Design Challenge Project Report

CSE 508: Network Security

Prabuddha Kumar (SBU ID: 112076553)

Title:

DataOwn: Separating data from web based services

Aim:

The aim of the project was to demonstrate that it is possible to have applications where the data storage is separate from the application service. This would prevent service providers from accessing the user's personal data.

For the project, I have developed a browser extension that injects user data into the website template provided by the service provider. This allows the user's data to be stored where the user wants, whether on the system, or on Firebase or some other cloud based service. This effectively shifts the ownership of the data back to the user, as should be.

Problem:

Most applications these days run on a model where a user's data is stored on the application's servers. The privacy practices of these companies are a cause of concern, who do not wish for their data to be on the servers of a third party, potentially being misused or sold. The recent scandals surrounding Google, Facebook, among other companies, fuels the argument against the traditional data storage methodology.

A number of services (Inrupt, Elasto etc) are working towards creating a paradigm of applications, where separating a user's data from the core service is the norm. For example, in the case of Inrupt, users can build applications on a platform, where data is not stored in the application's servers, leaving users with the flexibility of storing data elsewhere. The real challenge is that this solution required a rethink of the application architecture, with limited to no backward compatibility with the current style of applications.

My project showcases a proof of concept that demonstrates the ease with which one can run applications that do not rely on service providers having access to the users' data. The data belongs solely to the user who should have autonomy to store them wherever they wish.

The use cases enlisted below are some of the cases that can be tackled with the approach used:
- Calendar Applications
- User Contact Storage
- Personal Management Applications like Trello
- Health and Fitness application

These applications do not require data to be stored on the service provider's servers, and can be severely misused (especially health data) if entrusted with rogue third parties. This data can be stored wherever the user wants, provided it is in the same format as can be used by the application, and can be rendered inside the browser without having to come in contact with the application. In effect, we will have complete separation of the data and the application.

Solution Setup:

The setup comprises of three components:

- The website code (html+css):
  - The website code is loaded from a website server, locally hosted on Apache, it sends the layout and styling file, but no javascript.
  - The rationale for this is to prevent any means for the service provider to get access to the data.
  - As a next step, one could implement ways to prevent access to the data while having javascript files sent from the service provider for enhanced navigation etc.

- The backend code (NodeJs+ExpressJs+MongoDb):
  - The server code serves data requests for adding a card, updating an existing card and deleting existing cards.
  - For this project, I have chosen NodeJS, Express and MongoDB, but the important point here is that the users have complete autonomy over their data, and can store as per their preference in such a paradigm.

- The extension code (the javascript for injecting data)
  - The extension code is responsible for identifying the right website and plugging in the data from the server, stored as the user wants.
  - The extension is written for Google Chrome
  - It identifies whether the website currently active on the user's browser is the one that we are targeting. It then injects the HTML through javascript, assigns listeners and handles the calls to the local server.

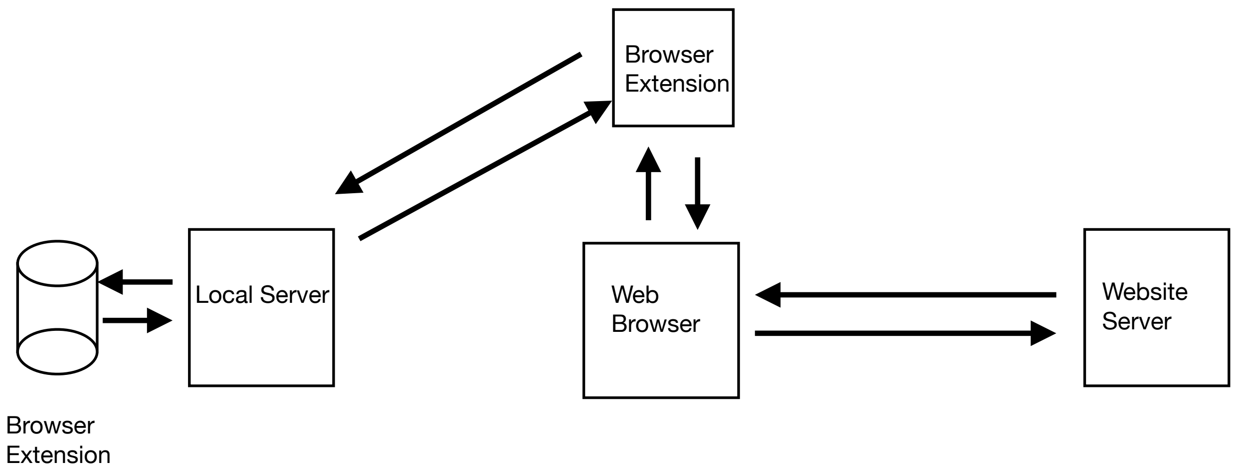


Fig 1. System Design

The project in action:

A) HTML/CSS is loaded from the website server

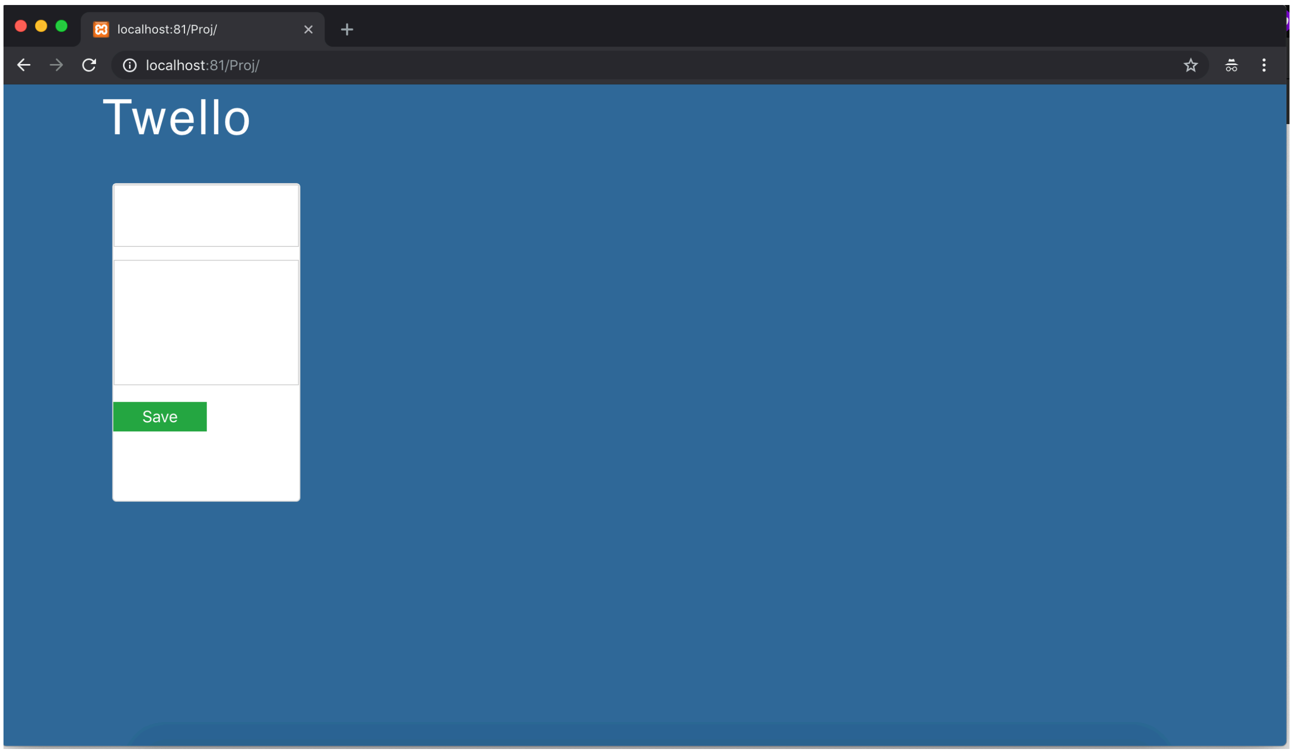
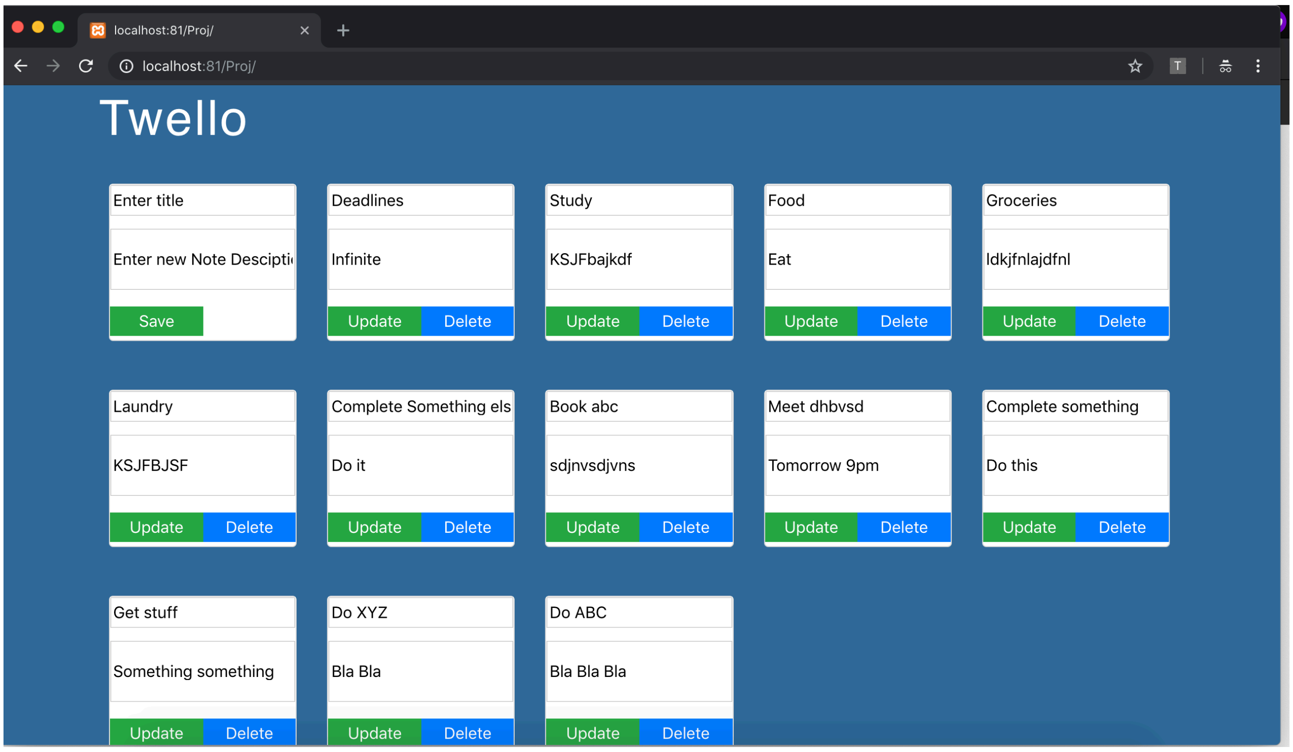


Fig2. Just the skeleton of the server is loaded.

B) The Browser Extension loads the cards



Enter title | Enter new Note Descipti
Deadlines | Infinite
Study | KSJFbajkdf
Food | Eat
Groceries | ldkjfnlajdfnl
Laundry | KSJFBJSF
Complete Something els | Do it
Book abc | sdjnvsdjvns
Meet dhbvsd | Tomorrow 9pm
Complete something | Do this
Get stuff | Something something
Do XYZ | Bla Bla
Do ABC | Bla Bla Bla

Fig.3 Loaded Content

The browser generates multiple cards and stacks them.
Note that the first card is the card in which to add new entries. All the other cards can be updated or deleted.

Design Decisions:

- **Extension**: A chrome browser extension was chosen given the ease with which the implementation can be generalized to multiple websites. The extension would be able to note if a website is compliant with the design standard and inject the relevant javascript to the layout and populate the data. The ability to passively be present and populate the data with no hassle makes the idea of an extension alluring.
- **NodeJS+ExpressJS+MongoDB**: Ease of development and deployment.
- **Card Like Website:** Simpler to convey the proposed idea by creating a website where it makes sense to have the data locally.

Limitations of the approach:

The following types of applications are beyond the scope of the proposed approach, because of their need for personal data in order to fulfill their service:

- Movie booking portals
- Travel booking portals (www.booking.com)
- Online shopping portals (www.amazon.com)

Recommendation systems are difficult to implement if the flow of data to companies is starved off.

- A potential solution: Sending anonymized data via opt-in with the users can solve this problem without compromising privacy. It is important to note that the anonymizing should be done via a trusted third party.

Future Scope:

- As a next step, one could implement ways to prevent access to the data while having javascript files sent from the service provider for enhanced navigation etc.
- One can implement a layer between the extension and the local server such that the layer would provide a uniform API for CRUD (Create, Read, Update and Delete), allowing for a uniform interface whereby the users can store their data wherever they wish.

References:

Used and modified the following code:

https://github.com/hayjay/node-api