

# Password Manager - Generate & Check passwords (Console)

---

**Problem** Passwords need to be dealt with locally to ensure best security practices.

**Scenario** The password generator solves the part of the problem where the user is dependant on third party password generators. The user is in need of locally generated passwords. Complementary to the existing gpassword manager a password generator is needed.

## User stories:

1. As a user, I want to see the list of commands.
2. As a user, I want to select generation options of the password.
3. As a user, I want to see the history of the generated passwords.
4. As a user, I want to get feedback on the passwords strength.

## Use cases:

- Show possible commands
- Generate password (possible options)
- Show password's strength
- Encrypt and decrypt passwords
- Access User and Password information (from `passwords.db`)
- Managing passwords

---

## Project Requirements

Each app must meet the following three criteria in order to be accepted (see also the official project guidelines PDF on Moodle):

1. Interactive app (console input)
2. Data validation (input checking)
3. File processing (read/write)

---

### 1. Interactive App (Console Input)

---

The application interacts with the user via the console. Users can:

- View the possible commands
- Generate passwords
- Manage users and passwords
- Receive a feedback on the strength of the passwords

---

### 2. Data Validation

The application validates all user input to ensure data integrity and a smooth user experience. This is implemented in `main.py` as follows:

- **Generate password (possible options):** When the user wants to create a password, the program checks if the input is a digit and within the valid menu range:

```
if not length.isdigit() or int(length) < MIN_LENGTH:
    print(f"⚠ Invalid password length. Password length must be at
    least {MIN_LENGTH} characters length and fully digit.")
    continue
```

This ensures only valid menu items can be generated.

- **Verify URL (password management):**

When the user wants to enter a platform link, the program checks if the input is a valid URL (must contain scheme and domain):

```
from urllib.parse import urlparse

parsed = urlparse(platform)
if not all([parsed.scheme, parsed.netloc]):
    print("⚠ Invalid URL. Please enter a valid link including scheme
    (e.g., https://example.com).")
    continue
```

- **Password management options:** Ensures only valid command options are accepted. Invalid arguments are handled by extending `argparse.ArgumentParser`, allowing customized error messages and cleaner output.

```
class PasswordManagerParser(argparse.ArgumentParser):
    def error(self, message):
        print(f"❌ Unknown command or argument: {message}")
        self.print_help()
        exit(2)
```

These checks prevent crashes and guide the user to provide correct input, matching the validation requirements described in the project guidelines.

---

### 3. File Processing

The application reads and writes data using files:

- **Input/Output file:** `passwords.db` — Contains the entries, one item per line in the format `username, platform, password`.

- The application interacts with this file depending on the command specified.
- The application writes data to file when needed.

## Implementation


### Technology

- Python 3.x
- Environment: GitHub Codespaces
- No external libraries

### Repository Structure

```
Password-Generator/  
├── main.py           # main program logic (console application)  
├── datastore.py      # SQLite handling  
├── cli_parser.py     # contains PasswordManagerParser  
├── help.txt          # help command list  
├── passwords.db      # database with passwords  
├── docs/             # optional screenshots or project documentation  
└── README.md         # project description and milestones
```

### How to Run

 Adjust if needed.

1. Open the repository in **GitHub Codespaces**
2. Open the **Terminal**
3. Run:


```
python3 main.py
```

### Libraries Used

- **os**: Used for file and path operations, such as checking if the menu file exists and creating new files.
- **argparse**: Used for parsing command-line arguments and handling user input with custom error messages.
- **sqlite3**: Used for database storage and management of password records.
- **urllib.parse**: Used for validating and parsing platform URLs.



These libraries are part of the Python standard library, so no external installation is required. They were chosen for their simplicity and effectiveness in handling file management tasks in a console application.

## Team & Contributions

 Fill in the names of all team members and describe their individual contributions below. Each student should be responsible for at least one part of the project.

Name	Contribution
Andrii Vlasov	Menu reading (file input) and displaying menu
Aaron Casula	Order logic and data validation
José Gédance	Invoice generation (file output) and slides

## Contributing

-  This is a template repository for student projects.
-  Do not change this section in your final submission.

- Use this repository as a starting point by importing it into your own GitHub account.
- Work only within your own copy — do not push to the original template.
- Commit regularly to track your progress.

## License

This project is provided for **educational use only** as part of the Programming Foundations module.

[MIT License](#)