Rafe Ibrahim

Development of Hybrid Full Stack Application for Vacation Rentals

Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Bachelor's Thesis
1 November 2022
Abstract

This application has been developed for a start-up xxx. In recent years Pakistan has witnessed exponential growth in tourism sector. Starting from second highest mountain K2 in Himalayas to the long beaches of Arabian sea, Pakistan offers a variety of different landscapes. In addition, Pakistan also has remains of one of the oldest civilizations in the world and has religious sites sacred to Buddhists, Hindus and Muslims. Currently, there is no platform which provides one stop solution to all needs of the tourists visiting Pakistan. Services are scattered across many platforms and in some areas of tourism industry, there is no digitisation at all. Over the past decade, there is huge investment by the government in the networking and information technology infrastructure throughout Pakistan. This makes it an ideal time to digitise tourism sector in Pakistan.

This start-up aims to tap into this huge potential and provide a multidimensional platform which will cater all needs of the local and international tourists visiting Pakistan. Prominent services to be provided by start-up include ability to book and post rental places, book air, bus and train tickets and travel guides for travel spots throughout Pakistan. This start-up is in planning phases. This application is one of the many applications developed for this platform. The final application will provide features such as the ability to book rental places, communication through application’s in-built messenger with the rental place owners, the ability to pay using local payment methods such as mobile wallet and even cash. Its first prototype has been developed for this final year project. The application includes backend and front-end hybrid mobile client (for both iOS and Android) development. Back end comprises of a server, database and cloud storage. The server is written with Node.js and Express. MongoDB will serve as database for this application. For file storage Amazon S3 (object-based storage) provided by Amazon web services has been used. In the first prototype, users can search and post hotel rooms, apartments, villas, camping sites etc. Users can also log in and save a list of favourite rental places in their profile. Next steps involve testing this prototype with real users and finalising requirements for second prototype.

Keywords: React, React Native, Node.js, Express, MongoDB, Amazon S3, Hybrid Application Development
List of Abbreviations

**DOM:** Document object model. It represents a webpage in the form of a JavaScript object which can then be manipulated programmatically.

**API:** Application programming interface. It provides a way to programmatically access different available features.

**Attr:** Attribute. Attribute refers to different properties of the JavaScript object.

**JSX:** JavaScript XML. This syntax is used to create UI for the React components.

**Props:** Properties. These are used for passing data from parent component to child component in React.

**NoSQL:** Not only structured query language. It includes different databases which store in nontabular format.

**JSON:** JavaScript Object Notation. Used as a standard for transferring data between clients and servers

**CRUD:** Create, Read, Update and Delete. Used for describing operations on databases.

**Npm:** Node Packet Manager. Node package manager takes care of dependencies and list them out in package.json file.
1 Introduction

Pakistan is a medium sized country located in South Asia with a population of more than 250 million people [1]. In the last decade tourism industry in Pakistan has started to grow. It is fuelled both by local and international tourists. Pakistan has the right ingredients to become one of the top tourist destinations in the world. From geographical perspective, it offers one of the unique mixes of different landscapes in the world. In the north, the country has a collection of highest mountains in the world followed by plains and deserts in the centre to a coastal line of 800 kms with Arabian Sea in the South. It offers four different climatic zones which includes highland climate, lowland climate, Desert/Arid climate and coastal / maritime climate. Pakistan is culturally very diverse and there are more than dozen primary languages spoken. Pakistan also has remains of very old civilizations such as Mohenjo-Daro and Harappa dating back to 2500BCE. Moreover, Pakistan has a number of religious sites sacred to Buddhists, Hindus, Muslims and Sikhs. During the last couple of decades due to war in neighbouring Afghanistan, tourism industry in Pakistan could not catch up with the rest of the world. Therefore, Pakistan offers a variety of unexplored landscapes for tourists to discover and enjoy. [2]

There have been large investments in information and communication technology infrastructure in recent years. Cellular subscribers in Pakistan have reached a massive number of 189 million i.e., 90% of total population. Broadband internet subscribers have increased to 110 million users and there are 99 million smartphone users in the country [3]. To sum it up, there is currently a digital revolution going on in the country. Unfortunately, services for tourists in Pakistan are scattered across many different platforms and in some areas, there is no digital support available. The start-up for which this project is carried out aims to fill this gap by developing a platform which will address all needs of tourists.
The start-up xxx aims to provide a multidimensional platform which will encompass all needs of local and foreign tourists visiting tourist destinations in Pakistan. The clients of the platform include tourists, hotel owners and private people renting their properties. The platform will provide clients with the ability to search, post and book accommodations. It also provides features to search and book travel tickets for air, railway and bus travel. It will also provide holiday packages and travel guides for famous travel destinations throughout Pakistan. Lastly, it will provide customer support based locally to guide its clients in real time. The platform will revolutionise tourism sector by digitisation. It will help boost tourism industry in Pakistan which will in turn boost standard of living of local population.

The target of this project is to provide the start up with the complete backend and a hybrid front end. Backend will consist of backend server, document-based database and object storage for storing files e.g., images for the application. Backend server will be written with Node.js which is a JavaScript runtime, and Express, a framework for Node.js. MongoDB will be used as a database and will be hosted on MongoDB Atlas which is a multi-cloud database service. Amazon S3 will be used as object storage to store images and other assets of the applications. Hybrid front end refers to mobile client written using React Native which will provide native looking apps for both Android and iOS platform.

Section 2 of thesis will provide a review of main libraries and technologies used for development of this project. Third section will document implementation of both the front end and the back end. Lastly, the conclusion of the study discusses how this application has been able to fulfil requirements of the start-up. It will list obstacles faced during development of project. Future steps will be mentioned and a set of features required for the next version of the application.
2 Background

2.1 Tourism in Pakistan

Pakistan is a country located in south Asia and it is the fifth most populous country in the world while it is the second largest Muslim country. Pakistan shares borders with India, Iran, China and Afghanistan, and a coastline with Arabian sea. This geographical location not only makes Pakistan a strategically important place but it also grants the same a bundle of opportunities for tourism. This geographical location makes the country experience a variety of different climate regions. There are Northern highlands with Karakoram, Hindukush and Pamir Mountain ranges, there are fertile Indus plains, there is Balochistan plateau and Deserts in the east. All these regions have different climates, flora and fauna, and lifestyles. All these geographic and climatic conditions make Pakistan a place worth visiting. [2.]

Pakistan is a country with various ancient cultures. There is almost 9000 years old Neolithic site of Mehrgarh in Balochistan (surrounded by Quetta, Qalat and Sibbi). There are ruins of Indus valley civilization which are dated back to Bronze Age. There are remnants of Gandhara civilization which is sacred for Buddhists and Greeks. Pakistan was the realm of many dynasties and empires. The most notable of them are Achaemenid (Alexander the great), the Maurya, the Gupta, the Umayyad Caliphate, the Hindu Shahis, the Ghaznavids, the Delhi Sultanate, the Mughal dynasty, Durranis, the Sikh empire and most recently the British Indian empire. All these empires and dynasties ruled modern Pakistan in ancient times. They left their architectural wonders behind them. These sites are now sacred to Hindus, Buddhists and others equally. There are probably very few countries in the world which offer such rich cultural and historical background. Nowadays these sites are tourist attractions. A number of social media influencers are visiting these sites and letting the world know about the vibrant historical and cultural aspects of Pakistan. [3.]
Unfortunately, Pakistan was hit badly by terrorism. There was a continuous war between the Army, which is world's sixth largest standing army, and terrorists. There were scanty roads transportation system too. But now the things have changed for the better. Now the terrorism is eradicated from the country. Almost a decade ago the government of that time-built roads and bridges hence connecting the Northern Pakistan to the rest of the country. All these efforts combined made Pakistan a dream destination for tourism. Now national and international tourists are returning. This is the Age of social media, when tourists post stories about the beauty and peace of Pakistan, other people get influenced and it is like a chain reaction. Most of the international tourists are social media influencers too, their words carry weight that is why Pakistan is becoming increasingly popular among international travellers. [4.]

Perfection is yet to be achieved in the field of tourism in Pakistan. There are many things which have improved, i.e., the roads, transport, the overall security system of the country but there are some aspects which are need to be addressed. For example, there must be an easy way to get visa for Pakistan. Secondly the areas which are an attraction for tourists must be made environmentally protected. The eco systems of these sites should be protected by giving tourists guidance about the flora and fauna of the areas. [5.]

Pakistan can be rightfully called a land of opportunities. A country with a booming population of almost 240 million, with the world's sixth largest standing army, with almost 10 million expats around the world, with the world's highest rising mountain ranges, with the most fertile plains, with the most hospitable and peace-loving people, with one of the most scrumptious cuisines in the world which is a blend of Asian, Irani and Afghan cuisines, Pakistan has way more to offer which cannot be covered in a few words. [6.]
2.2 Overview of client

One of the biggest challenges faced by tourism in Pakistan is lack of information. Lack of digital information means that potential tourists cannot get basic information about tourism sites, arrange for their accommodations in advance, booking means of transport, visa requirements and restrictions for foreigners and so on. There is no single reliable platform in Pakistan which can address this problem. Information is scattered across many government departments working under tourism, internal affairs and foreign affairs ministry. In addition to that, private sector providing different services such as transport, accommodations and tourist packages is even more disintegrated. This situation can be quite intimidating to potential tourists. [5.]

Information technology is revolutionising many industries around the world. The start-up xxx wants to revamp tourism industry of Pakistan so that it reaches its true potential. It aims to bring all different stakeholders in tourism industry to one portal. The portal will be the first of its kind in Pakistan. This portal will be tailor-made for local needs of tourism entrepreneurs. This portal will act as a bridge between tourists and entrepreneurs. Users of this platform can rate different services by an inbuilt feedback system. As there will be multiple entrepreneurs providing services on this portal, this feedback system will lead to improvement of the services. Another important quality of this platform will be to provide local based custom support. Using local based customer support will result in quick resolutions of problems and conflicts and will further augment confidence of users in this portal. The application developed for this final year project will provide back end and front-end hybrid mobile client for accommodation booking services of this portal. The back-end server is developed modularly so that it can be extended to provide back-end support for other services provided by this portal.
3 Key Technologies

3.1 Node.js and Express

Node.js is an open-source cross-platform runtime environment for executing JavaScript code outside of a browser. Mostly, Node is used to build back-end services like APIs. These services are responsible for powering up our client applications such as web apps and mobile apps. Client applications need to fetch and send information from some back-end service to show users up to date data and perform certain operations on that data. [7]

Node is highly suitable for building highly scalable, data intensive and real time apps. There are many benefits of using Node as compared to other tools for developing back-end services such as Django and ASP.net. It is easy to get started and can be used for prototyping and agile development. It is also very fast and highly scalable. It is used already in production by large companies such as PayPal, Uber and Netflix. Node applications use JavaScript which is a default programming language for web development these days. If the front end of an application is written with JavaScript, the same team can use its JavaScript skills to write a back-end server for it. This turns into a more consistent code for front end and back end and also saves loads of time and financial resources. As node has been around for some time it has many open-source libraries available online. [7]

JavaScript used to run only inside browsers. In 2009, Ryan Dahl came up with Node which provides a runtime environment for running JavaScript code outside the browser. He created Node by using Google v8 engine and encapsulating it in a C++ program. Google v8 engine is the fastest JavaScript engine. Node provides different ways to interact with the environment hosting it such as functions for reading file systems and creating HTTP servers and listening to the port and so on. [7]
Node applications are fast because they execute code using asynchronous architecture. All requests are handled by a single thread in Node. Whenever a request operation has been carried out such as a database query, it is added to the event queue of the Node. Node continuously watches for events reported to the events queue. Whenever there is a new event, it handles it quickly. This method of handling requests makes Node ideal for network intensive applications. On the other hand, Node should not be used for CPU intensive applications. The reason is that Node works using a single thread and while it is handling one CPU intensive request other requests are waiting for their turn which makes the server very slow. [8.]

Node uses a modular system. Some functions are available universally in Node and they can be accessed using a global variable. One example of a function available globally is console.log. Every file in a node application is treated like a module and all variables and functions in that file can only be accessed within it by default. This helps in avoiding name conflicts. In order to make any variable or function available to any other module we have to explicitly declare it using module.exports keyword. These exported items can then be used in other modules using a require statement. [8.]

Node provides a couple of built-in modules. A Path module provides functions to manipulate paths to fit our needs. The OS module provides utilities to interact with the operating system on which node is currently running. File system module has methods to work with files and directories. One of the most important modules provided by Node is an events module. This module is required to implement many core functionalities like listening to a port using HTTP server. Event emitter class of events module can be used to create custom events and their handlers can be registered with call back functions. Another important module provided by Node is the HTTP module. It is used to create a server and send requests to other entities like a database and a file storage. [8.]

A simple server can be created with a HTTP module of Node. But as the complexity of the server grows it becomes increasingly difficult to handle it with
this module. It has been mentioned that Node comes with a robust system of libraries which can be used for different use cases. Express is an open-source framework built for Node to handle backend server code elegantly. It helps to write code for handling different routes in different files while keeping it maintainable. [9.]

Install

```bash
$ npm i express
```

Repository

[github.com/expressjs/express](https://github.com/expressjs/express)

Homepage

[expressjs.com/](https://expressjs.com/)

<table>
<thead>
<tr>
<th>Weekly Downloads</th>
<th>24,964,156</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.18.1</td>
<td>MIT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unpacked Size</th>
<th>Total Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>214 kB</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Issues</th>
<th>Pull Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>47</td>
</tr>
</tbody>
</table>

Figure 1. Screenshot from express package page of npm official website [10].

From figure 1 above, it is clearly visible that it has been downloaded more than 25 million times in a week which makes it a very popular framework. Express provides very basic functionalities itself but software developers have published online many middleware packages to help with many problems. This also turns out to be a little bit challenging to choose the right middleware for the required functionality. Another problem with the minimalist framework such as Express is that there are many ways to structure an application which leads to incoherent code in different implementations of the server. [9.]
3.2 NoSQL Database and MongoDB

Every modern application needs to store some form of data. This data needs to be stored at one place so that all its clients can access the data and if a single client make changes to the data, it can be updated to all other clients. Database is an ideal solution for this kind of a problem. Databases store data in a structured way. There are many different kinds of databases. They can be largely divided into two categories SQL and NoSQL databases. Sometimes, they are also referred as relational and non-relational databases. SQL databases store data in relatable tables and enforce a strict structure like the type of data which can be stored in different columns of the table. NoSQL databases on the other way allow for non-structured and semi-structured database to be stored in it. This means in NoSQL databases there is no need for strict definition of data objects to be stored and their relationship with each other. [11.]

NoSQL databases are very common these days due to their availability to scale, performance optimization and high availability. One of the most popular NoSQL databases is MongoDB. It stores data in the form of collections of documents. Documents are stored in the form of BSON. BSON stands for Binary JSON. Documents actually look quite similar to JavaScript objects or JSON objects. Developers can interact with MongoDB in JSON format. MongoDB drivers usually take care of converting JSON to BSON and vice versa behind the scenes when querying the database. This database is very famous with JavaScript developers and is an integral part of MERN and MEAN stacks. MERN stack refers to the stack comprising of MongoDB, Express server, React and Node.js and MEAN stack consist of MongoDB, Express server, Angular and Node.js for developing full stack applications. The biggest advantage of MongoDB is that it is very flexible. In SQL databases, tables and columns need to be configure beforehand but in MongoDB your database structure can grow with the needs of application very easily. [12.]
Figure 2 shows document object returned by MongoDB when queried with its id. The data is returned in JSON format. However, it is stored in database itself using BSON.

3.3 React

The JavaScript library React is used to build reusable user interfaces. React utilises component-based approach. Component based approach is all about creating independent components once and reusing them repeatedly. MVC is widely used design pattern for mobile and web applications these days. MVC stands for Model, View and Controller. The most common way to incorporate React in an MVC architecture is in the view part. Model-View-Controller (MVC) is a widely used software design pattern which separates business logic and display. By abstracting the actual DOM, React offers better performance and it only updates the actual DOM when its content have been changed. Jordan Walke, a Facebook software engineer, developed React in 2013. Since then, it has competed with other UI libraries, such as JQuery and Angular. The popularity of React increased in 2015 when Netflix introduced React to build their UI. React popularity was further augmented when it gained popular support libraries such as React Router and Redux. These libraries eased state management and routing in React applications. Moving forward next major milestone in React history is the release of React Fiber in 2017. With Fiber, internal execution mechanism of React was completely revamped resulting in significant improvement in performance. With the release of Hooks in 2019 React code became more readable, reusable and organised. React core building blocks
include JSX, virtual DOM, components, props, state and context. They are explained in sections below. [14.]

3.3.1 Virtual DOM in React

It is important to understand the concept of virtual DOM to fully understand the power of React. DOM stands for data object model. It is composed of structure and content of a web document represented in the form of a data model. It can also be referred to as a programming interface. It provides a way for programs to change content and structure of a web page. It contains all properties, methods and events required to manipulate and create web pages. The DOM API can be accessed directly from within a script where script is a program implemented by a browser. Different data types used when interacting with DOM include Document, Element, Node, NodeList, Attr and NamedNodeMap. [15; 16.]

In modern web development to render a dynamic user interface, DOM must be updated frequently. Updating DOM is a very slow progress and reduces the performance of an application. Rather than directly modifying DOM, react uses its in-memory version of DOM called virtual DOM. It first calculates the changes made by an application in Virtual DOM and then smartly figures out a way to update a change in a web page so that real DOM is updated minimum possible times. The following diagram fully explains this process.
Figure 3. Figure depicting update of actual DOM by Virtual DOM [15].

Figure 3 demonstrates that as soon as there are changes to be made to the webpage, React smartly implements those changes in Virtual DOM behind the scenes without changing actual DOM. The actual DOM is only changed once. If there is no Virtual DOM every time there is a change of content it results in re-rendering of the complete DOM. Changing virtual DOM as compared to changing the actual DOM is very fast. Each time there is a change in virtual DOM, React smartly compares it with actual DOM with a process called diffing. After this comparison React only updates required parts in the actual DOM not the whole DOM. This makes applications or webpages more responsive. Using virtual DOM results in significant performance optimization. [15.]

3.3.2 Use of JSX in React

JSX is an essential part of how React works. It stands for JavaScript XML. It is a syntax extension for JavaScript. It is recommended way to describe how UI elements should appear. In the end, JSX gets converted to React elements. The core React philosophy is that rendering logic is inextricably tied to other user
interface logic. Following example is being created by author of this document following guidance in official React page to describe use of JSX in React.

```javascript
const textToDisplay = "Name to display"
const textElement = <h1>Hello, {textToDisplay}</h1>

ReactDOM.render(
    textElement,
    document.getElementById('root')
);

Listing 1. Use of JSX in React Component [15.]
```

In listing 1, the name constant is declared with an example name. Subsequently, an element constant is declared which contains a JSX expression. The JSX contains a name constant enclosed in curly brackets. In JSX, any expression written inside curly brackets is executed like a standard JavaScript. JSX is then converted to React.createElement() function calls by Bable. Bable is an open-source JavaScript trans compiler which converts JavaScript code into a backward-compatible version of JavaScript which can be run by older JavaScript engines. Conversion of JSX to createElement function calls is demonstrated in an example below. [15.]
Patch 1
const component = (  
    <h1 className= "class1">  
        Example text!  
    </h1>  
);  

Patch 2
const component = React.createElement(  
    'h1',  
    {className: 'class1' },  
    'Example text!'  
);  

Patch 3
Const component = {  
    Type: 'h1',  
    Props: {  
        className: 'class1',  
        children: 'Example text!'  
    }  
}  

Listing 2 An example of converting JSX to React.createElement() function call [15].

In Listing 2, there are three code snippets. The first two code blocks are identical. In first code patch constant component is declared using JSX. Second code block shows the form taken by component after Bable compilation. Last code patch shows an object which describes what needs to be displayed on the screen. [15.]

3.3.3 State Management in React

A state is just a plain JavaScript object that represents information about the current status of a component. A state is managed as a normal variable within a component. A state changes based on number or triggers such as input of user and triggering of an event. The state of React components controls how they are rendered. The state is declared with initial data. Consequently, when state changes, React gets notified and immediately re-renders the DOM. Only the
component with changed state is re-rendered not the entire DOM. In early days of React state could only be used in class-based components and not in functional components. Functional components were known as state less components in those days. Since introduction of hooks, state can now be declared and used in both class bases and functional components. Use of state in React is being demonstrated by an example below.

Class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            count: 0
        };
    };

    addCount = () => {
        this.setState({
            count: this.state.count + 1
        });
    };

    subtractCount = () => {
        this.setState({
            count: this.state.count - 1
        });
    };
}

Listing 3. Example code declaring react component state [20].

In Listing 3, React component Counter is declared. It uses class-based syntax. There is an initial value of 0 set for the counter in the state. Two methods are then declared to increment and decrement this state. These methods can be invoked in response to any event as required by the application. State management is an integral part of React implementation. Each component in React maintain its own state required for its user interface. [19; 20.]
3.3.4 Use of Hooks in React

Before react version 16.8 a developer is forced to write a class-based component to get access to many core features of React like maintaining a state. In practice, this leads to a complex structure of encapsulated components. These days almost all features of React framework can be implemented by functions known as hooks. The name of hooks always starts with the prefix “use”. React has ten built-in hooks. In addition, a hook can be written by a developer for some custom functionality. [21.]

`useState` is the most commonly used hook. It is used to maintain the data for a component. A class-based example of maintaining state is shown under the heading 3.2.4. Whenever the data stored in the state hook is changed it causes the component to re-render. It takes in as an argument the initial data and returns an array. This array can be de-structured to get the current state and a function to change the state.

```javascript
import React, { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Counter = {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
export default Counter;
```

Figure 3. Use of `useState` hook in functional component counter [21].

The figure 3 above shows an example of a functional component using a `useState` hook. A state is declared in this component with an initial value of zero.
The variables count and setCount are destructured from the returned array to access current count and function to change count respectively. The variable count is used to display the value of current state in p tags. Next, a button is implemented which uses the setCount function to increment current count by one every time it is pressed. The setState function should not directly manipulate the state; instead, it should be given as an argument to the new value for the state. [21; 22.]

Another React hook used for maintaining complex state is called useReducer hook. In addition to the initial state it takes as an argument a function known as reducer. The reducer function is defined by the developer which takes in current state and an action. Based on an action it manipulates the state in the required way. UseReducer hook returns an array which can be destructured to get the current state and a function normally called dispatch to update the state. Whenever the current state needs to be changed the dispatch function is called with proper action. This leads to an updated state based on the implementation of the reducer function. [21; 23.]

useEffect is another React hook used to run side effects every time a React component is re-rendered. These effects were executed in a class-based component with the different life cycle methods such as componentDidMount and componentDidUpdate. It takes in as an argument a function needed to run as a side effect and a dependency array. In case the dependency array is empty the effect function will run only once when the component is first mounted. If the second argument i.e., dependency array is not provided at all the function provided to useEffect hook is run each time the state of component is updated. If the dependency array contains any variables, then the effect function runs only when one of those variables have been changed.
useContext is another react hook used to consume Context API by react into the project. When the application size grows significantly, it becomes increasingly difficult to share the data or state with the components deep in the component tree. React Context API solves this problem by storing state which needs to be shared among all components in a component created with createContext function. The common parent of all the components which need access to this state is wrapped with the Context provider component. Whenever any component in this tree needs to access this context, it uses useContext hook to access it. Use of hooks has completely revolutionized React development. Using hooks can significantly the required amount of code and makes the code more scalable and readable when compared to class-based implementation. [21; 25.]
3.4 ReactNative

React Native is a framework for building native apps using JavaScript. It was created by Facebook in 2015 and over the last few years it has become very popular. It was launched keeping in view the success of the React for developing UI of the web applications. The team at Facebook aimed to translate React success into developing native UI with React Native. Android applications are written with Java or Kotlin and iOS applications are written with Swift. JavaScript is the programming language for developing applications with React. With React Native a complete native application can be developed without having dedicated developer teams for iOS and Android. React Native uses all the features of React behind the scenes to build native mobile application. [26.]

![React and ReactNative workflow](image)

Figure 5. Showing React and ReactNative work flow [18].

Figure 5 above shows how react native workflow relates to that of React. In Web development, ReactDOM library is used to translate react components for the web browser. In case of mobile client, React Native is used by React to convert React components into native UI elements. Image component changes to ImageView and UIImageView in Android and iOS respectively. The architecture of React Native is such that there is a native thread running on mobile along with the bridge. The bridge communicates with the JavaScript thread created by React Native in case of any events in the mobile device. [18.]
4 Project Implementation

4.1 Project Description

The aim of the project is to develop the first prototype of mobile client and back end for it. As per instructions of the client back-end server is written with Node.js and Express. Front end is required to be developed as a hybrid mobile application written with React Native. Ideally, application development should have started after the detailed survey conducted by start-up in the norther areas of Pakistan which is their opening market. This survey was planned to be conducted in April 2022. But unfortunately, it has been delayed two times so far. First it got delayed due to political turmoil in Pakistan in April 2022 which has led to large public protests and economic instability [27]. Second time it was postponed due to floods in large areas of Pakistan in the monsoon season of 2022 [28]. It is expected to be completed by the end of November 2022 according to current timeline. To save time, it was decided to go ahead with the project containing backend and functional front-end client implementing basic features. Functional front-end client means it should have basic UI for supporting fundamental features required by the application. Final UI will be implemented in the light of the results of survey expected to be available in about two months’ time. It will be included in the requirements of second prototype. Requirements for first prototype were agreed with the client for both front end hybrid mobile client and back end which includes backend server, database and object storage. A summarized version of requirements is described in the following paragraphs.

First requirement agreed with the client regarding back-end implementation is authentication handling. Authentication handling means that back end should be able to support sign-up, login and token verification for restricted routes. Restricted routes refer to those routes which can be accessed only after logging in to the application. Backend should have a data model for both users of application and rental properties posted by the users. It should support routes for
creating, reading, updating and deleting user. The user implementation is required for authentication handling. Backend server should also implement CRUD (create, read, update and delete) routes for rental properties. The data of the users and properties needs to be stored in MongoDB. MongoDB as mentioned earlier is a NoSQL document-based database in which documents are saved in JSON like format. The database is needed to be deployed online using MongoDB Atlas. MongoDB Atlas is a multi-cloud database service of MongoDB and is developed by same team which develop MongoDB itself. With MongoDB Atlas you can deploy your database on all famous cloud services providers such as AWS, Azure and Google Cloud [29]. The last requirement for the project is storing the files such as images in cloud storage Amazon S3. Amazon S3 is an object storage service offered by Amazon Web Services [30]. One of its most important benefits over storing files in database is cost-effectiveness.

First requirement of the front-end hybrid mobile client is to provide authentical workflow in the application. Authentical workflow refers to providing users with a functionality to authenticate themselves to the backend server [31]. It includes sign up and login screens and logout functionality. Second requirement is a screen displaying all rental places stored in the database. Third requirement is to display details about a chosen rental place. Screen for uploading new rental place to the application is also needed. This functionality is limited to authenticated users only. Lastly profile screen should be available for logged in user which lists rental places published by this user. Implementation details of back end and front-end hybrid mobile client are elaborated in subheadings 4.2 and 4.3 respectively.

4.2 Project Architecture

The architecture of the project is such that backend server is deployed on Heroku. The backend server is connected to Mongo database deployed on MongoDB Atlas. Back-end server also has permissions for reading and writing to cloud object storage hosted by Amazon Web Services. Mobile client communicates
with the backend server using JSON formatted data. Backend server handles all requests related to user authentication and manipulation of places data stored in the database and cloud storage. Figure 6 gives a visual depiction of how different entities in this project interact with each other.

Figure 6. Illustration of architecture of the project.
4.3 Backend Server

4.3.1 Application Structure

Back end is implemented with Node.js and Express. At first, project is created using npm command init. npm stands for Node Packet Manager. As its name describes npm provides a way to manage project dependencies by listing them out in a file called package.json [32]. Below is the screenshot of directory structure of a back-end server for this project.

![Directory structure of backend server](image)

Figure 7. Directory structure of backend server.

Figure 7 illustrates how implementation for server is divided into number of modules to make it easier to manage and scale the application. The models
directory contains files for defining the MongoDB schemas for both a user and a place. The controllers directory contains files for defining routes handled by the server. The utils folder contains different files for configuring environment variables such as URLs for database and implementation of custom middlewares for route handlers. Middlewares are functions which are executed before the route handler is executed [33]. They can execute various tasks such as running a validation check on request object before it is handled by the route handler.

```
{
    "name": "rental_server",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "start": "node index.js",
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "rafel",
    "license": "ISC",
    "dependencies": {
        "aws-sdk": "^2.1216.0",
        "bcrypt": "^5.0.1",
        "cors": "^2.8.5",
        "dotenv": "^16.0.2",
        "express": "^4.18.1",
        "express-async-errors": "^3.1.1",
        "jsonwebtoken": "^8.5.1",
        "mongoose": "^6.6.0",
        "multer": "^1.4.3-lts.1",
        "uuid": "^3.0.0"
    },
    "devDependencies": {
        "eslint": "^8.23.1",
        "eslint-config-airbnb": "^19.0.4",
        "eslint-plugin-import": "^2.26.0",
        "eslint-plugin-jsx-a11y": "^6.6.1",
        "eslint-plugin-react": "^7.31.8",
        "eslint-plugin-react-hooks": "^4.6.0"
    }
}
```

Figure 8. Contents of package.json file of the backend server.

Figure 8 shows contents of package.json file which is created by npm init command as discussed earlier. It lists out basic details of the application, scrips
to start it and more importantly it lists out dependencies required for running this server. It also includes a separate object containing dependencies required only during development process. The contents of Procfile are shown in the figure below.

![Procfile](image)

Figure 9. Contents of Procfile file of the backend server.

Procfile shown in figure 9 is required for the deployment of the backend server to the heroku. Heroku is a cloud computing platform providing utilities like platform as a service [34]. It provides a cloud environment for running Node applications. Procfile contains the script for telling Heroku how to run this server code. The server starts with the contents of index.js file. Index.js is listed in directory structure in figure 7. The figure below shows the contents of the index.js file.

![index.js](image)

Figure 10 The index.js file on the back-end server.

Figure 10 is displaying contents of index.js, an http server is created with the app variable imported from app.js file and Port is configured for the server to listen to. Implementation of routes and connection to mongo database is done in the file
from which the variable app is imported. Figure below shows the implementation of app.js file.

```javascript
const app = express();
const cors = require('cors');
const mongoose = require('mongoose');
require('express-async-errors');
const placesRouter = require('./controllers/places');
const usersRouter = require('./controllers/users');
const loginRouter = require('./controllers/login');
const middleware = require('./utils/middleware');
const logger = require('./utils/logger');

logger.info('connecting to', config.MONGODB_URI);

mongoose
  .connect(config.MONGODB_URI)
  .then(() => {...
  })
  .catch((error) => {...
  });

app.use(cors());
//app.use(express.static('build'));
app.use(express.json());
app.use(middleware.requestLogger);

app.use('/api/login', loginRouter);
app.use('/api/places', placesRouter);
app.use('/api/users', usersRouter);
app.use(middleware.unknownEndpoint);
app.use(middleware.errorHandler);

module.exports = app;
```

Figure 11. The app.js file of the back-end server.

Figure 11 shows that in app.js file variable app exported to index.js file is configured with Express. Express according to its documentation is a minimalist Node web framework. It provides out of the functions for handling HTTP requests such has GET, POST, PUT and DELETE and handlers for different URL paths can be configured easily. After setting up express an attempt is made to connect to the MongoDB using library named mongoose. Mongoose is used for creating
object models for storing them to the database [35]. In the following lines express
is configured to use multiple routers and middleware required to handle requests
made to the server.

4.3.2 Data Models

The user.js and place.js file in model’s directory define mongoose schemas for
interaction with the mongo DB database. The figure below shows the user
schema.

```javascript
models > user.js ...
1 const mongoose = require('mongoose');
2 const userSchema = new mongoose.Schema({
3   email: String,
4   name: String,
5   phone: String,
6   passwordHash: String,
7   places: [
8     {
9       type: mongoose.Schema.Types.ObjectId,
10      ref: 'Place',
11    },
12  ],
13});
14
userSchema.set('toJSON', {
15     transform: (document, returnedObject) => {
16       returnedObject.id = returnedObject._id.toString();
17       delete returnedObject._id;
18       delete returnedObject.__v;
19       // the password hash should not be revealed
20       delete returnedObject.passwordHash;
21     },
22   });
23
const User = mongoose.model('User', userSchema);
24
module.exports = User;
```

Figure 12. The user.js file in models directory of backend server.

Figure 12 shows the implementation of the user schema. Email, name, phone,
and password hash are stored as a string in the database. For security reasons,
instead of password strings their hashes are stored in the database. Formation
of password hash will be discussed under the heading 4.3.3 Controllers when the route handling creation of new user is explained. The most interesting attribute of userSchema is places. It contains an array of the ObjectIds of the places advertised by this particular user. After schema declaration the next code block is used for transforming the object returned from the database to suit the needs of the application. In this case the attribute _id is converted to id for convenience in handling. The attributes _id, __v and passwordHash are deleted from the returned object. The application has no use for the attributes _id and __v. The attribute passwordHash has been deleted for security so that the server does not send back password hash when the new user is created or logged in.

![Figure 13. The place.js file in models directory of the backend server.](image)

Figure 13 shows the implementation of the place schema. The attributes title, street address, postal code, city, rent and description is saved as strings in the database. The date attribute is saved using Date object of JavaScript. The
location attribute is saved as an object with the longitude and latitude attributes. Images attribute is saved as an array of objects containing an image URL as an only attribute. Lastly, user attribute is saved as an object which refers to the owner of this rental place advertisement with its ObjectId. These ids can be later used to fetch information about the users who created this rental place.

4.3.3 Route Handlers

Controllers are used for implementation of the routes handled by the backend server. They provide functions to perform CRUD operations on the database, storing images and other assets to the Amazon S3 object storage and sending JSON data back to the front-end hybrid client. The controllers directory contains three files. Login.js file implements route handlers related to login functionality. Users.js provides implementation for handling requests like creating a new user, getting all users and getting user by id. Places.js file provides request handlers for getting all rental places, getting a rental place with id, creating a new rental place and lastly deleting a rental place. Routes handling functionality for logging in and adding a new rental place is discussed next. The figure below shows login handler implemented in the back-end server.
Login functionality is handled by the code shown in Fig 14. First of all, backend server checks if the user exists with this email address in the database. Then the password received in request body is compared with the password hash stored in the database. Password hashes are created by bcrypt npm library everytime a new user is created through sign up process. Bcrypt is a password encryption system developed by Niels Provos and David Mazières in 1999 [36]. Comparing of passwords is also done with the helper function compare of bcrypt library. If the password is found to be correct a new token is created by jwt npm library. JSON web tokens provide a safe way to send information between client and server and vice versa. It is created with JavaScript object containing attributes needed to be stored in the token and a secret string. Moreover, in order to improve security a time duration can be configured after which token is expired.
The idea is that every time a new user is created or an existing user signs in, the backend server responds with an encrypted JWT token. The mobile client can then use this token to access restricted routes meant for authenticated users only. In this application, the token is sent in the authorization header with the word `bearer` and an empty space attached to it in the beginning.

The second most important route is for creating a new rental place. The client sends a POST request with endpoint `/places` to upload data for a new place. As this is a restricted route, the user needs to send a token to access it. The token sent is then verified and decoded to get details about the user who sent the request. Multer is used to handle multipart form data containing images and other data about rental places [37]. Multer provides a function which acts as a middleware and gives back files sent by the front-end client as an attribute of the request object. To fulfill the needs of our back-end server, Multer was required to be configured to store image files only in memory, accept only JPEG or JPG files, and limit the number of uploadable files to 5. It is achieved by the code shown in Figure 15 below.
After getting both the file objects and other data in the body of the request the next step is to upload the images to the Amazon S3 object storage. An amazon S3 bucket has to be created using AWS console and the user which in this case is a backend server is being given a read/write permission to it. AWS access key id, key secret and bucket name is written in the .env file for the local development environment and for the production environment these variables are uploaded to the Heroku console as configuration variables. To upload files to S3 a parameter object is created for each file in req.files array. Figure 16 below shows the parameter object created for each image of the rental place.

Figure 15. Code snippets showing configuration of multer for file uploads.
Figure 16. Code snippet showing parameter object for AWS S3 file upload.

```
const params = request.files.map((file) => {  
  // bucket that we made earlier  
  Bucket: process.env.AWS_BUCKET_NAME,  
  // Name of the image  
  Key: `${uuid()}-${file.originalname}`,  
  // Body which will contain the image in buffer format  
  Body: file.buffer,  
  // defining the permissions to get the public link  
  ACL: 'public-read-write',  
  // Necessary to define the image content-type to view  
  // the photo in the browser with the link  
  ContentType: 'image/jpg',
});
```

The file is uploaded to AWS S3 with the name specified in the key attribute of the parameter object. If we only use original name of the file with which it is stored in the client’s machine there is a chance that we run into the name conflict and we upload file to AWS S3 with the same name again. In this case it will just replace the old object stored with the same filename with the new one. There is no mechanism built-in to AWS S3 to configure this behaviour. This problem is solved with the use of npm package named uuid [38]. This package is used for generating unique ids for the data stored in the database. Behind the scenes this package uses the crypto module provided by Node to generate UUID which stands for Universally Unique Identifier. In this case we use uuid to create unique identifiers and then append them to original file names. The next step is to upload the files to AWS S3. The figure below shows the code snippet for uploading files.

```
const results = await Promise.all(  
  params.map((param) => s3.upload(param).promise())
);
```

Figure 17. Code snippet for uploading files

Uploading files is handled with the help of upload function provided amazon software development kit. The upload function returns a promise i.e., it is an asynchronous function. Promise.all function provided by JavaScript is used to
wait for all uploads to finish before we move on with the rest of code. The result object returned by the Amazon S3 contains the URL for that specific image. A place object is created using schema discussed in 4.3.2. After saving the rental place to database its id is stored in the array of places of the user who created it.

4.4 Hybrid Mobile Client

4.4.1 Application Structure

Front end hybrid client is developed with React Native which is discussed in detail in key technologies. Project folder is initiated with the help of Expo. Expo provides a framework for building react native apps and gives access to native APIs for accessing location, motion sensors and accessing camera. Figure 18 below shows the directory structure for the hybrid mobile client.
The application starts with the contents of App.js which can be seen in Fig 18. In app.js a navigation container is created and is wrapped with context providers. A navigation component is defined which returns conditionally rendered auth Stack and authenticated stacks wrapped in navigation container. The contents of these stacks are discussed in detail in 4.4.2 Navigation Workflow. This component is then rendered by the App component encapsulated in context providers. Following figure 19 shows the initiation code of the App component.
Screens directory in figure 19 contains all the main views for this application. All screens are shown in figure 20 and 21 in their respective stacks. The store directory contains files that define Auth context and rental places context. Auth context basically provides information about the authentication state of the application. Rental places context stores data of the rental places fetched from the back-end server in its state and provides functions to manipulate this data. Components folder is further divided into three folders namely Auth, Places and UI. Auth contains react components for rendering authentication related screens such as login and sign-up. Places folder contains all the components related to screens displaying rental places and form for adding new rental place. The UI folder is for small UI components used across all screens such as buttons and overlay screen for error. The rest of the files in the figure x define configuration for expo and dependencies required for running the application.
4.4.2 Navigation Workflow

The application is built in such as way that it has two navigation stacks. One for unauthenticated users called auth stack and one for authenticated users called authenticated stacks. A react context is implemented to determine which stack is mounted on the application. Auth context maintains a state for authentication token returned by the server and user id of the logged in user. A function is also implemented in auth context which on being called store the token and user id to the context. In addition, there is a function for logging user out which sets the token and user id back to null. The whole application is wrapped with auth context provider and all the components in the application can access to variables stored in the context to determine whether a user is currently logged in or not. Figure 20 below shows the structure of authenticated stack.

![Auth Stack Diagram]

Figure 20. Navigation Flow of Auth Stack

In auth stack mobile application starts with home screen which displays all rental properties stored in the database in the form of tabs. A user can click on a tab to navigate to the rental place details screen. It shows image slider for all images stored for that rental property, detailed information of the rental property and contact information of the advertiser. A user can navigate to the login screen using icon on the top right corner. Login screen provides input fields for the email and password. User who are not registered with the application can press the button sign up instead to navigate to sign-up screen. A sign-up screen contains multiple input fields required of the new user.
Figure 21 shows all the screens in the authenticated stack. User is directed to authenticated stack by either logging in or signing up with the application. User can access the home screen, add rental place screen and profile screen from bottom tabs. Add rental place screen provides a screen to enter required data of the rental place and ability to either take photos of rental place by camera or pick them up from the library. Profile screen displays all the rental places published by logged in user in the form of tabs similar to home screen. The only difference from the home screen is that user can delete their uploaded rental places visible on this screen. Authenticated stack is replaced with the auth stack once the user presses the logout button on the top right corner of the bottom tabs navigator.

4.4.3 Screens

In this section, two most important screens are discussed in detail. First one is Home Screen which shows all rental places with an image and selected details in the form of cards. Figure 22 below shows the real user interface of the home screen in both android and iPhone.
Figure 22. Home screen on iPhone (left) and android (right)

Home screen is the first screen visible to user in the application. RentalPlaceList component handles displaying of rental places. It uses FlatList component by react native to display scrollable list of cards. Every time home screen becomes visible to user it fetches all rental places from the backend server. This functionality is implemented by using useEffect hook from react and useIsFocused hook from react navigation library. The code for this is shown in figure 23 below.
Figure 23. Code snippet from AllRentalPlacesScreen.

Whenever home screen is loaded isFocused variable returned by useIsFocused hook returns true. This leads to execution of fetchRentalPlaces function which send axios GET request to the backend server. Fetched rental places are fed to the rental places context which in turn updates the home screen.

The second most important screen of this application is AddRentalPlaceScreen. It is responsible for sending data to the back-end server for creation of adding new rental place to the application. The layout of this screen is shown below in figure 24.
Figure 24. Add rental place screen on iphone (left) and android (right)

This screen uses custom made component AddRentalPlaceForm to show a form in which user can enter data of the rental place. It also provides a functionality for adding images for the rental place. Adding images is done with the help of dedicated component named ImagePicker.js. This component uses library provided expo-image-picker to let users either pick images from their gallery or take a photo with their camera. It adds images to array of selected images maintained in the state of form component. The form component makes use of FormData() provided by JavaScript to create data with key value pairs which are then sent to the server with content type set to multipart form data. The following code snippet in figure 25 shows creation of form data for rental place.
const createFormData = () => {
const data = new FormData();

selectedImagesArray.forEach((image, index) => {
    data.append('rentalPlaceImage', {
        name: 'image' + index,
        type: image.type,
        uri: image.uri.replace('file://', ''),
    });
});

data.append('title', inputs.title.value);
data.append('streetAddress', inputs.streetAddress.value);
data.append('postCode', inputs.postCode.value);
data.append('city', inputs.city.value);
data.append('rent', inputs.rent.value);
data.append('description', inputs.description.value);
// dummy lat and lng values for server.
data.append('latitude', '35.8819');
data.append('longitude', '76.4643');

return data;

Figure 25. Code snippet for creation of form data.

This form data along with token fetched from auth context is then sent to the backend server with axios post request. http file in utils folder implement all functions for sending http requests to the server. Content type is set to multipart form data and authorization is set to bearer with space and a token. Next figure 26 shows the implementation of a function for sending the request.
4.5 Results

The implementation of backend-server largely satisfies all the requirements of the client. However, during the testing phase a couple of observations were made to further develop the server and were shared with the client. Firstly, backend server is tested manually with the help of Postman and mobile client. This consumed many hours and testing process was very tedious. Therefore, it is strongly recommended to bring in automation testing by writing tests for each module of backend server. This will save time and will be the only feasible solution once application grows even further. Secondly, the backend server is currently saving public URLs of the image files stored in the Amazon S3 service. These URLs are sent back to the mobile client when list of rental places is requested. This leads to security issue as a user of this application gets his hand on public URL of the bucket saved in Amazon S3. Therefore, it is recommended to give some kind of encrypted URL to the user which expires after certain amount of the time.

Hybrid mobile client is currently fulfilling all the requirements of the client except the location functionality. The location functionality should have provided user
with the ability to select location pin on the map view or use their current location for their rental place. This location data containing longitude and latitude should then be uploaded to the database. Some progress has been made on this feature but it could not get ready due to time constraint of this prototype. It has been mutually agreed with the client to shift its implementation to the second prototype of this application. Apart from this feature many improvements are required to make it launch worthy. As mentioned earlier a survey about this application is currently under progress. It includes among other things different prototypes for user interface. As soon as the results of the survey are ready a more concrete list UI improvements can be made for the next prototype adding such as features like more details about the user signing up and finalizing theme of the application. Similar as the back end currently all features of this hybrid client are only tested manually. Therefore, it is strongly recommended to write test code for all classes integrate automation testing for front end hybrid client.

Summing up, this project has provided a solid foundation for bringing this idea to a reality. It has provided a complete backend with a database and cloud storage providing fundamental features required by the application. Mobile client also implement basic features for this rental places platform. This whole structure of this project can be carried forward to incorporate new features and improve UI design to better suit the needs of the target users.
5 Conclusion

Information technology has helped in overcoming challenges in almost all fields of life. It can be used to overcome hurdles in tourism industry in Pakistan. After this tourism portal has been completely deployed it will give a major boost to the tourism industry in Pakistan. This application is an integral part of this portal. It will digitize rental accommodation sector of this industry. Currently with this application all hotel owners and local entrepreneurs can post their properties online. The clients can then see pictures of rental property and other details such as address, rent per night and description. The clients or tourists can then contact advertisers of rental properties by their contact information. The backend of the application is written in Node.js and it stores all the required data in the online database. It is currently deployed on Heroku cloud runtime environment. Images of the rental properties are stored using Amazon S3 bucket. The URLs of these images are then stored in the database. A document-based database named MongoDB is used for storing data. It is deployed online using MongoDB Atlas.

As mentioned earlier there are some areas of improvement noted during testing phase of this application. These shortcomings would bar this application to reach its full potential. There are also recommendations for adding new features in the future. They have been shared with the start-up client and added to the back log for the next prototype. Some of these shortcomings and suggestions for new features are discussed here. Firstly, the application was tested manually and there is no automation testing suite for both the back-end server and front-end hybrid mobile client. The automation testing will make application development faster and more efficient. Secondly, users cannot use their current location to mark their rental property on the map. This feature will make it easy for the rental property owners to upload their places to the application. Thirdly, clients are not able to contact the owners of rental properties through the application. The inbuilt chat system in the application Clients cannot see the reservation calendar for the rental properties. Fourthly, the application should provide some kind of payment
mechanism which must include popular payment methods used in Pakistan. Fifth, a calendar must be available for each rental place so that clients can see its available days. Sixth, application should provide a rating and review system. It will increase confidence of the application clients in this portal. Additionally, the data models for both user and rental place will be updated as soon as the survey results are available. Lastly, it is strongly recommended to only continue with improving and adding features after the results of the application survey are available. However, writing testing suites and implementation of CI/CD for the whole project can be started immediately.

As with all real-life projects, this project has also faced a lot of challenges and delays during the development. The biggest challenge was to agree about the basic requirements with the start-up client especially when the survey was still under process. The other technical challenges include writing platform specific code for accessing camera and handling photo gallery permissions on both android and iOS platform, setting up IAM policy for the Amazon S3 bucket, dealing with android emulator bugs and using axios library to send HTTP requests to back-end server deployed locally. The application is developed using best coding practices and modularly. Modular structure will help to improve current features and add additional features efficiently. This project has provided complete implementation framework for this portal. Summing up, this project has been able to satisfy all the needs of the start-up client. It has also been successful in implementing first concrete application for this tourism platform.
References


