WHAT ARE THE OBSESSIONS OF AN EXPERIENCED SOFTWARE ARCHITECT FOR GLOBAL LIFE SATISFACTION PROJECT

I have been doing production coding, i.e. software engineering since 1995 and my feel for software engineering is extremely backend-oriented. This does not lead to a lot of nifty front-end magic but I am obsessed with backend because that is where the real work happens.

In the case of Global Life Satisfaction, I am obsessed by simplicity and efficiency, canonical code for simple basic things so that when we scale to running these things on 1000-2000 nodes in production, basic simple operations are robust, efficient, fast.

See, you can make a fancy gizmo widget-gidget FX show on the front end that works on a fast PC in demo mode and then hogs up huge resources, is slower than your grandmother doing C++ code, and totally unscalable for 8 billion user delivery.

The job of a great software engineer is to have a deep feel for the fast efficient canonical code that is both fairly easy to read and is extremely elegant and fast and robust. You want code that will not have to be touched afterward, very basic code that you can trust.

I have been slowly working towards generic ways of producing code to handle survey type front-end with scalability, efficiency, and robustness in mind.

When you have apps being used by billions of people, you will never know what broke at what esoteric layer of depth and if the bottom is rickety the whole enterprise will be hacking and using work-arounds and it will be a travesty. I try to eliminate the future problems by paying exquisite attention to the lowest basic layers and tighten it. This seems simple but it is not because we don't have empirical reaction from users when we are working on these problems.

You say, well it's trivial to do a JSON representation of Survey questions. You can hack something together, and you can hack something that displays it on web or mobile. Well the problem is that the kiddo approach of just get something working is inappropriate. You need extremely strong robust well-designed components precisely at the bottom layers where things are simple. Your choice of JSON representation will stick around for a century and if you get it wrong, you will be hated by all the people working on top of this. You need extreme efficiency and strong code structure and design because inefficiencies will end up propagating to some bloke in Sudan who will hate your guts because the app is too slow to load and not responsive fast enough to be usable.

It is the mark of an amateur coder who cuts corners on the deep lower backend and rushes to gizmos widgets midgets and flashy front-ends to impress all sorts of onlookers. I am not in that business. I want strong solid backend where language choice and very elementary operations are rock solid. This then gives confidence to layers on top that will go closer to the receiver of services.

Date: August 13, 2021.

Generic survey display and database management ought to be done perfectly flawlessly so that inefficiencies do not propagate from the lowest layer components. Then the lowest layer ought to be sealed for generic use in production and should not change for decades. People should not have to bandage this code in 3-4 years because we were not professional enough.

I go slowly here. I chose Meteor and Blaze Templating to do I/O. I keep doing tests of simple functions to gain sharper sense of what is clearest expression in code that will last, what is most efficient and clear, and what is shiny rock solid production quality components. Smart ambitious engineers hate working on simple components like this when they are young. I am 48 so I know this is much more important than it seems, and it is not trivial at all.

I have many tests on simple things. Let us look at a sample JSON of a questionnaire.

```
[
{
    "No":1,
    "Q": "You and your spouse make up after a fight.",
    "A": "I forgive him or her.",
    "B": "I'm usually forgiving.",
    "V": "B",
    "ZPmG": "1"
},
    {
"No":2,
"Q": "You forget your spouse's birthday",
"A": "I'm not good at remembering birthdays",
"B": "I was preoccupied with other things",
"V": "B",
"ZPmB":0
    },
    {
"No":3,
"Q": "You get a flower from a secret admirer",
"A": "He/she likes me",
"B": "I'm popular",
"V": "B",
"ZPmG": 0
    },
    {
"No":4,
"Q": "You run for a community office position and you win",
  "A": "I devoted a lot of time and energy at campaigning",
  "B": "I work very hard on everything I do.",
"V": "B",
  "ZPvG": 0
},
    {
"No": 5,
"Q": "You miss an important engagement",
  "A": "Sometimes my memory fails me.",
  "B": "I forgot to check my appointment book",
  "V": "B",
  "ZPvB": 1
    },
    {
"No": 6,
"Q": "You and your spouse make up after a fight.",
  "A": "I forgive him or her.",
  "B": "I'm usually forgiving.",
"V": "B"
},
    {
"No": 7,
"Q": "You host a successful dinner",
"A": "I was particularly charming that night.",
"B": "I'm a good host",
"V": "B",
"ZPmG": 1
    },
    {
"No": 9,
"Q": "You and your spouse make up after a fight.",
"A": "I forgive him or her.",
"B": "I'm usually forgiving.",
"V": "B"
    },
    {
"No": 10,
"Q": "You fail an important examination.",
"A": "I wasn't as smart as the others taking the exam.",
"B": "I did not prepare well for the exam.",
"V": "B",
"ZPvB": 0
    },
    {
"No": 11,
"Q": "You prepared a special meal for a friend who hardly touched it.",
"A": "I am not a good cook.",
"B": "I prepared it in a rush.",
"V": "A",
"ZPvB": 0
    },
    {
"No": 12,
"Q": "You lose a sporting event for which you trained for a long time.",
"A": "I'm not very athletic",
"B": "I am not good at that sport.",
"V": "A",
"ZPvB": 0
    },
    {
"No": 13,
"Q": "You lose your temper with a friend.",
"A": "He/she is always nagging me.",
"B": "He/she was in a hostile mood.",
"V": "A",
"ZPvB": 0
    },
    {
"No": 14,
"Q": "You are penalised for not returning your income tax forms on time.",
"A": "I always put off doing my taxes.",
"B": "I was lazy about doing my taxes this year.",
"V": "A",
"ZPvB": 0
    }
]
```

So I am practicing just using elegant Meteor to display it by Blaze templates. This is an art to do this in such a way that it does not look like someone killed the cat in the kitchen with a pick-axe. There are many bad ways of making this unreadable and too complex. However I believe there are canonically best ways of doing these things. And that takes time to reach.

Let us take a look at some attempts here. I want to show you how simple the code looks in Meteor. First, let's look at the javascript and then the Blaze template. This is working test. The 'main.js' looks like this. The key here is loading json with 'require' and getting a helper to give access to it by putting it into an instance variable.

```
import { Template } from 'meteor/templating';
var opt = require('./opt.json')
import './main.html';

Template.optimism.onCreated(() => {
  this.opt = opt;
});

Template.optimism.helpers({
  opt: () =>{
    return this.opt;
  },
});
```

Then in template we print out something reasonable.

```
<head>
  <title>test-optimism</title>
</head>
```

```
<body>
  <h1>Welcome to Zulf's Optimism Survey!</h1>

  {{> optimism}}
</body>

<template name="optimism">
  {{#each a in opt}}
    <p>
    {{a.No}}
    </p>
  {{/each}}
</template>
```

This is very clean code and it shows us how hardcoded json questionnaires can be accessible to front-end. This test is nice and clear. We can then change this gently to database-based surveys instead of hardcoded, we can add user input with other controls etc.

1. A Second Pass

I will show you next some improvements over the code above which includes using a form and submit and radio buttons. On coding technique here you will see me use a parametrised template but I won't do anything now on submit. This is test code.

The code is now in github in `https://github.com/zulf73/test-optimism`. A number of things need to be included such as submit-processing, managing collections of survey response JSON data records in the MongoDB so that on submit we have (receiver, responses) stored, syncing of display of questionnaire for partially completed surveys so that the receiver gets a partially filled questionnaire on re-log. Those things are a bit of work.

I updated the questionnaire a bit more but it is still not exactly 32 questions of Seligman's *Authentic Happiness*. I will just show you the new Blaze Template. I will then point out that this is fairly *generic* so that the exact same code can handle arbitrary two-value questions.

```
<head>
  <title>test-optimism</title>
</head>

<body>
  <h1>Seligman's Authentic Happiness Optimism Survey!</h1>

  {{> optimism}}
</body>

<template name="optimism">
  <form>
  {{#each a in opt}}
    <p>
    {{> entry q=a.Q n=a.No a1=a.A a2=a.B}}
    </p>
  {{/each}}
  <input type='submit'>
  </form>
</template>

<template name="entry">
  <p>{{q}}</p>
  <div>
    <input type="radio" name={{n}} id="a" value={{a1}} checked>
      <label for="a">{{a1}}</label>
  </div>

  <div>
    <input type="radio" name={{t}} id="b" value={{a2}}>
      <label for="b">{{a2}}</label>
  </div>
  </template>
```

2. A Return To Substance

All of this coding is to produce infrastructure for scientific psychologists to improve Global Life Satisfaction. Our goal is not just sell some products to lots of people. Our goal is to literally actually improve the Life Satisfaction of eight billion people. We're going to have a couple of companies. Thyself Inc. proper will use infrastructure to develop Quantitative Positive Psychology. The example in this not is Martin Seligman's Optimism Questionnaire of Chapter 6 of *Authentic Happiness*. Now the field of Positive Psychology is new and advancing every day. We will want to have ability to rapidly create new questionnaires measuring new metrics and deploy it for 8 billion and then analyse the results to improve Quant models of Positive Psychology. As a matter of fact that is the major innovative direction I am pioneering. We want good engineering to make lives much easier for this more substantial project.

What you see here is the capability of *generic questionnaire output to arbitrary people. We'd need to use Meteor's user-password module to tie this to individual receivers, and Meteor Collections to sync responses to database records, and then we need analytical tools on massive MongoDB data stores of Quant Positive Psychology data.*

We might decide to change and refine metrics and questionnaires. We might decide to add arbitrary new questionnaires and we might decide to track writing samples and self-disclosure of individuals. This simple test gives you a realistic sense of the feasibility of doing this by the technology I have chosen. Our goal is not to sell some products that are attractive but to manage a gigantic distributed I/O with infrastructure so that Quant Positive Psychology can use it as a tool like a telephone to rapidly develop psychological tools and do quantitative analysis of the results.

To be clear, we need great technology tools for I/O from eight billion people. The actual Global Life Satisfaction project has its work not in this part, which I think is clear is achievable, but in the continual process of developing quantitative models that involve many different questionnaires necessary to improve Global Life Satisfaction and managing the resulting data in MongoDB (cloud database).

3. A Slightly Fuller App

Actually collecting the responses from the receiver (I don't like 'user' and prefer 'receiver') and putting them in MongoDB is one of the major positives of Meteor. We want to take the input of the receiver and produce a JSON data structure easily and here we use a debugged package instead of hacking, following best practices of professional coders. When a debugged package satisfies the use-case it is inadvisable to produce new code that might introduce bugs. The package is pcel:serialize.

```
import { Template } from 'meteor/templating';
import 'meteor/pcel:serialize';
var opt = require('./opt.json')
import { Answers } from '../lib/answers.js';
import './main.html';

Template.body.helpers({
  answers() {
      // Show newest tasks at the top
      return Answers.find({}, { sort: { createdAt: -1 } });
    },
});
Template.optimism.onCreated(() => {
  this.opt = opt;
});

Template.optimism.helpers({
  opt: () =>{
    return this.opt;
  },
});

Template.optimism.events({
  'submit form' : (e) => {
    e.preventDefault();
    var formObject = $('form[name="qf"]').serializeJSON();
    formObject['createdAt'] = new Date();
    console.log(formObject);
    Answers.insert( formObject);
  }
});
```

So here we are invoking the pcel:serialize package to translate input on the front-end to a JSON. This is very very powerful because this code will work for arbitrary surveys *and not just the one we are testing. We can change the questionnaire and the same code will work without problems.*

Now Meteor tutorials all fail to tell you clearly that new collections work when the collection code is not in imports/api directory but 'lib' directory. This is very important because Meteor has directory conventions that are silent. In the 'lib' directory we have 'answers.js':

```
import { Mongo } from 'meteor/mongo';

export const Answers = new Mongo.Collection('answers');

if (Meteor.isServer) {
    // This code only runs on the server
    Meteor.publish('answers', function answersPublication() {
      return Answers.find();
    });
}

Meteor.methods({
    'answers.insert'(doc) {
        //check(text, String);
        // Make sure the user is logged in before inserting a task
        //if (! this.userId) {
        //  throw new Meteor.Error('not-authorized');
        //}
        Answers.insert(doc);
      }
});
```

There is a complex manner in which client side database operations interact with the server side database operations which is the magic of Meteor. Zulf is wise and does not want to know how this works and just carefully tries to do as little work as possible to leverage this beautiful sophisticated and mysterious engineering. I just want sync-ing between MongoDB and front-end. I don't really care how it's done as long as it is robust and it works.

Now the github will contain prototype that actually does update the database with the survey answers. And this is scalable and robust as is. The code is relatively simple and clean and it is fairly clear the potential for making the front end look more polished.

4. A Different Way Of Seeing Global Life Satisfaction

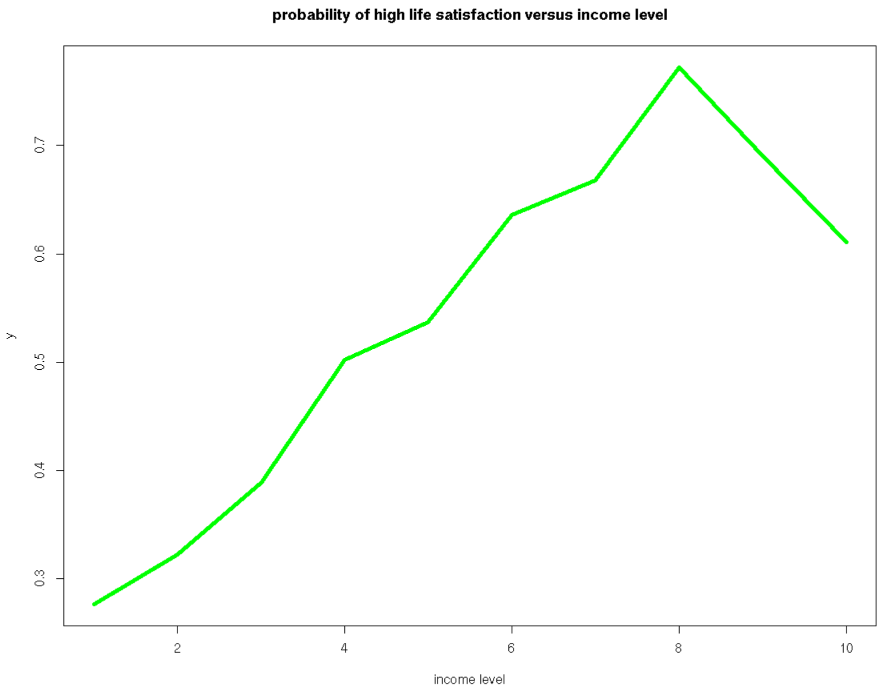
Global Life Satisfaction as a project can be seen as implementation of statistical models of Life Satisfaction of individuals as a function of a large number of factors with the aim of affecting positively Life Satisfaction of the eight billion individual human beings using algorithmic strategies. In other words, it is not a matter of subjective considerations. To put things in context, Bill Gates, based on his own esoteric beliefs thinks that money is the major factor in Life Satisfaction. This is known not to be quite true. There is a delicate relationship. Let me show you some results next.

First, here is a simple result that does support the richer people have higher life satisfaction hypothesis.

Probability of high life satisfaction given higher income is 67%; probability of high life satisfaction given lower income is 43.1%. So this does indicate that higher incomes increase the probability for higher life satisfaction. And these are global numbers from World Values Survey that I calculated just now – August 13 2021 – myself.

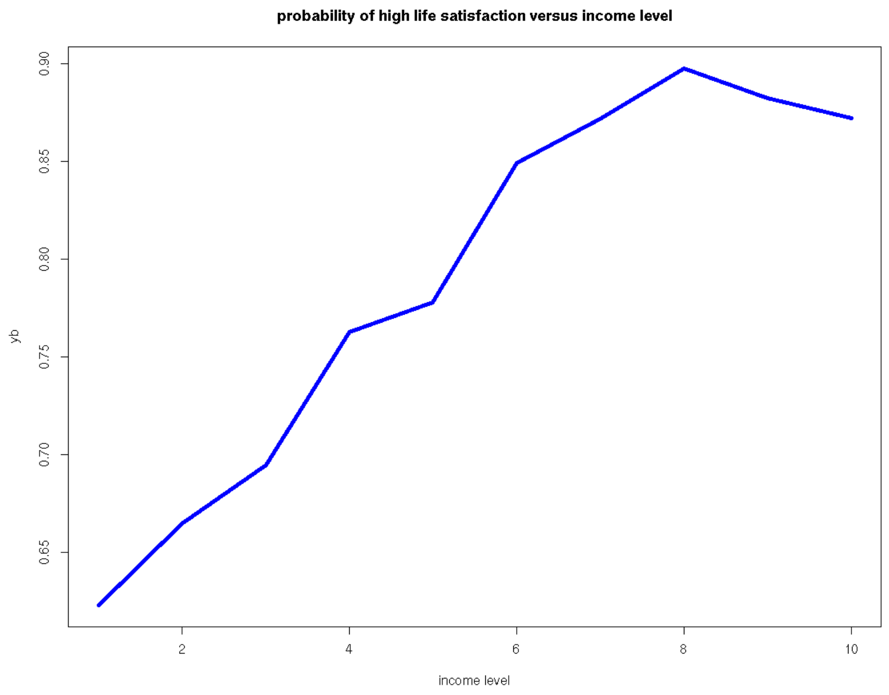
But it is not true that the major determinant of life satisfaction is income at all.

Aha, I have discovered the truth regarding income level and happiness. It is monotonically increasing to around level 7-8 out of 10 and then starts falling.



This is actually quite clear what the issues are. There is a reversal of probability of high life satisfaction beyond a threshold.

Ah, so that graph had normalized first on rows of Q49 first versus Q288. I removed the normalisation to get a similar pattern slightly tamer but it's clear that income level 8 is a maximum and 9, 10 go down monotonically.



Now don't get any ideas. These are my discoveries of August 13 2021, not yours ok?