# MODELLING A WEB BASED REAL TIME APPLICATION BUILDER WITH REACTJS AND NODEJS TECHNOLOGIES

**Master of Science Thesis**

**EMRAH ÖZ**

**Eskişehir, 2017**

# MODELLING A WEB BASED REAL TIME APPLICATION BUILDER WITH REACTJS AND NODEJS TECHNOLOGIES

**EMRAH ÖZ**

**MASTER OF SCIENCE THESIS**

**Computer Engineering Program**

**Supervisor: Assoc. Prof. Dr. Özgür YILMAZEL**

**Eskisehir**

**Anadolu University**

**Graduate School of Science**

**December 2017**

**JÜRİ VE ENSTİTÜ ONAYI**

**(APPROVAL OF JURY AND INSTITUTE)**

**Emrah Öz**'ün, "**ReactJS ve NodeJS Teknolojileri ile Web Tabanlı Gerçek Zamanlı Uygulama Tasarlayıcı Modellenmesi**" başlıklı tezi 22/12/2017 tarihinde aşağıdaki jüri tarafından değerlendirilerek "Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliği" nin ilgili maddeleri uyarınca, **Bilgisayar Mühendisliği** Anabilim dalında Yüksek Lisans tezi olarak kabul edilmiştir

| | Ünvanı Adı Soyadı | İmza |
|---|---|---|
| Üye (Tez Danışmanı) | : Doç. Dr. Özgür Yılmazel | ……..... |
| Üye | : Yrd. Doç. Dr. Muammer Akçay | ……..... |
| Üye | : Yrd. Doç. Dr. Ahmet Arslan | ……..... |

……..... 

**Enstitü Müdürü**

## ABSTRACT

**Master of Science Thesis**

**MODELLING A WEB BASED REAL TIME APPLICATION BUILDER WITH REACTJS AND NODEJS TECHNOLOGIES**



**Emrah Öz**

**Anadolu University**
**Graduate School of Sciences**
**Computer Engineering Program**

**Supervisor: Assoc. Prof. Dr. Özgür YILMAZEL**


**2017, 76 pages**



A real-time web based application builder technique on reactjs and nodejs that users/developers can design and generate codes without writing manually by hand from scratch is presented. The designing application is running on the browser while it is generated by platform. The project has a graphical user interface that everybody can use, users do not need to be a software developer to design an application. With this work, code generation and hand coding compared side by side in dimensions “coding quality”, “speed”,” error / bug” rate. Code generation platform makes the development lifecycle efficient and easy. The quality and speed of the development process increases by %50 for recurrent tasks.



**Keywords:** ReactJs, Webpack, NodeJs, Code Generation, Rapid Application Development, Javascript

# ÖZET

**Yüksek Lisans Tezi**

## REACTJS ve NODEJS TEKNOLOJİLERİ İLE WEB TABANLI GERÇEK ZAMANLI UYGULAMA TASARLAYICI MODELLENMESİ

**EMRAH ÖZ**

**Anadolu Üniversitesi**
**Fen Bilimleri Enstitüsü**
**Bilgisayar Mühendisliği Anabilim Dalı**

**Danışman: Doç. Dr. Özgür YILMAZEL**
**2017, 76 sayfa**

Kullanıcıların/geliştiricilerin sıfırdan elle yazmadan kodları tasarlayıp üretebildikleri, reactjs ve nodejs üzerinde gerçek zamanlı web tabanlı bir uygulama oluşturucu tekniği sunulmuştur. Tasarlanan uygulama, platform tarafından oluşturulurken tarayıcıda çalışır haldedir. Sistem, herkesin kullanabileceği bir grafik kullanıcı ara birimine sahiptir ve uygulamayı kullanmak için yazılım geliştiricisi olması gerekmez. Bu çalışma ile "kodlama kalitesi", "hız", "hata" boyutları altında, kod üretme ve elle kodlama yanyana karşılaştırılmıştır. Kod üretme platformu uygulama geliştirme yaşam döngüsünü verimli ve kolay kılmaktadır. Yazılım geliştirme süreci kalitesi ve hızı %50 oranda tekrarlı tasklarda artmıştır.

**Anahtar Kelimeler:** ReactJs, Webpack, NodeJs, Kod Oluşturma, Hızlı Uygulama Geliştirme, Javascript

## ACKNOWLEDGEMENTS

I would like to thank to my supervisor Assoc. Prof. Dr. Özgür YILMAZEL for his patience, advice, criticism and encouragements for all those years.

I would like to thank Dr. Alp Vasfi ASUTAY, Louisiana State University Faculty of Engineering Center of Advanced Computer Studies Department, Louisiana / United States for discussing about advanced javascript topics.

I would like to thank to my wife (also an academic person) Çağla TERZİOĞLU ÖZ for helping me out about proofreading and keeping the thesis document clear.

I would like to thank to ReactJS, Webpack and NodeJS open source project contributors to all for creating such nice libraries and platforms.

Emrah Öz

December, 2017

22/12/2017

## ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ

Bu tezin bana ait, özgün bir çalışma olduğunu; çalışmamın hazırlık, veri toplama, analiz ve bilgilerinin sunumu olmak üzere tüm aşamaşarında bilimsel etik ilke ve kurallara uygun davrandığımı; bu çalışma kapsamında elde edilen tüm veri ve bilgiler için kaynak gösterdiğimi ve bu kaynaklara kaynakçada yer verdiğimi; bu çalışmanın Anadolu Üniversitesi tarafından kullanılan 'bilimsel intihal tespit programı'yla tarandığını ve hiçbir şekilde "intihal içermediğini" beyan ederim. Herhangi bir zamanda, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçları kabul ettiğimi bildiririm.

…….....

Emrah Öz

**TABLE OF CONTENTS**

## LIST OF TABLE

## LIST OF FIGURES

## ABBREVIATIONS

| | |
|---|---|
| **DB** | **:** Database |
| **DOM** | **:** Document Object Model |
| **ES** | **:** EcmaScript |
| **GUI** | **:** Graphical User Interface |
| **HMR** | **:** Hot Module Replacement |
| **HTML** | **:** Hypertext Markup Language |
| **JS** | **:** Javascript |
| **JSON** | **:** JavaScript Object Notation |
| **LDAP** | **:** Lightweight Directory Access Protocol |
| **NPM** | **:** Node Package Manager |
| **SPA** | **:** Single Page Application |
| **UI** | **:** User Interface |
| **XML** | **:** Extensible Markup Language |
| **WYSIWYG** | **:** What You See Is What You Get |

# 1. INTRODUCTION

## 1.1 Comparison to Similar Systems

Code Generation is a very old topic if considered the subject on code generation, lots of articles, applications and open source projects can be found. But when focused to code generation on javascript applications and narrowed it down to ReactJS based single page applications this project becomes unique on its kind [http-1]. The main research is not the only code generation, but also generation after the codes modified by hand.

When code generation term used, mostly generation of the code is one way, which means; generate the source and use it and cannot generate it again. This is one of the unique difference that is used in this research.

The term bidirectional code generation is used to name the two-way code generation. Two way means, users can generate the codes from scratch and they can also generate the sources after developer coded by hand. The generated codes can be modified by developers by hand coding and those modified sources can be loaded and generated/modified again.

## 1.2 Literature Review

Since bidirectional term is unique in its kind, similar systems have been focused that exists in the community. The javascript projects are commonly open sourced and can be examined in detail. On the next chapter, some of the code generation platforms are examined.

## 1.3. Similar Platforms Comparisons

There are so many code generation platforms on the web. Those are also web based projects like appbuilder itself. The main difference is loading existing projects and start from where the project is.

### 1.3.1 Forms.IO

Forms.IO is a web based application generation platform, most of the features and ideas are similar to appbuilder platform. With Forms.IO user can generate form based applications and can gather data from those screens.

The second difference is, forms.io creates an entire project after user finishes the design. Since the codes are generated from scratch, it lets to output a few technologies based app like ReactJS, Angularjs etc.

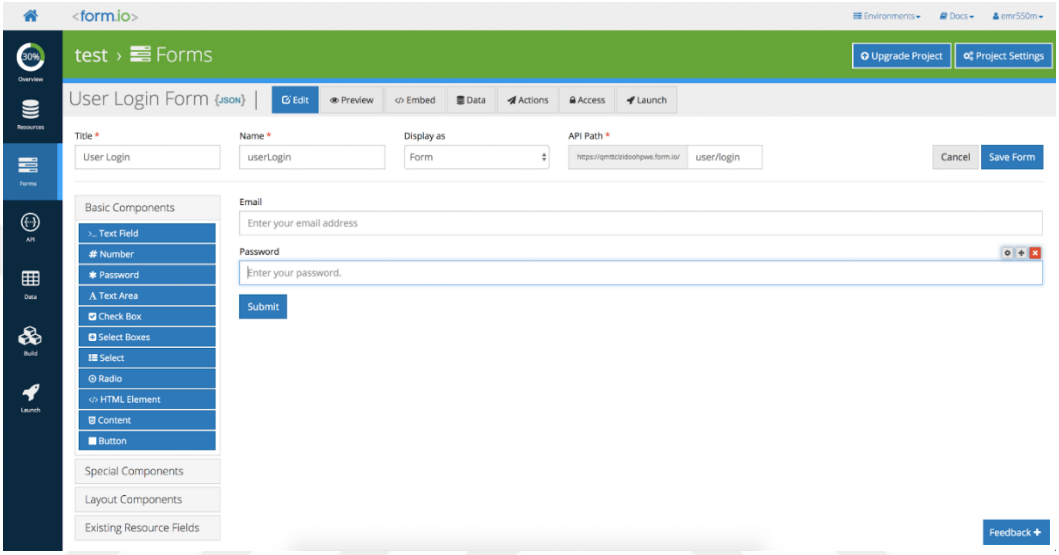


**Figure 1.** *Forms.IO designer screen*

As shown in the Figure 1. Forms.IO designer screen the generation layout seems similar to each other. The side by side comparison is shown in the Table 1. Forms.IO & Appbuilder Comparison. Plus side of appbuilder is, code generation is two ways, weak side is Forms.IO supports for AngularJS and ReactJS as an output but appbuilder supports only reactjs.

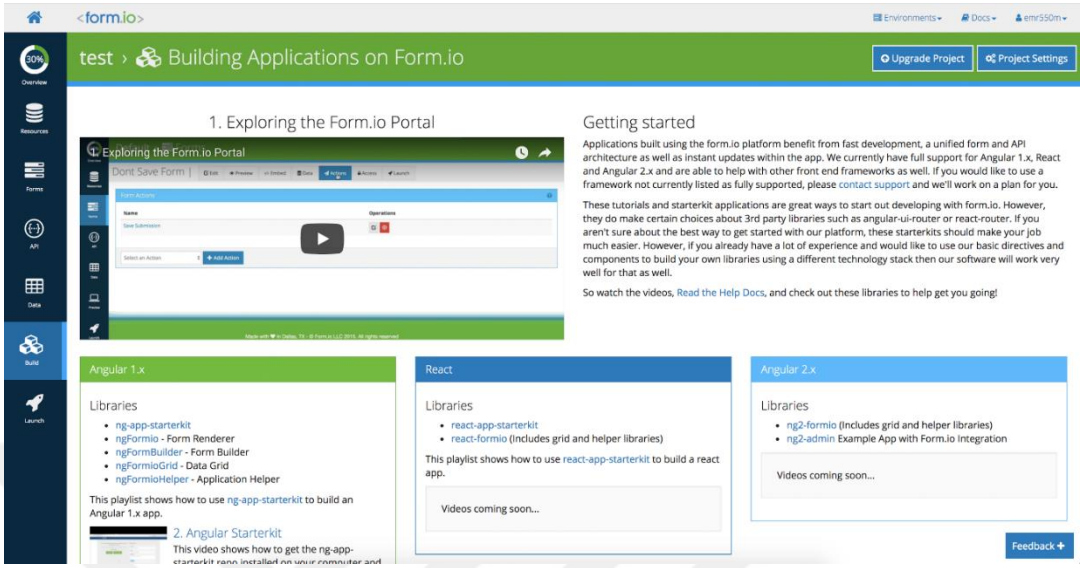


**Figure 2.** *Forms.IO export screen*

**Table 1.** *Forms.IO & Appbuilder Comparison*

| AppBuilder | Forms.IO |
|---|---|
| Each page/form can be designed from scratch or loaded and old design. | Each page/form can be designed from scratch or loaded and old design. |
| Project loaded from its own source at start and after. | Project created from scratch, after creating it can be loaded again and modified. |
| Original source can be loaded and can be allowed for design. | Only its source project definition can be loaded. Users own source cannot. |
| Bidirectional code generation allowed. | One way code generation is supported. |
| ReactJS is supported. | Angular JS and ReactJs is supported. |
| Backend mocking allowed | Backend mocking allowed |
| Source can load from any git repo | Projects load from its own database. |

Forms.IO is a commercial product, users have to pay for each project. It is not an open source project.

### 1.3.2 Gatsby app

Gatsby is an open source project for code generation and make javascript development easy. But main problem with the gatsby system is, it is not drag & drop style designer.

It just creates a workspace and a hot loading javascript project environment to developers and leaves it there. The main idea is also the main code generation idea, create source code from intermediate languages like json, markdown etc.

**Table 2.** *Gatsby & Appbuilder Comparison*

| AppBuilder | Forms.IO |
|---|---|
| Each page/form can be designed from scratch or loaded and old design. | Pages generated from source language or console. No graphical interface. |
| Project loaded from its own source at start and after. | Project loaded from its own source at start and after. |
| Original source can be loaded and can be allowed for design. | Original source can be loaded and can be allowed for design. |
| Bidirectional code generation allowed. | One way code generation is supported. |
| ReactJS is supported. | ReactJS is supported. |
| Backend mocking allowed | Backend mocking allowed |
| Source can load from any git repo | Source can load from any git repo |

Gatsby an open source project, so anybody can contribute, but it seems it is not a well-stable project yet.

### 1.3.3 Helmetrex.com - structor project

Helmetrex is the start point of this study. It's first sparkles comes from helmetrex project. Appbuilder starts with helmetrex original source, but in time everything needs to be changes.

The project is an open source project that is served over github.com [http-2]. The aim of the project is to generate and design an app without coding it.

The structure and the ideas are similar as a running application with appbuilder and helmetrex's structor project. They both gives users a GUI for designing applications.

But the main difference between the appbuilder and the structor project is like all the others, code generation is bidirectional on appbuilder, but in structor, each project is generated from json source files. So, code generation is one way.

**Table 3.** *Helmetrex.Com - Structor Project & Appbuilder Comparison*

| AppBuilder | Forms.IO |
|---|---|
| Each page/form can be designed from scratch or loaded and old design. | Pages generated from source language (JSON). |
| Project loaded from its own source at start and after. | Project loaded from its own database, not original source, only its own project types can be loaded. |
| Original source can be loaded and can be allowed for design. | Only special project type to itself can be loaded and designed. |
| Bidirectional code generation allowed. | One way code generation is supported. |
| ReactJS is supported. | ReactJS is supported. |
| Backend mocking allowed | Backend mocking allowed |
| Source can load from any git repo | Source can load from any git repo |

There is a general idea on code generation on projects and articles related to them, there is a source language and a target language. Generation is one way from source to destination, for designer perspective, designing creates the source language files, the generation engine creates the target sources files from those source files.

Like all the similar projects inspected above, each project has its own source generation files to create / generate target source code. There comes the main difference between appbuilder project and the others. Appbuilder uses directly source files themselves to create new ones.

## 2. METHODOLOGY, TECHNOLOGIES AND ARCHITECTURE

### 2.1. Summary and Used Technologies

It is every software team's goal to design applications by just drag and dropping application items. The idea starts with "Code Generation" or the term "Rapid Application Development". To decrease cost of production, most of companies believe code generation may be a solution.

In most code generation techniques, there are a pre-stage of the codes that generated from a second language like XML or JSON. That makes the code generation one way, which means users can generate the sources from second language to target one. But cannot reload current/latest source code and redesign it, or it is very difficult. Because their development team has already changed the sources, and most of the case, those codes cannot be decompiled back to source code generation language.

Solution is the keywords above. If users can combine them all together, it is possible to create sources from nothing (there is no second language), users can reload their current / latest source code from source versioning tool and start designing / code generating (For specific programming languages).

Sources taken from version control system (git/tfs etc.), generated-designed with a WYSIWYG (what you see is what you get) editor, and committed back to source control. This technique told in this document (thesis) will be called Application Builder later on.

Application Builder framework provides each user a webpack sandboxed execution unit. Clients directed to the chain of webpack middleware during the design stage of their projects [http-3].

Users create/modify/manage their pages, modules and components within their project and webpack sandboxed unit provides a transparent passage from server to the UI. The transparency involves [http-4]:

-Server-in-memory packaged ReactJS based Single Page Application (SPA) Project

-Hot module replacement (HMR) to inject updated modules into the active runtime

-Preserve DOM and React component state when components are saved

-Record Compilation timings and status

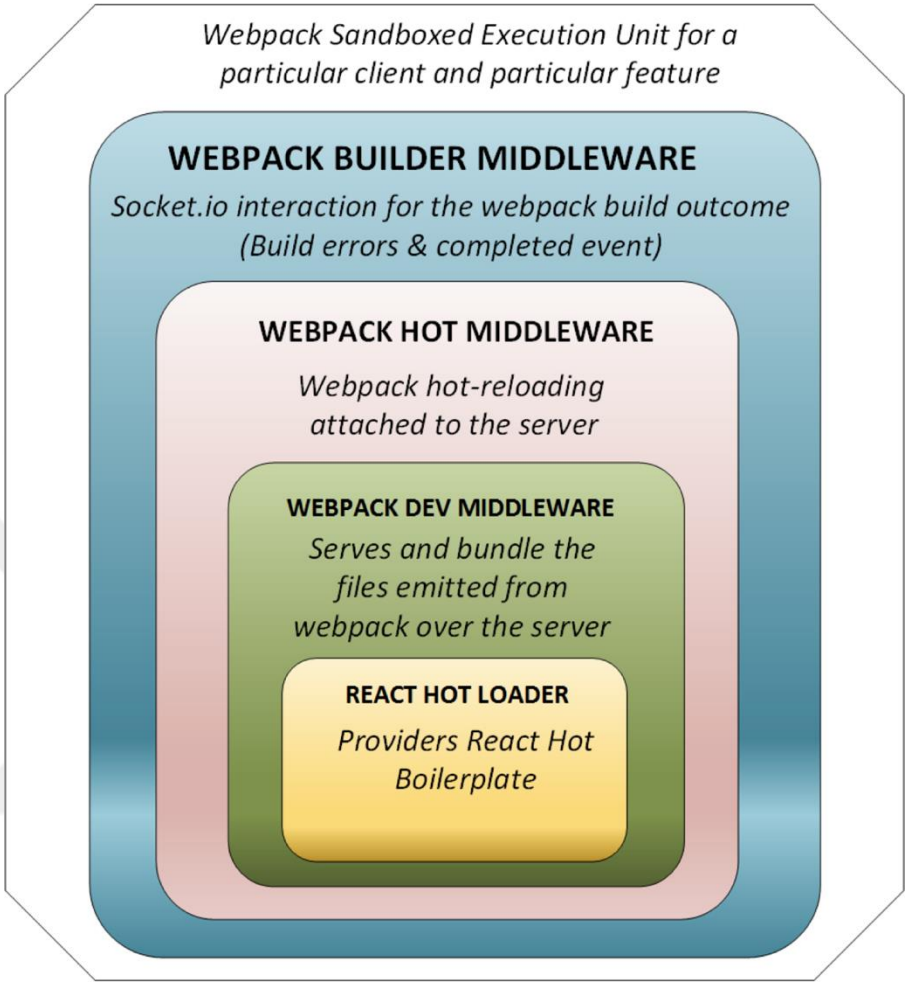


**Figure 3.** *Common structure of a single user's project structure*

Webpack sandboxed execution unit provides a chain of 4-layer middleware infrastructure. At the top, there is the Webpack Builder Middleware provides a websocket connection to the client based on the immediate HMR compilation status. Build errors and succeeded changes are published to the clients through websocket.

Webpack Builder middleware wraps the HMR middleware as the next layer in the build pipeline. HMR middleware provides the underlying mechanism for replacing Javascript modules on the fly.

Webpack hot middleware wraps the dev-middleware. Webpack dev-middleware provides an in-memory bundling the target SPA Project. It splits the files into chunks,

separates the vendors and parallelize the build process to improve immediate builds triggered by client changes.

Webpack dev-middleware wraps the react-hot-loader layer deepest in the hierarchy. This layer attempts to build on top of Webpack HMR and preserve DOM and React component state when components are saved.

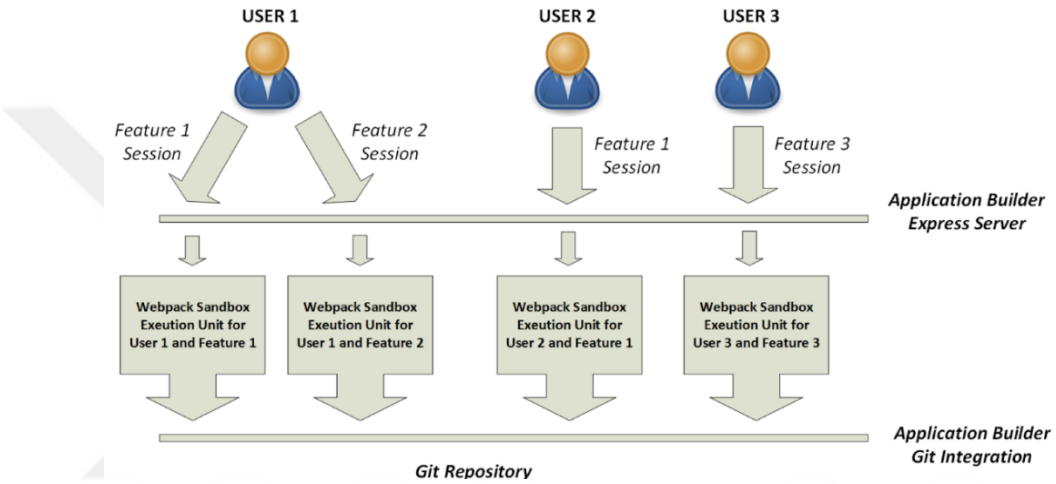


**Figure 4.** *General structure of app builder project*

Webpack dev-middleware wraps the react-hot-loader layer deepest in the hierarchy. This layer attempts to build on top of Webpack HMR and preserve DOM and React component state when components are saved.

Each user login to App Builder, open a Project feature session. Users can have multiple feature sessions allocated to them. Webpack Sandbox execution units are spawned through the system as an isolated mini server providing a transparent gateway to the target project feature. Project features changed are published to their target repository through Application Builder.

By combining applicable technologies together, it is possible to create an application builder platform, which can build applications while they are even running on the browser. In this thesis, every step will be defined clearly [http-3].

### 2.1.1 Node.JS

Node.JS is the core system that lies on everything and enables the whole project possible. All components on the project runs on the Node.JS infrastructure. Node.JS is an asynchronous event driven Javascript runtime that runs on Google V8 engine. It uses a very light ware and efficient event driven non-blocking I/O model. Node.JS also has a big open source package library ecosystem inside (npm).

NPM is a part of the Node.JS platform. Developers can share libraries over npm and also use other libraries that is shared by others before. At first, npm is used to share Javascript code blocks for Node.JS projects, but today, any kind of code can be shared over npm.

The first use of Node.JS to run Javascript code on server side over network. But it goes furthermore and users can create large scaled network apps with high performance today.

Node.JS platform is completely open source based over GitHub. Anybody can contribute to project. Currently more than 250 people is contributing to project and this number is increasing over time. With Node.JS it is very easy to create web applications;

```
http = require('http');
http.createServer(function(req, res) {
res.writeHead(200, {'Content-Type': 'text/html'});
res.write('<p>Hello World</p>');
res.end();
```

These 5 lines of code is enough to create a simple web application that says hello. Only Node.JS is needed to run this app, not even need a web server is needed. If these lines compared to PHP it‘ll be written down;

```
<?php
echo '<p>Hello World</p>';
?>
```

These lines above. For example, if those two lines compared for performance [http-5]; For about 100.000 request with 1000 concurrency, the results below have accuired.

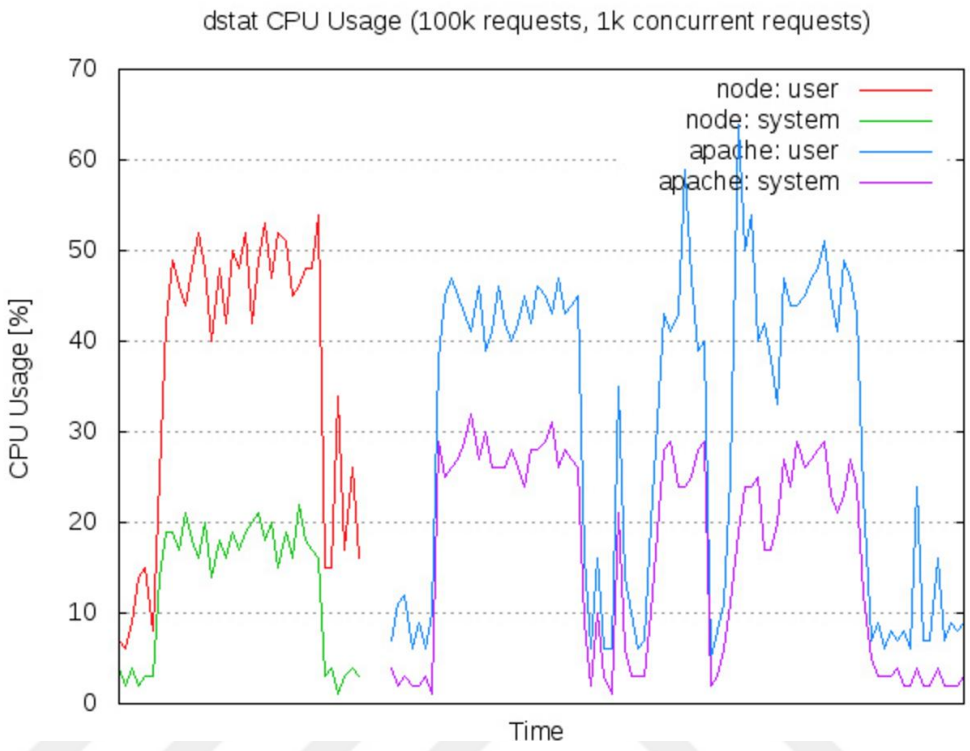


**Figure 5.** *100K request with 1K concurrency*

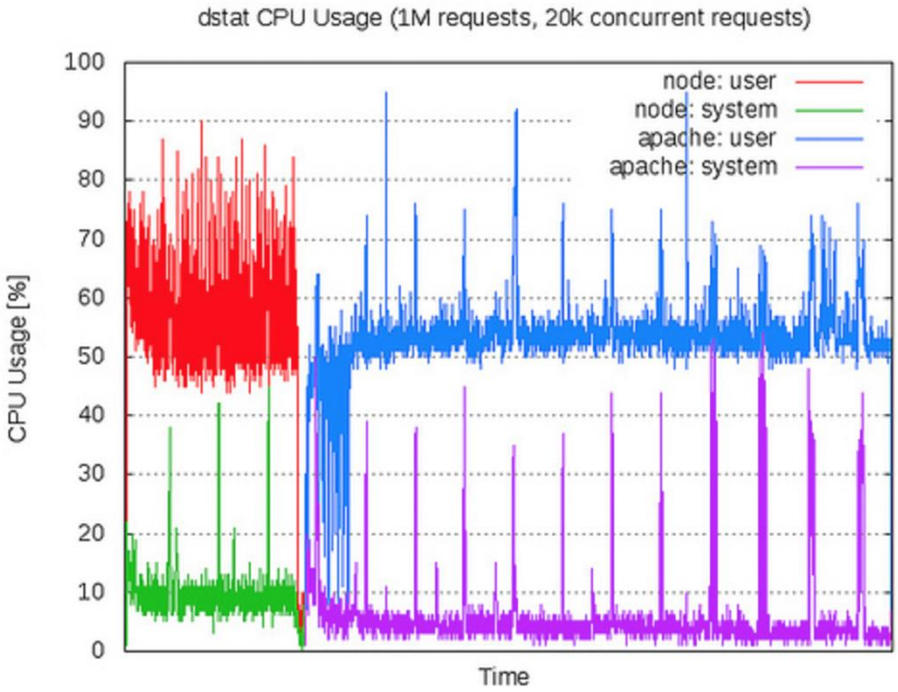


**Figure 6.** *1M request with 20K concurrency*

With higher values, users can see the node performance more clearly. Users can find many performance tests that is done by users on the internet [http-5].

With Node.JS there is a second subject called ECMAScript. ECMAScript is the language library version of javascript. It is also called as ECMAScript (ES). Currently there are ES5, ES6 and ES7 versions of ECMAScript. Standard browsers are all supporting ES5, ES6 is partially supported, and ES7 is the future [http-6]. This thesis uses version ES6 and ES7 which will be described later [http-7].

### 2.1.2 React.JS

React JS is a javascript library that makes custom web components possible. Users can create even custom html tags and attributes as if they really exist.

React makes it painless to create interactive UIs. Design simple views for each state in applications and React will efficiently update and render just the right components when data changes. In general, react components, will be executed and run on client side (web browsers), but with combining Node.JS, react can be execute also on server side.

React.JS creates a copy of DOM (called virtual DOM), at first, creates the operations (changes in HTML) on the virtual DOM, calculates the differences to the real DOM on the browser, makes only the necessary changes with the smallest change. DOM operations cost much for browsers (performance perspective), thus, making small changes on DOM makes React.JS a better JS library [http-1].

### 2.1.3 Redux.JS

Redux is a predictable state container for JavaScript apps. It helps users to write applications that behave consistently, run in different environments (client, server, and native), and are easy to test [http-8]. On top of that, it provides a great developer experience, such as live code editing combined with a time traveling debugger.

Javascript applications (client or server), needs a structure, a living data structure that data can be stored, and notified when changed to listeners. This called generally “state”. Redux javascript library’s goal to achieve that with perfection.

The state of whole application is stored in an object tree within a single store. This makes it easy to create universal apps, as the state from server can be serialized and hydrated into the client with no extra coding effort.

In Redux, state object has to be read only, and can only be modified with actions. This ensures that neither the views nor the network call backs will ever write directly to the state. Instead, they express an intent to transform the state. Therefore, Redux can tell the state listeners that data is changed.

Changes are made with pure functions. To specify how the state tree is transformed by actions, pure reducers needed. Reducers are just pure functions that take the previous state and an action, and return the next state. Remember to return new state objects, instead of mutating the previous state. User can start with a single reducer, and as app grows, split it off into smaller reducers that manage specific parts of the state tree. Because reducers are just functions, user can control the order in which they are called, pass additional data, or even make reusable reducers for common tasks such as pagination.

### 2.1.4 Webpack

Webpack is a module bundler system for javascript applications. Users can combine and bundle javascript codes together into single or partitioned files. Multiple file formats are also supported. Which means, user can bundle images into javascript applications is also can be done [http-3].

Not only javascript or image files supported, it is possible to bundle any kind of file by writing an appropriate loader for it. Existing module bundlers are not well suited for big projects. The most important property of webpack is Code Splitting. Users can split bundles into small pieces that have been decided.

The main goals of webpack project are;

1. Split the dependency tree into chunks loaded on demand
2. Keep initial loading time low
3. Every static asset should be able to be a module
4. Ability to integrate 3rd-party libraries as modules
5. Ability to customize nearly every part of the module bundler

6. Suited for big projects

In the heart of this thesis, webpack means a lot for bundling modules (javascript code blocks). The most important spec of webpack for this thesis is the hot loading ability of webpack.

Webpack HMR, is a great technology that can create an interactive bundle mechanism, that can be modified at run time. This means, users can change the source of their bundle while running.

The idea of hot loading makes this project possible. If the project that can change source files at runtime, Webpack HMR can serves them live. This is the core of the App Builder technology.

### 2.1.5 Webpack hot module replacement - HMR

Webpack has an embedded specification called HMR. Webpack adds a small HMR runtime to the bundle, during the build process, that runs inside the application. When the build completes, Webpack does not exit but stays active, watching the source files for changes.

If Webpack detects a source file change, it rebuilds only the changed module(s). Depending on the settings, Webpack will either send a signal to the HMR runtime, or the HMR runtime will poll webpack for changes. Either way, the changed module is sent to the HMR runtime which then tries to apply the hot update. First it checks whether the updated module can self-accept. If not, it checks those modules that have required the updated module. If these too do not accept the update, it bubbles up another level, to the modules that required the modules that required the changed module. This bubbling-up will continue until either the update is accepted, or the app entry point is reached, in which case the hot update fails.

This hot loading mechanism enables us to create the real-time app builder mechanism. At the overall hierarchy, a hot loader structure was set up over source code, after that, build a project that can modify the source code with technique what you see is what you get kind and finally webpack HMR does the rest, it directly compiles the code and displays on the browser.

### 2.1.6 ES5 / ES6 / ES7

With the technology advancing in programming languages, some new titles occurred in literature. For example, javascript had started a versioning like .NET or any other library. And also, these versions welcomed and accepted by the w3 consortium [http-9].

Currently there is 3 major versions on javascript. The supported by every browser and what users have known till now is the version ES5. For recent years, new coding techniques are added to the standard and it was called ES6. End the next coming version is called ES7. On this project, ES7 level code is used for the latest abilities.

### 2.1.7 Babel JS

The ES5, ES6, ES7 which are the core versions of the javascript, are not supported by all of the browsers yet. ES5 is covered by all of current browsers, ES6 is mostly covered, ES7 is the least covered version.

To solve this support problem, a platform called Babel JS has been created. Babel JS is a javascript compile engine, that converts ES6 and ES7 code into ES5, which is understandable for every browser.

Because ES6 and ES7 is new, they are not supported by every browser. Babel fixes this problem for us. The compile engine used in this project is Babel JS. It is not enough just compiling and converting the code for the browsers. If the browser has not the functionality to support ES6 or 7 features at all, it has to be poly filled. Polyfilling is also done by Babel JS.

### 2.1.8 NPM

Node Package Manager (NPM), is the package management for javascript projects. All the packages used in this project are publicly deployed to npm and they are used from there in this project.

Npm comes with Node JS (embedded), by installing Node JS , users can start to use npm from their computer's console. It is possible to search the packages through its web portal [http-10] and easy to install whatever needed. Like components ReactJS, Redux JS, Babel and etc. all packages are retrieved from npm network.

## 2.2 Project Structure

App builder project has two main parts, one is the backend part and one is the front-end part, both parts have significant contribution to the project. Backend part does the most work like pulling project code from version tool, package and bundle the code, serve the project like a web site, do the code generation and at the and push back the code to the version tool (Git / TFS Etc.).

Frontend part is a SPA (single page application), written in reactjs, has two main roles, first role, displaying the designing project inside itself like a sub-application, second role, create a user interface for building and generation. This generator interface is a second layer above the designing project and they seem as a whole project. But in reality, there are two projects running on the browser (as called front end).

### 2.2.1 Project architecture

As displayed in Figure 3. Common structure of a single user's project structure, app builder project has some layers and modules inside. Both backend and frontend parts have a substructure inside them.

#### 2.2.1.1 *Project backend structure*

App builder backend is written in Node JS in ecmascript 5 the technology mentioned in 2.1.6 ES5 / ES6 / ES7 section. This project has several modules inside that does' different jobs for the main goal.

The backend project is a Node JS Application. Because, Node JS applications becoming very popular in recent years. With the improvements on the cloud computer systems, creating and serving nodejs apps are very easy today. To create and run a Node JS app, only Node JS library is needed installed on target PC/Server. As already mentioned in 2.5.1.1 Node.JS section, NodeJS is an easy to use platform.

NodeJS applications are generally web-based (runs on the browser) kind applications. But it is possible to create console applications too. In fact, with nodejs users can only create console applications right now, because Node JS is not a web server or does not include a web server technology, it only executes and outputs the javascript sources on console. App builder backed also is just like this [http-10].

To make NodeJS a fully capable web application, users need express package. As mentioned in 2.1.8 NPM, users can install it from NPM. Express package is one of the mostly used package on Node JS applications. It makes node JS to behave like a web server, users can serve web pages or web services by node JS with express.

Express package is the core part that makes app builder to behave like a web server and serve backend services. The backend projects can be summarized as submodules below.

#### 2.2.1.1.1 *Core module*

Core module is the main module of the backend part. This module keeps everything together and it also includes the express module. Core Module creates a web server and a web service, from those, the app builder user interface served and generator commands are executed.

Core modules also boots up the app builder project and does what need to be done at project start. It boots up the database if it is not running, it reads the start-up parameters, it executes the update process, and starts the web server for serving user interface and also it starts the web service layer for listening user interface for commands.

#### 2.2.1.1.2 *Database manager*

This module is responsible for database and database storage operations. By default, app builder project has its own database inside mongodb. If there is no database configured externally it boots up its own database and start to use it.

Database manager module also has business functions for database operations like select, insert, update and delete. Currently app builder is working with mongodb, but buy using alternative database drivers, it is also possible to use other databases like Microsoft SQL server or Oracle Database, but users also need to update the database functions inside the module (select, insert, update, delete).

App builder project does its most of the job on the source files, so database operations aren't that much, just a few database tables are using to keep things organized like user logins, project names and definitions. There are no code generational database tables at all.

#### 2.2.1.1.3 *Bidirectional middleware compiler*

As shown in Figure 3. Common structure of a single user's project structure javascript (webpack) middleware compilation on the hot is very important for this project. As mentioned in 2.1.5 Webpack Hot Module Replacement (HMR) section, thanks to webpack javascript files can know be replaced and reloaded while they are executing. This module creates the webpack hot loader middleware layer for each designer project and puts it to the express located in the core module, so that updates on the source javascript files can be sent to the user interface over express from javascript sockets.

Each manipulation on source javascript files creates an update on the webpack middleware compiler, those updates sent to the client (which is the user interface runs on the browser on users) over socket. Socket called websocket in general term, is a live connection between backend and frontend layers that communicates several things. Webpack hot middleware changes is one of them.

#### 2.2.1.1.4 *Dummy service manager*

The designing projects may have backend connections. For example, the project that will be designed in the app builder may have a login page, in app builder the designing application is running on the browser while it is designing, so it may not be possible to log the user inside the app while it was designing.

For such those cases, users need to simulate the backend that the designing application uses. The simulation of the backend is called dummy service or in general term “Mocking”.

This module does the backend service mocking of the designing application. It is possible to create mock/dummy service calls from app builder project. For example, user can create a fake reply for login feature. It is possible to make fake replies to all the backend calls that the designing application makes.

#### 2.2.1.1.5 *Export manager*

App builder project can download source files for the designing project from its source control system, and can commit back to the result. Today there are some highly common source control system like Git or TFS. App builder is currently works with Git versioning system. But it is possible to make it work with other source control systems easily.

This module does the commit back the changed files to source control system. App builder makes changes to all files to make the real time designing happen. But, when source codes need to commit back to source control system unnecessary changes needed to remove (the changes that app builder does for itself). At the result, there must be only changes that the user designed.

This module does the cleaning of the source files to original and makes the commit back operation. It also creates a new branch from the original branch and does the commit over that branch. In Git, it is very important to work in branches.

#### 2.2.1.1.6 *File format module*

This modules formats (beautify) the generated source files. Mostly generated files won't seem nice. The indentation may be bad, this module uses esformatter npm package to format the generated source files.

After formatting files, they will look like they will have written by hand on a nice IDE. The rules of the formatting can be modified and told to the esformatter. Current settings for the formatter are the general rules in the literature on the javascript. Here is the configuration for the formatting source files;

```
// this is the section this plugin will use to store the settings for the jsx formatting
"jsx": {
  // by default, is true if set to false it works the same as esformatter-jsx-ignore
  "formatJSX": true,
  // keep the node attributes on the same line as the open tag. Default is true.
  // Setting this to false will put each one of the attributes on a single line
  "attrsOnSameLineAsTag": false,
  // how many attributes should the node have before having to put each
  // attribute in a new line. Default 1
  "maxAttrsOnTag": 2,
  // if the attributes are going to be put each one on its own line, then keep the first
  // on the same line as the open tag
  "firstAttributeOnSameLine": true,
```

```
    // align the attributes with the first attribute (if the first attribute was kept on the same line as on the open tag)
    "alignWithFirstAttribute": true,
    "spaceInJSXExpressionContainers": " ",
    "htmlOptions": { // same as the ones passed to jsbeautifier.html
      "brace_style": "collapse",
      "indent_char": " ",
      //indentScripts: "keep",
      "indent_size": 4,
      "max_preserve_newlines": 2,
      "preserve_newlines": true
      //unformatted: ["a", "sub", "sup", "b", "i", "u" ],
      //wrapLineLength: 0
    }
  }
```

As it seemed on the configuration lines, it is possible to create very clean and good-looking source files while generating them.

#### 2.2.1.1.7 *File manager*

While code generation, there are so many file operations, like read file, save file, modify file and etc. This module contains the file operations inside it. To keep the project files clear (app builder itself), file operations are gathered together on this module.

#### 2.2.1.1.8 *File parser*

Appbuilder supports ReactJS as UI javascript library. ReactJS has its own file/coding syntax. So, a special parser for those files needed. Parsing javascript source files to javascript objects for processing is done in this module.

#### 2.2.1.1.9 *Generator manager / template manager*

Appbuilder project can create file or file groups together with a given template files. This is used to create pages in designing apps. ReactJS applications may need more than one file while creating a page in the application.

This can change designing applications structure. For complex projects that contains multiple subpages, there may need to create several files to create a single page to the project. If Redux JS used as application state, these files also need to be created for that too.

In appbuilder administration pages, users can define page outline (template) definitions, so that appbuilder can use those definitions and creates files. Those modules do the page generation operations that users have created as templates. While creating source files the defined parameters in the template definition are gathered from the user.

#### 2.2.1.1.10 *Git manager*

Appbuilder project can download project from version tool (Current supported versioning tool is Git. After downloading the project, it builds the source code and runs it. So that user can start designing To do that, users need a running and a hot loading supported reactjs web application in Git repository. After the designing and code generation finished, appbuilder also can put (commit) back the generated files to its original versioning tool. Those code downloading and uploading (pull & push in Git Terms) operations done in this module.

#### 2.2.1.1.11 *Index manager*

In ReactJS, web components will become very popular term. In reactjs projects every item in project (objects) called as “Component”. So, users can create whatever they need as component and use them as much as they want.

Appbuilder can use those web components as design items in the project. Users can define their components to appbuilder as design items and appbuilder can identify those items on source files and can modify them. The identification of source files (finding component definitions) is done on this module.

#### 2.2.1.1.12 *Login manager*

Appbuilder can be integrated with different authentication mechanisms, users can integrate it with LDAP or oAuth, currently a regular authentication is used and auth info stored in local database. The authentication functions and logic stored in this module.

#### 2.2.1.1.13 *Npm manager*

In ReactJS applications, most of the libraries are on npm. So, users need features for installing and removing npm packages to their designing project. NPM operations like install and remove are stored in this module.

#### 2.2.1.1.14 *Socket manager*

Appbuilder can run multiple project designers at the same time. Different user can log in and start different projects concurrently. Each project creates a separate workspace and works standalone.

But each workspace needs its own and private socket connection between appbuilder and client browser to communicate. As mentioned in 2.3.1 Webpack Hot Module Replacement (HMR), when javascript sources changed and files compiled, client browser is notified over socket connection. If there are multiple socket connections at the same time, users need to manage them and keep them separate, this is done by Socket Manager module.

#### 2.2.1.1.15 *Export manager*

Appbuilder can download and upload project source files from source versioning tool in addition to that it can also create a compressed single file that contains every file on the project, and give it to the user - by a browser file download. To do that, app builder needs some processing on files, after processing all the files compressed together and sent to user browser. This operation done in export manager module.

#### 2.2.1.1.16 *State manager*

Appbuilder project has its own application storage called state manager. State means something different in ReactJS applications but since appbuilder is a nodejs application, it

also needs a space to put settings and global variables somewhere. State manager module stores and handles the changes application wide variables and settings.

#### 2.2.1.1.17 *Storage manager*

As mentioned in 2.2.1.1.8 File Parser section, ReactJS has its own file structure and syntax. So, users need special functions to parse, add and remove lines from source files. ReactJS source specific parsing and manipulations functions stored in storage manager module.

#### 2.2.1.1.18 *Webpack builder middleware*

As mentioned in 2.1.5 Webpack Hot Module Replacement (HMR), when javascript source files changed, webpack hot module replacement module creates some update operations, those operations created some events, appbuilder use those events to feed the frontend project. Webpack hot module configuration and management operations done in this module.

### 2.2.1.2 *Project front end structure*

Appbuilder projects frontend part is a single page application written in reactjs. Mainly, it creates a user interface for users, for administering appbuilder itself and let the projects can be designed.

It creates a WYSIWYG (what you see is what you get) style builder interface while user is in designer mode. Designing application is running on an inside frame, appbuilder puts a hidden layer above it, with this layer it can give to user designer capabilities.

This frontend ReactJS application has some modules and folder structures inside described below. Frontend project codes are located on the static folder at the root of the project.

#### 2.2.1.2.1 *API files*

Frontend needs specific functions like searching through designer components, creating UI overlays etc. are located inside api directory.

#### 2.2.1.2.2 *Static files*

Every web application uses some css files, font files, external javascript libraries. Those files that are not written or maintained by this project are located under static folder. Generally, they were published directly to workspace (project output), without doing extra process like bundling.

#### 2.2.1.2.3 *Components*

As mentioned in 2.2.1.1.11 Index Manager section, appbuilder frontend application has some components that used inside its own pages. Those components are located in components directory.

#### 2.2.1.2.4 *Middleware*

Redux JS library has a middleware property, means that users can hook every change operation in redux data store. In appbuilder, middleware is used to do some UI specific modifications while redux storage items are modifying.

#### 2.2.1.2.5 *Appbuilder pages*

Appbuilder project frontend applications has some pages itself. Those pages are divided into two main categories.

##### *2.2.1.2.5.1 Administration pages*

Appbuilder project needs some definitions to run. For example, the definition of the source control system or the definition of users and may be the most important part is the definition of designing components. Appbuilder administration pages stored in pages directory.

##### *2.2.1.2.5.2 User Pages*

Appbuilder project has some pages to run the designer process and gives an output to user what was done before. Those pages will be described in detail later and they are stored in pages folder.

#### 2.2.1.2.6 *Redux files*

Redux JS has a specific folder structure, users need to define some sub files and objects to run it smoothly. In appbuilder, redux needed files stored under redux folder.

As described in 2.1.3 Redux.JS section, redux has some layers inside it. Files are created related to those layers.

#### 2.2.1.2.7 *Route definitions*

ReactJS applications are single page applications as described in 2.1.2 React.JS section. They are called single page but this doesn't mean they contain only single page inside it. ReactJS applications may contain multiple pages inside like appbuilder frontend.

If there are multiple files, some routing information needed to define which page is which route. Those route information's are stored in rotes folder.

#### 2.2.1.2.8 C*onfiguration files*

Every ReactJS application need some configuration for webpack bundling, ecmascript execution and npm package management. Those files are located at the root folder of frontend project.

### 2.2.2 User interface and usage guide

Appbuilder has an easy to use user interface for both system administrators and users. Node JS server-side application serves the user interface UI. This UI is a single page application web application.

The UI interface connects to the backend by JSON backend calls. User commands like insert/update/delete operations done on the Node JS backend application. UI pages can be separated into two parts. User Pages and Administration Pages.

User pages are the regular pages to design applications. Appbuilder users logs in and create workspaces for designing applications. Administration pages helps to define and modify system parameters that appbuilder needs to run.

#### 2.2.2.1 *User pages*

Appbuilder application has its own authentication system. The system cannot be used without authorization and permission. Users can only use the approved projects that administrators give to them.

##### 2.2.2.1.1 *Login page*

Both admin and user level users use the same login screen. This is a simple login only uses username and password. Login can be integrated with LDAP services easily for commercial or corporate usage.

Login connects to backend NodeJS app. NodeJS app checks the login and if it is valid user redirected to dashboard screen. Users are defined by system administrator, which is another application user defined with admin role. The screen outline is shown in Figure 7. Login page screen.

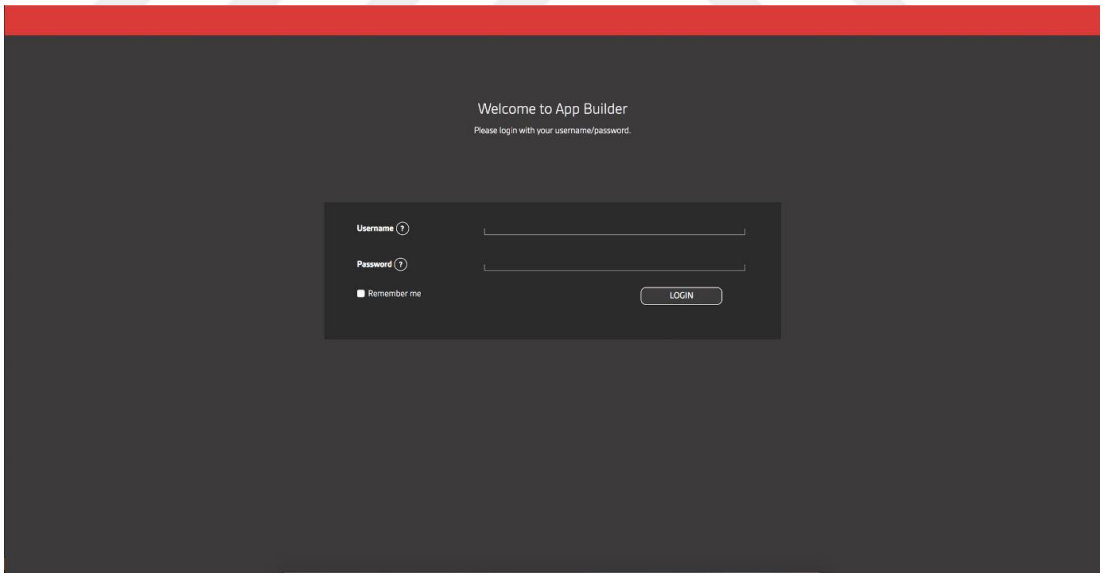


**Figure 7.** *Login page screen*

##### 2.2.2.1.2 *Dashboard page*

Both admin and user level users redirected to dashboard page after successful login. This screen is a summary screen that contains the previous work of the users. The screen layout shown in Figure 8. Dashboard page screen.

There is a navigation menu for each page that user can navigate. The menu is changed to the authorization level of the user. Administration pages shown if user has admin role.

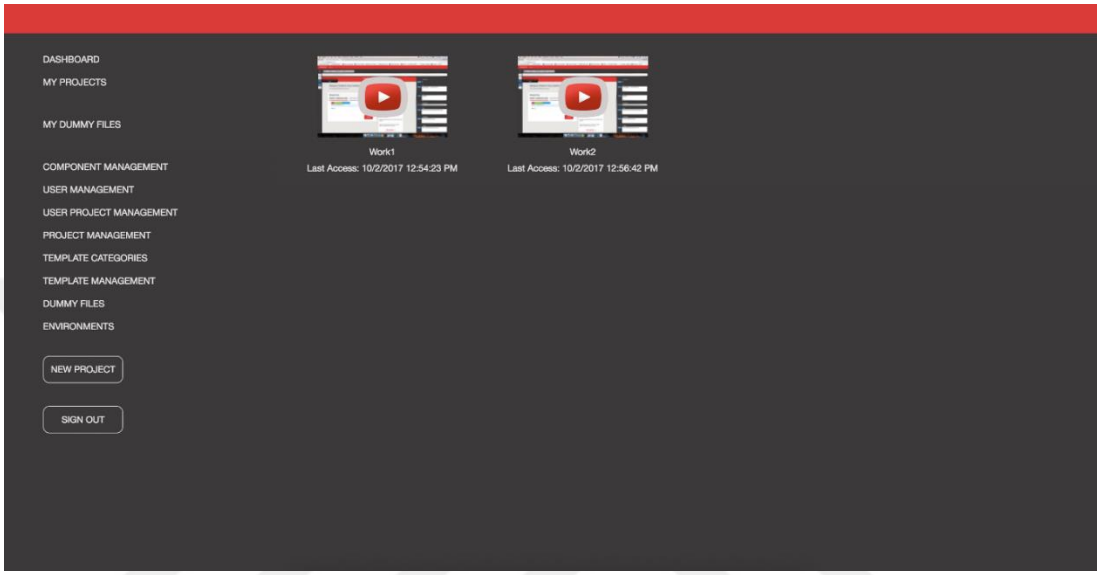


**Figure 8.** *Dashboard page screen*

### 2.2.2.1.3 *My projects page*

Dashboard page has only quick access/start to user's previous projects. User can manage their projects (delete operation) in My Projects Page. Users can also delete the project in this page and can see additional information about their previous works.

My Projects page shown in Figure 9. My projects page screen. Users can create as much as project they want, with the physical server limitations and what permission they have.

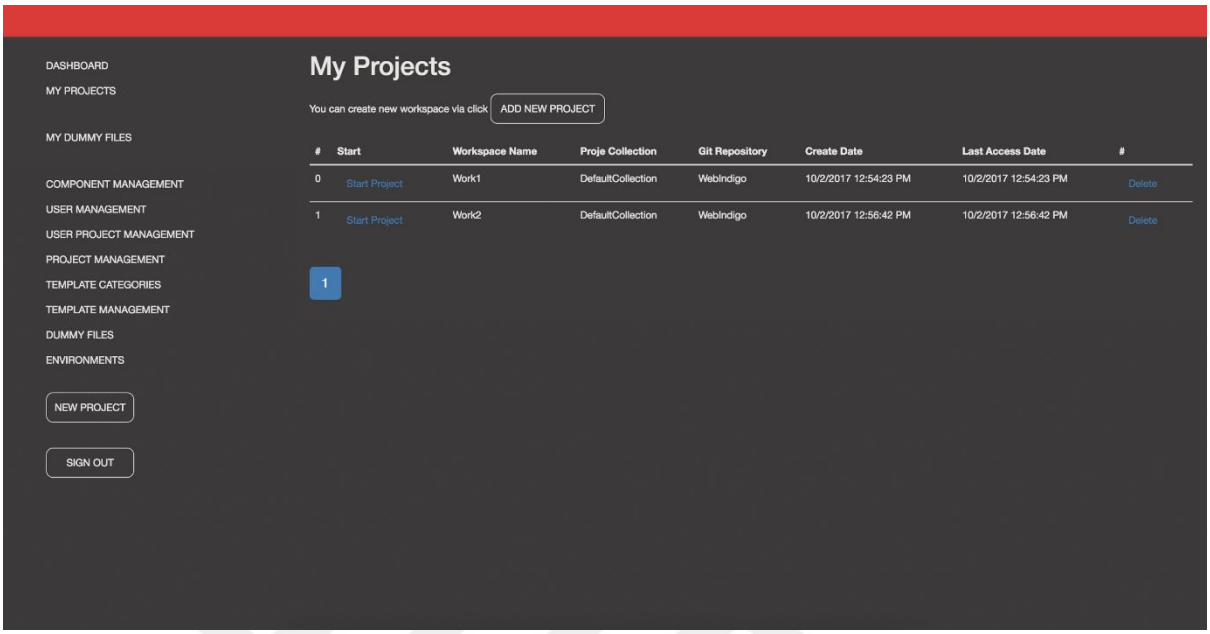


**Figure 9.** *My projects page screen*

### 2.2.2.1.4 *User dummy files page*

ReactJS single page web applications communicate backend with JSON requests. It can be said those requests are “Backend Calls”. While designing an application on appbuilder, those backend calls needs to be simulated. Because it may not always possible to run backend of the designer application.

Simulating backend calls is named service mocking. Service mocking needs dummy responses to run. They need to what to reply while giving a fake reply to request.

Users can load their own fake backend replies as JSON format from this User Dummy File Screen. Both admins and users can access and use this page. User Dummy Files page shown in Figure 10. User dummy files page screen.

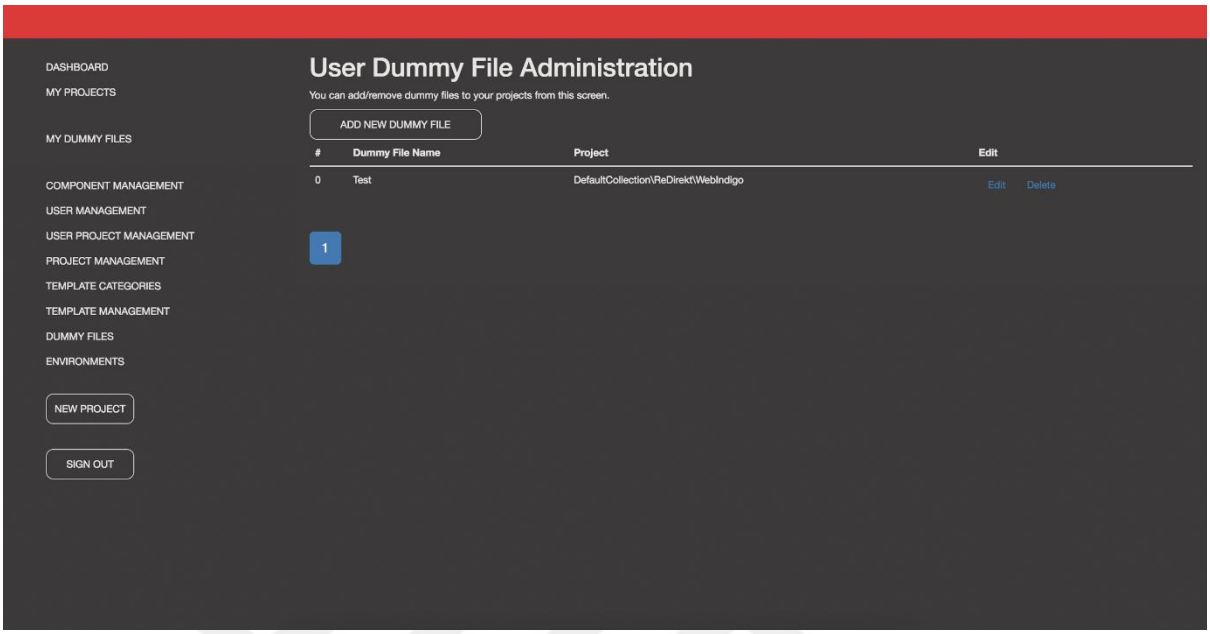


**Figure 10.** *User dummy files page screen*

### 2.2.2.1.5 *Start new project page*

To start a new application design, users select the source branch to load initial project and gives a name to work. Project name also Git repository branch name. So, naming convention has to be like Git branching name (No empty or blank characters).

This screen is working by user permissions. Administers give users to access projects. Appbuilder can support to design different projects at the same time. Unless they are in reactjs web application that has some configuration in it, appbuilder can start a designing project from its source control.

When user clicks to the ENTER button, the project will be downloaded from its original version control system and appbuilder navigate to the loader screen. On that loader screen, some outputs displayed to the user. While waiting the user in that screen, project will be initialized in the background. Since it is a javascript application the project is bundled, npm packages will be installed and all those outputs displayed to the user at that time.

When the initialization ends, the designer page will show up. Start new page shown in Figure 11. Start new project page screen. After this screen, project loading screen showed in Figure 12. Project loading page screen.

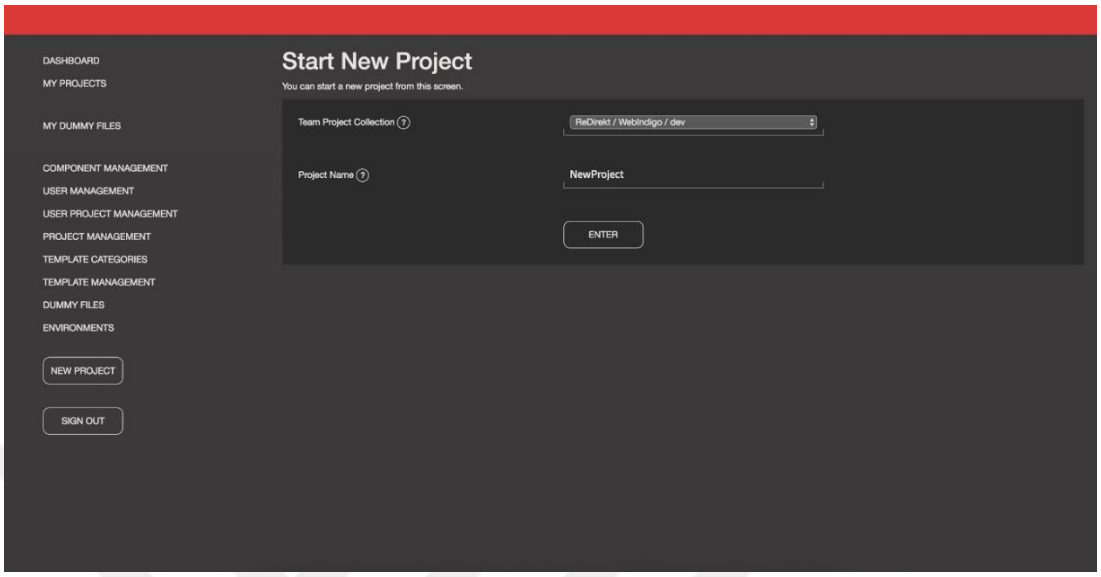


**Figure 11.** *Start new project page screen*

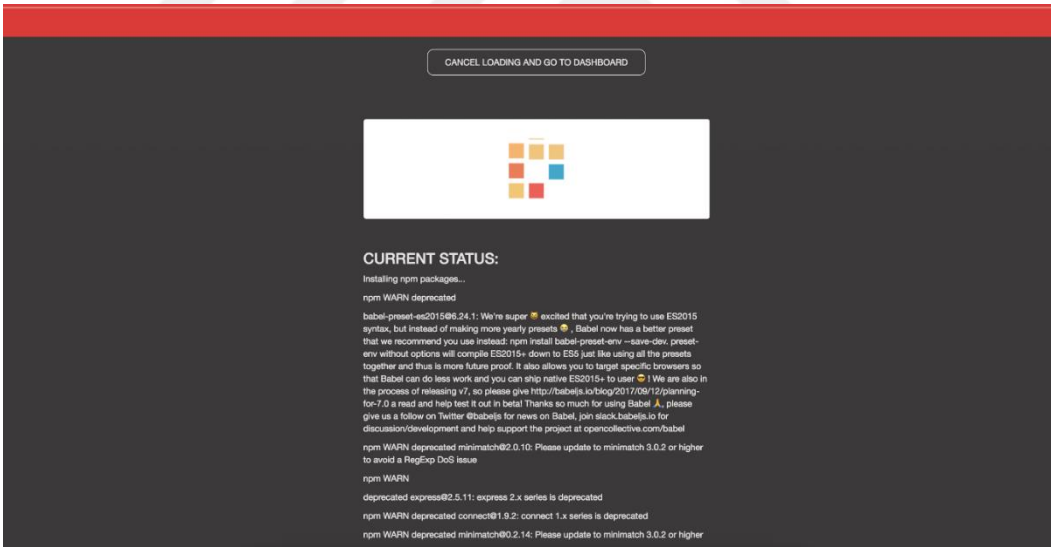


**Figure 12.** *Project loading page screen*

### 2.2.2.1.6 *Application designer page*

After loading project, the designer screen displayed to the user. This screen is the most important UI screen and everything is done here. Users can see what components are installed in app builder, they can add new pages, modify current ones.

This page, creates a hidden layer onto the designing application inside. While users moving the cursor above the screen, the components on the designing application is highlighted and can be selected.

Application designer screen shown in Figure 13. Application designer page screen. This page will be described in detail.

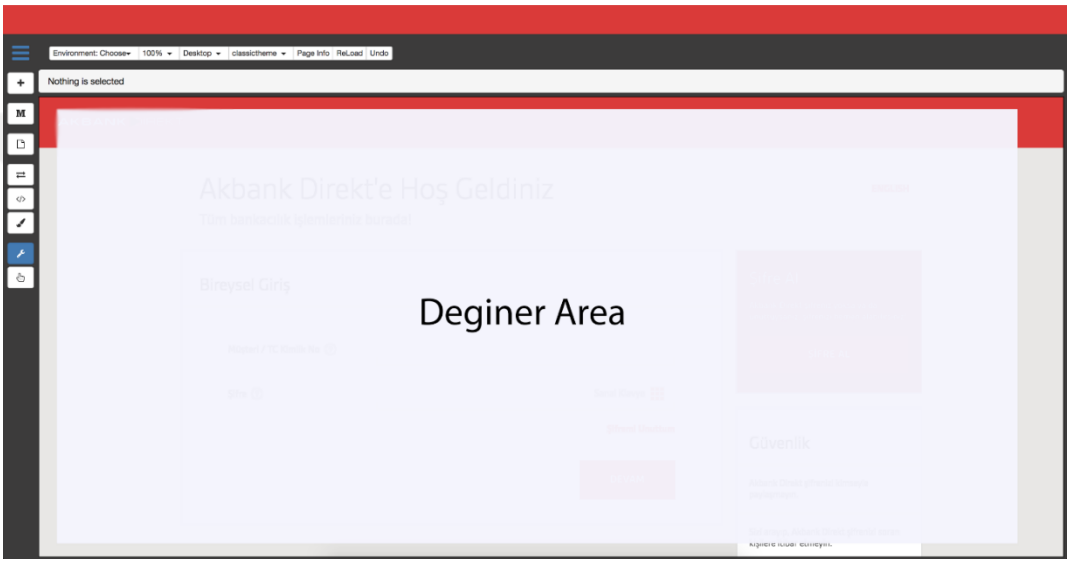


**Figure 13.** *Application designer page screen*

### 2.2.2.2 *Administration pages*

Appbuilder has a dynamic format structure. It can be used in any kind of reactjs web application. It just needs some meta descriptions to identify the applications structure to design. Each definition needed for appbuilder can be managed over administration pages.

#### 2.2.2.2.1 *Component management page*

The most important part of the appbuilder designer property is to define and identify components in the designing application. To do that, appbuilder needs the definitions of the components. Those are mainly the name of the package (npm package name), the object name of the component, a category information for displaying the component in the component toolbox, and finally the definition of the properties.

These component metadata is managed in component management page. Each definition needed can be defined here. Component Management page is shown in Figure 14. Component administration page screen.

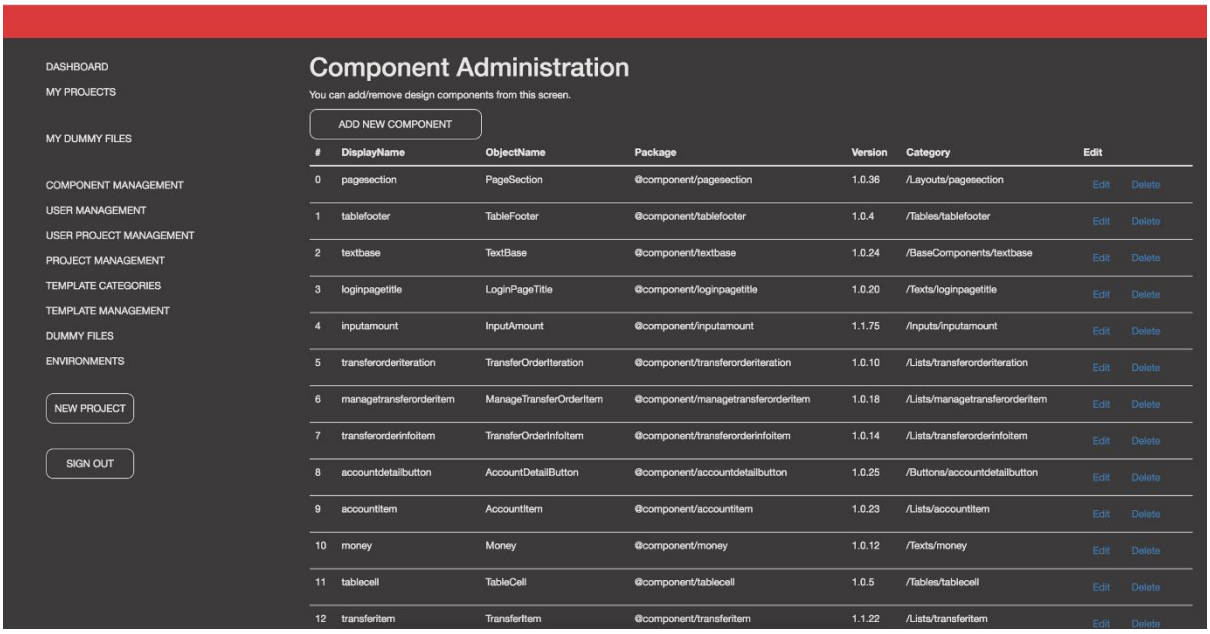


**Figure 14.** *Component administration page screen*

### 2.2.2.2.2 *User management page*

Appbuilder currently has a built in simple user and permission manage0ment. This management can be improved with LDAP like services if needed.

User are defined and permissions are granted in this screen. Each user login information and project specific permissions given here. User management screen shown in Figure 15. User management page screen.

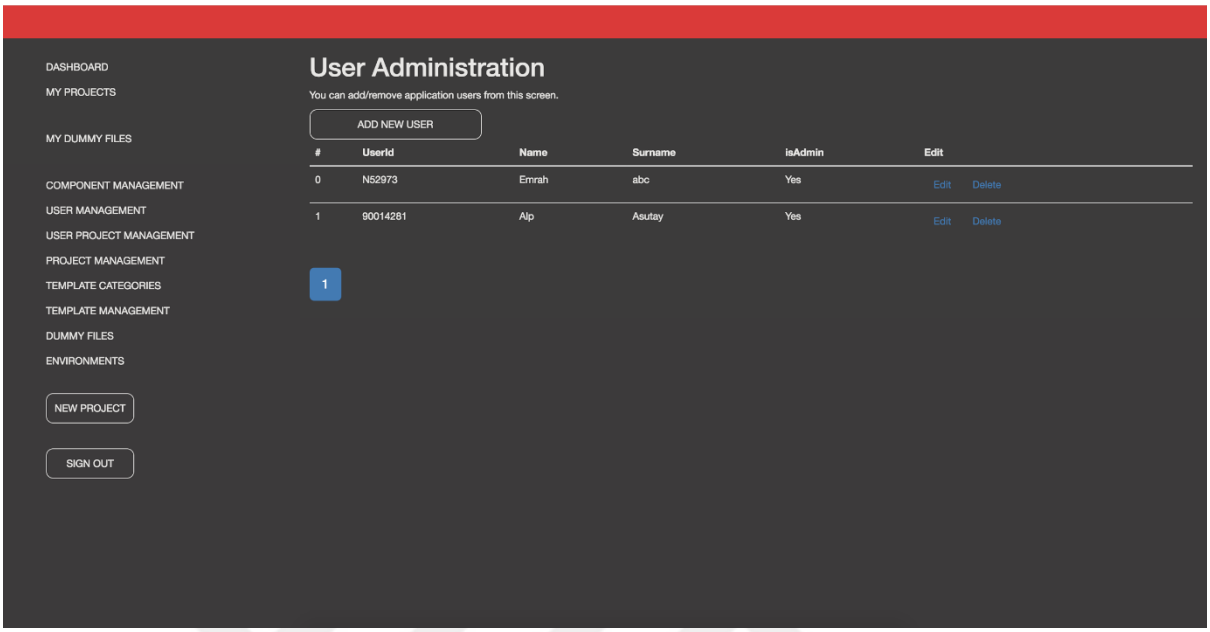


**Figure 15.** *User management page screen*

### 2.2.2.2.3 *User project management page*

Appbuilder has a permission based workspace system. Each user can access only their projects (workspaces). But as an admin user, all of the active projects can be displayed and deleted if needed.

ReactJs web applications has may have so many files (like node_modules directory) inside. So, deleting old projects keeps the server clean. User project administration page shown in Figure 16. User project management page screen.

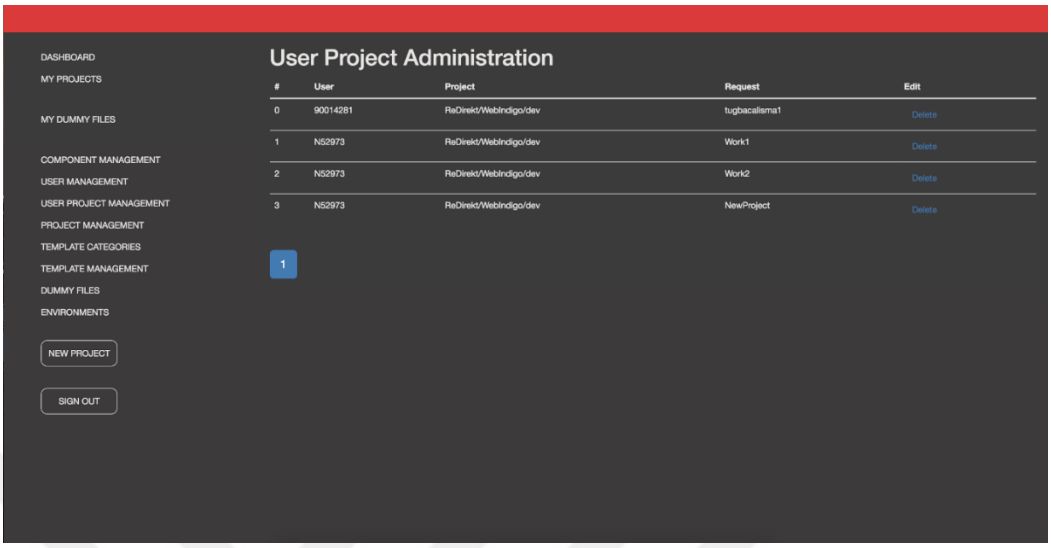


**Figure 16.** *User project management page screen*

### 2.2.2.2.4 *Project management page*

Appbuilder can download projects from their own version control system. Each project needs some definition for that.

By default, Git is used for projects, so Git repository and branch name is required for each project. There can be multiple projects and multiple repositories at the same time.

Project definitions are managed in project management page. Project management page is shown in Figure 17. Project management page screen.

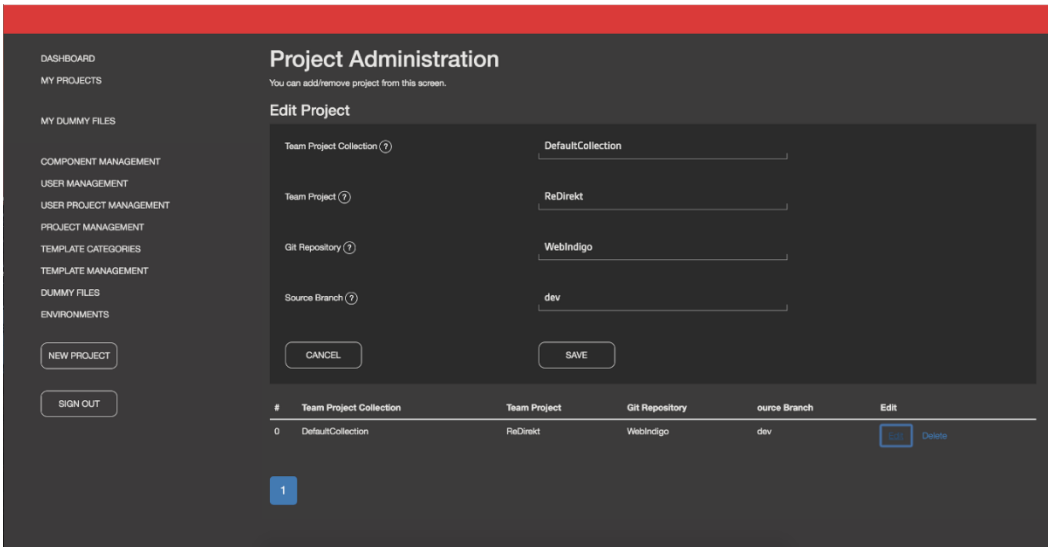


**Figure 17.** *Project management page screen*

#### 2.2.2.2.5 *Template category management page*

Appbuilder can create multiple files concurrently at in a project. This means, users can define some file creation templates (also the body of the files), so that users can create and add to their projects.

With this technique, user can add new pages at the same time. React JS web applications with are using redux framework for state management, may need multiple files to add just a page.

For a big project, there can be several types of template definitions, so they need to be categorized. The templates categories are defined in template category management page. Template category management page is shown in Figure 18. Template category management page screen.

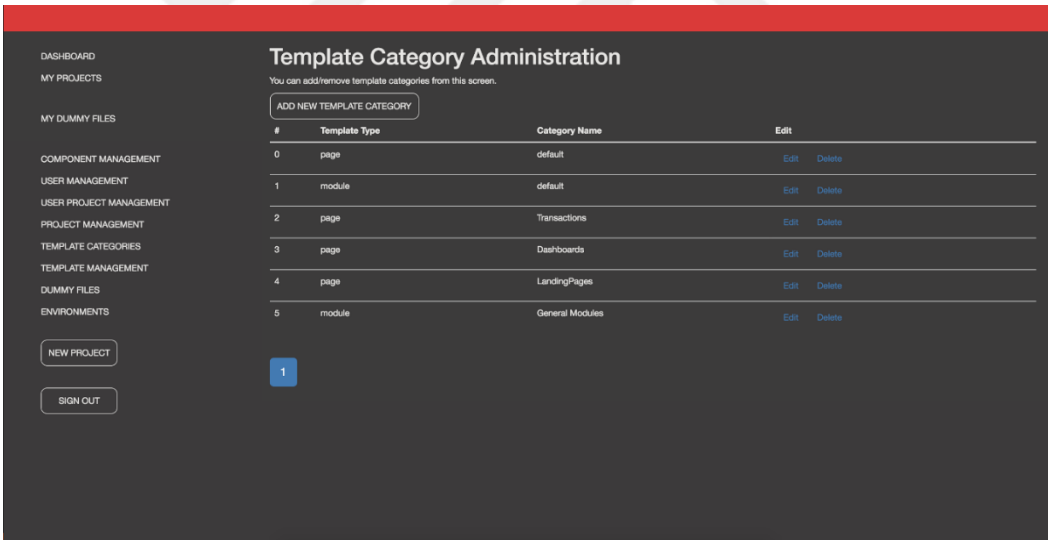


**Figure 18.** *Template category management page screen*

#### 2.2.2.2.6 *Template management page*

As described in 2.2.2.2.5 Template category management page section, appbuilder has need template definitions for file creation operations. Each project has their own file structures and types. So for each project there has to be some template definitions. Template definitions done in template administration page and shown in Figure 19. Template management page screen.

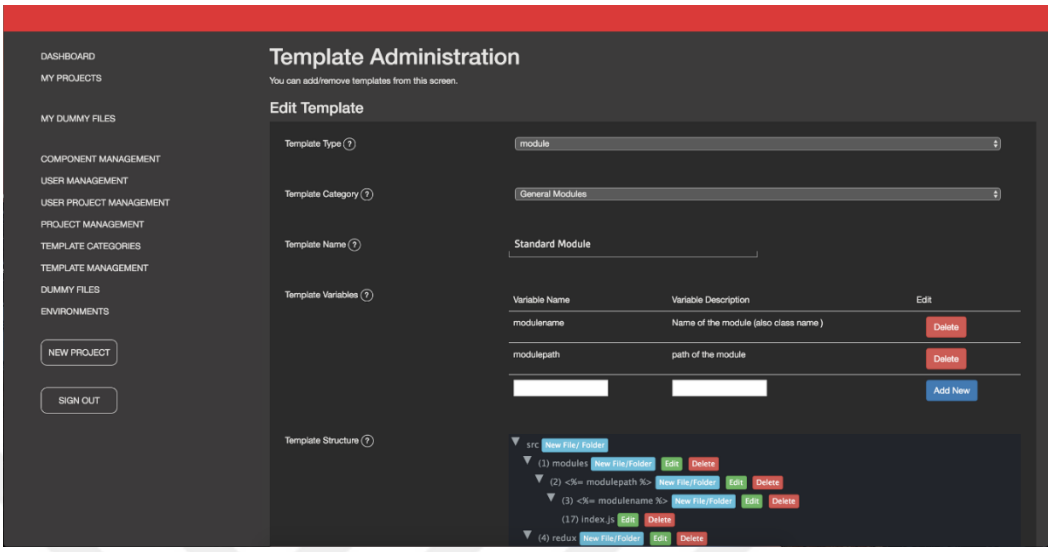


**Figure 19.** *Template management page screen*

### 2.2.2.2.7 *Dummy files management page*

As described in 2.2.2.1.4 User dummy files page section, designing projects need dummy file definitions. Those dummy definitions can be done by each user for themselves only and also there can be definitions that applies to all users. These dummy files that can be used by all users and defined in dummy file management admin page. Dummy file management page shown in Figure 20. Dummy file management page screen.

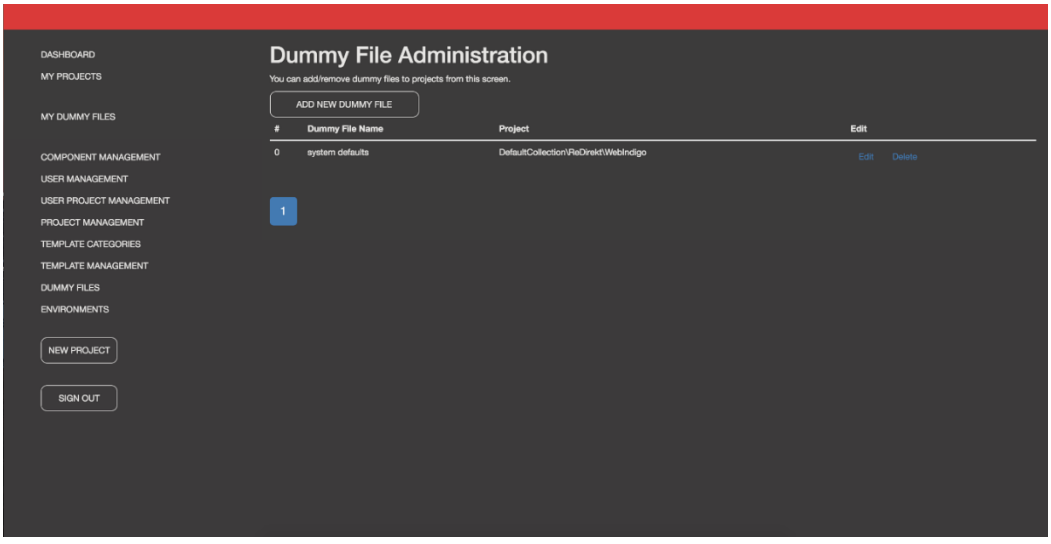


**Figure 20.** *Dummy file management page screen*

#### 2.2.2.2.8 *Environment management page*

Each single page web application has a backend to process operations. For example, to log in users to system applications needs to verify the data entered. Bu single page web applications runs in client's browsers, so it can't do database operations or transactions.

Those operations done in server side and this server side application called backend in generally. In big projects or in big corporates there can be several environments of the same application platform. Those environments generally called "testing environment", "production environment" and etc. This means, there are several backends to the application.

These environment definitions done in environment management page. Appbuilder can change environments of the running application while designing it. Environment management page shown in Figure 21. Environment management page screen.

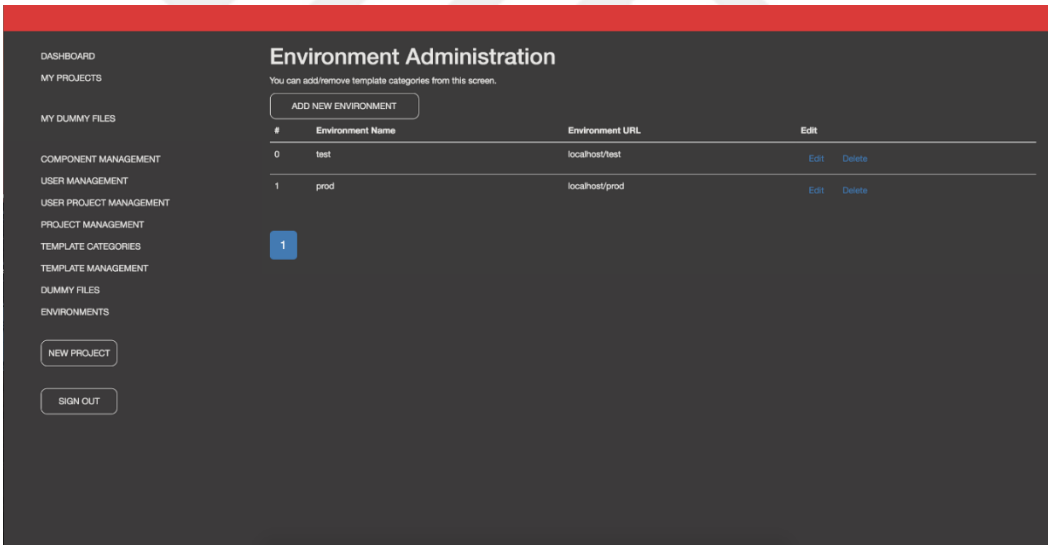


**Figure 21.** *Environment management page screen*

## 2.3 Project methodology

Appbuilder platform has a unique technique that differs than others. The approach makes the application building phase can be done at run time (while designing application is already running).

At a summary ReactJS library is modified a little bit, that components on the running application can be identified. Each file in the project is processed and every component is tagged with some meta data.

Since ReactJs is a %100 component based library, every component in the designing application needs to be tagged. After tagging components in the project, modifications on them can be made.

### 2.3.1 Webpack hot loading

As mentioned in 2.1.5 Webpack Hot Module Replacement (HMR) section, webpack javascript bundling library, has hot module replacement feature. This means, if users bundle together some files (in reactjs applications this means entire application source codes), when user change any source file in the bundle, webpack automatically updates the javascript bundle output with the changes that user have made.

This hot loading feature is one of the main concepts in appbuilder. With the technique that will be told later, source reactjs application files are modified, after sources modified, (a webpack hot loading mechanism already being setup) webpack does the rest and updates the rest of the work.

But for react js applications it is not the only thing for hot loading source files, in react js applications users also need a special package for updating the UI. Webpack only updates the bundle files, it did not update the UI, to do that users have to reload the browser.

ReactJS has react-hot-loader package for dynamically updating the UI when webpack updates occurs. If users do not use this package, they have to refresh the UI every time. This might be a problem if application uses application state, for example user can lost their login session and start from login again. Especially for application state using reactjs applications, -for example application needs login for operations if users don’t have hot loading, developers need to login and may be move across a few pages before what they have coded.

With Webpack Hot Loading and React Hot Loader module together, developers can open their project, runs the hot loader, they code and test together at the same time simultaneously. This speed up the development process very much. In appbuilder, webpack hot loader and react hot loader modules are used in after code generation step to display to user chat changes are made.

### 2.3.2 Changing source files

The entire code generation operations are done in code generation steps. Appbuilder platform processes every file in the designing project while loading. In this process step, every component definition is found and identified.

ReactJs applications has a component driven architecture. Every UI items are components and have to be like that. So, identifying them is easy. The JSX structure is simple.

After project loaded in appbuilder, it knows every page and every item (component) in it. So changing them (source files) can be done easily. For example, app builder modifies the code below [http-1].

Source Code Sample Original:

```
<Box context={this.state.context} clearfix style={{ paddingTop: '15px' }}>
  <Text context={this.state.context} text={this.lang.MainCardNo}
typo='standardBodyCopy' style={{ float: 'left' }} />
  <Text context={this.state.context} text={this.CardDetail.MainCardNumber}
typo='standardBodyCopyBlack' style={{ float: 'right', textAlign: 'right' }} />
</Box>
```

Source Code Sample Processed:

```
<Box uuid="box-10705-tw866hfc1k"
pageid="/Users/emr550m/Documents/Templates/N52973_Work1/src/modules/cards/cardd
etails/virtualcarddetail.js" context={this.state.context} clearfix style={{ paddingTop: '15px'
}}>
   <Text uuid="text-10982-6cslwm4rdb"
pageid="/Users/emr550m/Documents/Templates/N52973_Work1/src/modules/cards/cardd
etails/virtualcarddetail.js" context={this.state.context} text={this.lang.CardExpiryDate}
typo='standardBodyCopy' style={{ float: 'left' }} />
  <Text uuid="text-11308-a7lwxqqjmf"
pageid="/Users/emr550m/Documents/Templates/N52973_Work1/src/modules/cards/cardd
```

*etails/virtualcarddetail.js" context={this.state.context} text={this.CardDetail.ExpireDate} typo='standardBodyCopyBlack' style={{ float: 'right', textAlign: 'right' }} />*
*</Box>*

As shown in the code samples above, each file is processed while loading the application. In source code that is processed, each component definition has 2 extra properties after processing. One called uuid and the second called pageid. UUID property is a unique value for every component in the entire project. PAGEID property is the physical file that stores the component source code.

Appbuilder injects those property values into their objects in the browser. So when user selects a UI component, appbuilder knows where is the definition of the component (which file and line number in the file).

It is just basic string operations to modify files after knowing the location of the change. It is possible to add new properties, modify current ones, insert new components. Regular code generation operations can be easily done.

### 2.3.3 Running designing application

Appbuilder enables users to run the designing application at the same time while designing (code generating) it. Each project workspace creates a working directory identical to developers local working (project) directory and they have their own webpack and hot loader modules separately. Those separate projects served under a core express web server. Each project has its own route under express. These routes added dynamically to express server when user creates a project workspace.

For example, appbuilder is running in address http://localhost, when a user creates a project, appbuilder downloads its sources and creates a web project for it. When appbuilder finishes starting project, an address available for viewing the project like http://localhost/user1_Project1.

The designer page described in 2.2.2.1.6 Application designer page section, this url is shown inside the designer as an inline frame. So that, appbuilder displays the designing app while users designing it.

### 2.3.4 Parsing designing application

As described in 2.3.2 Changing source files section, each file in the designing project is processed and parsed before the designer starts. While parsing the javascript source files in the project a JSX parser library is used. This library called astparser and a running sample [http-11].

Ast is a meta description file that stores data about source files and its structures. Appbuilder uses ast definitions of javascript source files (compilers also use ast files in compilation). Esprima-fb package is used to get ast definitions of source files.

Example ast structures can be shown in astexplorer.net's web site. Source code files described as parsed objects. For appbuilder perspective, these parsed objects are very important to identify the source files (pages in the designing application). By extracting ast (parser) definitions from source files, appbuilder can identify components, variables or any other definitions in the sources [http-1].

### 2.3.5 Identify components & component selection

As mentioned in 2.1.2 React.JS section reactjs based applications are coded completely from components. Like object oriented programming languages, reactjs is component oriented programming language.

Every UI element in a reactjs application has to be in reactjs component object. In reactjs, those components used in JSX language syntax. As mentioned 2.3.4 Parsing designing application section source jsx files are parsed and as mentioned in 2.3.2 Changing source files section each parsed source files items are tagged with metadata.

By using injected metadata in the source files, appbuilder identifies the components displayed to user. This done by modifying the reactjs source itself.

In reactjs source code, the rendering components to dome part is modified a little bit. Normally, ReactJS did not cares what is included dynamically to components. Because components don't have these two properties in their definitions (injected properties mentioned in 2.3.2 Changing source files section). What is changed in reactjs source itself is reading those extra two properties and inject them to the DOM at the runtime.

This modification makes us to know which component is on the UI at any time. Because inside the browser's UI object (DOM), our component meta descriptions are also included.

Now the components at the source code is well known, also the objects at the UI, so it is easy to match them together. Appbuilder has a feature highlighting UI component while user moves the mouse over them at the designer.

Because what was inside the DOM is already known, there is a hidden layer above the designing application, appbuilder gives a colour to it, makes its boundaries equal to the components boundaries. So, user feels like the component is highlighted. When user clicks to this coloured layer, it is already known which component is related to this layer, so a callback to appbuilder backend is made and the metadata is retrieved from backend. That's how the selected components information displayed to the user at the properties window.

### 2.3.6 Modifying pages & components

As described in 2.3.5 Identify components & component selection section, each component rendered to UI on the designing application is known. They are also known where they are on the UI, what is the sizes etc.

Figure 22. Highlighting components shows a sample form application running on the appbuilder. When user moves the cursor above elements, the components highlighted.

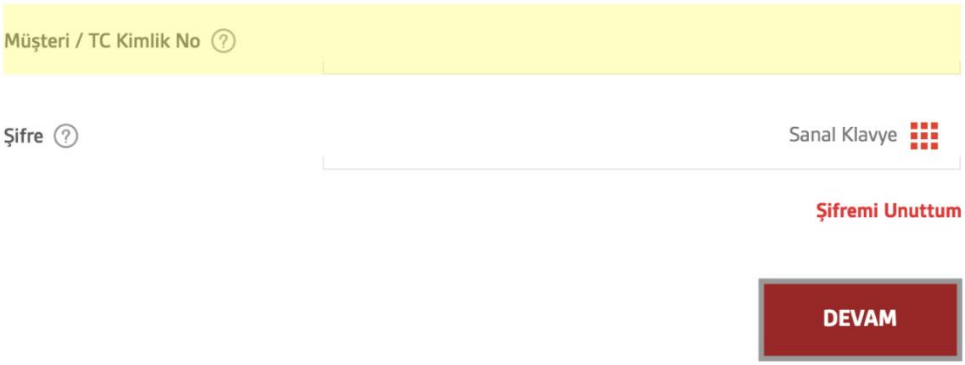


**Figure 22.** *Highlighting components*

If user selects (clicks) a highlighted item, component is selected as shown in the Figure 23. Selecting components. The allowed operations displayed above the selection highlight.

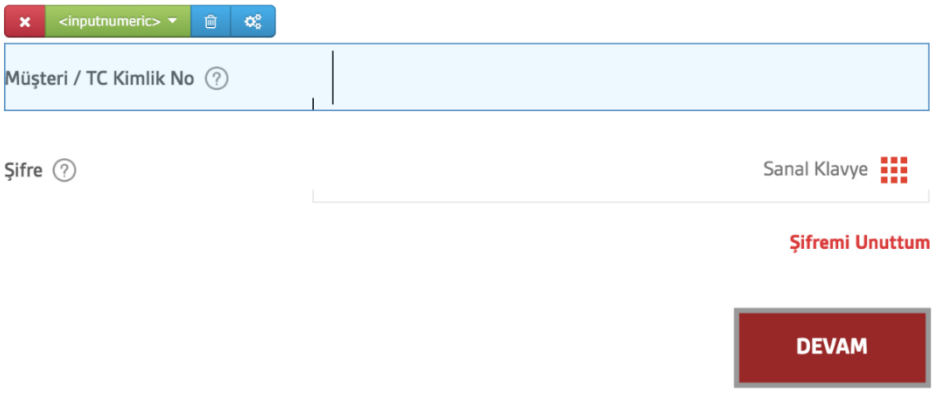


**Figure 23.** *Selecting components*

When a component selected, users can modify it. The property toolbox window can be opened from main left bar as shown in the Figure 24. Property toolbox.

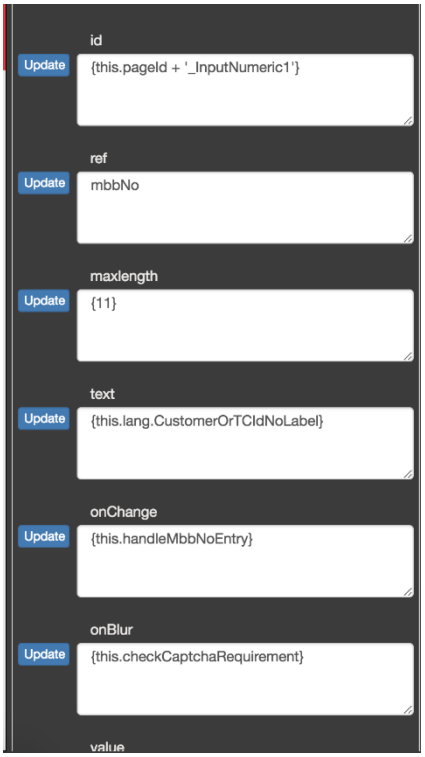


**Figure 24.** *Property toolbox*

When users change a value in property toolbox and click save button, the requested operation sent to backend with a backend call, because it is already known which component is selected, it is also known where is the file and which lines in the source file (from metadata that was inserted before). With simple text and string manipulation operations, the modification is done. Because reactjs uses JSX syntax it is easy to parse and modify.

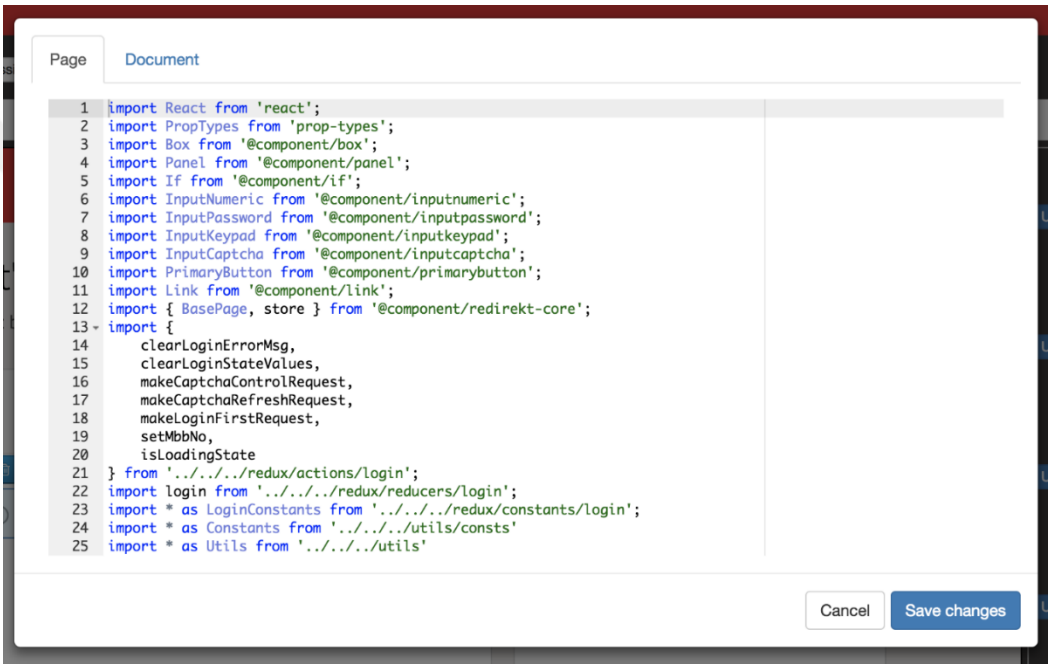


**Figure 25.** *Source file editor*

Appbuilder has an embedded source file editor. The editor itself is an open source npm package called ACE editor. When user select an item on the UI and click the source button “ ”, appbuilder finds the target file from the meta definition and opens up the ace editor with the target files source in it.

Users can modify the source files if needed. This is an advanced mode for appbuilder. The main idea in appbuilder project is to generate source code, not modifying the source by hand. But if users are a software engineer and they know what they are doing, it is okay to change the source and examine the output directly.

In a single page web application, there can be multiple pages in the app. So, user need to navigate to those pages while application is running. Because appbuilder can design and modify the current page (what is rendered to the user screen).

Because the app is physically running inside the appbuilder, users can navigate to the page that they want to modify by using applications menu or flow. Bu when they create a new page from scratch, there are no links to their page. That's why appbuilder has a routing table and users can navigate to any page they want.

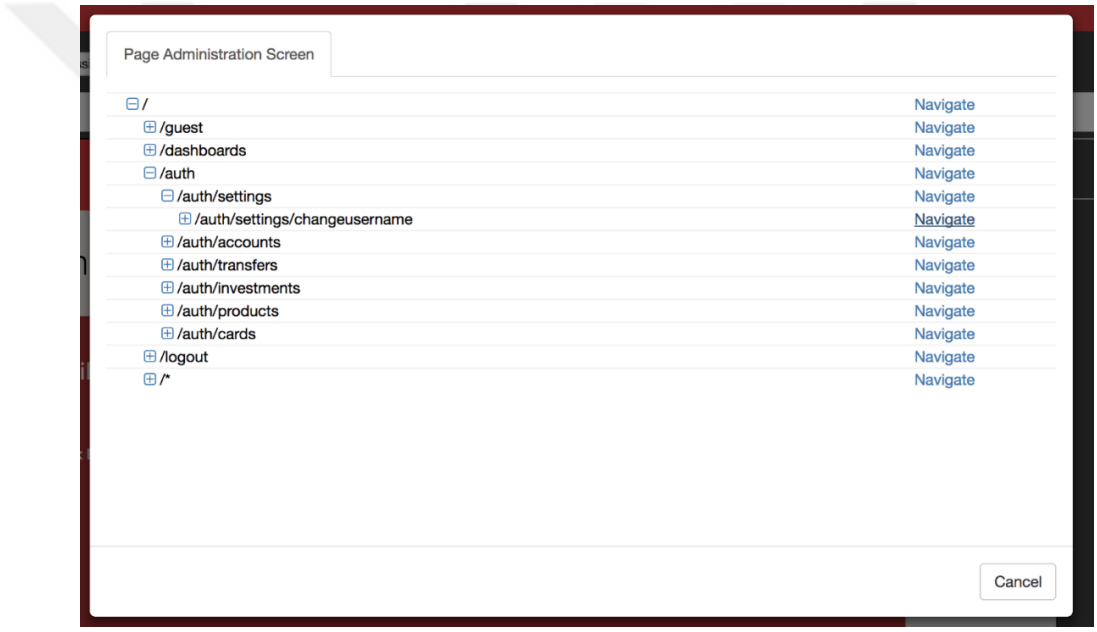


**Figure 26.** *Sample application page selection and navigation*

As shown in the Figure 26. Sample application page selection and navigation, appbuilder identifies what pages application has and can navigate the screen to that page.

But in most cases, applications have built in authentication mechanisms, this means users can't navigate directly to these pages. So, they may need to use mocking operations or they many need to login their application while it is running.

Every design option also available when users navigate to another page. Because metadata is injected to every page while loading the project from source control and they have been modified the reactjs source control, when a new page will be displayed, all metadata will be injected to the new page while it will be rendered to UI.

### 2.3.7 Communication between backend and frontend applications

Appbuilder has two main parts in the project. First part is a nodejs applications and the second part is a UI application served from that nodejs backend part.

Since there is a 2-layered structure there has to be a communication between them. There are two types communication between these two parts. One is a live connection between the nodejs backend, one is a callback type not alive connection.

Live connection between UI and the nodejs backend is called socket connection and based on socket.io framework. This socket framework used for sending webpack hot loader messages to the UI. This has to be alive because at any time there can be change occurs on the source files.

Callback communications are used to send commands from UI application to backend. These are generally code modification request. They are one-time operations that sends a command gets a reply type. These callbacks are not keep alive kind calls, they ‘ll expire and timeout if they take too much time to execute. So, they have to be quick while using those callbacks.

For example, the webpack bundling at the project designing start takes between 1 to 4 minutes depends on the project, for that time they can't hold a backend call because it will timeouts and terminates the call. For that kind of call backs, socket infrastructure is used, and data sent over socket without timeouts. Because socket is open all the time and won’t close until the project closed [1].

### 2.3.8 Modify Reactjs source code for appbuilder

As mentioned before, reactjs libraries original source code is modified to the code generation two way (from generated to hand coded, hand coded to generated). The modification is so simple in the reactjs library, a few lines added to it to magic happen.

In the source file ReactCompositeComponent.js;

```
if(inst.props.uuid) {
  var domNode = findDOMNode(inst);
  if(domNode) {
    var idMap = -1;
```

```
    var targetuuid = null;
    do {
     idMap = idMap + 1;
     targetuuid = idMap == 0 ? inst.props.uuid : inst.props.uuid + "_" + idMap;
    } while(window.uuidHashMap[targetuuid]);
    window.uuidHashMap[targetuuid] = true;
    domNode.setAttribute('data-uuid',targetuuid);
    domNode.setAttribute('data-pageid',inst.props.pageid);
    if(inst.props["data-locked"]) {
     domNode.setAttribute('data-locked',inst.props["data-locked"]);
    }
   }
}
```

These code block injects the metadata inside to the real DOM object. There is an index of injected meta data also created for easy access and use in the UI. For module definitions, there is also a special lock metadata inserted that will described later.

### 2.3.9 Modules, components and more

In reactjs, every item is called components. But in projects, users may need some complex components that may be contains a few components together and they may need to use this complex component item more than once.

Those items called “modules” in appbuilder. Users can create modules and use them as much as they want in their pages.

Pages are also having special meaning in appbuilder. Pages are also reactjs components but appbuilder has to say something page to allow users to design it.

So as an hierarchy, appbuilder can be summarized into three main concepts. Pages, Modules and Components. As described in 2.2.2.2.1 Component management page section, components can be managed by appbuilder administration page. But for modules and pages they are not common for every project and they cannot be administered.

When appbuilder loading a project, it scans every file for page and module definition. There is a static rule on these. Page definitions are extracted from reactjs applications route definitions. Module definitions are extracted from modules folder.

Each project has a folder structure inside. Appbuilder expects there is a modules folder inside the project and every component definition inside it called and indexed as module. Just like the components as administered from appbuilder, module list is sent to UI for using in designer purposes.

As shown in Figure 27. Sample modules tree for a sample project figure, module definitions extracted from designing project listed to the user. When user selects one of these modules, it can be inserted to any location on any page.

By using modules, each project can create their project specific parts, declare them, design once and reuse everywhere. These modules also can be created in appbuilder. An empty module can be created with the Create New Module section shown in Figure 27. Sample modules tree for a sample project figure.

As mentioned in 2.2.2.2.6 Template management page section, the modules code structure can be defined to use here. Each module template definition is displayed in the new module section, users can create a module just giving the module path and module name. After adding the module, it automatically added to the list and user can select and insert to any page.

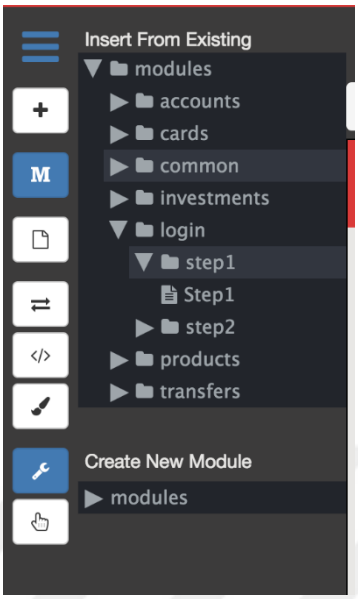


**Figure 27.** *Sample modules tree for a sample project*

When user inserts a module to a page, by default it is locked for modification. Because modules can be used on many pages, modifying a module by accidently, cause every usage in the project to fault.

So, user need to unlock a module in the page before modifying it. This is done the lock and unlock button at the top of the highlight frame as shown in the Figure 28. Module locking and unlocking.

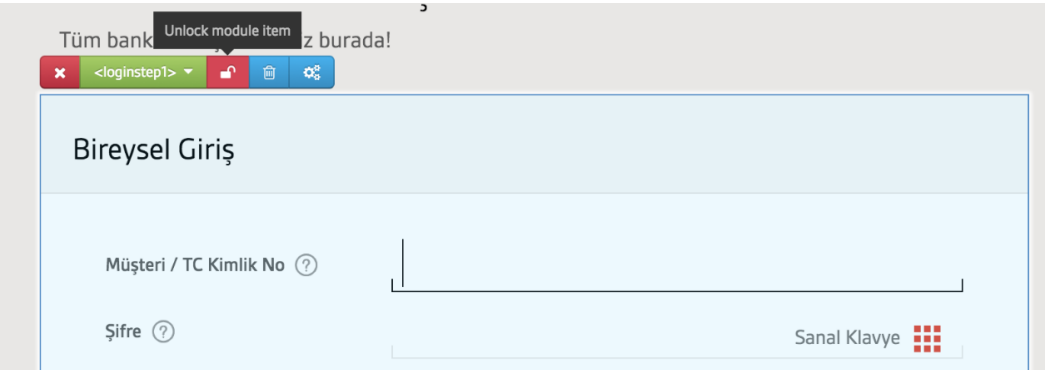


**Figure 28.** *Module locking and unlocking*

Just like appbuilder does for the components, for modules their source files metadata injected to its DOM object. So, when user modifies something in it, it changes on its very own source.

### 2.4 Multiple User Interface

Appbuilder is a multi-user project. Multiple users can use at the same time. There is no limit about that. By default, regular login is used. User definitions stored in its own database [2].

#### 2.4.1 Multiple user support

Each user can login concurrently to the appbuilder and can start their own project. And also, a single user can create multiple users at the same time too [3].

#### 2.4.2 Multiple project support

Appbuilder can create multiple hot loading units at the same time as mentioned before. As mentioned in 2.1 Summary and Used Technologies section, each designer unit has a big infrastructure inside it (Figure 3. Common structure of a single user's project structure).

With multiples running at the same time concurrent designing tasks can done at the same time. This is limited with the physical limitations of the server that runs the appbuilder. Multi user structure shown in the Figure 4. General structure of app builder project.

### 2.5 Designer/Code Generator Page Details

Most of the project operations occurs on the appbuilder designer page. It is the core of the project. Users do the code generation and appbuilder operations on this page [4].

#### 2.5.1 Management panel

Main code generation operations done in the left main panel.

**Figure 29.** *Main management panel (left panel)*

This management panel has these features on it.

#### 2.5.1.1 *Main application menu*

The main application menu contains (Figure 30. Main application menu)

**Export Project**: This creates a single zip files that contains the designer applications entire source. This export feature is described before.

**Publish Project**: This feature publishes (commit back) the sources that generated in the project to the source control.

**Go to Dashboard**: This feature ends the designing session and navigates user to the dashboard screen.

**Sign Out**: Signs out the user from appbuilder.

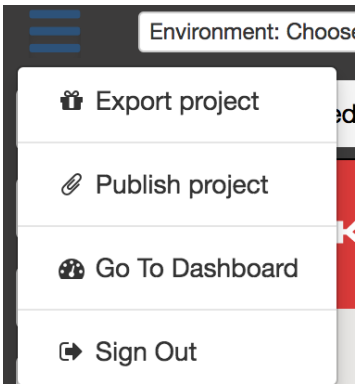


**Figure 30.** *Main application menu*

#### 2.5.1.2 *Component management panel*

Components that are defined for designing operations are displayed and selected for adding to the project from here. The plus sign (+) displayed in Figure 31. Component management panel figure.

Components are displayed in categories in this panel. That category information is extracted from component definitions or can be administered from by component administration pages.

When user selects a component in the selection bar the selected components name appears and user understands that he/she can insert this item to the UI wherever he/she wants. Component items also can be dragged and dropped to any place.

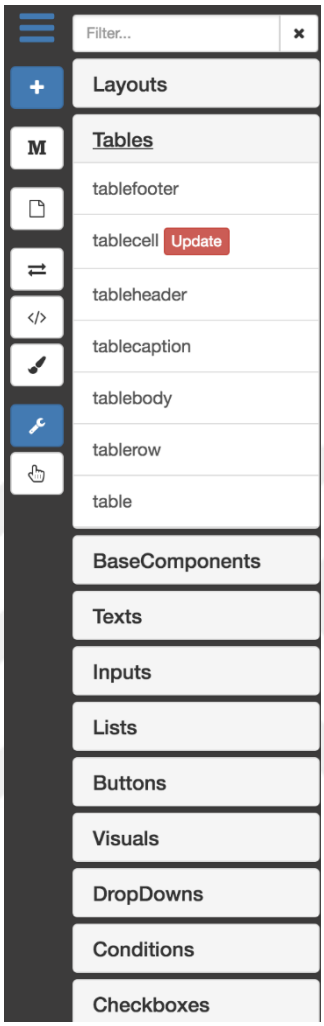


**Figure 31.** *Component management panel*

### 2.5.1.3 *Module management panel*

As already described in 2.3.9 Modules, components and more section identified modules are displayed in modules panel (opens with M icon in the Figure 29. Main management panel).

Like shown in the Figure 27. Sample modules tree for a sample project figure, new modules also can be added to the application from here. Selected modules inserted to the project just like components. When a module is selected, it will be displayed in the selection bar and the user can insert it any place on the project. Module items also can be dragged and dropped to any place.

#### 2.5.1.4 *Page management panel*

Appbuilder has the ability to add new pages to the project. Those pages created from template definitions that described in 2.2.2.2.6 Template management page section.

Each template definition has its own parameters that prompted to the user while creating the page. These parameters like the page name, page route or back route information.

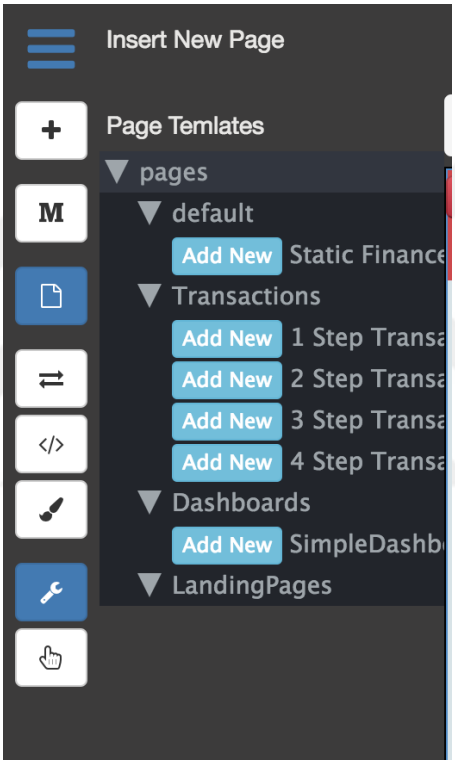


**Figure 32.** *New page creation menu*

#### 2.5.1.5 *Dummy files management panel*

Dummy files may be needed for some backend using applications while designing the application.

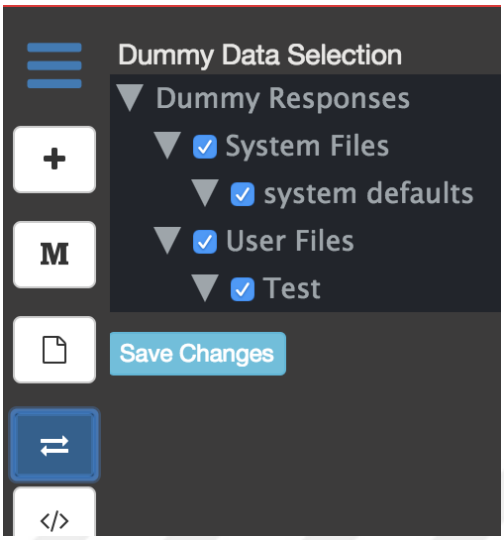


**Figure 33.** *Dummy file selection menu*

As described in 2.2.2.2.7 Dummy files management page and 2.2.2.1.4 User dummy files page files page sections, there can be several dummy definitions for backend calls. There can be even more than one definition for the same service. Because there can be several scenarios on the same screen. Users selects the active dummy files from this screen, the selected dummy files are served from node js backend if the application is running on dummy mode.

#### 2.5.1.6 *Component hierarchy panel*

When appbuilder parsing every file at start, it is also possible to create a component hierarchy tree for selecting items in the screen. React JS is a component base language and components can be structured nested form in pages.

Some components may not have a width or height or some components may not even display in the screen. For example, there can be If, Select or Loop components (logic components). Those components do not have a volume on the screen so it cannot be selected on the screen.

With the component hierarchy tree window, users can select any component for modification. Hierarchy panel is shown in Figure 34. Component hierarchy panel figure.

**Figure 34.** *Component hierarchy panel*

#### 2.5.1.7 *Properties management panel*

Properties windows can be opened with the properties button in the main menu. The opening toolbox is shown in the Figure 24. Property toolbox. This toolbox is one of the most important panel on the application.

#### 2.5.1.8 *Designer mode button*

Appbuilder has the capability to design the application at the run time. Users can modify the pages while it was actually running on the browser. So there is a switch needed between designer mode and preview (running) mode. These options are on the main panel. As shown on the Figure 35. Designer mode button figure, user can switch between designer and preview mode.

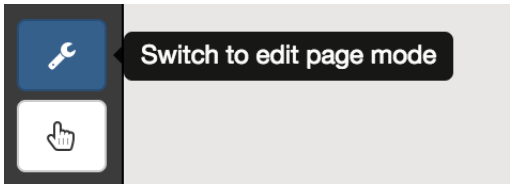


**Figure 35.** *Designer mode button*

#### 2.5.1.9 *Preview mode button*

When user clicks the preview mode, appbuilder designer add-ons (hidden layer to highlight components) removed above the project. Because the application is also running inside the appbuilder, the rest after the layer removing is the application itself. So it can be said that preview mode is running the application (Figure 36. Preview mode button).

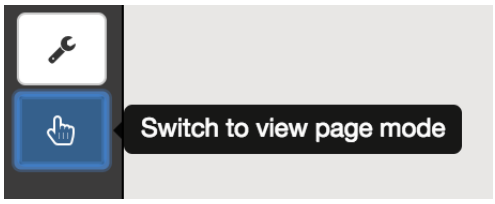


**Figure 36.** *Preview mode button*

### 2.5.2 Configurations panel

If an application is running on a web browser, it needs so many extra features except its own business. For example, web browsers run on any device, any size of screen, so application needs to be responsive in the browser.

There are also different platforms that uses the web. Like mobile phones, tablets and thousands of types computers. Designer application may be needed to be different on each platform.

Appbuilder has the ability to change the platform (by simulating it), it can simulate variety of screen sizes while designing the application. For example, user can see the app in a mobile size (320 pixel) width if they want.


**Figure 37.** *Configurations panel*

From configurations panel, there are several options can be changed:

**Environment Choose**: As mentioned before, designing application can have multiple backend environments like, production, testing etc. Users can define and select the running applications environment from this menu. Dummy mode also enabled from this section.

**Screen Size Selection**: Users can change the running applications screen size to some predefined screen sizes. Various screen sizes supported for testing the applications screen. From here as shown in the Figure 38. Screen size selection.

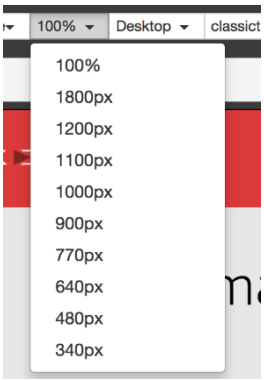


**Figure 38.** *Screen size selection*

**Platform Selection**: There are mobile, tablet and desktop mode simulation selected from here. Appbuilder acts as if is running inside the selected platform.

**Theme Selection:** If the components supports theming option, it can be integrated and changed from that option.

**Page Info**: The routes and pages defined in the designing application is displayed from here. User can navigate to any page from the routing menu (Figure 26. Sample application page selection and navigation).

**Reload**: Sometimes users need to reload the application runs inside the appbuilder (The designing application). Reload button reloads the application and the appbuilder metadata operations done again.

**Undo**: Each code modification done in appbuilder can be undone with this button. Every step is saved in a designing session. Each step can be undone one by one. When user closes the session undo buffer cleans out. This means users can't undone the previous design sessions.

### 2.5.3 Selection panel

In appbuilder designer screen, selection on the running designing application and selection from the components tree has special meanings. The selection panel displays what is currently selected. So, users can do different operations on what is selected. On the lower line the selected component on the UI is displayed. On the upper line, the selected components tree is displayed. The selection panel is shown in Figure 39. Selection panel.


**Figure 39.** *Selection panel*

### 2.5.4 Properties panel

Properties panel also shown in Figure 24. Property toolbox, is used to modify what is already inserted to the application. Current values also displayed to user and user can modify them too.

### 2.5.5 Page component tree panel

As mentioned in 2.5.1.6 Component hierarchy panel section, the component tree is used to select and summarize the page's component hierarchy.

# 3. SCIENTIFIC FACTS AND RESULTS

## 3.1 Scientific Facts

In this section, why this technique is needed described with facts.

### 3.1.1 Why this technique is needed

In software development lifecycle, development phase is the most expensive and time consuming one. Everything that is done for speeding up the development phase, speeds up the projects.

Most of the corporates that does products for themselves or their customers, has similar business development steps. First development need or job comes from business line or management. After main product described by them, details are outlined by analysis people or product owners. By having a described task, developers can do their job. But all that procedure has circular repeat over and over again because job needs may change, or business line changes opinions after what they seen that coded.

To speed up those procedures, this appbuilder platform can generates the UI of the project easily by any level of people, even managers, analysis people or product owners. Job can be seen and examined even before it will be coded. So mostly repeated coding sessions can be skipped that way.

### 3.1.2 Benefits of code generation

For developer side of code generation, most repeating tasks can be done in seconds. Every project has repeating patterns inside. For example, each unit function has same outline or template as said. First, code generation saves too much time for repeating tasks or codes.

With code generation technique, also mistakes and bugs are decreased tremendously. Because there is no handwriting error, only mistakes can be occurred from false definitions. If users can define the templates correct for once, there won't be any mistakes any time after that.

For DevOps, code generation also can be used. If users can generate source codes, they can also generate unit tests too. That means, for continuous integration steps users can take benefit too.

### 3.1.3 Possible problems

It is not always easy to generate every code or page in applications. To generate codes without having hard time, users need to keep things simple and clear. If they have complex architecture in their applications generation may be convert their life more complex.

In ReactJS based applications and for appbuilder platform, pages must be declaratively written in render block. If pages are generated by javascript blocks and executions, appbuilder platform cannot identify the component blocks and it won't' run correctly.

## 3.2 Performance Metrics

To proof what have done so far, side by side comparison handwriting and code generation that is done by appbuilder is needed. To do that, a sample 20 people (they need to be developers, so that they can write the same task) is selected. Each 20 people does the same task with handwriting and by using the appbuilder. Each test subject also has the ability to write in reactjs and they know to use appbuilder platform. There are several tasks from easy to hard. Each task is examined below;

### 3.2.1 Basic page development in app builder versus hand coding

In this test, test subjects were asked to create a simple page that has two inputs on it. Users first do the job by handwriting and after by appbuilder; on the same project and starting at the same point. Each test iteration page was asked to developers with different input names and types but same amount number two.

**Table 4.** *Average Times Taken in Basic Page Development*

| | | Average Time Taken | Number of Users in Iteration |
|---|---|---|---|
| **Iteration 1** | App Builder | 5.6 min | 3 |
| | Handwriting | 12.4 min | 3 |
| **Iteration 2** | App Builder | 7.8 min | 5 |
| | Handwriting | 16 min | 5 |
| **Iteration 3** | App Builder | 6.3 min | 15 |
| | Handwriting | 14.7 min | 15 |

As shown in Table 4. Average Times Taken in Basic Page Development number of development count used in tests did not makes changes too much the average times. For a basic page, it can be said that nearly %50 of time saved while developing with appbuilder platform.

This happens because page templates defined in appbuilder and users can create the page outlines just in seconds. The time taken in giving the properties of the inputs. This means, more detailed templates give more generated codes.

### 3.2.2 Advanced page development in appbuilder versus hand coding

In this test, test subjects were asked to create a complex page that has two subpages on it. Each subpage has 5 inputs in it. Users first do the job by handwriting and after by appbuilder; on the same project and starting at the same point. Each test iteration page was asked to developers with different input names and types but same amount of inputs.

**Table 5.** *Average Times Taken in Complex Page Development*

| | | Average Time Taken | Number of Users in Iteration |
|---|---|---|---|
| **Iteration 1** | App Builder | 12.2 min | 3 |
| | Handwriting | 35.3 min | 3 |
| **Iteration 2** | App Builder | 15.2 min | 5 |
| | Handwriting | 42 min | 5 |
| **Iteration 3** | App Builder | 16.1min | 15 |
| | Handwriting | 39.4 min | 15 |

As shown in Table 5. Average Times Taken in Complex Page Development number of development count used in tests did not makes changes too much the average times. For a complex page, it can be said that nearly %65 of time saved while developing with appbuilder platform.

This happens because page templates are defined in appbuilder and users can create the page outlines just in seconds. The time taken in giving the properties of the inputs. This means, more detailed templates give more generated codes.

It is understood that when codes or pages becomes complex, the time that users have gained increases. But there is a break point for that, if things gone very complex appbuilder loses the generation ability and times changes and loses the gain over handwriting.

### 3.2.3 Bug rates in basic page in appbuilder versus hand coding

Each test iteration was evaluated (code reviewed) after developers finishes their work. Bug's called miss-writings or malfunctioning on the code. Since appbuilder generates the codes by itself and the templates defined for those pages are bug clear, users do only definition errors. But in hand coding, users define and writes everything and this also reflects the bug counts.

**Table 6.** *Average Bug Count in Basic Page Development*

| | | Average Bug Count | Number of Users in Iteration |
|---|---|---|---|
| **Iteration 1** | App Builder | 1 | 3 |
| | Handwriting | 3 | 3 |
| **Iteration 2** | App Builder | 2 | 5 |
| | Handwriting | 4 | 5 |
| **Iteration 3** | App Builder | 2 | 15 |
| | Handwriting | 5 | 15 |

As shown in Table 6. Average Bug Count in Basic Page Development, appbuilder decreases bug/error very much (Two or three times better).

### 3.2.4 Bug rates in advanced page in appbuilder versus hand coding

When coding becomes complex, the rate of bugs increases. But in appbuilder, the only difference is the definitions are more. So, the amount of bug increases both sides. But appbuilder also a winner here.

**Table 7.** Average Bug Count in Complex Page Development

| | | Average Bug Count | Number of Users in Iteration |
|---|---|---|---|
| **Iteration 1** | App Builder | 3 | 3 |
| | Handwriting | 14 | 3 |
| **Iteration 2** | App Builder | 5 | 5 |
| | Handwriting | 16 | 5 |
| **Iteration 3** | App Builder | 4 | 15 |
| | Handwriting | 15 | 15 |

### 3.2.5 Appbuilder in continuous delivery

Since continuous delivery is a hot topic these days, every step that users have done for it makes development life clear and fast. Continuous delivery means, they have a running application or software and users are adding new features to it without breaking its working and doing the development process seamlessly. Seamless means, any part of the development procedure did not become an anchor. For example, bad coding makes the testing phase longer, poor analysis makes turn backs from coding etc. If users can define a perfect environment for all their development lifecycle that any phase did not blocks the procedure this means they have a continuous delivery ecosystem.

Where appbuilder can used in continuous delivery is all its users choice. Users can use the appbuilder in design or analysis phase to get clear definitions, they can use appbuilder in development phase to get perfectly written codes or pre-written unit tests that checks users coding.

The base idea in appbuilder to generate codes but it can be use in different areas. So, users can use the appbuilder in multiple steps of continuous delivery.

### 3.2.6 Appbuilder in devOPS

In recent years a new term is created, DevOPS. That word means development operations. This term is used to define every step together that users have done for continuous delivery. Most of the work in dev-ops done in build mechanisms or in source control mechanisms like TFS or Git.

In projects, to keep things clear and well, some methodologies used like test driven or behaviour driven developments. Writing tests is the main idea on those concepts and running those tests while modifying the project source itself. Since it is possible to generate unit tests by appbuilder platform, it can be said that it is also useful in devOPS ecosystem.

### 3.2.7 Appbuilder in cloud platform

Cloud platform is a hot topic in recent years. Since appbuilder is a web based application, it is very possible to run and use appbuilder in cloud platform.

Because appbuilder may need computer power to do multiple designing sessions in parallel, it is ideal to run it on cloud platform. It can be easily scaled over many servers on cloud platform. To use appbuilder in cloud, all users need is node js as described before.

## 4. USER COMMENTS

In this section, user's opinions are gathered and summarized. Users are divided into two section, developers and users. As software developer perspective things and opinions may differ than regular users. Appbuilder can be used by both regular users and software developers.

### 4.1 User Comments

Önder Akar - General Manager of VOLT Bilişim A.Ş.**:** There are several problems on the business software development process. Some of these uses infrastructures that helps to write less code to adapt changes fast. While trying to manage these criteria, both software companies and companies that use business applications are faced with projects that are not able to meet the needs properly and dragged to chaos in time. For this reason, some frameworks automatically generate code blocks and screens. Those auto generated codes has some standards but they are not able to modify by hand and in time inconsistencies occurred. Bidirectional code generation may help to answer the quickly changing needs in time easily and may help the projects can go live fast.

### 4.2 Instructor Comments

Assoc. Prof. Dr. Özgür YILMAZEL: For big enterprises, it will always be a dilemma to use or not use code generation techniques. Because, in most cases, generated codes are become useless or will be modified too much in time. Since the code generation is done from a source language most of the time, those source files will deprecate in time. Described Bidirectional code generation pattern, may be a solution for this problem.

Since the latest source codes used for code generation as start, there won't be any deprecating source files for code generation. The idea of code generation after manual coding on the same source may be the key point of the work and what makes it valuable.

### 4.3 Software Engineer Comments

İlhan Erikçi - Software Architect - Digital Banking Applications - AKBANK T.A.Ş.: In 1990s and early 2000s we had WYSIWYG editors to build web pages. These editors

gave us the comfort of seeing the effects of the changes we were making. In fact, the page was being built in design mode by drag'n drop.

While new technologies and new patterns arose, developers lost this comfort. Now designers and coders work separately, merge their work after finishing and hope to run smoothly.

A real-time application builder is the solution for this complexity. With such an application developer can drag and drop existing "components" and see the effects like the way WYSIWYG editors provided them before. This is a huge step for web development.

Alper Soğukpınar - Senior Consultant – Microsoft: In large applications, page development time is long and business needs are taking long time to come up. There may thousands of pages in an institutional application. Therefore, reducing this page development time will be a significant gain in terms of cost and speed of development.

Although page templates, design, and design rules for such large applications are developed manually, these pages can easily become a non-standard structure. At the point where the pages, components are out of the standard, the user experience is difficult and the ease of using the application is lost. On the other hand, pages and components are not standard, but they are problematic in maintenance in the future. In practice, when a radical design change is made, it is necessary to change all the pages one by one.

The code generation tools will speed up code development and make possible future maintenance easier and faster, since the pages and components used are standardized. Initial generation of pages with one-time code generation will be fast, but after development, code generation will be disabled. With the two-way code generation, the corresponding tool will continue to be used for further development of the pages.

Finally, this tool allows the design and creation of static pages to be done by analysts, possibly future text changes, etc. it could also be done by analysts. With such a tool, all these changes will need to be made by the developers, which will extend both the development cost and the duration

# 5. CONCLUSION AND WHAT IS NEXT

## 5.1 Conclusion

Appbuilder methodology, is a code generation and designer tool that is presented in this thesis. This tool uses NodeJS, ReactJS and some side technologies together to generate sources from a graphical user interface. The designing application is running inside the web browser while user designing and generation sources in it. So, users can see whatever they design directly.

This appbuilder tool aims for non-developer users at first. But developers can also use and do hand coding it together with appbuilder. The purpose of this tool is to speed up development life cycle. Analysis and design step of a software can be done quickly in appbuilder. The output of the work directly converts to source codes. Developers can take the source codes from there and do the rest. They can also use the appbuilder for hand coding or they can use their own development tool. Since appbuilder is embedded with version tool, either way it is supported.

The appbuilder tool was examined under dimensions “coding quality”, “speed”,” error / bug rate”. A people of 20 selected for measurements. They all know good knowledge for reactjs. Since the generated sources in reactjs they all need to know hand coding on it.

After doing different iterations with different number of groups on variety of designs, it was measured that the time efficiency on appbuilder is at least %50 better than hand coding.

The average bug rate that is done by developers decreased by two or three times with appbuilder. The codes are computer generated, the bug rates become user input mistakes, since users gives less inputs for code generation, bug counts decreases.

The code generation standards are given to system as templates, so each code generation makes the same output. So, it can be said that the quality and the standard of the code is always the same. But in hand coding, each user does their own style, so the output differs, and the quality is not the same.

Appbuilder currently do design kind (User Interface) code generations, so this means backend or business execution algorithms needs to be done by developers. If those algorithms have standard structures they can also be generated but mostly they are not.

Appbuilder mostly suitable for huge or big web applications that always in progress or new functionalities added frequently. Since the productivity in software development is increased, the production speed and rate increase with it.

### 5.2 Future Word

Appbuilder uses very recent technologies like ReactJS or NodeJs. That means, things will change and grow super-fast. For example, each library releases at least 2 version per month. So appbuilder needs to be updated all the time. If users are using those technologies, they have one static task is to keep up to date.

It will be a great service that serve appbuilder over cloud platform and keep it open source. More users that uses appbuilder, more ideas and requirements will come.

For functional perspective, appbuilder generates UI currently. Javascript based code generations and designer parts can be added. For example, managing global state like Redux on designing applications can be a great plus.

Appbuilder currently generates UI only, but it will be great if also backend services can be generated too. For a Node JS based backend, it will also be possible to create backend services on the fly.

As described before, appbuilder has many useful usages. But for my perspective, it will be a best usage by non-developer kind people. Because, idea is to generate codes, it will be meaningless to generate codes with developers. Developers can do that already, to gain speed, if other people uses like business line or analysis people, development phase speeds up because, developers get pre-coded sources and start the job with a boost.

Since now, appbuilder get the sources, generate new ones, and commit back them to the source control back. But it will be great if appbuilder has publisher mechanisms embedded. So that generated codes can be published and tested on its original environment. Simple publisher structure can be added that works with the cloud system automatically, that publishes the application to its original serves.

## REFERENCES


[http-1] GitHub, Inc. https://github.com/facebook/react (Date of access: 04.09.2015)

[http-2] GitHub, Inc. https://github.com/ipselon/structor (Date of access: 08.10.2016)

[http-3] GitHub, Inc. https://github.com/webpack (Date of access: 18.09.2016)

[http-4] GitHub. https://webpack.github.io (Date of access: 01.02.2015)

[http-5] Benchmarking Node.js - basic performance tests against Apache + PHP. http://zgadzaj.com/benchmarking-nodejs-basic-performance-tests-against-apache-php (Date of access: 14.01.2015)

[http-6] Can I use ES6 ? https://caniuse.com/#search=ES6 (Date of access: 24.05.2016)

[http-7] Node.js Foundation. https://nodejs.org/en/ (Date of access: 01.06.2015)

[http-8] Redux. https://redux.js.org/ (Date of access: 03.02.2016)

[http-9] W3C. http://www.w3.org (Date of access: 04.11.2017)

[http-10] NPM. https://www.npmjs.com/ (Date of access: 14.08.2016)

[http-11] AST Explorer. http://astexplorer.net (Date of access: 23.10.2016)

[1] Lara, J. A., Lizcano, D., MartíNez, M. A. & Pazos, J. (2013). Developing front-end Web 2.0 technologies to access services, content and things in the future Internet, Future Generation Computer Systems, v.29 n.5, p.1184-1195, July.

[2] Comai, S., & Mazza, D. (2012). A model-driven methodology to the content layout problem in web applications. *ACM Transactions on the Web (TWEB), 6* (3), 10.

[3] Basta, M. & Willer, M. (2017). *Adaptive Determination of Dynamically-Composited Web Page Elements in A Web Application,* US 20170132185 A1. California. BOX, INC.

[4] Leece, M. (2013). Declarative show and hide animations in html5, US 20130346851 A1. Washington. Microsoft Corporation.

[http-12] GitHub. https://git-scm.com/book/en/v1/Git-Branching-What-a-Branch-Is (Date of access: 17.05.2016)

[http-13] W3C. https://www.w3.org/ (Date of access: 06.02.2017)

[http-14] GitHub. https://github.com/jquery/jquery (Date of access: 15.10.2016)

[http-15] MongoDB, Inc. www.mongodb.com (Date of access: 08.12.2016)

[http-16] Node.js. https://github.com/expressjs/express (Date of access: 24.10.2017)

[http-17] NPM. https://www.npmjs.com/ (Date of access: 04.12.2017)

[http-18] The PHP Group. http://php.net/ (Date of access: 14.11.2017)

## GLOSSARY

**Ace Editor:** Ace editor is an open source project that creates a web based source file formatter and editor.

**Ast:** Ast is a meta description format of source code files. Compilers used those ast format to compile and identify the source files.

**Babel JS:** BabelJs is an open source platform that converts an compiles ES6, ES7 to ES5 javascript. It can be found on https://babeljs.io

**Branch:** A branch in git is a very widely used term that has a big definition [http-12].

**CSS:** Cascading Style Sheets is the general term of styling web applications. In can be independent ".css" files or can be written directly inside html files.

**DevOPS:** DevOPS is the shortest term for developer operations, which means the complete cycle for software development, like testing automating, building deploying and etc. Recent years, importance of devOps is increases very fast.

**DOM (Document Object Model):** The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data [http-13].

**ECMAScript:** ECMAScript is the subset / version of javascript that started used in terms recently. Currently there is ECMAScript version 5, 6 ,7 are defined. It is very possible to released some more soon.

**Es-Formatter:** This is a npm package to format (beautify) source files visual structure. Not the code itself, only the visual appearances.

**Esprima-FB:** This library is the extension for ast decoders to understand jsx source files.

**Express Package:** Express is an npm package and an open source project, that developers can create web servers easily with it on node js [http-16].

**Github.com:** GitHub is an open source project platform and it also gives free code storage repository to developers.

**Git Pull-Push Operations:** Sending codes to central code repository is called push, taking latest codes from central code repository means pull operation in Git.

**Git / TFS Code Share Tools:** Code version tools are used for team collaboration when a group of people codes a software all together. A centralized code repository is located on the server and each client (developer) pulls the code from there and pushes their codes back. TFS and Git are the mostly used ones.

**Hot module replacement:** HMR is a feature in webpack, to inject updated modules into the active at runtime. This means, users can update source codes at run time [http-14].

**JSON:** JSON is a format for storing and exchanging data. It is mostly used in javascript. It can be said it is invented for javascript needs, but used very widely.

**JSX:** JSX is the language syntax of react js library that used to define custom web components.

**LDAP:** Lightweight Directory Access Protocol, is an Internet protocol that email, and other programs use to look up information from a server. It mostly used in companies for authentication mechanisms.

**Mocking:** Mocking term in computer science means, simulating or replicating some code execution. Mostly in backend services, mocking term is used. In unit testing environments, mocking has huge importance.

**Mongodb:** MongoDB is a document database with the scalability and flexibility that users want with the querying and indexing that they need. It has both open source and licenced parts [http-15].

**NodeJs:** Nodejs is a built in javascript runtime that runs javascript codes in console. It is also an open source project.

**Npm:** Npm is the package manager of NodeJS platform. Developers can share codes or libraries over npm. It is free to use platform and can be used in enterprises locally with licensing [http-17].

**oAuth:** OAuth (Open Authorization) is an open standard for token-based authentication and authorization on the Internet.

**PHP:** PHP is a widely used, open source project, that is especially used for web development purposes. It is a server-side web programming language [http-18].

**Polly Filling:** Polyfilling term is used when a web browser did not support a new javascript functionality, users need a simulation code for this on that browser. That simulation codes are generally called polyfills.

**Pure Javascript Function:** The pure function is always returns the same result for same arguments passed in. It does not depend on any state, or data, change during a program's execution. It must only depend on its input arguments. Pure functions do not produce any observable side effects such as network requests, input and output devices, or data mutation.

**Pure Reducers:** Reducers written as pure functions called pure reducers.

**React Component:** A re-usable code block called component for web applications built with ReactJS library.

**React Hot Loader:** React JS libraries webpack hot module replacement plugin extension. Used to hot load reactjs applications.

**React JS:** ReactJs is a javascript library for building User Interfaces. It is owned by Facebook and it is also open source project.

**Redux JS:** Redux is a predictable state container for JavaScript apps. It is an open source project.

**Responsive:** Responsive is a common term used for applications supporting multiple screen sizes at the same time. It is used mostly for web and mobile applications.

**Single Page Application:** The term "single-page application" (or SPA) is usually used to describe applications that were built for the web. These applications are accessed via a web browser like other websites, but offer more dynamic interactions resembling native mobile and desktop apps. The difference is all the application is on just single html file.

**Socket:** Socket is a common term that is used to name the connection between 2 applications.

**Socket.io:** Socket.io is an open source project for giving applications a web socket infrastructure.

**Webpack:** Webpack is a static module bundler for modern JavaScript applications. It's an open source project.

**W3 Consortium:** W3 org is the consortium that defines the standards of web. The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth of the Web.

**RESUME**

| | |
|---|---|
| Name | Emrah Öz |
| Languages | Turkish, English |
| Place and date of birth | Ordu/Ünye TURKEY - 1983 |
| Email | emr550maranello@gmail.com |

Education and Working Experiences:

- 2006 Anadolu University, Faculty of Engineering, Computer Engineering/Student
- 2017 Anadolu University, Graduate School of Sciences, Computer Engineering / Student
- 2005-2006 Genç Anadolu Reklam Tanıtım , Eskişehir , Company Owner
- 2007-2008 Eroğlu Yazılım , Eskişehir / Software Engineer
- 2010-2012 Uyumsoft , İstanbul / Software Engineer
- 2012-2017 Akbank T.A.Ş. / Senior Software Architect