LECTURE NOTE 6 [CSC 421]

BUILDING A WEB APPLICATION (APP) CONT'

5.2.7 **Step 7 – Seek early validation**

You now have a beautiful wireframe/prototype that visually describes your web app. It is time to show your beautiful prototype to the world.

At this stage, we want constructive feedback. Simply asking your friends if they would use your new web app is not enough. You should start with a small number of representative users. Visit your target market's forums, verify the problem with them, and present your solution. Try to build a rapport with these representatives as they could become your customers.

I like to use this stage to test my sales pitch - the ultimate tokens of validation are pre-launch sales. Takes notes that keep a record of feedback. The learning from these meetings will help direct the development of your MEP (Minimal Excellent Product).

Before developing the web app, I would advise you to take note of the following tips:

1. Attempt to get a small section of your app fully working.
2. At the start - expect things to change a lot as you learn and discover what you have not thought about.
3. Spend time learning your tools.
4. Do not avoid issues that need to be fixed.

5.2.8 **Step 8 – Architect and build your database**

So, we know roughly our web application's functionality, what it looks like, and the pages required. Now it is time to determine what information we will store in our database.

DATABASE

A database is simply a collection of data. Data can be stored on a disk, in memory on a server, or both. You could create a folder on your hard drive, store a few documents, and call it a database. A Database Management System (DBMS) is a system that provides you with consistent APIs to (most commonly):

- Create databases, update and delete databases
- Read and write data to databases
- Secure access to a database by providing leveled access to different areas and functions

What data you need to store and what your users need to do, will determine the type of database required to run your web app.

Database Types

There are many types of databases for many different purposes. A web app will most commonly use one of the following:

1. **SQL**

You should use a SQL database if your data is very relational. Your data is relational if you have multiple, well-defined record types that have relationships between them.

For example, a “Customer” may have many “Invoices” stored against their record. Typically, you would create a customer table and an invoice table - which could be linked together by “Foreign Key” columns. E.g., Customer.Id = Invoice.CustomerId.

SQL databases have an extremely powerful query language that allows you to present your data in all sorts of useful ways. They have been around for decades, are very well understood, and are usually a safe choice. MySQL, Postgresql, and Microsoft SQLServer are some of the most common - along with many more modern offerings.

The downside of SQL databases is that you must declare all your tables and columns up front. There can be a lot of overhead to manage. If you have never used one before – you are in for a pretty steep learning curve. However, there are plenty of learning resources available, and it is always a great skill to have.

2. **DOCUMENT DATABASE**

You should use a document database if your data is not very relational. Document databases store “documents”. Each record in your database is simply a big blob of structured data - often in JSON format.

If you need to store relationships between your records, you will have to write code to manage this yourself. However, many other aspects of using document databases are much simpler. Your database can be “schemaless” - meaning that you do not have to declare your records’ definitions upfront.

Generally speaking, the bar to entry to a document database is much lower. They also tend to be much more scalable than SQL databases. They usually offer some querying capabilities, although sometimes not as powerful as SQL. Examples of document databases are MongoDb, CouchDb, Firebase (serverless), and Dynamo Db (AWS).

Decide how to segregate your data

Each of your clients has its own, private dataset. One of the worst things that can happen to your app is for one client’s data to be seen by another client. Even if there is only a small amount of non-sensitive leaked data, and no damage is done, an event like this will massively erode trust in the security of your app. You must architect a solid strategy for segregating your clients’ data to make sure that this never happens.

Broadly speaking, you have two options - *Physical Separation and Logical Separation*.

Physical separation

Every one of your clients has a separate database (although could share a database server with others). This makes it much more difficult to make a mistake that leads to data leakage.

Pros:

- ❖ Most secure
- ❖ More scalable

Cons:

- ▪ Managing, maintaining and upgrading are more complex
- ▪ Query all your clients' data together is more difficult

Logical separation

All of your clients are stored in one giant database. Every time you need to get data for a single client, you must remember to include a filter for the client. E.g., 'select' from customers where customerClientId = "1234"

Pros:

- ❖ Easier to get started
- ❖ Easier to maintain and upgrade
- ❖ Can easily query all your client's data with one query

Cons:

- ▪ Easy to make a mistake that will result in a data breach
- ▪ More difficult to scale

5.2.9 **Step 9 – Build the frontend**

In reality, you will build your backend and frontend at the same time. But for this course, we'll keep it simple.

Front-end

The front end is the visual element of your web application. It defines what you see and interact with. The front end is developed with HTML, CSS, and JavaScript.

You'll need to set up your development environment. The components of this will be:

1) A code editor, such as VS Code, or Sublime Text.
2) A compilation, and packaging framework:
   - ➢ Webpack
   - ➢ Gulp
   - ➢ Grunt

   This is also used for serving and "Hot Loading" your application at development time, on a nodejs web server, running on localhost.
3) A frontend framework.
   - ❖ React
   - ❖ Ember
   - ❖ Vue
4) Configuring your packaging tool to talk to your backend

5.2.10 **Step 10 – Build your backend**

The backend is typically what manages your data. This refers to databases, servers, and everything the user cannot see within a web application. Building your backend is one of the toughest parts of web app development. If you feel overwhelmed, a tool like Budibase can take away many of the complexities.

When building your web app, you need to choose between:

- Server Pages (Multiple Page Application) and
- Single Page Application

If you have chosen Server Pages, your backend will also be generating your front-end and serving it to your user.

With a single-page app, the backend will simply serve your static front-end files (i.e., your "Single Page" and it is related assets).

The primary jobs of the backend will be to:

- ✓ Provide HTTP endpoints for your front-end, which allow it to operate on your data. E.g., Create, Read, Update, and Delete ("CRUD") records.
- ✓ Authenticate users (verify they are who they say they are: a.k.a log them in).
- ✓ Authorization. When a logged-in user makes a request, the backend will determine whether they are allowed (authorized) to perform the requested action.
- ✓ Serve the front-end.

When choosing your backend:

- Use tools that you're familiar with (you can try Budibase).
- Server page frameworks like Django, Express, and Flask.

5.2.11 **Step 11 - Host your web application**

Hosting involves running your web app on a particular server. When using Budibase, this step can be automated with Budibase hosting. With Budibase, you are still required to buy a domain. If you are not using Budibase to host your web application, follow these quick steps:

I. Buy a domain - Namecheap
II. Buy/Setup an SSL certificate - Let's Encrypt
III. Choose a cloud provider:
    - ✓ Amazon
    - ✓ MS Azure
    - ✓ Google Cloud Platform
    - ✓ Heroku and Firebase are interesting alternatives that aim to be faster and easier to get things done.

These hosting options will almost certainly provide you with everything you need. They have ample documentation and community support and are generally reliable options.

5.2.12 **Step 12 - Deploying your web application**

You have sourced your idea, validated it, designed and developed your web app, and chosen your hosting provider. You are now at the last (deploying) step.

This deployment step includes how your web application gets from your source control on your computer to your cloud hosting provider chosen in step 11.

The following development tools provide continuous integration and will help you with deploying your web app to your cloud hosting:

- GitLab
- Bitbucket
- Jenkins