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/CO12

Divjot Singh 262/CO/12

GUIDED BY:

Dr. Sushma Nagpal



COMPUTER ENGINEERING (COE)

NETAJI SUBHAS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF DELHI

2014-15

**Table of Content**

1. Detailed Problem Statement

1.1. Introduction

1.2. Motivation

1.3. Problem Statement

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

3.1.1. Purpose

3.1.2. Scope

3.1.3. Definitions and Acronyms

3.1.4. Technologies Used

3.1.5. References

3.1.6. 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**

**Chapter 1: Detailed Problem Statement**

1.1. Introduction

1.2. Motivation

1.3. Problem Statement

**1.1. Introduction**

The use of mobile devices and apps is growing at a tempestuous pace. Smartphones play progressively important roles in people’s lives as mobile apps are used more than ever for entertainment, socializing, shopping and payment purposes. It’s a new change to our lifestyle, apps becoming each year more and more creative and useful.

We are probably witnessing the biggest shift in technology since the emergence of internet. And the future looks bright in the app landscape as the rapid growth in smartphone and tablet sales drives an impressive growth of app downloads.

There are so many aspects that are required for the successful "making" of an app:

1. Planning

- Figure out what to put and not put in the app, know when to stop

- Decide on the audience

- Choose what platforms to support

- Imagine a design

- Prototype and refine the design

2. Development

- Implement the data structures, custom objects/classes, DB, create cohesive system

- Break down the functionality of your app into little parts, and implement them

- Design & Create the graphical resources

- Implement the UI

- Implement and refine the UX

3. Testing

- Spend your nights crying over some stupid bug that you cannot figure out.

- Try every combination of hardware and configuration to fix the bugs that could and will occur

We plan to automate the entire task of making of an app in a modular and component based approach so that a person with an idea but lack of computer engineering skills can also use this utility and immediately analyze his/her idea directly and reduce software development time.

**1.2. Motivation**

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.

**1.3. Problem Statement**

A common pattern is observed in modern day applications that we often use whether it is a restaurant finder app *Zomato* or a movie booking app *BookMyShow* and many more such applications are out there in the world of web apps. They are all a search engine around a particular entity.

All such apps have a home feed section where user can look at trending entities, sort and search entities based upon popularity and attributes of the entity itself and also a section where the user read and give reviews and comments around the entity.

The only thing these apps differ in is the “entity” they allow searching around but they share the same functionality and even aesthetics around the other features like reviewing, searching and contacting the entity via mail/phone/map directions and other mediums.

In order to automate the task of developing these applications which differ in entities, we developed a generator to 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.

**Chapter 2: Literature Survey**

2.1. Babel

2.2. Redux

2.3. React

2.4. Express

2.5. Component Based Software Development

2.6. Reactive Programming

2.7. Agile Software Development

\*\*few more to be added

**2.1. Babel**

Babel has made a big impact on the community. It allows us to use features from the future of JavaScript. It will transform your futuristic code to a format browsers understand. You can even use it to develop your own language features. Babel's built-in JSX support will come in handy here.

Babel provides support for certain experimental features from ES7 beyond standard ES6. Some of these might make it to the core language while some might be dropped altogether.

Babel can be used with Webpack easily throughbabel-loader. It takes our ES6 module definition based code and turn it into ES5 bundles. Install *babel-loader* with

*npm i babel-loader babel-core --save-dev*

*babel-core* contains the core logic of Babel so we need to install that as well.

To make this work, we need to add a loader declaration for babel-loader to the *loaders* section of the configuration. It matches against both .js and .jsx using a regular expression (/\.jsx?$/).

To keep everything performant we should restrict the loader to operate within *./app* directory. This way it won't traverse node\_modules.

**2.2. Redux**

Redux allows you to manage the state with a minimal API but completely predictable behaviour – a predictable state container for JavaScript apps.

Redux is a Flux implementation but that does not use Flux. It is inspired by functional programming and immutability (Elm, Clojure) and written by Dan Abramov.

Redux makes you think of your application as an initial state being modified by a sequential list of actions. Redux enables tools like logging, hot reloading, time travel, record and replay with no extra work.

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* : If one is familiar with Node.js middleware like Express or Koa, Redux middleware works similar to them except, it solves different problems.  
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. Redux makes you a better UI engineer by forcing you to handle state changes explicitly.

**2.3. React**

React is a library for creating user interfaces. It renders your UI and responds to events aka: the V in MVC. Because it is not as opinionated as many of other front-end JavaScript libraries, it plays nicely with your stack, whatever it maybe, Ruby, Java, Node.js or .NET.

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.

Templates encourage a poor separation of concerns. Angular style directives are a good example of this. The View Model tightly couples template to display logic. This is mentioned in Angular docs:

However isolated scope creates a new problem: if a transcluded DOM is a child of the widget isolated scope then it will not be able to bind to anything. For this reason the transcluded scope is a child of the original scope, before the widget created an isolated scope for its local variables. This makes the transcluded and widget isolated scope siblings.

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.

*Components* are:

* Reusable
* Composable
* Unit testable

Make components small and only put display logic in your components.

*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.

State means:

* No magical data binding.
* No more explicit DOM operations – everything is declarative.
* No model dirty checking.

*Virtual DOM* : You can’t just throw out the DOM and rebuild it on each update. It’s too slow and you’ll lose form state and scroll position. So Facebook built a Virtual DOM and events system.

It is optimized for performance and memory footprint. Let’s look how React use virtual DOM.

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 (new in 0.4) 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).

**2.4. 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.

**2.5. 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.

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).

Object-oriented modeling results in a plethora of fine-grained classes, objects and relationships. It is very hard to discover reusable parts among these smaller units. The idea behind CBD is to integrate the related parts and reuse them collectively. These integrated parts are known as components.

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

**Chapter 3: Software Requirement Specification**

3.1. Introduction

3.1.1. Purpose

3.1.2. Scope

3.1.3. Definitions and Acronyms

3.1.4. Technologies Used

3.1.5. References

3.1.6. 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 this utility to immediately analyze 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**