Dataccount System Overview

Problem statement

Many companies maintain databases of personal information about individuals, which they use for their own purposes:

- Web sites like Facebook, Amazon, and Google monitor your online activities, then sell the information to advertisers for their own profit, with no compensation to you.
- Roomba's robot vacuums make a map of the interior of your home and send it back to the company. Roomba uses it to improve their robot algorithms, again with no compensation.
- Amazon keeps a permanent database of your personal home address and credit card information, which they can use however they see fit.
- Doctors keep your sensitive medical history in their own databases. You have no control over it.

Thus, the first problem is that information about yourself is stored by many companies, who promise to keep it safe, but you have no concrete way of verifying it. The companies may accidentally expose information which is private or embarrassing, or which allows other people to impersonate you. They can make money from your information without compensating you. You have to trust them.

The second problem is for companies. They are exposed scandal, lawsuits, etc. if they lose their customer's information to hackers.

Proposed solution

This document proposes a combination of computer systems and laws that would:

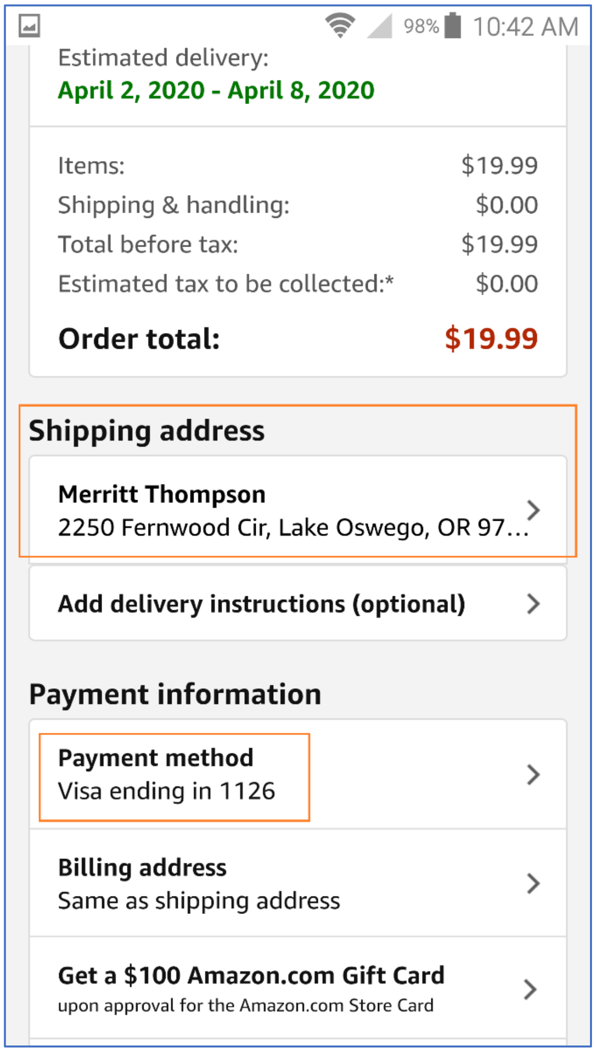
A. Allow companies to access personal information under the control of the individual.
B. Allow individuals to track accesses to their own personal information.
C. Prevent companies from storing personal information.

A. Allowing access under individual control — Example

When I order something on Amazon today, they show me a screen that contains my address and credit card, as shown on the left. This illustrates how Amazon maintains my personal information in their private database.

The image on the right shows the same screen after Amazon has upgraded to use this system. All personal information is gone. Instead there is only a web address for a “dataccount” (“data” + “account”). This illustrates how Amazon no longer keeps my personal information. They get it from my dataccount, which is at “https://merrittmarcthompson-dataccount”:

Today


Estimated delivery:
April 2, 2020 - April 8, 2020

Items:	$19.99
Shipping & handling:	$0.00
Total before tax:	$19.99
Estimated tax to be collected:*	$0.00
Order total:	**$19.99**

Shipping address

Merritt Thompson
2250 Fernwood Cir, Lake Oswego, OR 97…

Add delivery instructions (optional)

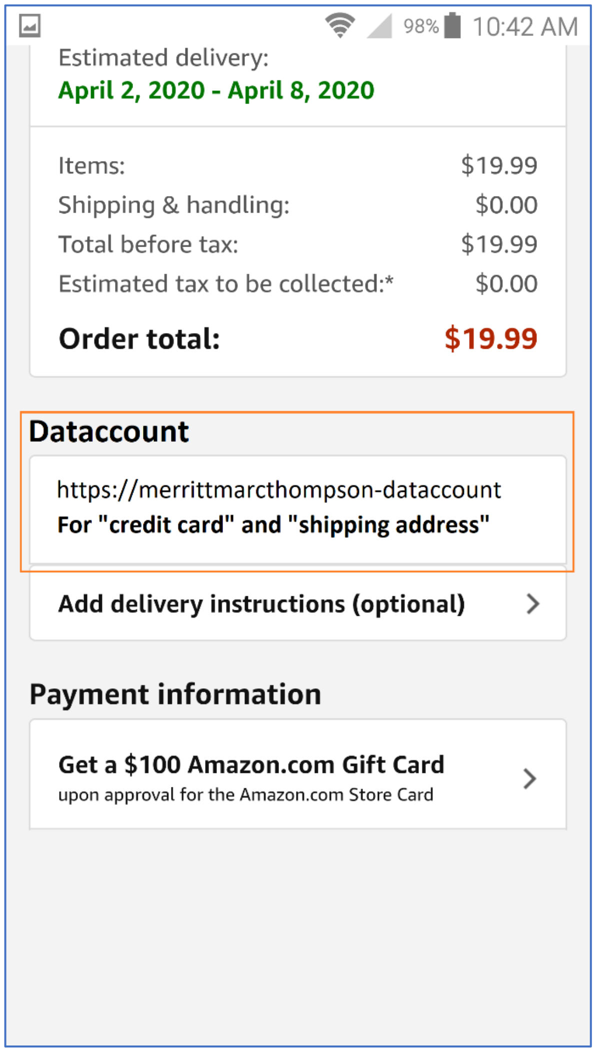
Payment information

Payment method
Visa ending in 1126

Billing address
Same as shipping address

Get a $100 Amazon.com Gift Card
upon approval for the Amazon.com Store Card

Upgraded with Dataccount


Estimated delivery:
April 2, 2020 - April 8, 2020

Items:	$19.99
Shipping & handling:	$0.00
Total before tax:	$19.99
Estimated tax to be collected:*	$0.00
Order total:	**$19.99**

Dataccount

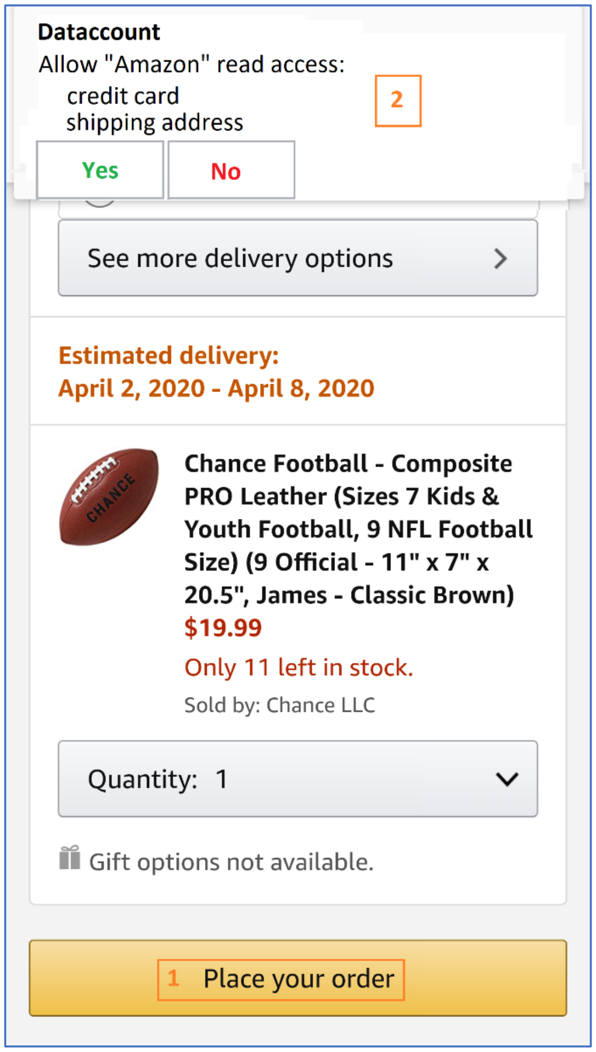
https://merrittmarcthompson-dataccount
For "credit card" and "shipping address"

Add delivery instructions (optional)

Payment information

Get a $100 Amazon.com Gift Card
upon approval for the Amazon.com Store Card

When I click the Amazon "Place your order" button (1 below), Amazon tries to access my dataccount. That causes a push notification from the Dataccount software on my phone (2), which lets me approve the access. This allows Amazon to bill my credit card and print a shipping label without permanently storing my personal data:



Dataccount
Allow "Amazon" read access:
credit card
shipping address

Estimated delivery:
April 2, 2020 - April 8, 2020

Chance Football - Composite PRO Leather (Sizes 7 Kids & Youth Football, 9 NFL Football Size) (9 Official - 11" x 7" x 20.5", James - Classic Brown)
$19.99
Only 11 left in stock.
Sold by: Chance LLC

Quantity: 1

Gift options not available.

B. Tracking access to personal information

When a company accesses my data, my dataccount software logs the access and its purpose. You can view the accesses in the dataccount management web UI.

Access date/time	Accessor	Data Item	Access	Allowed?	Purpose
4/7/2025 19:03:27	www.amazon.com	credit card	Read	Yes	Purchase $19.99 football
4/7/2025 19:03:27	www.amazon.com	shipping address	Read	Yes	Print delivery label

C. Restricting storage of personal information

Thus far, this system gives external entities a way to access necessary information without storing it, but it doesn't stop them from doing so anyway. They can still access the information from the dataccount, then also save it privately for their own use or profit.

This would be prevented by a new law that would criminalize storage of other people's personal data for purposes other than the stated ones.

In the example above, if Amazon got your shipping address from your dataccount, then used it for any purpose other than printing a delivery label (for example, if they sold it to a mass-mailing company), they would have committed a crime.

The key concept

This law would criminalize what is now an extremely common practice. You might think that companies would oppose this restriction, but it is more likely that they would ultimately approve of it, because as long as they follow the rules, they are completely absolved of liability for individuals' data losses.

Thus, this isn't a magical technological system for tracking information across the web. Instead, the goal of the technology and law is simply to make it possible to create a new social norm:

> You never store individuals' information. Instead, you always request the information directly from the individual, who provides it only at their own discretion.

About compensation

In this scheme, your Roomba vacuum could make a floor plan of your house, then send it directly to your dataccount (sending it anywhere else would be illegal). Roomba could then buy your floor plan from you. I'm not sure if that would make you much money, but Roomba might give customers perks of some kind for it. You could also sell it to researchers other than Roomba, since it is your data, not Roomba's.

Another example like this would be selling medical data to researchers. If you have a rare or interesting condition, you might get a reasonable amount of money for your medical records.

Automatic approval or denial

Automatic approval allows the owner to specify that certain accessors can automatically access certain data items without manual owner approval, because the owner trusts the accessor and is tired of manually approving their requests. For example, you could fill in a form like this in the owner UI:

Automatic approval

For accessor:	www.amazon.com
Data items:	Amazon Shipping Address Amazon Credit Card Number
Restrictions:	Up to 5 accesses per month

Implementation overview

Dataccount location

A dataccount can go on any server that is on the web, so long it supports the dataccount web API. Thus, you could have your own server locked in a closet in your house. Or you could contract with a “data bank” company that provides encrypted dataccount services for many individuals.

Data format

Conceptually, personal data would be stored in a *data items* table in the form of key/value pairs, for example:

"Home address"="Jane Smith, 123 Shady Lane, Tulsa, Oklahoma, 74101"

Nondisclosure of data in keys

Imagine that you want to send a diamond ring to your mistress Sheila's home address. Further, you already have a data key called "My mistress Sheila's home address". So, you might fill out a field like this on the Amazon web site:

Shipping Address Data Key: `My mistress Sheila's home address`

This would be less than ideal, as it would disclose personal information. To prevent this, you could create an *alias* in your dataccount:

"Amazon shipping address"=@"My mistress Sheila's home address"

This means, “Whenever someone asks for ‘Amazon shipping address’, give them ‘My mistress Sheila’s home address’ instead”.

Amazon would default their form to the following, so you probably wouldn’t have to enter anything:

Shipping Address Data Key: `Amazon shipping address`

APIs

There would be two API sets:

- One API set is for individuals (“owners”) to manage their own personal information. This includes functions that support the on-the-fly push notifications for Allow and Deny illustrated previously.
- One API set is for companies (“accessors”) to access personal information of others. This is just two functions, one to read and one to write.

Owner authentication

Owners can be authenticated for the owner API using HTTP Basic authentication or OAuth. Encrypted HTTPS transport must be used for HTTP Basic to work, as the password is merely base-64 encoded.

Accessor authentication

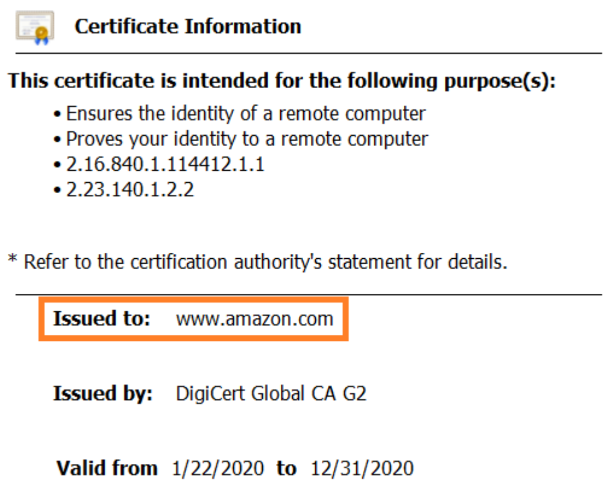
The most important thing for the accessor API is to authenticate the accessor. For example, if someone tried to access your dataccount, said they were “Amazon”, and the dataccount simply believed it without authentication, anyone could impersonate Amazon and get your information.

Transport Layer Security (TLS), the basis of the HTTPS communication protocol, normally requires the server to prove who they are to the client before any data can pass between them. The server does this by using a secret private key to encrypt some data. If the client can successfully decrypt the data using the public key, it proves to the client that the server is who they claim to be.

But that process is the opposite of what this system requires. In this system the server (the dataccount) must obtain proof that the client (the accessor, ex. Amazon) is really who they claim to be, not the other way around.

Fortunately, there is a "two-way" variant of TLS which provides exactly this function. It both verifies that the server is who they claim to be to the client (as normal) and the client is who they claim to be to the server (needed for this system).

As an example, here's the TLS certificate for Amazon. The "Issued to" value becomes the unique name for the accessor:



Certificate Information

This certificate is intended for the following purpose(s):

- Ensures the identity of a remote computer
- Proves your identity to a remote computer
- 2.16.840.1.114412.1.1
- 2.23.140.1.2.2

* Refer to the certification authority's statement for details.

Issued to: www.amazon.com

Issued by: DigiCert Global CA G2

Valid from 1/22/2020 **to** 12/31/2020

Because TLS does the verification, the server only receives verified requests and needs to do no further validation.

Push notifications and state-free web API flow

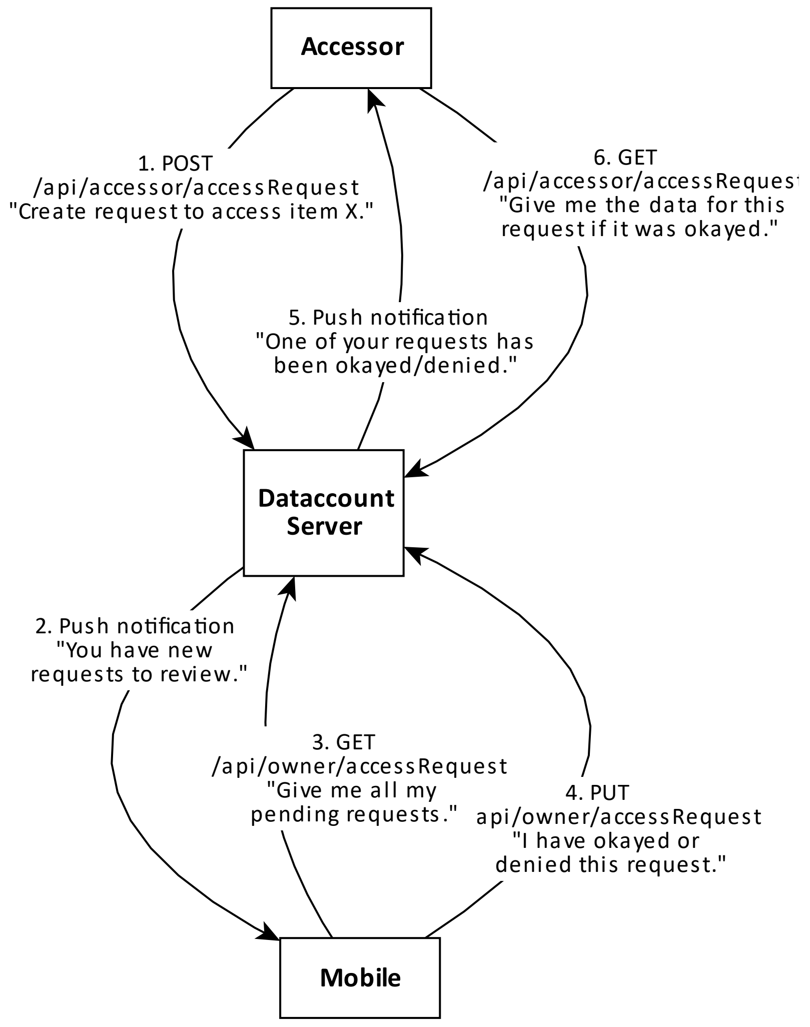
Ideally, when a company contacts your dataccount web API to access data, your server should immediately notify your mobile device for approval. This can be implemented using a "service worker" on the mobile device via JavaScript, which always runs and waits for such notifications.

But even so, the system is designed to not rely on push notifications. A typical use case is like this, with Amazon as an example:

1. Amazon wants to get your shipping address, so it requests it from your server. But it doesn't wait for the data to return, as it might take any amount of time to get the authorization from the owner.
2. The next step for the server would be to make a push notification to the owner's phone, telling you that you have something to approve, but this may or may not get through.

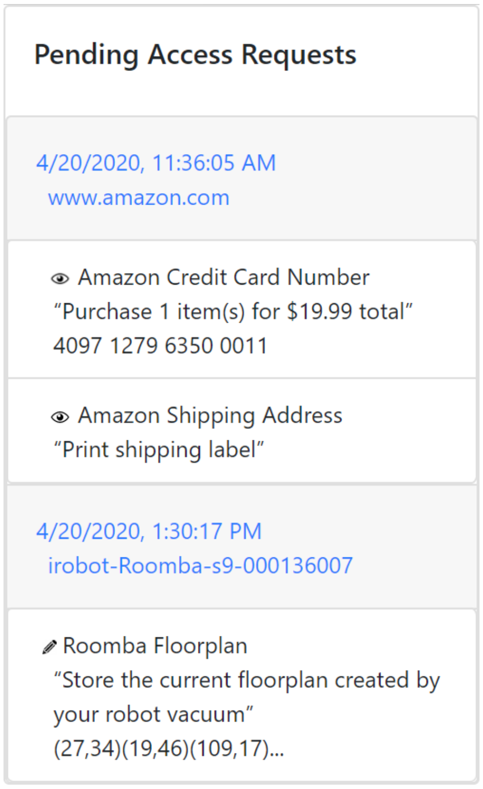
3. Next, at some point, either because the owner did see the notification or because they happened to check your pending requests, they see Amazon's request and tell the server to okay it.
4. Again, as the next step, the server may or may not be able to send a push notification (or web hook) to Amazon, telling them that they have an okay.
5. Finally, at some point, because of a notification or because they check periodically, Amazon requests the okayed address data.

The following diagram show more details of how the APIs work in the ideal case. In typical REST API form, everything is mediated by a resource, in this case, the Access Requests table. Since all state is stored in the table, the system still works even if this ideal sequence of events doesn't happen. For example, there can be a server crash right after step 1 (POST) or step 4 (PUT). When the server restarts, the system will carry on based on the table data as though nothing happened:


Accessor
1. POST /api/accessor/accessRequest "Create request to access item X."
6. GET /api/accessor/accessRequest "Give me the data for this request if it was okayed."
5. Push notification "One of your requests has been okayed/denied."
Dataccount Server
2. Push notification "You have new requests to review."
3. GET /api/owner/accessRequest "Give me all my pending requests."
4. PUT api/owner/accessRequest "I have okayed or denied this request."
Mobile

Example use case coding

A common use case would be “The owner can approve or deny pending access requests”. Here’s a simple UI for it. It shows a list of all the access requests waiting for approval or denial. It uses an “accordion” structure. It has the time of the request and the accessor as the “toggle” lines. When you click the toggles, they expand to show the collapsed details of what is requested. Some information may still need to be filled out by the owner (ex. the Amazon Shipping Address). Some information is to be read, others to be written (ex. the Roomba Floorplan):



Pending Access Requests

4/20/2020, 11:36:05 AM
www.amazon.com

Amazon Credit Card Number
“Purchase 1 item(s) for $19.99 total”
4097 1279 6350 0011

Amazon Shipping Address
“Print shipping label”

4/20/2020, 1:30:17 PM
irobot-Roomba-s9-000136007

Roomba Floorplan
“Store the current floorplan created by your robot vacuum”
(27,34)(19,46)(109,17)...

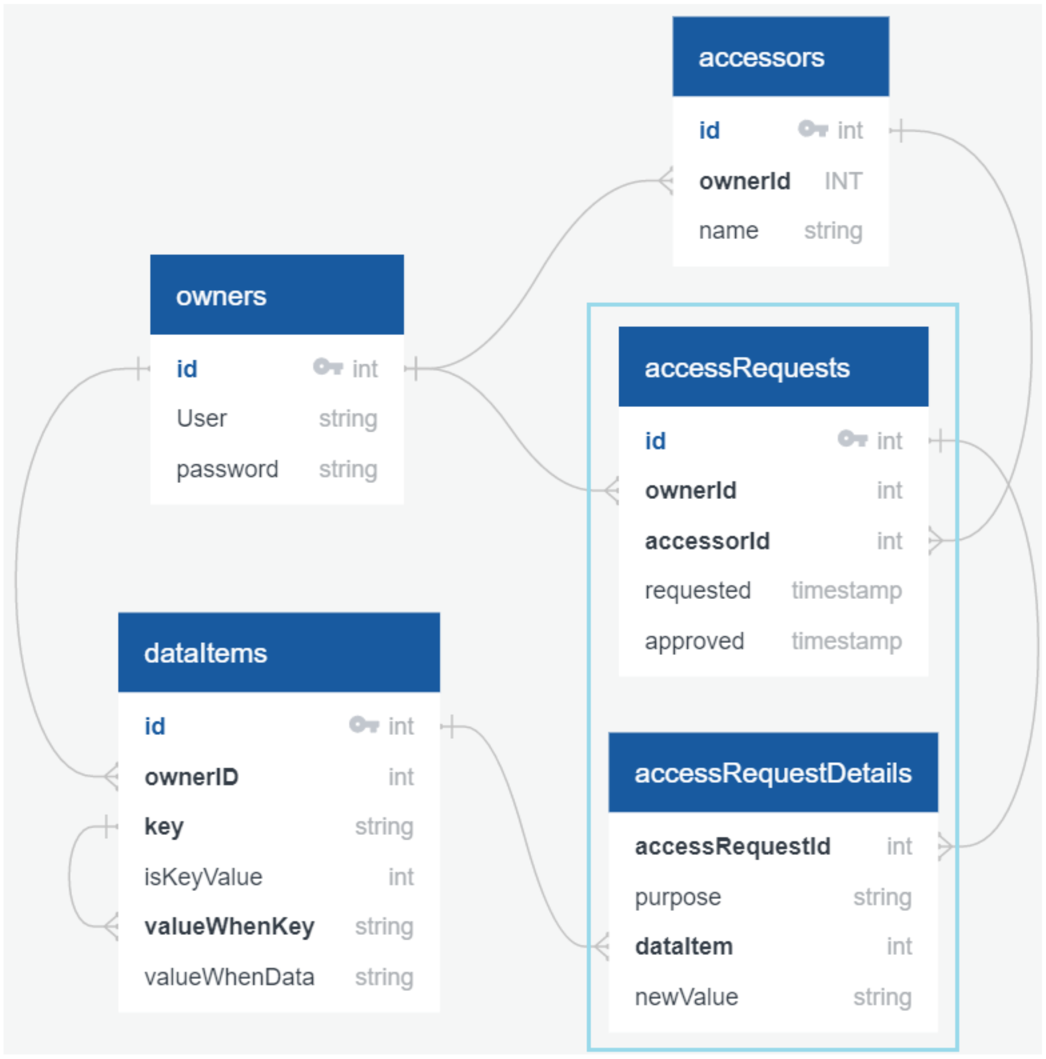
If the dataccount server returned the JSON below, it would be easy to code the UI, as this data format mimics the UI layout:

```
[
  {
    accessor: "www.amazon.com",
    requested: new Date("2020-04-20T11:36:05"),
    items: [
     {
       key: "Amazon Credit Card Number",
       purpose: "Purchase 1 item(s) for $19.99 total",
       value: "4097 1279 6350 0011",
     },
    {
       key: "Amazon Shipping Address",
       purpose: "Print shipping label",
     }
    ]
  },{
    accessor: "irobot-Roomba-s9-000136007",
    requested: new Date("04/20/2020 13:30:17"),
    items: [
      {
        key: "Roomba Floorplan",
        purpose: "Store the current floorplan created by your robot vacuum",
        value: "(27,34)(19,46)(109,17)...",
        newValue: "(27,35)(20,46)(110,19)...",
      }
    ]
  }
]
```

Here’s some simple React-Bootstrap code to get the JSON shown from the dataccount server via REST API and produce the accordion from it:

```
export default function OwnerScreen(props) {
  const [pendingAccessRequests, setPendingAccessRequests] = useState([]);
  useEffect(() => callWebApi(
    'GET',
    '/api/owner/accessRequests/pending',
    wrapper => setPendingAccessRequests(wrapper.pendingAccessRequests)
  ), []);
  return (
    <Accordion>
      {pendingAccessRequests.map((pendingAccessRequest, index) => (
        <Card key={index}>
          <Card.Header>
            <Accordion.Toggle as={Button} variant="link" eventKey={pendingAccessRequest.id} className=
"text-left" style={indentNextStyle}>
              {new Date(pendingAccessRequest.requested).toLocaleString()}<br />
              {pendingAccessRequest.accessor}
            </Accordion.Toggle>
          </Card.Header>
          <Accordion.Collapse eventKey={pendingAccessRequest.id}>
            <Card>
              <ListGroup variant="flush">
                {pendingAccessRequest.items.map((pendingAccessRequestItem, index) => (
                  <ListGroup.Item className="text-left" style={indentNextMoreStyle} key={index}>
                    <ReadWriteIcon read={pendingAccessRequestItem.newValue == undefined} />
                    {pendingAccessRequestItem.key}<br />
                    “{pendingAccessRequestItem.purpose}”<br />
                    {pendingAccessRequestItem.value}
                  </ListGroup.Item>
                ))}
              </ListGroup>
            </Card>
          </Accordion.Collapse>
        </Card>
      ))}
    </Accordion>
  );
}
```

I originally coded the server using a MySQL relational database. The relational database schema is conceptually like the following. The JSON denormalizes the Access Requests and Access Request Details tables into one JSON document (blue box):



It isn’t that easy to convert the two normalized tables directly into one denormalized JSON document using MySQL. But it was simple using a NoSQL database like MongoDB, which stores the data in the denormalized JSON format:

```
app.get(
  '/api/owner/accessRequests/pending',
  async (req, res) => {
    // The owner wants to see the access requests that need their approval.
    try {
      // Experience has shown that the db may not be ready yet the first time, so there may be a pause
 and something else may happen for a while if this is the first GET.
      db = await mongoClient.db("dataccount");
      // Make an array of the access requests that are pending approval.
      const accessRequestsCursor = db.collection("accessRequests");
      const pendingAccessRequests = await accessRequestsCursor.find({ approved: { $exists: false } }).
toArray();
      // Add a name to the array, just in case we want to add more results with different names to thi
s API later:
      //   { pendingAccessRequests: [...the pending access requests...] }
      const wrappedPendingAccessRequests = { pendingAccessRequests: pendingAccessRequests };
      res.status(200);
      const json = JSON.stringify(wrappedPendingAccessRequests);
      res.send(json);
    } catch (error) {
      console.error(error);
      // This should have worked. If it didn't, it seems like it's our (the server's) fault, i.e. 500.
      res.status(500);
      res.send();
    }
  });
```

The downside of the denormalized implementation is that some queries that would be easy in SQL (like getting a list of all companies that have asked what my age is) are difficult.