**Introduction:**

Contact Management System is an end to end application which helps manage contact list of names, multiple addresses, phones and events. The home screen houses a search bar and links to find, modify and delete contacts based on Name, Address, phone number or event date.

**Architecture:**

Application uses a three-tier-architecture. A client (browser), a web server and the database.

**Client (Browser, HTML, CSS, JavaScript)**

**Node JS + Express JS**

**Database - MySQL**

Client is built using HTML, CSS, JQuery and Embedded JS. Web-Server is built using Node JS, Express JS and the database used is MySQL.

Client uses HTTP protocol to make GET and POST requests. Creating new contacts and Modifying contacts are made through POST requests to the server whereas searching and deleting are made through GET requests.

The web pages are housed in the server and is used to render web pages to the client when necessary.

Client is used to request and insert data into in the database. The request is sent to the web-server to process the request and make the appropriate query to the database. The database sends the data as a response to the fetch query and success message for an update or insert request.

**Design and Assumptions:**

**Initial Setup:**

* Contacts.CSV file is parsed using Python using Comma as delimiter.
* The CSV file is then normalized and loaded into the MYSQL database into 4 tables – Contact, Address, Phone and Date.

**Searching Contacts: (API: GET /?search=query)**

* The searching is accomplished using a single search box which searches through the contact names, address fields, phone numbers and event dates.
* On the back-end, there is query joining all 4 tables to contact table on contact ID.
* The contact ID, First Name, Middle Name and Last Name are displayed as search results.
* Edit/Delete button are provided for each contact which appears as search result.
* Multiple strings can be provided for search. Contacts which matches ALL details will appear.
* The search query is built using the following technique:
  + The Contact table is “LEFT JOINed” (cascaded) with address, phone and date tables on the matching contact IDs.
  + The WHERE clause contains string matching LIKE conditions of every field.
  + The WHERE clause is built by concatenating the above LIKE conditions for every search query string entered by the user.
  + Finally, the count of the above search result is queried to be displayed to the user.

**Adding new Contact: (API: POST /add)**

* The fields displayed are First Name, Middle Name and Last Name, Address fields – Type, address line, city, state and zip, Phone fields – Phone Type and Phone number, Event fields – Event Type and Event Date.
* All fields are required and validated according to the following Reg Expressions:
  + First Name, Middle Name, Last Name, City, State: /^\w{1,20}$/ (Alphanumeric, max 20 characters)
  + Address Type, Phone type, Event Type: /^\w{1,10}$/ (Alphanumeric, max 10 characters)
  + Address Line: /^.{1,100}$/ (any string, max 100 characters)
  + Zip: /^\d{5,6}$/ (5 or 6 digit number)
  + Phone: /^\d{10}$/ (10 digit number)
  + Date: HTML 5 Date (YYYY-MM-DD)
* The form data is POSTed in JSON format to the server. On the back-end, the form data is received and parsed using the “body-parser” Node JS middleware. DB queried are made through Javascript Promises using the “my-sql” package.
* On the DB, first the contact names are inserted into the contact table, then using the contact ID (Foreign Key), the address table phone table and date tables are filled with the data entered by the user.

**Modify Contact: (API: POST /edit/:id)**

* Edit button are provided for each contact which appears as search result.
* The fields are same as Adding a contact, additionally the user will be able to delete existing address fields, phone fields and event fields or add new fields.
* All fields need to be filled according to the Reg Expressions.
* When the user modifies an address, phone or date field, the modified data is send along with the table unique ID as an HTML hidden input element. On the back-end, the corresponding table with unique ID is modified.
* When the user deletes an address, phone or date field, a HTML hidden input element is created with the ID as value and is sent along with the form. On the back-end, the corresponding table with unique ID is deleted.

**Delete Contact: (API: GET /delete/:id)**

* Delete button are provided for each contact which appears as search result.
* On the back-end, the address, phone and event information for the contact is deleted first, then the contact. This prevent foreign key violations.

**Entity – Relationship Diagram:**

