E M X

**Joby-Jobs Position Match   
Web & Mobile App**

**Software Design Document (Server Side)**

Enities List – רשימת ישויות מערכת

Seeker (Candidate) - מועמד

Employer – מעסיק (חברה)

Employer-Person – employer personal - עובד של המעסיק

Position - משרה

Field – תחום עיסוק

Profession – מקצוע עיסוק

Sub-Profession - תת מקצוע עיסוק

Interview – ראיון

Conversation (Chat Message) – הודעת צ'ט

Application (job application) – תהליך קבלה לעבודה של מועמד

Admin-Person – Administrator personal – מנהל מערכת (אדמיניסטראטור)

Reaction – a company's like of a worker, or the other way around. - תגובה / לייק

(ASP.NET Identity Objects)

User - משתמש

Role – קבוצה (של משתמשים)

Components:

Worker-Job-Interest – Holds the data about the interest of a worker around possible jobs (e.g. Min-Salary, Max-Salary, Distance) - מידע אודות דרישות משרה של עובד

Enums: - מבני אפשרויות-מרובות במערכת

ApplicationStatuses - סטטוס קבלה לעבודה

UserTypes - סוג משתמש

ReactionTypes - סוג לייק

User-Activities - פעולות משתמש

Basic Scenario

1. Employer-> Manager Employer Person signs in the system (with the employer) - <http://www.jobyjobs.com/Account/RegisterEmployer>

[Background call to issue an employer token: <http://www.jobyjobs.com/Employers/GenerateTokenEmployer>]

1. Employer-> Manager Employer Person invites another employer person to join the system via mail - <http://www.jobyjobs.com/Account/RegisterEmployer>
2. Employer-> Employer person receives the invite via mail and opens (views) it. to the application <http://www.jobyjobs.com/Account/GetRegisterInvite>
3. Employer-> Employer person receives the invite via mail, accepts the invite and registers to the application <http://www.jobyjobs.com/Account/RegisterEmployer>
4. Employer-> Employer person receives the invite via mail and rejects the invite [http://www.jobyjobs.com/Account/OnRegisterInviteRejected](http://www.jobyjobs.com/Account/OnRegisterInviteRejectedt)
5. Employer-> Manager Employer person initiatively cancels the invites [http://www.jobyjobs.com/Account/ OnRegisterInviteCancelled](http://www.jobyjobs.com/Account/%20OnRegisterInviteCancelled%20)
6. Employer-> Employer person posts a new position <http://www.jobyjobs.com/Positions/PostPosition>
7. Seeker- > Seeker registers to the system – <http://www.jobyjobs.com/Account/RegisterSeeker>

[Background call to issue an seeker token: <http://www.jobyjobs.com/Employers/GenerateTokenSeeker>]

1. Seeker-> Seeker commits a search on all position, with a criteria <http://www.jobyjobs.com/Account/RegisterEmployer>
2. Seeker-> Seeker selects a position via the UI, and enters it (views it) <http://www.jobyjobs.com/Account/GetPosition>
3. Seeker-> Seeker applies for a position <http://www.jobyjobs.com/Applications/ApplyForPosition>
4. Seeker-> Seeker removes his application for a position <http://www.jobyjobs.com/Applications/DropApplication>
5. Seeker-> Seeker removes all of his applications <http://www.jobyjobs.com/Applications/DropAllApplication>s
6. Seeker-> Seeker changes his work state (available, working etc.) <http://www.jobyjobs.com/Seekers/UpdateSeeker>
7. Employer->Employer views a position (same endpoint)

<http://www.jobyjobs.com/Positions/GetPosition>

1. Employer->Employer views all application(candidates) for the position <http://www.jobyjobs.com/Positions/GetPositionApplications>
2. Employer->Employer selects an application via the UI, and enters it (views it)

<http://www.jobyjobs.com/Positions/GetApplication>

1. Employer->Employer requests to view seeker's information via the UI.
2. <http://www.jobyjobs.com/Seekers/GetSeeker>

[Employer->Employer physically calls the seeker]

1. Employer->Employer updates the status of the seeker to "AfterPhoneInterview"

<http://www.jobyjobs.com/Applications/UpdateApplicationStatus>

[Employer->Employer physically appoints an interview with the seeker]

1. Employer->Employer sets an interview with the seeker.

<http://www.jobyjobs.com/Applications/SetInterview>

1. Employer->Employer wants to update the interview information.

<http://www.jobyjobs.com/Applications/UpdateInterview>

1. Employer->Employer chooses to cancel the interview.

<http://www.jobyjobs.com/Positions/CancelInterview>

1. Seeker->Seeker receives the interview invitation via mail, and opens it (views it)

<http://www.jobyjobs.com/Applications/GetInterview>

1. Seeker->Seeker chooses to accept the interview

<http://www.jobyjobs.com/Applications/AcceptInterview>

1. Seeker->Seeker chooses to reject the interview

<http://www.jobyjobs.com/Positions/RejectInterview>

Scenario 2

1. Seeker->Seeker searches for a position

<http://www.jobyjobs.com/Positions/SearchPosition>

1. Seeker->Seeker chooses to like a position

<http://www.jobyjobs.com/Reactions/SeekerLikesPosition>

1. Employer->Employer chooses to like a seeker for a position (a candidate)

<http://www.jobyjobs.com/Reactions/EmployerLikesSeekerForPosition>

1. Employer->Employer chooses to unlike a seeker for a position (a candidate)

<http://www.jobyjobs.com/Reactions/EmployerUnlikesSeekerForPosition>

1. Seeker-> Seeker requests to view all mutual likes (=matches).

<http://www.jobyjobs.com/Reactions/GetAllMatchesSeeker>

1. Employer-> Employer requests to view all mutual likes (=matches).

<http://www.jobyjobs.com/Reactions/GetAllMatchesSeeker>

1. Employer -> Employer wishes to view seeker information.

<http://www.jobyjobs.com/Seekers/GetSeeker>

1. Employer -> Employer requests to download seeker cv file.

<http://www.jobyjobs.com/Seekers/DownloadCVFile>

1. Employer -> Employer requests to view all draft positions.

[http://www.jobyjobs.com/Positions/GetPositionsByStatus (?status=1)](http://www.jobyjobs.com/Positions/GetPositionsByStatus%20(?status=1))

1. Employer -> Employer requests to view all active positions.

[http://www.jobyjobs.com/Positions/GetPositionsByStatus (?status=2)](http://www.jobyjobs.com/Positions/GetPositionsByStatus%20(?status=2))

1. Employer -> Employer requests to view all frozen positions.

[http://www.jobyjobs.com/Positions/GetPositionsByStatus (?status=3)](http://www.jobyjobs.com/Positions/GetPositionsByStatus%20(?status=3))

1. Employer -> Employer requests to view all archived(=deleted) positions.

[http://www.jobyjobs.com/Positions/GetPositionsByStatus (?status=4)](http://www.jobyjobs.com/Positions/GetPositionsByStatus%20(?status=4))

1. Employer -> Employer requests to view all positions.

<http://www.jobyjobs.com/Positions/GetAllPositions>

Scenario 2

[Employer->Employer opens main page(in angular client) and switches to "positions" page]

1. Employer->Employer views all positions

<http://www.jobyjobs.com/Positions/GetAllPositions>

1. Employer->Employer enters a particular position via the UI, and views position info

<http://www.jobyjobs.com/Positions/GetPosition>

Project Infrastructure

JobyJobs Solution(EMX.JobyJobs)

-> EMX.JobyJobs.ASPCoreFwk.API - Main Server WebAPI site (Presentation Layer (PL))

-> EMX.JobyJobs.Admin.ASPCoreFwk - Admin Side Server WebAPI site (Presentation Layer (PL))

-> EMX.JobyJobs.BL - Business Logic Layer (BLL) project

-> EMX.JobyJobs.DAL - Data Access Layer – DAL (including Object-Relational-Mapper (ORM) objects)

-> EMX.JobyJobs.ProxyServices - ProxyServices project- contains proxy classes for external services.

-> EMX.JobyJobs.Shared - Solution-Level shared library (all projects reference it)

-> Emx.JobyJobs.NotificaitonEngine.Service - allows notifications between different modules in the system.

-> Emx.JobyJobs.NotificaitonEngine.Logic - logic project for "NotificatioEngine.Service"

-> Emx.JobyJobs.TaskEngine.Console - allows scheduled operations (CRON jobs).

-> Emx.JobyJobs.TaskEngine.Logic - logic project for TaskEngine.Console.

Business Logic Concerns Scheme

**UsersBL(AccountBL) -** Handles all business-logic activities around users, roles, registration, logins,

passcodes, forgot-password, collaboration with facebook, google and so on. (also ASP.NET Identity functionality)

**SeekersBL -** Handles all business-logic activities around seekers, also contains general methods for the application.

**EmployersBL -** Handles all business-logic activities around companies, company persons etc.

**FieldsBL -** Handles all business-logic activities around fields, professions, sub-professions and so on.

**PositionsBL** - Handles all business-logic activities around position, searches, search tags, position precedence scheme and so on.

**ReactionsBL -** Handles all business-logic activities around reactions, that is company-likes-worker, worker-likes-company, and matches.

**ApplicationsBL -** Handles all business-logic activities around applications, interviews and so on.

**ConversationsBL -** Handles all business-logic activities around conversations (chat messages) etc.

Important Note:

The application is conceptually separated into the above modules, and this should also be utilized in the client-side implementation.

**Server Cross-Cutting solutions and Paradigms**

Logging:

Log4 net framework for ASP.CORE

All exceptions are logged in the target site, and also in the initial method.

Exception Handling:

Application Specific Exceptions are thrown (JobyJobsException) until the presentation Layer (PL), and from there are notified to the caller via HTTP satus codes / Reponse status object.

Authorization:

Users are divided into three primary groups: Seeker, Employer(Person) and AdminPerson, which are called "Roles".

Every method is intended for a specific role, while some are used by two or more of the groups.

After a user successfully registers to the system, he should call the "GenerateToken" API Call, and receive a token.

This token cryptographically holds the log-in data: user and password, and optionally may contain other data such as: expiration\_date – a date after which the token expires and the user should issue a subsequent token.

Supported Login providers: Native (Server Authrization module), Facebook , Google.

Account Management supported methods: Forgot Password(Reset Password)

Native Mobile Support

The server should also support native to server requests, and also push notifications (via push servers)

External Services Consumption

The Server is using the following services which can be configured in the application configuration file:

* Emails (SMTP Server)
* SMS (Goldman SMS)
* Native Push Services (not yet chosen)

Scheduled Tasks

The system supports scheduled tasks: such as send emails at the end of the day, commercial emails, notifications, data-deletion etc. none yet implemented.

Module-Notification-Mechanism

The System supports notifications between different modules: for example on-demand cache busting etc. none yet implemented.

Transaction Scoping and Compensation Mechanisms

The business logic layer in the program uses a compensation mechanism and transaction scoping for methods that are mutually-dependant and therefore should be accepted or rolled back as a single transation.

Internationality and time zone support

The server should fully support different time zones in the client side,

And initially the languages: Hebrew, English and Russian.

\*\* Server and client sides divide principality for the internationality, shown in the SoC section below.

System Abstraction and Unit-Testing

The Server side system is optimally abstracted in all levels, and testing (including mock-testing) may be used when needed.

API System Testing

The testing solution currently used is PostMan ©, other software may be used instead if needed: e.g. Advanced-Rest-Client© or API-Swagger©

Postman projects are including in the Delivery for all of the API calls for Seeker and Employer.

Admin side calls are not included in the delivery.

Server/ClientPrincipality and Separation of Concerns (SoC)

Server side handles all functionality and business logic around data set and retrieved from the db.

Client side handles UI-level internationality and local to utc time conversion, while the server handles on content-level internationality and later on currency (in future development).

Context Specific Objects Design

Objects are divided into three primary groups:

* DataObjects (ORM-Level objects)
* BusinessEntities (Business Layer objects)
* ServiceModels (WebAPI Service requests/responses)

\*\* Data mapping between the objects is done with the AutoMapper © tool and sometimes manually.

HTTP Protocol

The two methods used are: HTTP-Get and HTTP-Post, Whereas the "Get" should be used for "Read" calls and when the state of the server is not altered in a significant manner, and Post is used for other cases: Add-Update-Delete.

HTTP Status codes used are 200-OK, 301-Moved Permanently,302-Moved Temporarily(Found), 400-Bad Request, 401-Unauthorized, 403-Forbidden, and the general 500-Server Error, and other middleware(/IIS pipeline) codes: 404-not found, 502-bad gateway.

Other notions for status of the request should be notified via a designated Status code inside the response.

Server Side Caching

The Server holds cache for all frequently-requested resources, the cache is invalidated(e.g. busted) upon expiry and then reloaded.

Some of the data that is cache on the server: Fields, Professions, Cities.

Server Side Versioning

The server side code modules are not currently versioned and also the data base is not versioned, on later release version-management should be used.

Content File Management

User resources are currently provided simply by the File-System and are saved on server disk, later on Content-Delivery-Network (CDN) may be used.

Content Editing and Direct-DBs-Access

All contents should be ultimately inserted via the Administrator profile, and only on rare occasions be altered directly in the DB by a tech-person.

Server-to-Client Side Instrumentation

Module Division

As stated in the "Business-Logic-Concerns scheme" section of this document:

The application is conceptually separated into modules, and this should also be utilized in the client-side implementation.

Client-Side Consumption Support

The server primarily supports WebAPI infrastructure (for Angular/React), but can also be easily converted to Server Views (Model-View-Controller pattern) .

Client Side Ideality

The Client should hold identical objects for all the server's Transport-Level-Obects (ServiceModels).

The Client should hold an internal cache for all frequently-requested data, and approach the server only when needed.

**Relational Data Base Management System (RDBMS)**

Implementation:

The relational database is implemented using MySQL © db.

ORM level used: Entity Framework v6.0.

Initial Scripts:

Create\_Schema\_{date}.sql – create schema for initial db tables (no data)

Add\_static\_data\_{date}.sql – adds data to the static data tables \*

Add\_initial\_data\_{date}.sql – adds data to the initial data tables \* and static data tables

Add\_all\_data\_{date}.sql - adds data to the user data tables \*, initial data tables and static data tables

delete\_user\_data\_{date}.sql – deletes only user data

delete\_all\_data\_{date}.sql – deletes all the data from the db

\*\* all scripts belong to a specific db version.

\*\* static tables= non(to rarely)-changing data

\*\* initial tables= admin data

\*\* user tables=normal user data

Enumerators

Enumerators are implicitly used in the data base and are mostly not kept in any table. The server contains a tech-description file called : "app.info" in the root folder specifying all of the enums used by the db.

This file also contains other essential information such as: assumptions that the application utilizes, or static data.

Record Auditing and Slowly-Changing-Dimensions (SCD) Scheme

The data base contains tables which offer the state of Active/Inactive (active column), and in some tables, as required, LastUpdated column is used.

In some other cases: Archived column is also used to tell between actual data and archived data.

As a general practice: data is usually not deleted physically (but logically via active column)

Business Logic in the database (e.g. SP/UDF)

The database as a general note does not contain any Stored Procedures (SP) or User-Defined-Functions (UDF). Logic is centralized on the server app only.

Application to Database calls

All calls from the application to the data base are done via ORM (Entity Framework), except for rare occasions (Lookup delete, Bulk delete, etc.)

Versioning:

First db version: v1.0.1

**Document Version History**

**Date Version Change Author**

16-11-2017 1.0 Create Ori Ahdout

13-02-2018 1.01 Edit Ori Ahdout