**Cognitivity**

**High Level Design**

**Submitted by**: Ophir Katz

Mark Erlich

Ben Hunter

Guy Shemesh

Daniel Lyubin

Peer Sagiv

## Supervised by: Tanya Brokhman

**Table of Contents**

[Abstract 3](#_30j0zll)

[1. Introduction 4](#_3znysh7)

[1.1 General Project Description 4](#_2et92p0)

[1.2 Programming Environment 4](#_tyjcwt)

[2. Theoretical Background 4](#_3dy6vkm)

[2.1 Section 1 4](#_1t3h5sf)

[2.1.1 Sub Section 5](#_3rdcrjn)

[2.2 Section 2 5](#_26in1rg)

[3. Basic System Functionalities 5](#_lnxbz9)

[4. Software Implementation 5](#_35nkun2)

[4.1 Modules 5](#_1ksv4uv)

[4.2 Top-Level View 5](#_44sinio)

[4.3 Main Menu 5](#_2jxsxqh)

[4.3.1 User Interface 5](#_z337ya)

[4.3.2 Features 5](#_3j2qqm3)

[4.3.3 User Interface 5](#_1y810tw)

[5. References 5](#_4i7ojhp)

# 

# General guidelines for writing HLD

HLD is an initial contract between you and your customer on what functionalities will be provided by your program. It doesn’t go into the low-level details of how each function will be implemented, it doesn’t state ALL of the views that will be created for the user but it does state the main things.

For example: say I’m going to design an application for managing a supermarket. What are the main functionalities I’m going to support?

There will be 2 usage modes: one for customer and one for an employee. Each mode will provide different functionalities.

1. Customer mode:

The customer will be able to choose from a variety of the supermarket products, searching for a desired one according to the following filters:

Type of product (food, beverage….)

Calories range

Lacking specific ingredient (for example gluten free)

The information available to client on each product will be: ingredient list, calories, manufacturer, expiration data

Note that I didn’t explain **how** the views will look like, just what functionality will be available to the client.

1. Manager mode: …
2. Registration and access control

There are, of course, a lot more features that can be added to the supermarket application that are not mentioned above. As you can see, the description is very HIGH LEVEL (==HLD) and doesn’t go into technical details.

The following template is a suggestion from what to start. Its ok if some sections are not relevant for your project and if additional sections (that are not mentioned) should be added.

**To conclude**: when writing, think about the customer. This is a contract you’re committing to. All that is mentioned here must be provided by your product.

# Abstract

Creating cognitive tests for subjects can be tough work, more so when we want to consider different aspects of a cognitive process of a subject while completing a test. The Project mainly focuses on the hardship of creating a test, in a purpose, of course, to eliminate the tedious parts of the job.

# Introduction

## General Project Description

Cognitivity is a platform for creating and conducting cognitive tests for subjects all around the world.

The platform is implemented as a web application which can be accessed anywhere.

## Programming Environment

1. Java technologies : Spring MVC, Hibernate.
2. MySQL.
3. Angular 4 with TypeScript.
4. HTML 5. CSS (bootstrap, or other styling frameworks and libraries - TBD).
5. Node.js.

# Basic System Functionalities

1. Login as an administrator/test manager.
2. Create a cognitive test.
3. Manage data objects such as tests, subject’s information, questions etc.
4. Conducting a test.
5. Measure time units during questions and tests.

# Software Implementation

This is the programmer manual: Up to you – like:

## Modules

Tanya - you decide if we need this part..

## Top-Level View

The Project is implemented as a web application, so we will have backend modules and frontend modules.

**The Frontend:**

FILL IN HERE : Angular, HTML, CSS and other shit..

# Bootstrap (Discuss with team first)

# Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery. Bootstrap allows you to build responsive, mobile-first projects on the web.

# 

# Angular

# General Explanation

Angular (Angular version 4) is a cross-platform, MVC development framework, developed and maintained by GoogleTM.

Angular applications can be developed for desktop browsers, mobile web and even native mobile and desktop applications.

The framework applies for the frontend aspect of your application, while giving you the freedom of implementing the backend in any way you wish.

In a way, Angular is a design pattern for web applications.

# Architecture Overview

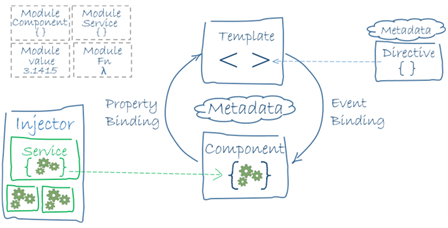


Figure 1: Architectural scheme

While it may seem that a website is built of html pages and serverside functionality, Angular’s Architectural view “breaks” the web application to different components that represent pieces of data that should be viewed as a single entity, hence improving cohesion and reducing coupling. Angular’s methodology is to encapsulate UI components for decreasing the dependencies between them to them minimum.

The basic parts of an Angular web app are as follows:

**Template and Component**

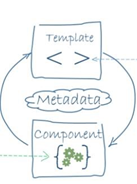


Figure 2: the VC in MVC

The View and Controller of the MVC design pattern. The Template is the UI as seen by the user, the regular HTML(5) components. The component is the controller. It holds the data that the template needs to show: lists, user data, etc. The component is an object containing the template’s state. The “Metadata” in Figure 2 is a class decorator that tells the component class which template it’s connected to, in what name should we call the component (a selector in HTML terminology)

Templates and components communicate in a “2-Way Binding”, The arrows in Figure 2. A template passes data to its component via “Event Binding” (the arrow directed from ‘template’ to ‘component’). User interactions with the UI trigger events which are functions declared in components’ classes. These events pass any data defined by the developer to the component’s object. For example, an Angular web app contains a registration form – a set of textboxes and a submit button. Each textbox is identified with a special identifier (which is a part of Angular’s syntax), and the form container (an HTML tag) identified with the same kind of identifier that combines all of the textboxes’ identifiers to a single value. The form also has an “onSubmit” event that passes data to the component.

Showing information on the screen is done with via “Property

Binding” (the arrow directed from ‘component’ to ‘template’). The information shown is a property of the component’s object. For example, we have a basic web app that shows a user’s first and last name, and the data is fetched from a server.

The names are stored in two variables, firstName and lastName which contain the names respectively. Both of these variables are properties of the class. In the template, we will show the names inside any HTML tag that shows text, an h1 tag for example. The names will be displayed with the following syntax:

<h1> {{ firstName }} {{ lastName }} </h1>

If we had a form that changes these names (a form with a structure similar to the previous example), when we’ll submit it, the names will change (after submission).

There’s also a way to dynamically change information presented on the screen as we edit the property (with a textbox bound to it, for example), called ngModel. With this feature, a property is both an input and an output. The figure bellow demonstrates and summarizes the types of data binding featured in Angular.

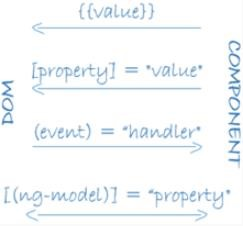


Figure 3. (DOM=HTML)

**Services**



A service is a class that contains a single logical behavior of the application. It can implement anything that’s needed for the app, from talking with a database or serverside program, to getting pictures from websites via HTTP requests. Services encapsulate the (ugly) implementations of backend behaviors, while letting the components talk with the backend via interfaces they provide.

Remember the gear drawings in Figure 2? It’s not coincidental, because components include services’ instances. The use of services dramatically improves the development process and provides code reuse, encapsulation, etc. A component *never* calls directly to backend functionality. Instead, he calls a service’s functionality.

**Modules**

Modules are classes that provide large functionality. There is usually one module that contains the developer’s components (UI), services, but it can contain only functions and services that serve the same purpose and should logically be together under the same module. For example, there is a module called BrowserModule that oversees rendering the web app on the browser.

The first component that’s loaded is a default component called AppComponent (can be named however desired) and it’s the root component. It can contain components which contain other components, and so on. Intuitively, the web application’s structure resembles a tree, where a node is a component and a child node is a component nested in another component (illustrated in Figure 4).

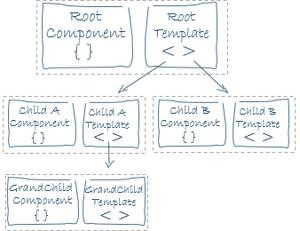


Figure 4: Component nesting tree

# Advantages

• Modular development

• Component encapsulation

• Component and code reusability

• Use of an object oriented (both statically and dynamically typed) language, which allows declarative programming

• Developed by GoogleTM and an open source project, so it’s supported and maintained (It won’t go away anytime soon)

• Deployed on a server quite easily

• Can run on anything - browsers, mobiles, tablets • Easy testing, because of modularity.

# Disadvantages

• Clientside development only. No framework for a serverside development, so without a database which provides a module that integrates with Angular, developing a custom DB can be tough and quite a headache, but with proper development it will be a one time headache

• Since Angular doesn’t have a common syntax, there is a learning curve. But once again, it’s a one time headache Angular is great. Use it.

**Sources:**

Angular Documentation:<https://angular.io/guide/architecture>

**POC for Our Project**[https://www.codeproject.com/Articles/860024/Quiz-Application-inAngularJs](https://www.codeproject.com/Articles/860024/Quiz-Application-in-AngularJs)

# TypeScript vs. JavaScript

TypeScript is a superset of JavaScript, meaning TS provides additional functionality to JS. TS eventually compiles to JS, so you can write JS code in a TS file.

**Pros:**

• **Provides optional static typing and static type checking, thus preventing errors in early development stages.**

• **Added OOP principles and features – classes, interfaces, inheritance, private/public members, etc.**

• **Easily understandable by javascript developers, especially by programmers who are familiar with OOP concepts.**

• **Comfortable, modular and organized development for large projects.**

**Cons:**

• **Javascript’s cons, except the problems TS solves.**

A StackOverflow thread on the topic:

[https://stackoverflow.com/questions/12694530/what-is-typescriptand-why-would-i-use-it-in-place-of-javascript](https://stackoverflow.com/questions/12694530/what-is-typescript-and-why-would-i-use-it-in-place-of-javascript)

**The Backend:**

As the most commonly used language for backend in web applications is **Java**, we chose to use it for our implementation of the backend logic.

**Java technologies for developing the backend side:**

Spring Web MVC (or just Spring MVC) is the original web framework built on the Servlet API and included in the Spring Framework from the very beginning. The Java Servlet API is just a standard for implementing Java classes that respond to requests. It is designed in a way that web based application (which naturally involve a lot of request based web page changes) could use it very easily. The Spring MVC framework allows for extensive configurations, either using the web.xml file or even via the Java code. Moreover, the MVC module provides default configuration (that can be changed easily) suitable for most applications. Spring offers a neat way to define REST clients for transferring data between the web application and the server side (for various purposes, such as storing data, or handling web pages requests). Spring MVC also offers a very comfortable way to test the entire backend, using Mocks, which is incredibly convenient for testing Java classes.

**Spring’s modularity for use with AngularJS:**

First of all, as stated before, Spring is entirely modular. This makes it possible (and easy) to program the backend logic without regard to the web technology used for the UI.

You can define controllers, adapters and more objects to implement logic of the backend, all in Java, while the configuration of the web technology used, is only done in the xml configuration files. For instance, declaring that a certain service of the UI is implemented in a file called service.js (or service.ts), can be easily be done by inserting a <script> tag in the corresponding .jsp file (like homepage.jsp, for example).

**Integrating with Angular’s services** is also easily done: Spring MVC defines a controller, which defines interfaces for conversing with a frontend, and Angular can define how to use these interfaces when implementing different types of services, such as pulling data from a data server (which is done via the server side), or requesting a web page, and so on… The RESTful services Angular calls are also integrated in the Spring framework, so it makes for an even easier development (because the design is already implemented).

The way it is done is by defining interfaces and then implementing these interfaces as services to offer from within the controller class. The controller is accessed from Angular.

**Learning curve with Spring MVC:**

As stated, this is just a Java library, so it is written in only Java, using some configurable content in xml. So, All we really need to know is Java, and to read a lot on the framework (it has alot of features, but what’s good is that we probably won’t use them all).

**Pro’s**

* Spring MVC is FREE!
* It is highly used in building web applications. Actually, it is the most used in Java, grabbing over 40% of the market.
* It is written in only Java - easy to learn + all of the team mates already know a portion of Java (as a prerequisite to this course).
* It has a lot of guide online, and a bunch of examples for building web applications, even using AngularJS for frontend.
* As the most used library out there, it is very unlikely that it will stop being supported. Since it is also open source - that is probably impossible.

**Java’s way to interact with a SQL data server:**

Since we are definitely using a SQL data server, Hibernate ORM is a great option:

**Hibernate ORM** (Hibernate in short) is an object-relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database.

The ORM model is THE way to use any relational data servers, for creating data, and reading from it. It interacts with such servers using JDBC ( = Java Database Connectivity is an application programming interface for the programming language Java, which defines how a client may access a database. It is Java based data access technology and used for Java database connectivity).

Hibernate was designed to work in an application server cluster and deliver a highly scalable architecture, which answers, in theory, any questions about number of users the application will have, regarding access to the data server.

Just like Spring, Hibernate is highly configurable.

Hibernate is well known for its excellent stability and quality, proven by the acceptance and use by tens of thousands of Java developers, which means it will be supported for a long time.

Hibernate is database agnostic. It doesn’t care if you use Oracle, SQL Server, MySQL, or about a dozen other relational databases. This means that without regard to the database we will use, Hibernate will be good with it. Specifically, as we will most likely be using a kind of SQL server, Hibernate is perfect.

**Configuring Hibernate to work with Spring:**

This configuration is done via just xml config file in the project directories. It is easy, and does not need further dependencies.

**Configuring Hibernate to work with the data server:**

This configuration is done via just xml config file in the project directories. It is easy, and does not need further dependencies. We also need to write a sql script file to define the tables in our database.

**To summarize:**

There really is no fault with using the Hibernate service. Both it, and the JDBC are highly used for accessing different types of data servers, and the only thing we need to do is to configure the correct one with the system, and implement the objects about which we want to converse with the server.

### Features

1. Login as an administrator/test manager.
2. Create a cognitive test.
3. Manage data objects such as tests, subject’s information, questions etc.
4. Conducting a test.

### System Functionalities

Login (Feature #1):

Requests a username and a passwords and connects the user if they exist in the database.

Creating a cognitive test (Feature #2):

- Can add an unlimited number of

### User Interface

FILL IN HERE : Ben’s features.

# References

# References for the backend technologies

* <https://blog.angular-university.io/developing-a-modern-java-8-web-app-with-spring-mvc-and-angularjs/>
* <https://spring.io/>
* <https://examples.javacodegeeks.com/enterprise-java/spring/mvc/angularjs-spring-integration-tutorial/>
* <https://www.journaldev.com/2882/hibernate-tutorial-for-beginners>
* <https://www.journaldev.com/3524/spring-hibernate-integration-example-tutorial>
* <https://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial>
* <http://hibernate.org/>