PandaManager

Nadav Shamir - 323833038
Maya Raskin - 209372325

Introduction:

PandaManager is a cutting-edge password manager application designed to address the growing need for secure and convenient management of user credentials in an increasingly digital world. As cyber threats continue to evolve, the importance of robust password management cannot be overstated. Our Password Manager aims to provide users with a reliable and sophisticated solution, employing innovative approaches in both password storage and encryption.

The primary purpose of the Password Manager is to empower users to store, organize, and access their passwords in a highly secure manner. By utilizing a self-implemented password-saving method and a proprietary encryption algorithm, the application prioritizes the confidentiality and integrity of user data.

Unlike traditional password managers that may rely on established methods, this application adopts a self-implemented password-saving method. This method ensures that user credentials are not only securely stored but also shielded from emerging threats.

Recognizing the importance of user convenience, a secure browser extension is introduced. This extension seamlessly integrates with popular web browsers, offering users a frictionless experience in managing and auto-filling passwords. This integration ensures that security and usability are not mutually exclusive, providing a comprehensive solution for individuals seeking a balance between accessibility and protection.

Features:

- *Auto-Save*:
  The Auto-Save feature operates dynamically, detecting when a user enters new login credentials.
  The browser extension will iterate through the DOM, extract the information from the relevant inputs. As soon as the user successfully logs in, the Password Manager recognizes the redirection and prompts the user to save the credentials.
- *Auto-Complete*:
  The Auto-Complete feature operates contextually, recognizing saved credentials for specific hosts or websites. When a user navigates to a login page for a saved host, the Password Manager intelligently identifies the site and offers to auto-complete the login fields.
  The browser extension provides an automatic prompt when detecting a login page for a saved host. The user has the option to accept or decline the auto-completion suggestion.
  Specifically, the prompt contains a dropdown or a selection menu, allowing users to choose the appropriate login credentials for the specific account they want to access.

  The user will also have the flexibility to customize Auto-Complete settings based on their preferences, by enabling or disabling this feature for specific hosts.

- *Browser Extension*:
The browser extension will provide a user interface with a few separate pages/views:

1. Login Page & Registration Page
2. Credentials Viewer (default page):
   The default page serves as a centralized hub for accessing and managing passwords. Users can access passwords by selecting them, triggering an API call to retrieve the complete password, which is then displayed briefly.

   Functionality includes editing and adding credentials directly from this interface.

   If a password for the current host is stored, it will be prominently displayed at the top of the credentials list.

3. An additional feature enables users to create robust passwords via our interface, offering customizable options:
   - Inclusion of special character
   - Incorporation of capital letters
   - Addition of digits, with the ability to set a minimum number
   - Specification of character length range.

Algorithms & Security:

<u>Password splitting</u>
Upon user submission of a chosen password for storage, the password undergoes encryption before being divided into two distinct segments. Each segment is subsequently stored within separate regions, enhancing security measures by decentralizing sensitive data components. This approach ensures a higher level of protection against unauthorized access or breach attempts, as it disperses critical information across disparate locations.

<u>HTTPS</u>
The communication between the client and server is fortified through the utilization of HTTPS I/O (Hypertext Transfer Protocol Secure input/output). HTTPS ensures data integrity and confidentiality by encrypting the transmitted information, mitigating the risks associated with potential eavesdropping or data interception during transit.

<u>Secrets</u>
Our backend securely communicates with the database instances, requiring a client certificate (e.g., in .pem format) to establish connections. A .pem file serves as a standard format for storing cryptographic keys, certificates, and associated data. This .pem file will be stored within the chosen cloud provider's environment, utilizing dedicated secrets management options.

<u>Encryption algorithm</u>
The AES256 algorithm will be used in order to encrypt the passwords.
The Advanced Encryption Standard (AES) is a widely used symmetric encryption algorithm, with 256-bit key size being one of its variations. AES256 operates on blocks of data, dividing them into 128-bit blocks, and encrypts them using a 256-bit key. It employs a substitution-permutation network, consisting of multiple rounds of substitution and permutation operations, to achieve high-level security. AES256 is recognized for its strength and is commonly utilized in securing sensitive data across various applications and industries.

Architecture, Technologies & Frameworks:

- Client: Browser Extension:
  Our browser extension is developed using TypeScript and Angular 17, leveraging the MUI library for seamless styling. TypeScript ensures type safety and enhanced code organization, while Angular 17 offers a robust framework for building dynamic, scalable web applications. MUI library facilitates rapid development with its pre-designed components and responsive design capabilities, streamlining the UI development process. Together, these technologies provide a powerful foundation for delivering a feature-rich and visually appealing browsing experience.

- Backend:
  Our backend infrastructure is built with NestJS, offering REST API endpoints and fortified security measures.

  Two backend instances are deployed, operating independently without mutual awareness. A dedicated proxy server is utilized to facilitate communication with both backend instances.

- Database:
  MongoDB is used as our database.
  The main collections in the DB are:

  - Users:
    - *_id:* ObjectId
    - *first_*name: string
    - *last_name*: string
    - *email*: string
    - *master_password*: *string*
    - *devices:* string[]
    - *created_at*: ISODate
  - Credentials:
    - *_id*: ObjectId
    - *user_id*: ObjectId
    - *display_name*: string
    - *host*: string
    - *login*: string
    - *password*: string
    - *deleted*: boolean
    - *created_at*: ISODate

  Two database instances are employed, with each instance dedicated to a specific provider. Each database is configured with a private endpoint that corresponds to its respective provider.

- Proxy Server/API:
  A web server is employed to forward client requests to the appropriate backend. The server leverages Redis to manage pairs of sequential requests. Each pair shares a header containing a client-generated UUID, which serves as a key in Redis. The value stored in Redis includes the HTTP method, URL, query parameters, body, and the provider that handled the request.

Comparison to Google Password Manager

The favored choice among users for a password manager is Google's Password Manager, integrated within the Chrome browser. While it offers convenience and simplicity in usage and upkeep, Google's password manager compromises some security aspects by storing passwords locally; e.g, in Windows and Google Chrome, passwords are typically stored in the directory *C:\Users\YourName\AppData\Local\Google\Chrome\User Data\Default*. If you're signed in to your Google account, they may also be saved in the cloud.
The passwords are stored within a file named "Login Data," which is an SQLite database–a compact and lightweight database contained within a file. This file can be accessed using an SQLite browser. Upon inspecting the file on our local system, we observed that passwords are encrypted and stored without modification, while other data is stored without encryption.
This observation led us to also consider that it may not be entirely unreasonable for passwords not to be split on the server side, although definitive evidence cannot be provided.
Consequently, it may be susceptible to exploitation through browser vulnerabilities.
Our password manager provides the same features as Google's password manager but without the associated security risks. Passwords are exclusively stored in the cloud. For trusted devices, only the master password is required for each login, whereas for untrusted devices, both the master password and OTP authentication are requested.

Main Flow:

1. The user inputs their username and password for storage.
2. Upon submission, the credentials undergo encryption on the client-side:
   User passwords undergo hashing via *bcrypt* to generate robust encryption keys.
   Each user receives a unique salt from *bcrypt*, enhancing security by thwarting attempts at password decryption through reverse engineering.
3. Subsequently, the encrypted credentials are transmitted to the backend, where they are stored across two distinct regions without the backend having access to their actual content.
4. Users have the ability to retrieve their encrypted data via the client-side extension. When a password is requested to be displayed, an API request is sent, and the client undertakes the decryption process using their individualized salt.

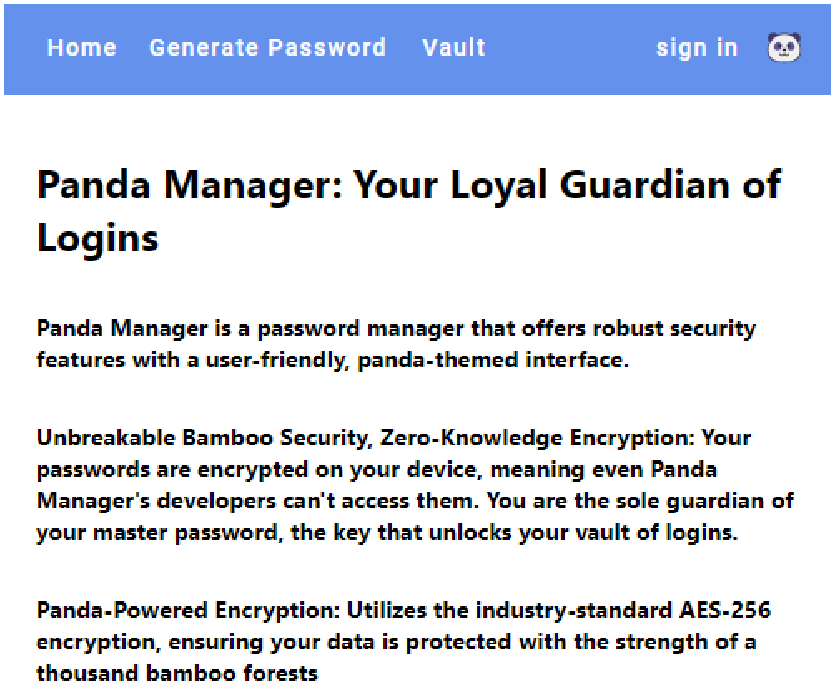
Chrome Extension

The extension consists of five pages：

- Register
- Login
- Home
- Generate Password
- Vault (must be logged in to enter)

Home

The home page, which is the default page the user sees when opening the extension, provides a brief explanation about the purpose and methods of the password manager.



Panda Manager: Your Loyal Guardian of Logins

Panda Manager is a password manager that offers robust security features with a user-friendly, panda-themed interface.

Unbreakable Bamboo Security, Zero-Knowledge Encryption: Your passwords are encrypted on your device, meaning even Panda Manager's developers can't access them. You are the sole guardian of your master password, the key that unlocks your vault of logins.

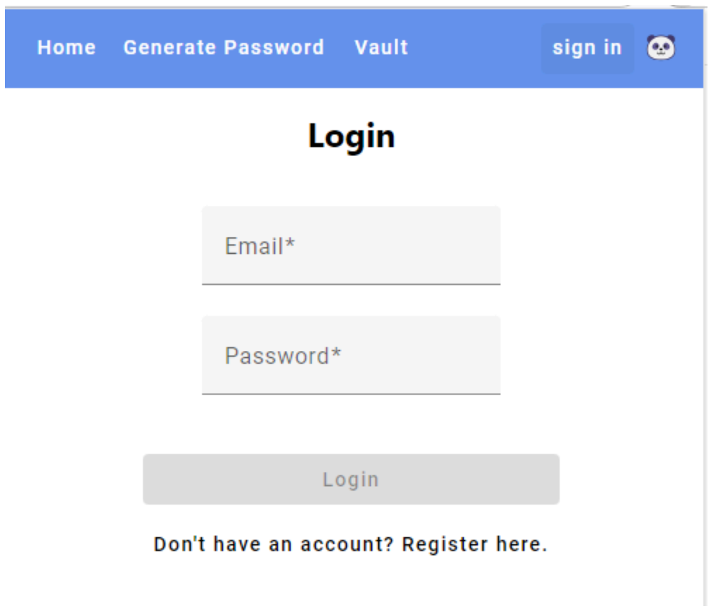
Panda-Powered Encryption: Utilizes the industry-standard AES-256 encryption, ensuring your data is protected with the strength of a thousand bamboo forests

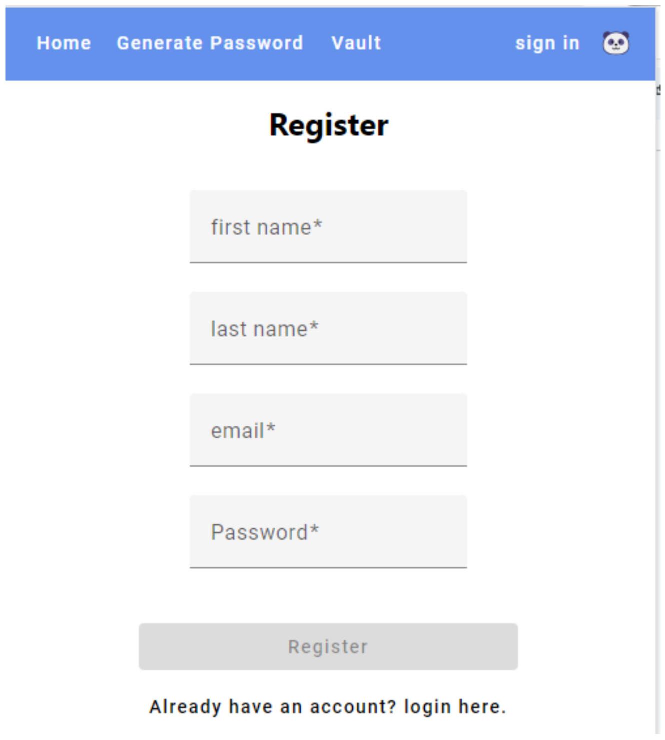
Register/Login

The registration process is enhanced with an OTP (One-Time Password) feature, ensuring that users verify their identity during sign-up. Upon registration, users are required to provide their basic information and receive an OTP, which they must enter to complete the process. This adds an extra layer of security, preventing unauthorized access.

For logging in, users submit their email and a password. The server then validates these credentials. If they match with a record in the database, the server generates a JWT (JSON Web Token) and sends it back to the client. The client then uses this token for following requests.

On the next visit, the presence of this token allows the extension to recognize and remember the user, providing a seamless login experience without requiring the user to re-enter their credentials. This approach not only enhances security but also improves user convenience by streamlining the authentication process. Overall, your login and registration system combines robust security measures with a user-centric design, making it both safe and efficient for users.





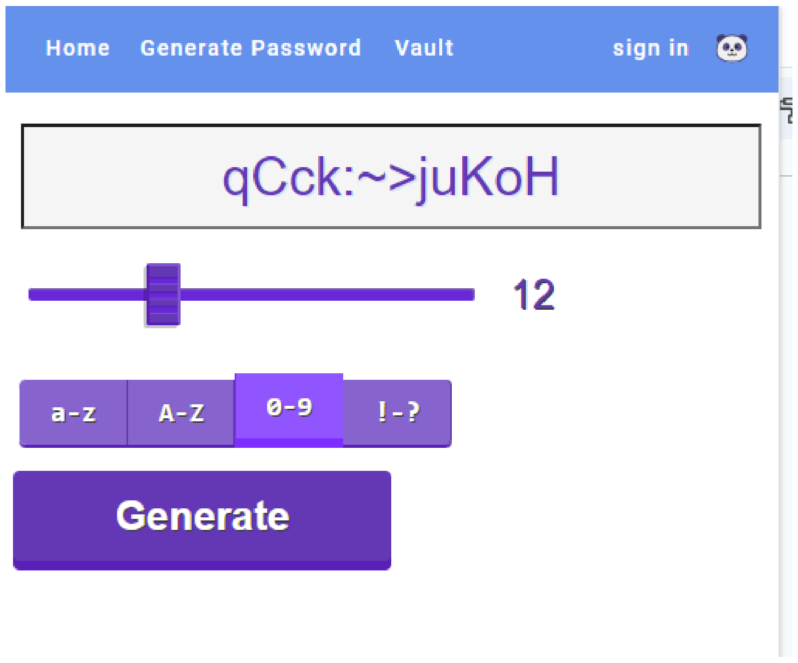
Generate Password

The "Generate Password" page is designed to help users create strong, customized passwords with ease. On this page, users select the desired password length using a custom-designed scroller, which offers a visually appealing and intuitive way to specify the number of characters.

Additionally, the page provides four options for character types to include in the password: lowercase letters, capital letters, numbers, and unique symbols (e.g., !, ?). Users can select any combination of these options to tailor the complexity and type of their password according to their security needs.

Once the desired length and character types are chosen, users can click the "Generate" button. This action generates a strong password based on the selected criteria and automatically copies it to the clipboard, allowing users to easily paste it wherever needed without manual copying.

This feature not only enhances security by encouraging the use of complex passwords but also improves user convenience by simplifying the password creation and copying process.



qCck:~>juKoH

12

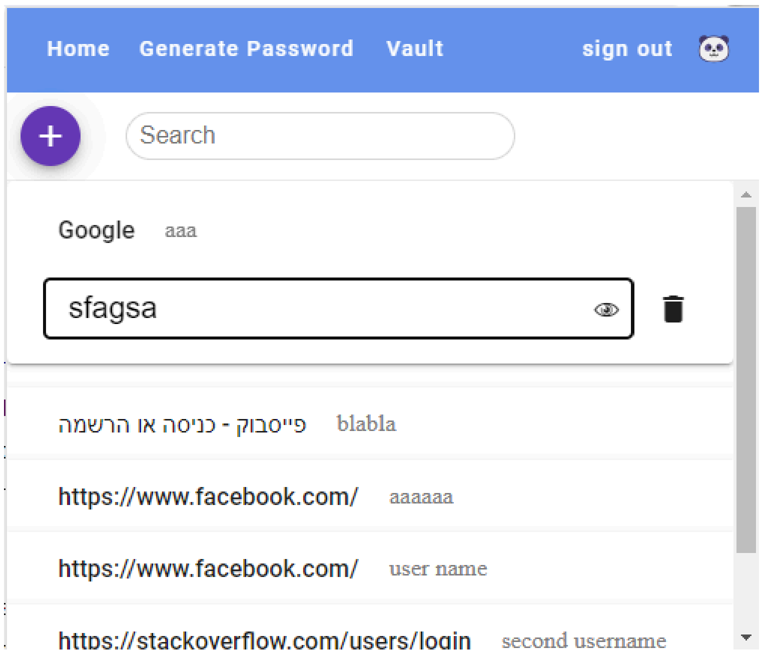
Vault

The "Vault" page serves as a central hub for managing all saved credentials securely and efficiently. Initially, this page displays a list of usernames and corresponding website URLs, but it does not show the passwords directly to maintain security. If a user wants to view a specific password, they must enter their master password, which is used to decrypt and display the password securely.

In addition to viewing passwords, the Vault page allows users to remove or update their saved credentials, providing flexibility in managing their accounts. This ensures that users can keep their credential list up-to-date and remove any outdated or unnecessary entries.

The page also includes an + button for saving new credentials. When adding a new password, the extension automatically injects the current page's URL into the host field. Users only need to fill in the username and password fields, making it quick and convenient to save new credentials.

Overall, the Vault page combines robust security features with user-friendly functionality, ensuring that users can manage their passwords and other credentials efficiently while maintaining a high level of security.



Google	aaa
sfagsa	
פייסבוק - כניסה או הרשמה	blabla
https://www.facebook.com/	aaaaaa
https://www.facebook.com/	user name
https://stackoverflow.com/users/login	second username

Backend

Code

The backend architecture is implemented in TypeScript utilizing NestJS. Adopting dependency injection as its primary design pattern, the backend is structured into modules, controllers, and services.

- **Controllers**: These components contain functions annotated with routing metadata, designating them as the handlers for incoming HTTP requests to specified routes.
- **Modules**: These define the dependencies, exports, and controllers associated with each module, ensuring cohesive organization and reusability of components.
- **Services**: Responsible for encapsulating the business logic, services typically interact with the database, performing the core operations required by the application.

This structured approach ensures a clear separation of concerns, enhancing maintainability and scalability of the backend system.

The backend offers protected routes, primarily for managing credentials. These protected routes are implemented using NestJS's Guards mechanism, ensuring that only authorized requests can access these sensitive endpoints.

DB Access & TypeORM

Our *MongoDB* databases reside in the cloud, employing secure communication through a key and certificate in PEM format. Each database is accessible solely through a private endpoint, guaranteeing that only the corresponding backend service has interaction privileges.

Furthermore, *TypeORM* is integrated into the backend to optimize the mapping of entities to their corresponding database collections, thereby improving overall functionality and data management efficiency. By simplifying operations such as querying, updating, and deleting records, as well as facilitating interaction with objects and classes, *TypeORM* enhances the development experience and fosters more robust database integration.

Deployment Process

As previously described, two instances of the backend are deployed, differing solely in their environment variables and secrets. The majority of these environment variables and secrets are managed in the VCS (GitHub).

To minimize the number of Docker images stored in the container registry (GitHub's container registry), environment variables and secrets are injected during the deployment phase rather than during the build process. This approach ensures efficient management and deployment of the backend using a single image.

The deployment process is initiated via GitHub Actions. Upon a push event on the *main* branch, a Docker image is built and subsequently pushed to the container registry. Following this, a rollout is executed on both providers.

Terraform will be employed to enhance Infrastructure as Code (IAC) practices and further refine the deployment process.

Proxy Server/API

Code

A web server is employed to forward client requests to the appropriate backend. The server leverages Redis to manage pairs of sequential requests. Each pair shares a header containing a client-generated UUID, which serves as a key in Redis. The value stored in Redis includes the HTTP method, URL, query parameters, body, and the provider that handled the request.

Upon receiving a request, the server checks if the corresponding key exists in Redis. If the key is found, the request is forwarded to the alternate provider. Once a successful response is received, the key is deleted from Redis. If the key is not found, the server randomly selects a provider, processes the request, and stores the relevant data in Redis.
When forwarding a request to the appropriate backend, the server copies the request headers, query parameters, body, and URL. Additionally, an *X-Forwarded-For* header is added with the client's IP address, and the host header is updated as well.

If the Time-to-Live (TTL) for a document in Redis reaches one minute, it indicates that the first request was handled successfully but the second request has not arrived or failed. In such instances, a rollback operation is executed on the respective provider, with varying functions (or routes) implementing distinct rollback procedures as needed. A revert also occurs if any failure happens during the handling of the second request before the key is deleted.