, lead=None)

Else:

Return render\_template(‘index.html’, error=”Missing required fields”, leads=None, lead=None)

# Handle edit lead

If request.method == ‘POST’ and action == ‘edit’ and lead\_id:

Company\_name = request.form.get(‘company\_name’, ‘’).strip()

Website = request.form.get(‘website’, ‘’).strip()

Contact\_email = request.form.get(‘contact\_email’, ‘’).strip()

Industry\_form = request.form.get(‘industry’, ‘’).strip()

Revenue = request.form.get(‘revenue’, ‘’)

If company\_name and industry\_form and revenue:

Try:

Revenue = float(revenue)

If revenue < 0 or not is\_valid\_email(contact\_email):

Return render\_template(‘index.html’, error=”Invalid email or revenue”, leads=None, lead=None)

Score = calculate\_score(revenue)

Cursor.execute(‘’’

UPDATE leads SET company\_name = ?, website = ?, contact\_email = ?, industry = ?, revenue = ?, score = ?

WHERE id = ?

‘’’, (company\_name, website, contact\_email, industry\_form, revenue, score, lead\_id))

Conn.commit()

Except ValueError:

Return render\_template(‘index.html’, error=”Invalid revenue”, leads=None, lead=None)

Else:

Return render\_template(‘index.html’, error=”Missing required fields”, leads=None, lead=None)

Return redirect(url\_for(‘lead\_list’))

# Handle CSV import

If request.method == ‘POST’ and action == ‘import’:

File = request.files.get(‘csv\_file’)

If file and file.filename.endswith(‘.csv’):

Seen\_names = set([row[1] for row in cursor.execute(‘SELECT company\_name FROM leads’).fetchall()])

Try:

Reader = csv.DictReader(file.read().decode(‘utf-8’).splitlines())

For row in reader:

Company\_name = row[‘company\_name’].strip()

If company\_name in seen\_names or not is\_valid\_email(row[‘contact\_email’]):

Continue

Revenue = float(row[‘revenue’])

If revenue < 0:

Continue

Score = calculate\_score(revenue)

Cursor.execute(‘’’

INSERT OR IGNORE INTO leads (company\_name, website, contact\_email, industry, revenue, score, created\_at)

VALUES (?, ?, ?, ?, ?, ?, ?)

‘’’, (

Company\_name,

Row[‘website’].strip(),

Row[‘contact\_email’].strip(),

Row[‘industry’].strip(),

Revenue,

Score,

Datetime.now().strftime(‘%Y-%m-%d %H:%M:%S’)

))

Seen\_names.add(company\_name)

Conn.commit()

Except (ValueError, KeyError):

Return render\_template(‘index.html’, error=”Invalid CSV format”, leads=None, lead=None)

Else:

Return render\_template(‘index.html’, error=”Invalid or missing CSV file”, leads=None, lead=None)

# Handle delete lead

If action == ‘delete’ and lead\_id:

Cursor.execute(‘DELETE FROM leads WHERE id = ?’, (lead\_id,))

Conn.commit()

# Show single lead details or edit form

If lead\_id:

Cursor.execute(‘SELECT \* FROM leads WHERE id = ?’, (lead\_id,))

Lead = cursor.fetchone()

Conn.close()

If not lead:

Return render\_template(‘index.html’, error=”Lead not found”, leads=None, lead=None)

Return render\_template(‘index.html’, lead=lead, leads=None, edit\_mode=action == ‘edit’)

# Filter, search, and sort leads

Query = ‘SELECT \* FROM leads WHERE 1=1’

Params = []

If industry:

Query += ‘ AND industry LIKE ?’

Params.append(f’%{industry}%’)

If min\_revenue:

Try:

Query += ‘ AND revenue >= ?’

Params.append(float(min\_revenue))

Except ValueError:

Pass

If search:

Query += ‘ AND company\_name LIKE ?’

Params.append(f’%{search}%’)

# Validate sort\_by

Sort\_by = sort\_by if sort\_by in [‘company\_name’, ‘revenue’, ‘score’] else ‘company\_name’

Query += f’ ORDER BY {sort\_by}’

Cursor.execute(f’SELECT COUNT(\*) FROM ({query})’, params)

Total\_leads = cursor.fetchone()[0]

Total\_pages = (total\_leads + per\_page – 1) // per\_page

Query += ‘ LIMIT ? OFFSET ?’

Params.extend([per\_page, (page – 1) \* per\_page])

Cursor.execute(query, params)

Leads = cursor.fetchall()

Conn.close()

Return render\_template(‘index.html’,

Leads=leads,

Lead=None,

Industry=industry,

Min\_revenue=min\_revenue,

Search=search,

Sort\_by=sort\_by,

Page=page,

Total\_pages=total\_pages,

Edit\_mode=False)

@app.route(‘/export’)

Def export\_leads():

Industry = request.args.get(‘industry’, ‘’)

Min\_revenue = request.args.get(‘min\_revenue’, ‘’)

Search = request.args.get(‘search’, ‘’)

Conn = sqlite3.connect(‘leads.db’)

Cursor = conn.cursor()

Query = ‘SELECT company\_name, website, contact\_email, industry, revenue, score FROM leads WHERE 1=1’

Params = []

If industry:

Query += ‘ AND industry LIKE ?’

Params.append(f’%{industry}%’)

If min\_revenue:

Try:

Query += ‘ AND revenue >= ?’

Params.append(float(min\_revenue))

Except ValueError:

Pass

If search:

Query += ‘ AND company\_name LIKE ?’

Params.append(f’%{search}%’)

Cursor.execute(query, params)

Leads = cursor.fetchall()

Conn.close()

Output = StringIO()

Writer = csv.writer(output)

Writer.writerow([‘Company Name’, ‘Website’, ‘Contact Email’, ‘Industry’, ‘Revenue’, ‘Score’])

For lead in leads:

Writer.writerow(lead)

Timestamp = datetime.now().strftime(‘%Y%m%d\_%H%M%S’)

Response = Response(

Output.getvalue(),

Mimetype=’text/csv’,

Headers={‘Content-Disposition’: f’attachment; filename=leads\_{timestamp}.csv’}

)

Return response

# Initialize database and populate leads

If \_\_name\_\_ == ‘\_\_main\_\_’:

Init\_db() # Always run to ensure schema is up-to-date

If os.path.exists(‘sample\_leads.csv’):

Populate\_leads()

App.run(host=’0.0.0.0’, port=8000, debug=True)

**Methodology**:

1. Analysis: Studied SaaSquatch Leads to identify gaps (e.g., no editing, limited filtering).

2. Design: Chose Flask for simplicity, single-file structure for minimalism, and SQLite3 for lightweight storage.

3. Development:

- Built `app.py` for all back-end logic, including database initialization and migration.

- Designed `index.html` with conditional rendering for list, detail, and edit views.

- Implemented features incrementally: CRUD, filtering, scoring, pagination, import/export.

4. Testing: Tested on Pydroid 3, ensuring mobile compatibility and error-free operation.

5. Optimization: Added deduplication, validation, and advanced styling within 5-hour limit.

**Conclusion**:

The Caprae LeadGen Tool delivers a robust, user-friendly solution for lead generation, addressing Caprae’s need for actionable, high-quality leads. Its minimal design (one back-end, one front-end file) ensures simplicity while offering advanced features like lead scoring, editing, and filtered exports. Built for Pydroid 3, it is responsive and error-free, overcoming previous SQLite3 issues. The tool aligns with Caprae’s MaaS model, enhancing acquisition workflows, and is extensible for future AI-driven enhancements.