B.E. PROJECT ON

**Minimum Viable Product Generator**

SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENTS OF AWARD OF

B.E. (COMPUTER ENGINEERING)

DEGREE OF UNIVERSITY OF DELHI

SUBMITTED BY:

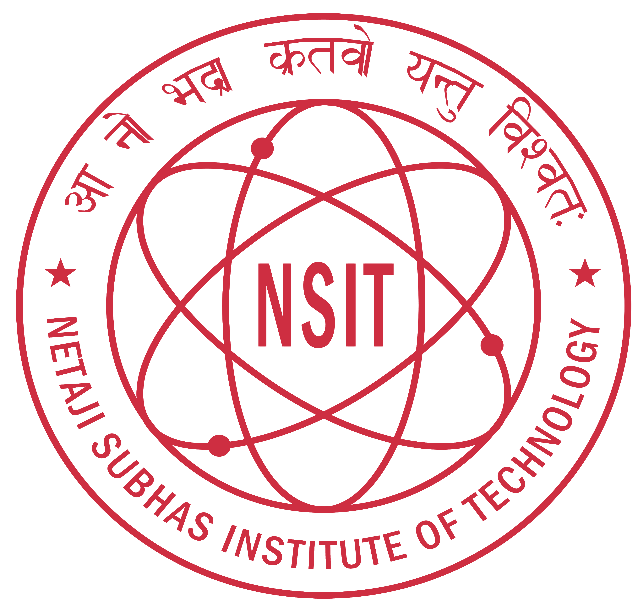
Akanshi Gupta 216/CO/12

Chitrasoma Singh 254/CO/12

Divjot Singh 262/CO/12

GUIDED BY:

Dr. Sushma Nagpal



COMPUTER ENGINEERING (COE)

NETAJI SUBHAS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF DELHI

2012-16



**CERTIFICATE**

The project titled **“Minimum Viable Product Generator”** by **Akanshi Gupta (216/CO/12), Chitrasoma Singh (254/CO/12) and Divjot Singh (262/CO/12)** is a record of bonafide work carried out by them, in the Division of Computer Engineering, Netaji Subhas Institute of Technology, New Delhi, under the supervision and guidance of Dr. Sushma Nagpal in partial fulfilment of requirement for the award of the degree of Bachelor of Engineering in Computer Engineering, University of Delhi in the academic year 2015 - 2016.

**Dr. Sushma Nagpal**   
Division of Computer Engineering   
Netaji Subhas Institute of Technology   
New Delhi

Dated:

**CANDIDATES’ DECLARATION**

This is to certify that the work which is being hereby presented by us in this project titled “**Minimum Viable Product Generator**” in partial fulfilment of the award of the Bachelor of Engineering submitted at the Department of Computer Engineering , Netaji Subhas Institute of Technology Delhi, is a genuine account of our work carried out during the period from January 2016 to May 2016 under the guidance of Dr. Sushma Nagpal, Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi. The matter embodied in the project report to the best of our knowledge has not been submitted for the award of any other degree elsewhere.

Dated:

|  |  |  |
| --- | --- | --- |
| Akanshi Gupta | Chitrasoma Singh | Divjot Singh |

This is to certify that the above declaration by the students is true to the best of my knowledge.

Dr. Sushma Nagpal

**ACKNOWLEDGEMENT**

No significant achievement can be done by solo performance especially when starting a project from ground up. This B.E. Project, on such a revolutionary idea, has by no means been an exception. It took many very special people to enable it and support it. Here we would like to acknowledge their precious co-operation and express our sincere gratitude to them.

**Dr. Sushma Nagpal** has again been very supportive and involved in yetanother student project. It was her support that helped the project to start in its earliest and most vulnerable stages. Her name opened many doors for us and persuaded many people. She was always found with energy and enthusiasm to make sure that we were provided everything we needed. No amount of words can express thanks to her. She was the one who backed us in providing any assistance we needed during the project work.

We are also thankful to our friends who motivated us at each and every step of this project. Without their interest in our project we could not have been gone so far.

And the most of all, we would like to thanks our wonderful parents who motivated us from day one of the project. You were the lights that lead us.

It was a great pleasure and honour to spend our time with all of them and there could be no better payment for the efforts put into completing this B.E. Project than their valuable presence. They are all very special to us.

**ABSTRACT**

Digital world is undergoing and experiencing the biggest change in the history, with not only the internet reaching almost every corner of the world, but also with the increasing use of web and mobile apps more than ever. New start-up ideas are evolving to make almost all services like entertainment, socializing, shopping, payments etc. just a click away through them. Application development for these platforms requires a talented team of software engineers working on each part of the software stack. This requires a lot of capital that is usually not available with start-ups initially.

Application developers have started focusing on developing search engines apart from traditional applications. They are focused search engines for entities like doctors, restaurants, movie bookings, service workers etc. A common pattern has been observed in their functionality that provides a great room for automation in their development process. MVP Generator focuses on automating this process, using component driven and agile software development methods to achieve high level of modularity. This can help the startups in saving cost and time, getting funds early and provide boilerplate code to experienced teams to work on.

**TABLE OF CONTENTS**

1. Detailed Problem Statement

1.1. Introduction

1.2. Motivation

1.3. Problem Statement

1.4. Definitions

1. Literature Survey
2. Software Requirement Specification
   1. Introduction

3.1.1. Purpose

3.1.2. Scope

3.1.3. Technologies Used

3.1.4. Overview

3.2. Overall Description

3.2.1. Product perspective

3.2.2. Product functions

3.2.3. User characteristics

3.2.4. Constraints

3.2.5. Assumptions

3.3. Specific Requirements

3.3.1. External Interface Requirements

3.3.2. Functional Requirements

3.3.3. Performance Requirements

3.3.4. Design Constraints

3.3.5. Non Functional Requirements

3.3.6. Other Requirements

3.4. Change Management Process

4. Model Proposed

5. Conclusion and Results

5.1. Result

5.2. Conclusion

6. Future Scope

7. References

**LIST OF FIGURES**

|  |  |
| --- | --- |
| Figure No | Description |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |

**Chapter 1  
Detailed Problem Statement**

1.1. Introduction

1.2. Motivation

1.3. Problem Statement

1.4. Definitions

* 1. **Introduction**

With internet reaching the next billion people, new startup ideas are evolving to provide more services on the backbone of internet and smartphones. We are probably witnessing the biggest shift in technology since the emergence of internet. The use of web and mobile apps is growing at a tempestuous pace. [1] As a matter of fact, apart from games and content consuming applications, search engines are a major part of application industry. From finding restaurant [2] to finding doctors [3], there’s an app for that. Digital world is progressively playing an important role in people’s lives as web and mobile apps are used more than ever for entertainment, socializing, shopping and payment purposes.

However, these applications aren’t as simple as static websites of 2000s. These often have a cloud-computing backed data-store, a server equipped with load-balancers, and complex UI loading content dynamically using swift HTTP transfers. The development of such an application usually requires a team of software engineers working on each part of the software stack. This isn’t a big problem for established companies with enough capital to invest in such applications, however being seed funded on basis of quality of Minimum Viable Product (MVP) is the only chance for a startup to break ice in the industry.

From application stores of mobile operating systems like iOS and Android, we observed that a good share of applications follow a common pattern when it comes to functionality. They are essentially focussed search engines for entities like restaurants, doctors, service workers etc. This pattern makes room for some automation in development process, which can not only help startups to get funded early, but also assist experienced teams with some boilerplate code to work on.

MVP Generator aims to do the same with the help of component driven development [4], industry best practices, and use of latest technology stacks that assure high level of modularity with good performance in order to generator such MVPs based on descriptions given about the entity their focussed search engine application is about.

* 1. **Motivation**

Modern lifestyle is being shaped by internet and mobile applications. Our initial survey of top 50 applications in Apple App Store and Google Play Store showed that 10% of applications are search engines. Applications like restaurant finder [2], car reviews [4], service worker finders [3] are top in that list.

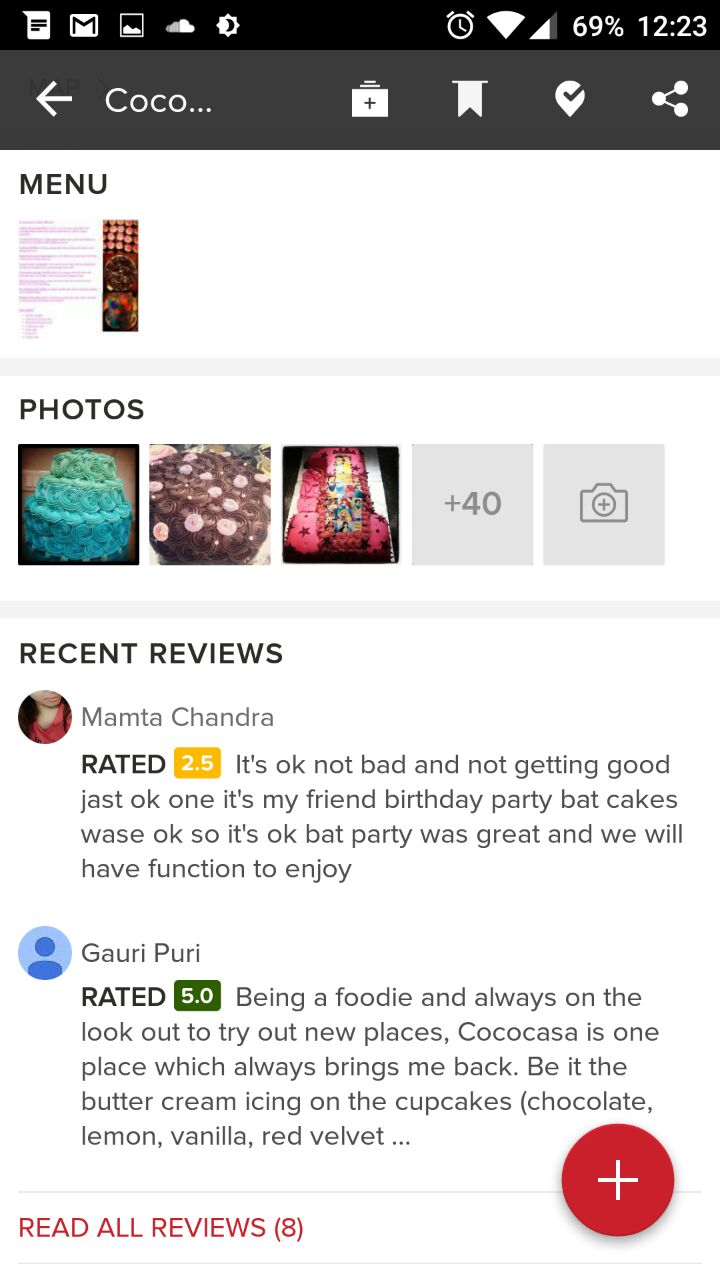
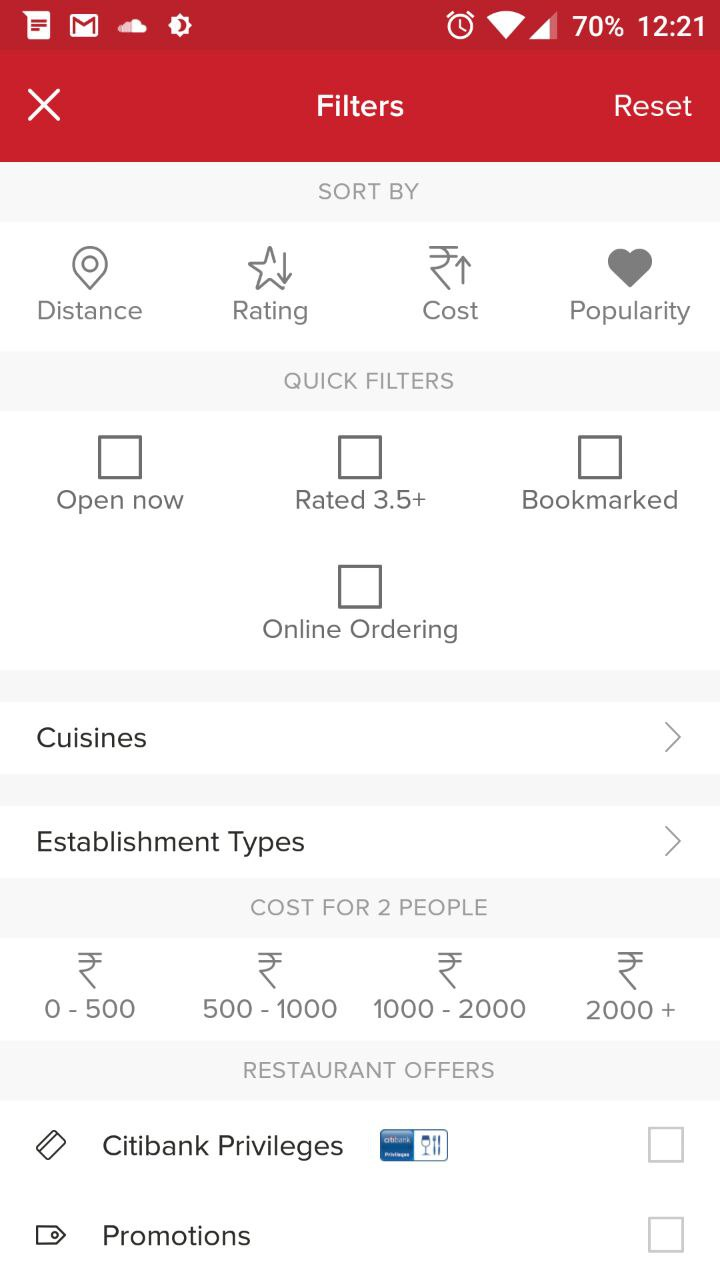
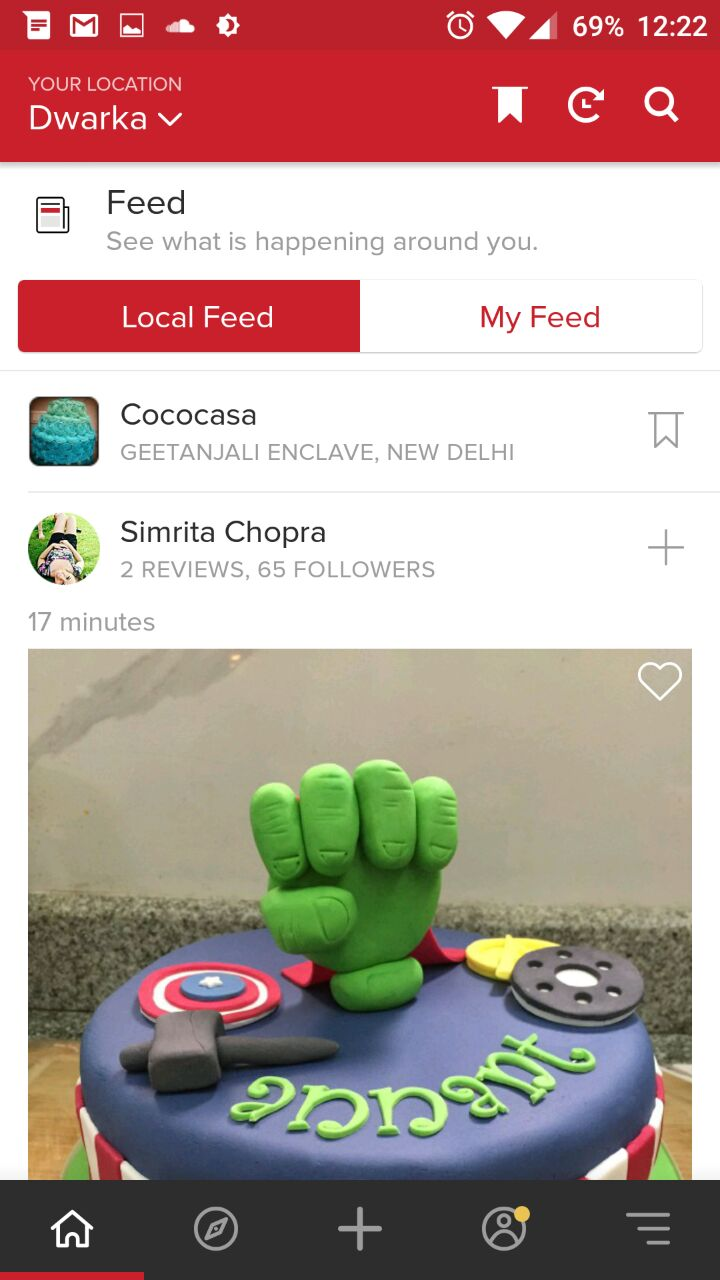
Often, such applications have a web application too, for desktop and laptop users. We observed that they all have common user interface involving following primary screens:

* Home feed showing popular or trending entities
* Entity page with reviews made by registered users
* Search page allowing users to use filters to fetch desired entity

Our case study of a few such popular applications highlights these features very well.

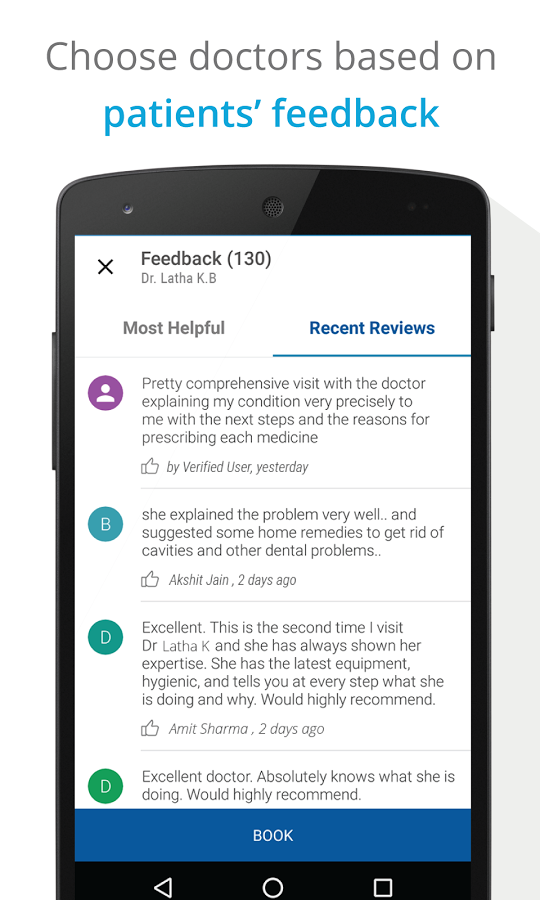
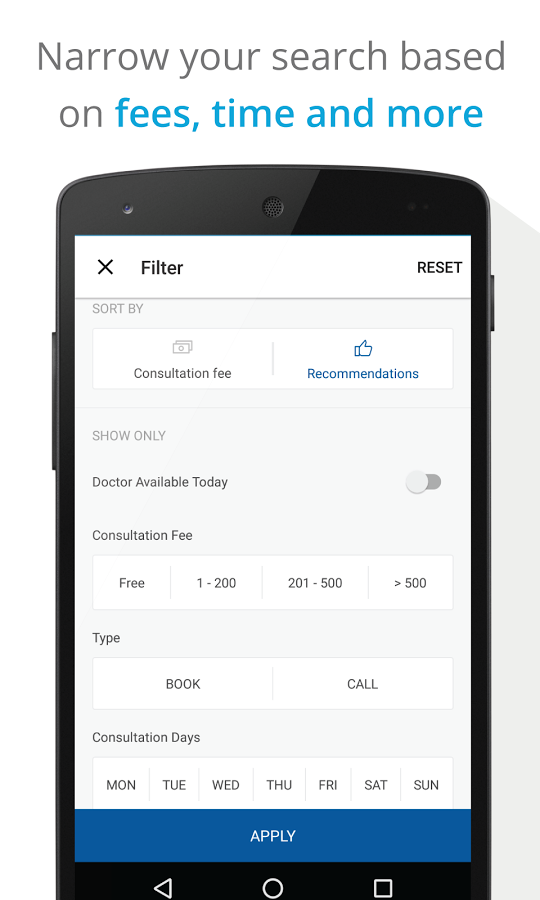
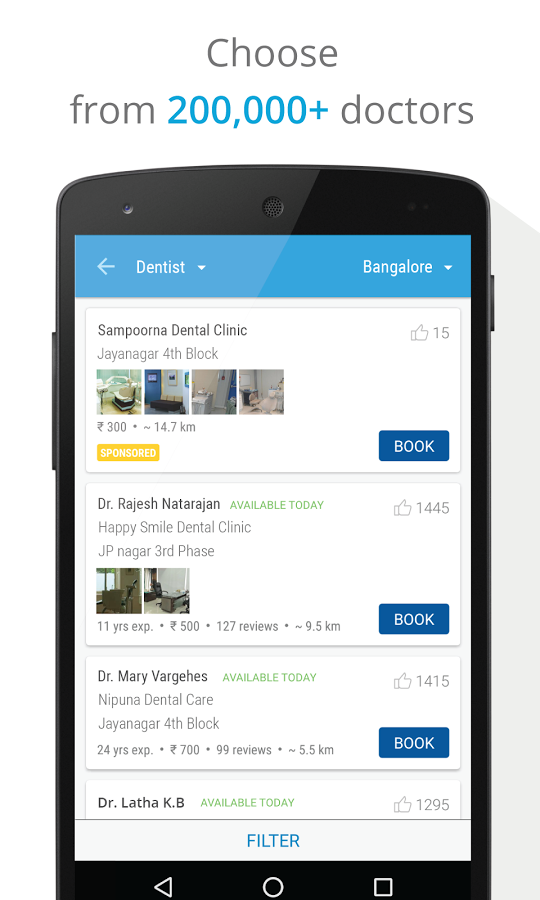
**Zomato (Restaurant Finder)**

Zomato is a restaurant search and discovery app and website, providing in-depth information for over 1.4 million restaurants across 23 countries. Zomato is used by consumers globally to discover, rate and review restaurants, as well as create their own personal network of foodies for trusted recommendations. It lets its users to search for meals through a number of filters, including location, price, cuisine, etc.



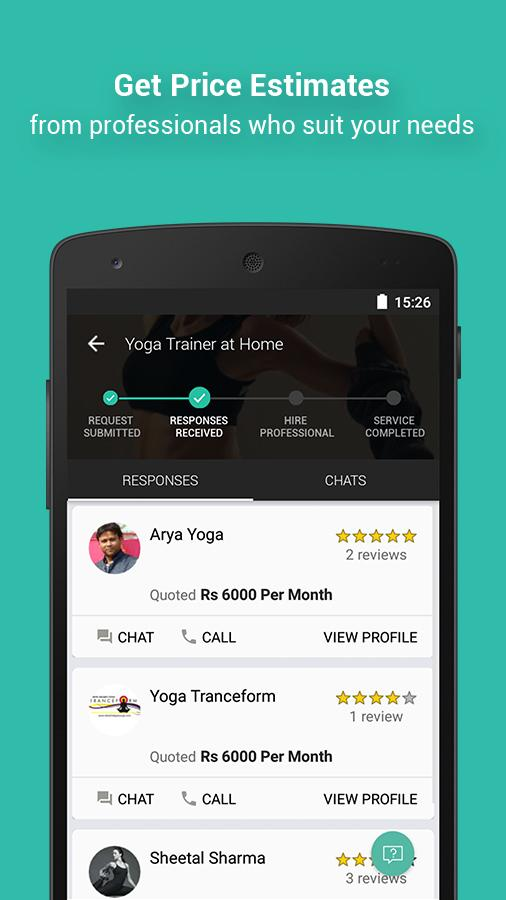
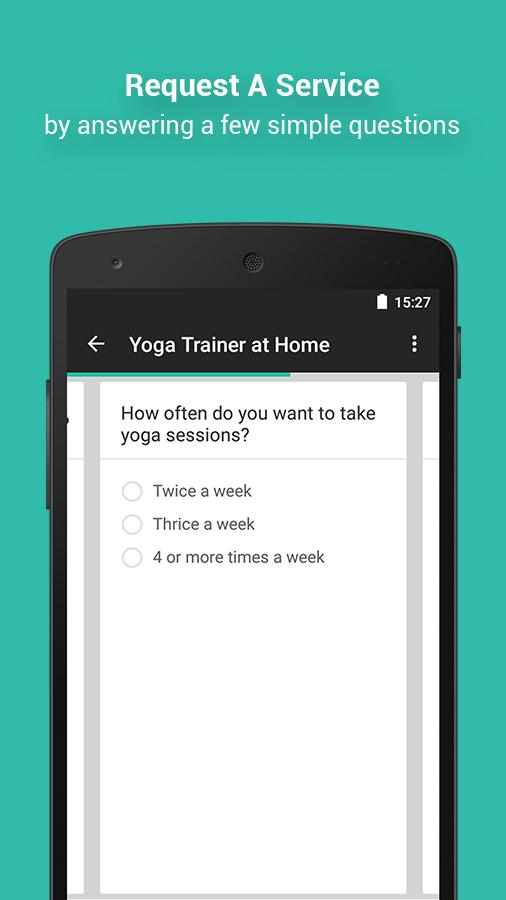
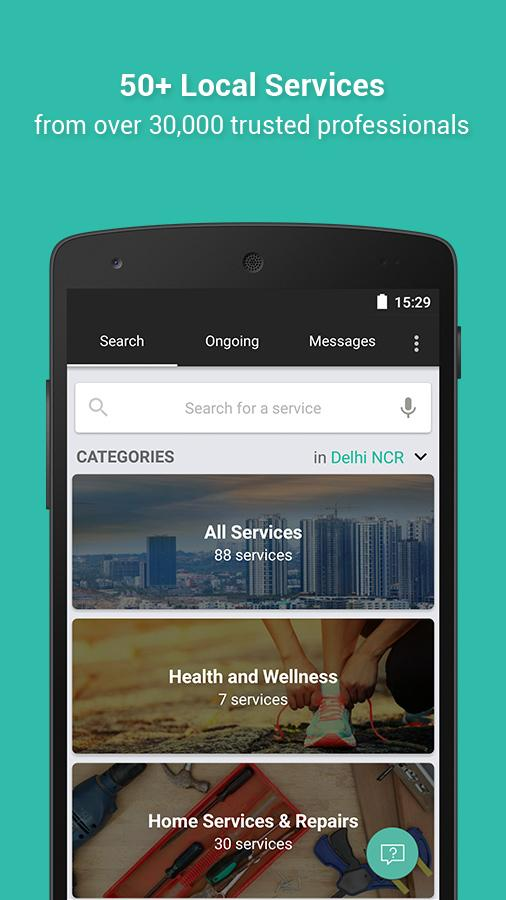
As you can see, the common features as we mentioned exist in Zomato. Note how the design is followed by the following applications too, even though they serve a very different purpose, i.e. the entity they search around is very different from restaurants.

**Practo (Doctor Finder)**



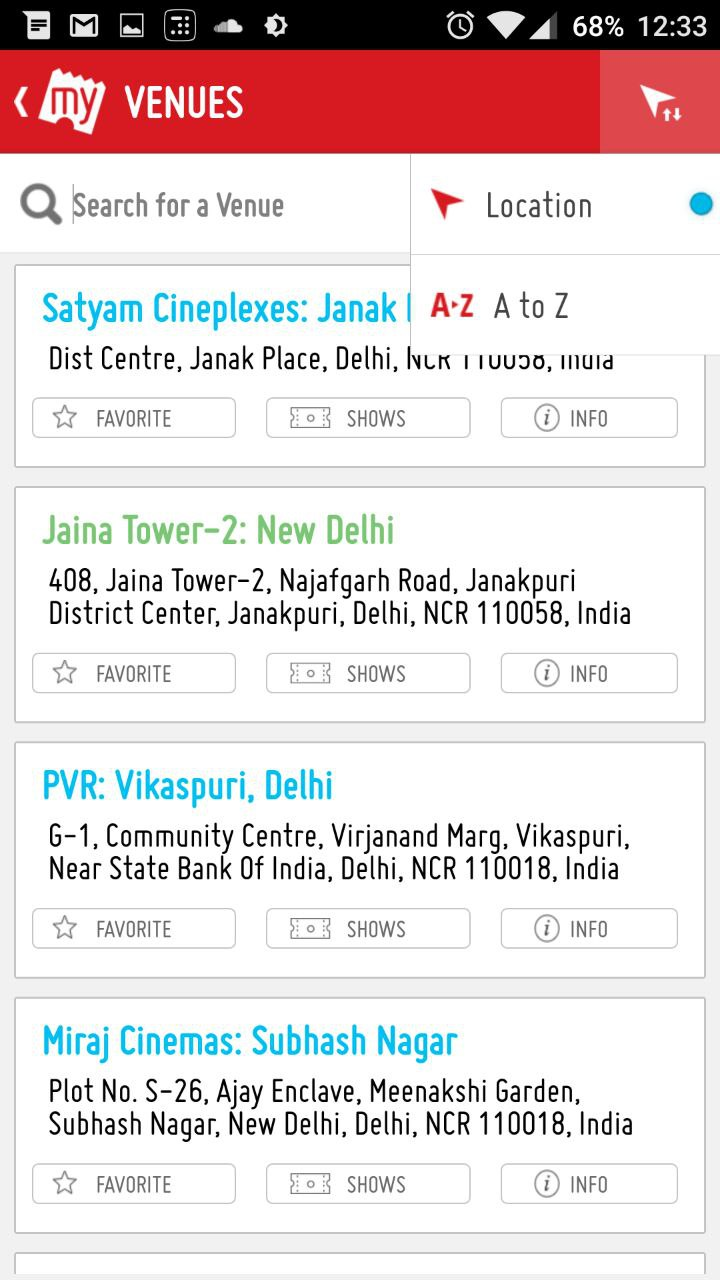
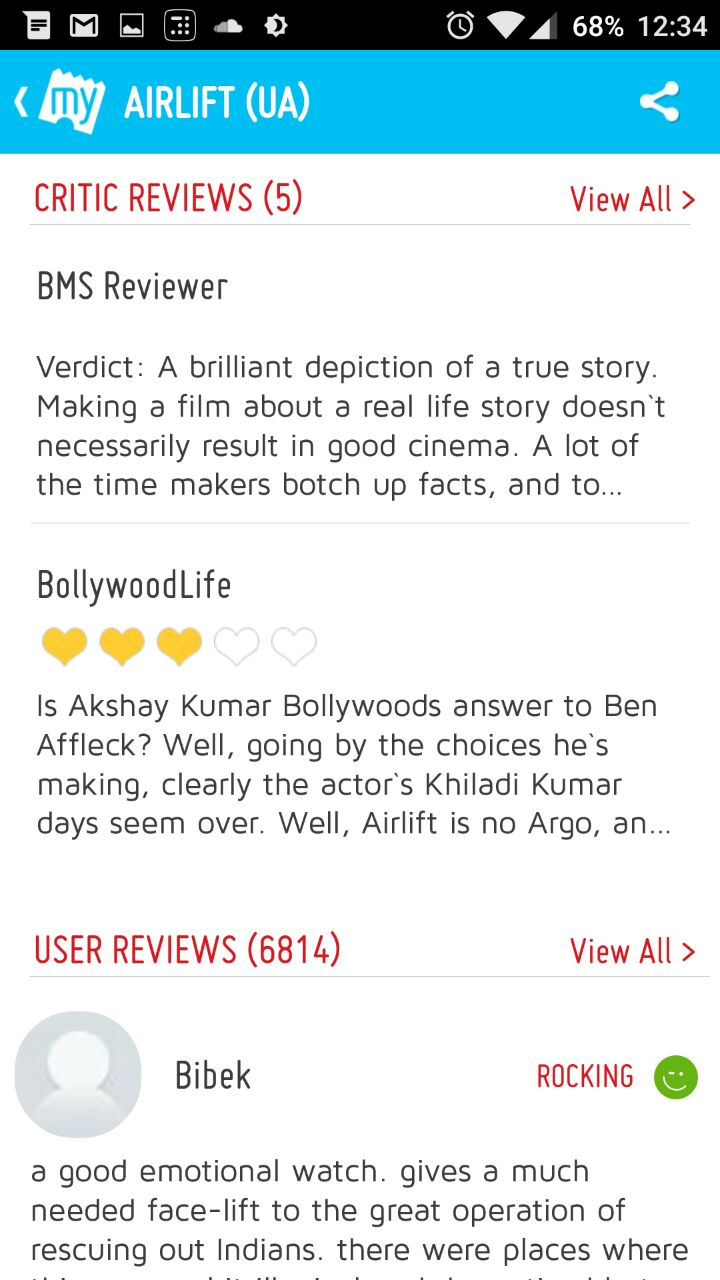
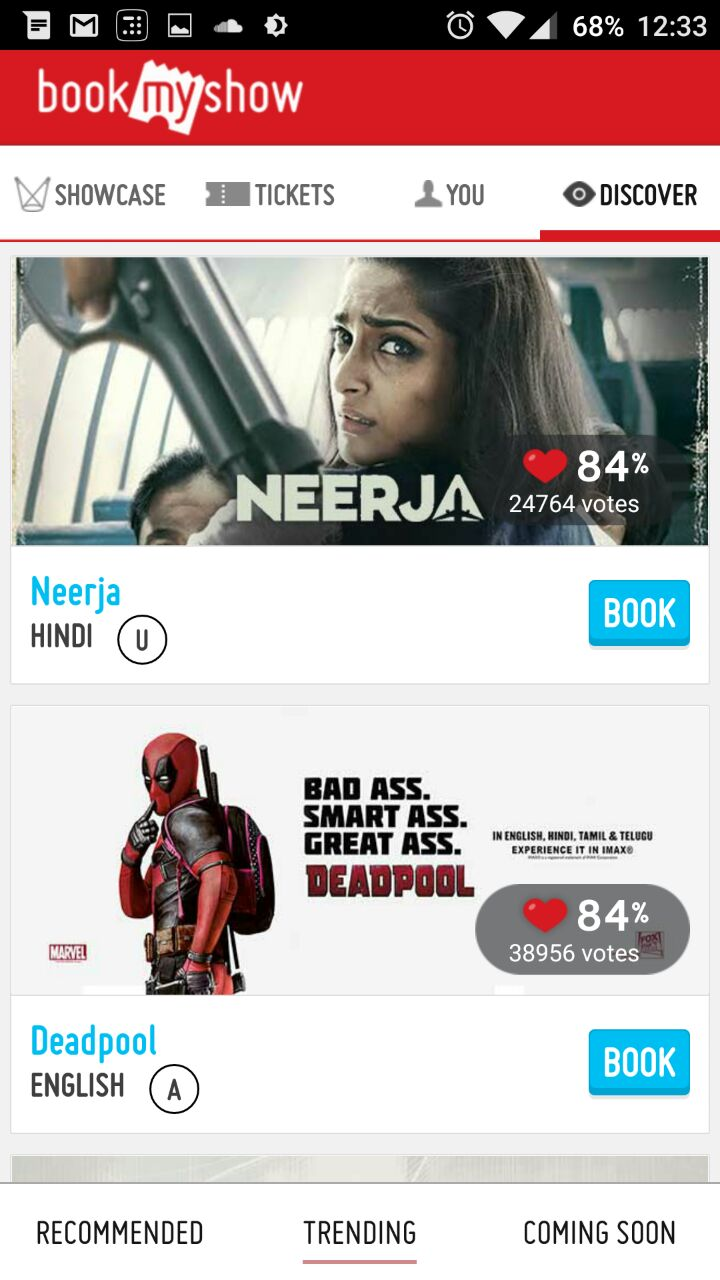
The next application is Practo. Note how the features as well as design is very similar to Zomato.

**Urbanclap (Service Worker Finder)**



This application has slight variations in design but the features are same.

**Bookmyshow (Movie Booking App)**



Once again, the feature set is very similar even though the entity is way too different.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Application/Features | Home Feed | Reviews | Searching | Elaborate Filters |
| Zomato |  |  |  |  |
| Practo |  |  |  |  |
| Urban Clap |  |  |  |  |
| BookMyShow |  |  |  |  |
| Car Trade |  |  |  |  |
|  |  |  |  |  |

These common features in search centric applications yet diverse deployment devices show how a project manager needs a developer for each mobile platform, one for web development and one for backend which ties everything together. This is all just to build a Minimum Viable Product. The pattern allows us to use automation techniques to cater this niche yet huge segment of applications, which ultimately lead to idea of creating MVP Generator.

* 1. **Problem Statement**

Develop a code generator program that generates an application for search centric services. The generator accepts inputs from user like application name, system details, and description of the entity around which the application is to be built, provided as a configuration file. The generator should work on Linux and OSX systems at least.

Generated code should be modular enough so that user can tweak it with minimum file changes. The cohesion should be high for the modules.

**INPUT TO THE SYSTEM**

A list of inputs like:

* Application name
* Name of entity
* Theme of application
* Database details
* Etc.

And a configuration file with details of search filters and attributes of entity that are unique to it.

**OUTPUT FROM THE SYSTEM**

A folder is to be created with server and application source code that works as per the inputs given by user.

* 1. **Definitions**

|  |  |
| --- | --- |
| **Word** | **Meaning** |
| Cohesion | Cohesion refers to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationship between pieces of functionality within a given module. |
| URL | A Uniform Resource Locator (URL), commonly informally termed as a web address is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. |
| API | In computer programming, an application programming interface (API) is a set of routines, protocols, and tools for building software and applications. |
| ORM | Object-relational mapping in computer science is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. |
| DOM | The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents. The nodes of every document are organized in a tree structure, called the DOM tree. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. |
| Pure Function | In computer programming, a function may be considered a pure function if both of the following statements about the function hold:   1. The function always evaluates the same result value given the same argument value(s). 2. Evaluation of the result does not cause any semantically observable side effect or output, such as mutation of mutable objects or output to I/O devices. |
|  |  |

**Chapter 2**

**Literature Survey**

2.1. Application Development

2.2. Component Based Software Development

2.3. Agile Software Development

2.4. React

2.5. Redux

2.6. NodeJS

2.7. Express

**2.1. Application Development**

Application development is a big software engineering task involving processes like prototyping, server development, database management, testing and so on.



The development process is initially subdivided as frontend (client side) and backend (server side) development. Frontend development deals with user and interactions with layout and user interface of the application:

1. Frontend: User interface is built using some toolkit that can help in rendering them as per the designs. Hyper Text Markup Language (HTML) and Cascading Style Sheets (CSS) help in building User Interfaces (UIs) in web domain, while each mobile application ecosystem has its own rendering logics. The UI is further composed of components that interact with the components or the server (backend). A user triggers events using these components. For backend interactions, applications communicate using Hyper Text Transfer Protocol (HTTP) requests. A payload is sent in the HTTP message, often in JavaScript Object Notation (JSON) in modern applications. This is the boundary of frontend.
2. Backend: This is the server side, which is an ongoing application on a remote machine which listens for HTTP requests on a certain port number, usually 80 for HTTP applications. Upon receiving requests, server processes it and responds back to client. Server code is again subdivided into software modules that are highly cohesive in nature for better software maintainability. There’s often a router layer that deals with routing different requests for different URLs to different functions of different modules. Usually these modules lie in the API layer. Often an ORM is used to handle database queries in a manageable way.

The two ends are core parts of application development. We’ll go through development strategies used for high maintainability and flexibility in industry.

**2.2. Component Based Software Development**

Component-based development (CBD) is a procedure that accentuates the design and development of computer-based systems with the help of reusable software components. With CBD, the focus shifts from software programming to software system composing.

The process of component- and component-based system development differs in many significant ways from the “classical” development process of software systems. The main difference is in the separation of the development process of components from the development process of systems. [5] This fact has a significant impact on the development process. Since the component-based approach is a relatively young approach in software engineering, the main emphasis in the area has been in development of technologies.

Component-based development techniques involve procedures for developing software systems by choosing ideal off-the-shelf components and then assembling them using a well-defined software architecture. With the systematic reuse of coarse-grained components, CBD intends to deliver better quality and output.

Component-based development is also known as component-based software engineering (CBSE).

Component-based development techniques consist of non-conventional development routines, including component evaluation, component retrieval, etc. It is important that the CBD is carried out within a middleware infrastructure that supports the process, for example, Enterprise Java Beans.

The key goals of CBD are as follows:

* Save time and money when building large and complex systems: Developing complex software systems with the help of off-the-shelf components helps reduce software development time substantially. Function points or similar techniques can be used to verify the affordability of the existing method.
* Enhance the software quality: The component quality is the key factor behind the enhancement of software quality.
* Detect defects within the systems: The CBD strategy supports fault detection by testing the components; however, finding the source of defects is challenging in CBD.

Some advantages of CBD include:

* Minimized delivery:
  + Search in component catalogs
  + Recycling of pre-fabricated components
* Improved efficiency:
  + Developers concentrate on application development
* Improved quality:
  + Component developers can permit additional time to ensure quality
* Minimized expenditures

The specific routines of CBD are:

* Component development
* Component publishing
* Component lookup as well as retrieval
* Component analysis
* Component assembly

**2.3. Agile Software Development**

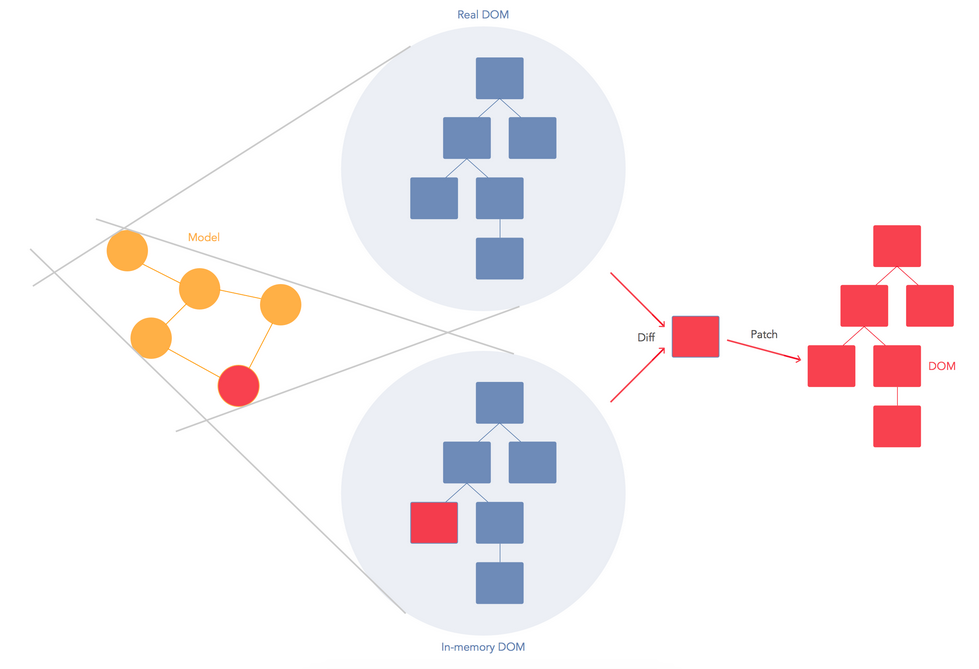
Software development is an organized thrives to deliver products in faster, better and cheaper ways. There have been many studies and suggestion in improving the development process. Recently, this interest has paved way to a new software development method called Agile Software Development. To overcome the fast changing organizational business needs using traditional methods agile methods were introduced. Agile methods aid in and focus on developing solutions more quickly and efficiently. Agile methods highlight customer satisfaction by structuring the development process into iterations where in each iteration produces sizeable amount of working code and artifacts of interest to customers. [6] It‘s a set of software development methods based on iterative and incremental development process, where requirements and development evolve through collaboration between self-organizing, cross-functional teams that allows rapid delivery of high quality software to meet customer needs and also accommodate changes in the requirements.[7] Compared to traditional software engineering, agile software development mainly targets complex systems and projects with dynamic, non-deterministic and non-linear characteristics, where accurate estimates, stable plans, and predictions are often hard to get in early stages—and big up-front designs and arrangements would probably cause a lot of waste, i.e., are not economically sound.

The Agile Manifesto is based on twelve principles: [8]

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

**2.4. React**

In order to use components in the user interface, ReactJS acts as a great library for the same. It renders user created components that can have their own state and can be in a component tree. The tree is saved in memory for optimizations before reflecting the changes to the tree directly to the DOM, which is often expensive on memory and can cause unpleasant user experience.



React combines DOM generation and display logic. It is designed to re-render the whole app on every update via state, and its implementation consists of Virtual DOM and synthetic events. These best practices make React a powerful tool for developing your web applications.

The framework cannot know how to separate your concerns for you. It should only provide powerful expressive tools for the user to do it correctly.

Instead of templates, we can use components to separate your concerns. React components are highly cohesive building block for UIs loosely coupled with other components. In React these components are written in JavaScript, which is highly powerful.

React Components are thus :

* Reusable
* Composable
* Highly Cohesive
* Unit testable

Some definitions:

*JSX* : JSX is an optional preprocessor to let you use HTML-like syntax. With JSX, it’s easy for designers to contribute code.

*State* : Re-rendering the whole app on every update makes React stand out from other front-end libraries/frameworks.

Data changing over time can make it really difficult for the front-end developer. When data changes, React re-renders the entire component. So the React components describe your UI at any point in time, just like a server-rendered app. This means every place data is displayed is guaranteed to be up-to date.

*Virtual DOM* : In memory snapshot of DOM used by React to diff and update only those components that are required.

React is optimized for performance and memory footprint.

On every update to the component, React:

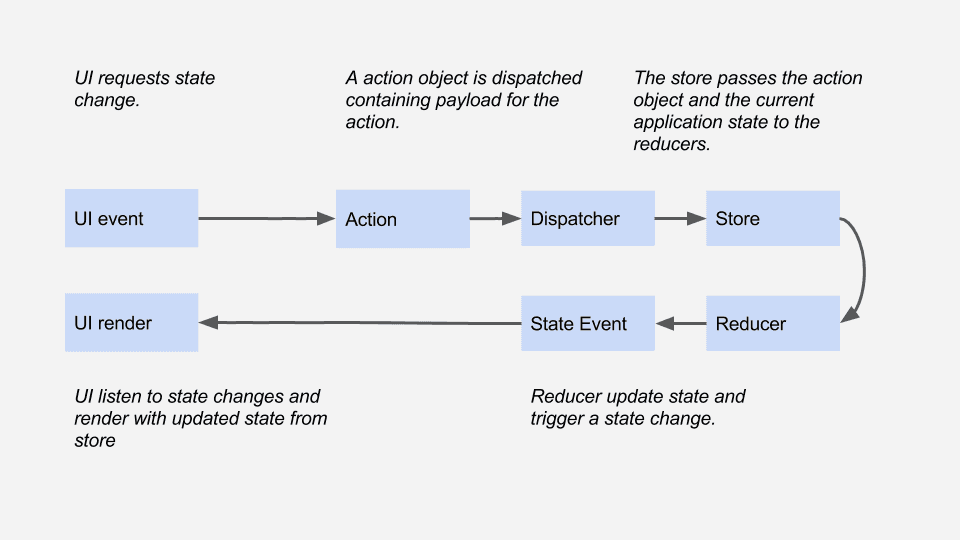
* Builds a new virtual DOM subtree
* Diffs it with the old one
* Computes the minimal set of DOM mutations and puts them in a queue
* And batch executes all updates

React can run in Node.js with optimizations based on app structure. React also has SVG, VML and <canvas> support, and can run the whole app in a Web Worker (experimental).

The component based architecture can be created using React, which allows for generating applications by simply replacing, removing or adding components.

**2.5. Redux**

Redux is a microlibrary often used on top of React to manage the state with a minimal API but completely predictable behaviour i.e. it’s a predictable state container for JavaScript apps.



Redux is:

* Simple, conceptually and in file size (2kb)
* Has predictable state transitions
* Single source of truth for UI state
* Highly performant
* Easily testable

To understand Redux, we need to look into the following concepts :

* Actions and action creators
* Reducers
* Store
* Middleware

*Actions* : Actions are payloads of information that send data from your application to your store. They are the only source of information for the store. You send them to the store using store.dispatch().

*Action Creators* : Action creators are the functions that return the body of an action.

*Reducers* : Actions describe the fact that something happened, but don’t specify how the application’s state changes in response. This is the job of a reducer. It updates the state according to those actions. Reducers are pure functions with the signature (previousState, action) => newState.

*Store* : Store is a wrapper around a JavaScript object (state). A store has two key methods: getState, and dispatch. If you are using something like react-redux, you don’t have to deal with these functions directly.

*Middleware*: It provides a third-party extension point between dispatching an action, and the moment it reaches the reducer. People use Redux middleware for logging, crash reporting, talking to an asynchronous API, routing, and more.  
Recommended middleware: thunk, redux-promise and batched-updates.

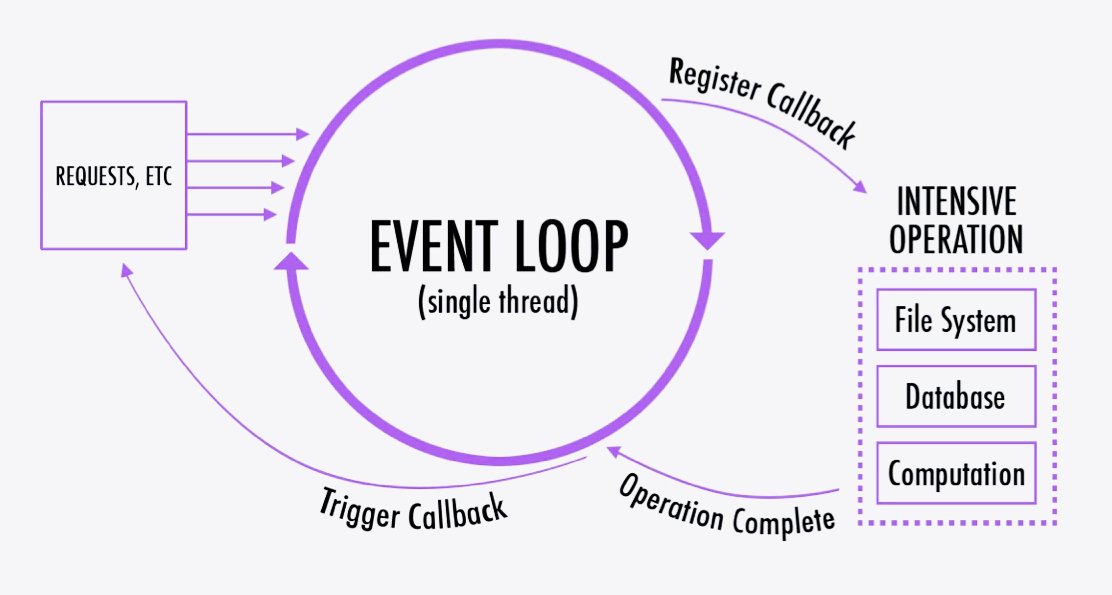
Hence, Redux is powerful for building predictable, testable, maintainable interfaces. This helps us further to maintain state in a highly cohesive system.

**2.6 NodeJS**

Node.js is an open-source, cross-platform runtime environment for developing server-side web applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, Linux, FreeBSD, NonStop, IBM AIX, IBM System z and IBM i. Its work is hosted and supported by the Node.js Foundation, a collaborative project at Linux Foundation.

Node.js provides an event-driven architecture and a non-blocking I/O API designed to optimize an application's throughput and scalability for real-time web applications. It uses Google V8 JavaScript engine to execute code, and a large percentage of the basic modules are written in JavaScript. Node.js contains a built-in library to allow applications to act as a stand-alone web server.

Node.js is used by IBM, Microsoft, Yahoo!, Walmart, Groupon, SAP, LinkedIn, Rakuten, PayPal, Voxer and GoDaddy.



**2.7. Express**

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates a rapid development of Node based Web applications.

Following are some of the core features of Express framework:

* Allows to set up middlewares to respond to HTTP Requests.
* Defines a routing table which is used to perform different action based on HTTP Method and URL.
* Allows to dynamically render HTML Pages based on passing arguments to templates.

**Chapter 3**

**Software Requirements Specification**

3.1. Introduction

3.1.1. Purpose

3.1.2. Scope

3.1.3. Technologies Used

3.1.4. Overview

3.2. Overall Description

3.2.1. Product perspective

3.2.2. Product functions

3.2.3. User characteristics

3.2.4. Constraints

3.2.5. Assumptions

3.3. Specific Requirements

3.3.1. External Interface Requirements

3.3.2. Functional Requirements

3.3.3. Performance Requirements

3.3.4. Design Constraints

3.3.5. Non Functional Requirements

3.3.6. Other Requirements

3.4. Change Management Process

**3.1. Introduction**

This document is a Software Requirement Specification (SRS) for the Minimum Viable Product (MVP) Generator. This is the initial draft for the SRS and it will be used for the extensions. This document is prepared following the standard IEEE conventions for software requirement specification.

Minimum Viable Product (MVP) generator is a tool that could automate the task of developing applications like Urban Clap (Service Finder), Zomato (Restaurant Finder), Practo (Doctor Finder), etc., and provide a Minimum Viable Product (MVP) to the end user that he/she could use to perform instant prototyping, pitch the idea to investors, and finally build the production code on top of the generated app.

The purpose of this project is to build an easy-to-use web service providing tool which a person with an idea but lack of computer engineering skills can use to immediately analyse his/her idea directly looking at the prototype.

**3.1.1. Purpose**

The aim of this document is to specify complete description of the Minimum Viable Product (MVP) Generator to be developed. It is the basis for agreement between suppliers and customers about the product to be developed. Through this document, the workload needed for development, validation and verification will ease. To be specific, this document is going to describe functionality, external interfaces, performance, attributes and the design constraints of the system which is going to be developed. Therefore, intended reader groups for this software requirement specification are customers or users.

**3.1.2. Scope**

I don’t know what language to use but this is how I have seen open source projects show their scope. Please research other SRS to get better idea, preferably look at YeoMan.

What this project isn’t

A non-modular code generator.

Hard to customize code generator

Responsible for generating problem specific code.

**3.1.3. Overview**

We are going to focus on describing the system in terms of product perspective, product functions, user characteristics, assumptions and dependencies on the following section of this document. Next, we will address specific requirements of the system, which will enclose external interface requirements, requirements of the system, performance requirements, and other requirements.

**3.2. Overall Description**

This section gives background information about specific requirements of the minimum viable product generator to be developed in brief. Although we will not describe every requirement in detail, this section will describe the factors that affect the final product.

**3.2.1. Product Perspective**

This software product is eventually intended to automate the process of development of modern day web based applications in a modular and component based approach. Product will be deployed to website and mobile platforms and users of the product will access it on these platforms. Website and mobile platforms will serve as the interface for the user data and the execution of provided functionalities.

This product generator cuts down the development process to merely seconds. The end-user will supply an options file describing the entity and other aspects of the project, and the generator would work its way out to compile entire service including Server, Database Configuration, Website & Mobile Applications.

The user will be able to interact with the generator using a Command Line Interface. For example (2.1.1):

$ mvpgenerate LawyerFinder options.json

> Parsing options file (options.json) (100%)

> Generating database modules (100%)

> Generating server modules (100%)

> Generating UI modules (100%)

> Optimizing & cleaning (100%)

Your project has been successfully generated in ./LawyerFinder directory !

**3.2.2. Product Functions**

This new product, minimum viable product generator, allow users to use functionalities which have been explained above in the introduction. Required functionalities of the product can be summarized in five categories; user management requirements, code editor requirements, debugger requirements, command line interface requirements and interface requirements. Overall description of the requirements can be found below;

**3.2.2.1. User Management Requirements**

This category of requirements is related to user authentication mechanism and workspace management of users. Each user will get to install this software on his/her system using command line interface and will be assigned to a workspace on the same system. Users will perform all the functionality over this workspace.

Project team is relatively small, hence GitHub is sufficient for code organization and collaboration. Telegram will be used for discussions while GitHub issues will be used for issue related discussions.

**3.2.2.2. Code Editor Requirements**

One of the most important functionality expected from such an application is a code editor which will ease the user’s life. Code editor will be the main interface that developers deal with. It supports variety of programming language with highlighting, syntax checking, auto-indentation and language specific auto-complete.

Vim editor will be used as the code editor with following plugins for JSX and JavaScript language:

vim-javascript by pangloss on GitHub

yajs.vim by othree on GitHub

vim-jsx by mxw on GitHub

**3.2.2.3. Debugger Requirements**

Debugger is the main tool that developers can test and debug their target program. Debugger of the product should allow setting and displaying breakpoints on the code. It will also provide functionality of stopping/continuing of the execution of debugger. Finally, it will provide an expression interface where user can enter an expression and observe the value of expression at each step.

React DevTools for Firefox & Chrome will be used to debug React App

**3.2.2.4. Terminal Requirements**

As an important part of the software development process, there should be a command line interface where user can work in old fashion and accomplish complicated tasks such as configuring NodeJS modules etc. The following scenario of generating a Lawyer Finder application tells how reports should be generated.

$ mvpgenerate LawyerFinder options.json

> Parsing options file (options.json) (100%)

> Generating database modules (100%)

> Generating server modules (100%)

> Generating UI modules (100%)

> Optimizing & cleaning (100%)

Your project has been successfully generated in ./LawyerFinder directory !

In case of error, proper details of the issue should be communicated to the user in a log file.

**3.2.2.5. Interface Requirements**

\*\*(to be added/edited with content)

**3.2.3. User Characteristics**

Users of this minimum viable product generator can be anyone of any field. Basically, a person with an idea but lack of computer engineering skills can use the utility to immediately to analyze his/her idea directly looking at the prototype. Also clear documentation and tutorials about the product feature will be provided.

**3.2.4. Constraints**

The project is expected to run well on unix based Operating Systems (Linux/ Mac OS X). Windows support isn’t required but would be preferable.

?? ismei aur kuch add karr saktey hain ??

\*\* need to write about portability and modularity of the product

**3.2.5. Assumptions**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

**3.3. Specific Requirements**

With this section and later, we will describe the requirements of the product in detail. Basically, we will categorize requirements in three which are namely external interface requirements, functional requirements and non-functional requirements. Except non-functional requirements, requirements of the product will be detailed under this section with brief information and later sample input-output sequence and low of events will be given.

**3.3.1. External Interface Requirements**

1. **User Interface Requirements :** Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
2. **Hardware Interface Requirements :** Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
3. **Communication Interface Requirements :** Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

**3.3.2. Functional Requirements**

**3.3.3. Performance Requirements**

**3.3.4. Design Constraints**

**3.3.5. Non Functional Requirements**

**3.3.6. Other Requirements**

**3.4. Change Management Process**

**Chapter 4**

**Model Proposed**

Our strategy to build the generator was to divide development process into two phases of Skeleton and Generator development.

Skeleton represents a basic structure of search centric application, including server side code. The more customizable skeleton is, the easier It becomes to generate code.

Second phase uses the Skeleton to tweak and adjust its code to fulfil the requirements of the user as described by the user.

**DFD goes here**

**ERD of Skeleton goes here**

**More technical description goes here**

**Chapter 5**

**Conclusion and Results**

The generator was successfully built meeting requirements as described in the SRS. We were able to generate diverse applications like School Finder, Gym Finder and a Banquet Hall Searcher.

**Chapter 6**

**Future Scope**

* The generator can support custom plugins that intercept the building process to inject, modify or even remove components to allow custom functionality provided to generator.
* Further, the generator can not only generate but update existing applications built using the generator to perform transformations to code like removal of certain components, replacement of component with some other alternative, etc.

**Chapter 7**

**References**

1. Mobile Internet Users In India 2016: 371 Mn by June, 76% Growth In 2015 - <http://dazeinfo.com/2016/02/08/mobile-internet-users-in-india-2016-smartphone-adoption-2015/>
2. India Restaurants - Find the best restaurants, cafés and bars in India on ... <http://zomato.com/>
3. Find & book appointments with doctors, diagnostic labs, salons, gyms & spas. Ask free health questions to doctors or get free tips from health experts. <https://www.practo.com/>
4. CarWale - Search New, Used Cars <http://carwale.com/>
5. Ivica Crnkovic, Michel Chaudron, Stig Larsson,“Component-based Development Process and Component Lifecycle”,2006
6. Kuda Nageswara Rao, G. Kavita Naidu, Praneeth Chakka, “A Study of the Agile Software Development Methods, Applicability and Implications in Industry”,2011
7. A B M Moniruzzaman, Dr Syed Akhter Hossain, “Comparative Study on Agile software development methodologies”,[4] Beck, Kent; et al. (2001).
8. "Principles behind the Agile Manifesto". Agile Alliance. Archived from the original on 14 June 2010. Retrieved 6 June 2010