Narendra Lama

Providing Native Experiences in Mobile with PWA

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

20 November 2019

Author Title Number of Pages Date	Narendra Lama Providing Native Experiences in Mobile with PWA 34 pages + 3 appendices 20 November 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of School (School of ICT)

The thesis aimed was to gain a deeper understanding of progressive web apps (PWAs) and how the knowledge obtained could be used to develop a functional PWA that provides the look and feel of a native mobile app while utilizing the power of the web. This is to explore the idea of breaking away from mobile apps ecosystem from development perspective which is expensive and time-consuming.

A bus booking web app was developed as a full-stack app using MERN stack. Since the communication between backend and frontend occurred through the REST APIs, it enabled for the development of frontend independently and enhance it with the use of modern web technologies to create PWA.

For a web developer with a fair share of knowledge of web technologies, it is relatively easier to implement PWA traits into the existing web app. From the user's perspective, the PWA experience is no different from a native mobile app with added accessibility of the web. Thus, the development of PWA is sure to remarkably benefit developers, businesses and users all over the world.

Keywords	PWA, native experiences with web

Contents

List of Abbreviations

1 Introduction 1

2 Background 2

2.1 Web Application 2

2.2 Mobile Application 3

2.2.1 Native App 4

2.2.2 Hybrid App 5

3 Progressive Web Apps 7

3.1 Advantages 8

3.1.1 Availability Regardless of Connection 8

3.1.2 Fast Load 9

3.1.3 Re-engagement 9

3.1.4 Home Screen Shortcut 10

3.1.5 Look of Native App 11

3.2 Core Tech of PWA 11

3.2.1 App Shell 11

3.2.2 Service Workers 12

3.2.3 Web App Manifest 16

3.3 Browser Support 18

4 Design and Modelling 20

4.1 Use Case Diagram 20

4.2 UI Design 22

5 Tools and Technologies 23

5.1 Database 23

5.1.1 MongoDB 23

5.2 Backend 24

5.2.1 Node.js 24

5.2.2 Express.js 24

5.3 Frontend 25

5.3.1 ReactJS 25
5.3.2 Redux 26
5.3.3 React-router 27
5.3.4 Bootstrap 27

6 Application Architecture and Development 27

7 Result and Discussion 30

8 Conclusion 31

References 32

Appendices

Appendix 1. Lighthouse Report for Desktop with Emulated Runtime Settings

Appendix 2. Search Result Page

Appendix 3: BookBus launched from the user's home screen

List of Abbreviations

API	Application Programming Interface
BSON	Binary JSON
CRA	Create React App
CSS	Cascading Style Sheet
CORS	Cross-origin Resource Sharing
DOM	Document Object Model
HTML	HyperText Mark-up Language
HTTP	HyperText Transfer Protocol
HTTPS	Secured HyperText Transfer Protocol
JSON	JavaScript Object Notation
MERN	MongoDB, Express, React and Node.js
ODM	Object Document Mapper
PWA	Progressive Web Application
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience

1 Introduction

Internet usage has come a long way since its advent in the 1980s [1] providing virtually every service with the click of button or tap on the screen. With the introduction of iPhone in 2007 [2], mobile internet has seen unprecedented growth within a decade, and it accounts for 51.65 per cent of global website traffic as of August 2019 [3]. This exhibits the importance of mobile internet users and how their consumption mode can affect business.

Building a mobile application which is platform and device agnostic has always been a challenge with an added burden of cost and resources. Likewise, developing a web app, considering the audiences from various geography mainly focusing on the internet availability and speed factor, has posed another grave headache for companies and developers. But with the advent of the idea of progressive web apps (PWAs), it is no longer such a feat. PWA combines the concept of a mobile app with web app thus providing a single solution for the growing number of mobile users including 2.71 billion smartphone users worldwide [4]. With the motto of offline first, developers can deliver the web app even for people having a weak internet connection and use the service without much distress. Even further, the PWAs can be installed on the user’s home screen offering a native app-like experience for users across devices and platforms enhancing their experiences while minimizing the development cost for the organization [5,22]. The later part plays a vital role given that more and more people around the world are relying exclusively on mobile devices to browse the internet [3].

To explore the idea of the thesis, ’BookBus’ was created which is a PWA implementing offline capability and ability to install on the home screen of the device. It is a full-stack application created with MongoDB, Express.js, React.js and Node.js (MERN) stack. The frontend is bootstrapped with Create React App (CRA) for easier and faster development instantiation.

The thesis is split into 8 sections for an effective elaboration of the topic. The first section talks about the background of PWA and the reasons for choosing this application development method. The second section explores the types of web application development and the third section dives deeper into PWA; its building blocks and working mechanism. The fourth section illustrates application modelling and design in an extended manner. The fifth section explains the tools and technologies used for application development while application architecture and development are detailed extensively in the sixth section. The outcome of the project and the thesis is articulated, and the future developments investigated and planned out in the following section. Finally, the thesis is concluded with the final thoughts, possibilities and what the future holds for PWA.

2 Background

A user of a mobile device has two options of accessing any application; through use of web app and mobile app which are introduced below.

2.1 Web Application

Web application (web app) is a computer program which runs in a web browser performing a specific function. A login form, newsletter signup form, simple survey in a webpage or a shopping cart are examples of the web app. This has been around since before the world wide web achieved conventional recognition. Since it runs in a web browser, the user can be in any platform or device to access it if s/he has a web browser. The content of the web app rendered depends on the browser, but it generally stays the same across platforms. Most web apps have a client-server architecture with client capturing the information and server storing and retrieving information for further use. The client-side is made up of HTML, CSS and often JavaScript while the server-side being built with a multitude of options such as CMS and servlets.

Websites used to be simple and static in past essentially allowing users to perform limited activities. But with the advancement of technology and availability of vast array development options, websites have transcended and can perform myriad activities competing with native applications. For example, photo editing, word processing, etc. which used to have specific software to perform such tasks are possible to do inside the web app. Web apps have come a long way from its same design across all devices to responsive web design, a word devised by Ethan Marcotte in his article in 2010 [6]. Two concepts were put forward before responsive design; graceful degradation and progressive advancement [7].

Graceful degradation starts with developing a website for larger screen devices such as desktops and building a full-fledged version with loads of functionality and trimming down those functionalities as the size of the screen goes down. While on the other hand, in progressive advancement, the website is designed for smaller screen first with basic features implemented; adding more interactions and effects to enhance the user experiences as the screen size increases. [7.] In 2010 mobile world congress event, Eric Schmidt, the CEO of Google encouraged designers to adopt the ‘mobile-first’ rule in product design which is a doctrine of progressive advancement approach addressing the continued growth of smartphone usage [8]. Since then, the mobile-first design has taken a huge leap and now almost every web application is mobile-centric or moving towards that goal.

2.2 Mobile Application

A mobile application is a type of software developed to work on devices such as phone, tablet or watch. It is generally only stated as the app as well. Such an app is a small unit and carries out a specific task only. The original purpose of apps was for productive and informative activities for instance calendar, email client, contact and weather information, nonetheless the exploding popularity of such apps and swift extension into additional sectors like mobile games, buying music and other media contents, factory automation, location-based services and shopping, prompted to have millions of apps to exists in the market. The mobile operating system owners operate application distribution platforms from where users frequently download the apps, for example, the Google Play or Apple Store (iOS). There are few independent app distribution platforms as well, such as Aptoide, F-droid and Cydia.

As of the third quarter of 2019, Google Play hosted 2.47 million apps with Apple’s App Store following the race with 1.8 million available apps [9]. Majority of the apps available on those stores are free while apps with price accounts for the remaining portion. The revenue generated from these paid apps is divided between the app developer and the distribution platform. With advancing technology, smartphones are being equipped with better capabilities along with growth in the number of apps available in app stores, users are installing an increased number of applications to their devices. According to the statistics of 2019, an average user spent 4 hours and 33 minutes in mobile of which 2 hours and 51 minutes is spent in smartphones and the remaining 1 hour and 43 minutes on tablets [10]. The latter is in decline, with mobile time continuing to unabatedly increase. What is more intriguing is that almost 90% of the mobile time is spent in apps [11]. Mobile apps provide some advanced features which enable them to be engaging and far more popular than web apps:

- The ability to send push notifications drive users back to the application even when the user is not using the app.
- Even though internet connectivity is required to accomplish most of the tasks in mobile apps, they can still offer basic content and functionality to users in *offline* mode.
- They can access device features such as camera, compass, accelerometer which can enhance user experience.
- Mobile apps can use advanced gestures such as pinch, tap, hold, drag and so on offering advanced functionality enabling users to execute a task gracefully.
- These apps store data locally on devices allowing quick data retrieval and thus performing actions much faster.

2.2.1 Native App

These types of applications are developed for a specific platform which means they are incompatible across platforms. In other words, an app meant for an Android device cannot operate on iOS devices. Consequently, these apps need to be developed for multiple platforms. Native Android apps are developed with Java, native iOS apps with Swift or Objective-C whereas Windows Phone apps are written with C#. The main purpose of developing such an app is to guarantee the best performance for a specific mobile operating system while encompassing consistency and better user experiences. Native apps also have easier access to the native features and hardware of the device such as compass and proximity sensor which enables to develop a more robust application.

Native apps act in tandem with the mobile device they are created for, so they work swiftly and with proficient, nonetheless, they are expensive to develop and time-consuming since different codebases are needed for the different platforms that the developer chooses to build in.

2.2.2 Hybrid App

Hybrid apps are a combination of the idea of native app and web apps. They are cross-platform and developed with web technologies, for instance, CSS, HTML and JavaScript. They are, in fact, a web app which runs in something called **WebView** or container usually provided by frameworks. Similar to native apps, they are circulated via the native app store and can access certain device features such as storage and compass. Since the same code base is used to develop the application, this essentially results in faster and cost-effective development in comparison to developing two native apps with half the numbers of developers. Or to put it another way, the development of the same application can be achieved in half the time by the same number of developers.

The main disadvantage of hybrid apps is exhibiting lower performance than their native counterpart. Since hybrid apps work inside WebView which is a browser-like component and is responsible for presenting the UI and executing the JavaScript, they completely depend on the performance of WebView. Due to this, apps fail to attain the same look-and-feel in different mobile operating systems. Apps developed using Sencha Touch, Apache Cordova and Framework7 fall into this category.

There is another category of application development which transcendence hybrid apps in the sense that even though being developed through web technology, which is JavaScript, these kinds of apps are converted into native modules rather than just outputting into WebView. Thus, without forfeiting the user experience or access to native APIs, they can obtain the advantages of cross-platform development. React Native, Titanium, Flutter and Xamarin are such frameworks that enable cross-platform native app development.

Even though mobile apps provide ease of use and better performance, there are many factors to discourage the development of such apps:

- Discoverability is the main issue with mobile apps. With over 2.47 million of apps in Google Play Store along with 1.8 million apps in App Store [9], that creating one that will be found by enough users to generate a return of interest (ROI) requires significant marketing spend. This spending is often twice/thrice the app development cost as mentioned by Brian Cardarella, chief executive officer (CEO) and co-founder of DockYard [5,22].
- Native apps that gobble up storage and require connectivity have long been a thorn in the side of international customers who may have limited storage on their phone.
- With many apps being discovered infringing the privacy of users and compromising their security, users have become more sceptical of downloading lesser-known apps.

Figure 1 depicts the individuals' apps usage based on time spent.

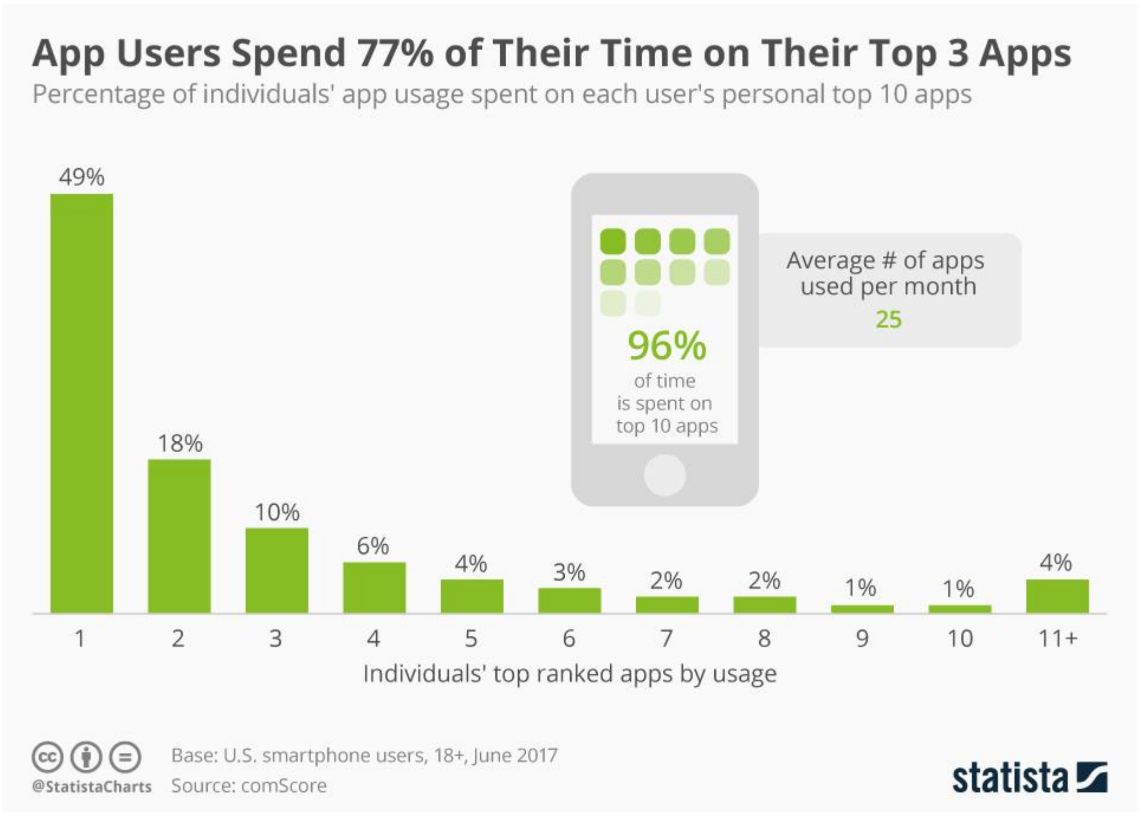


Figure 1. Individuals' time spent on apps copied from Statista (2017) [12]

While time spent on apps is an all-time high, the data from United States smartphone users shows that almost 80% is spent using favourite three apps and whopping 96% of the time is committed to their favourite ten apps as shown in Figure 1. This shows that it is hard to penetrate the native app market.

3 Progressive Web Apps

Progressive web apps (PWAs) are a new variety of web apps which conjoin the best of web app and native app providing preeminent experiences for users. They are developed entirely as websites, however, as the user interaction increases, they evolve into a traditional, native app [13]. In other words, while developing PWAs, it is not necessary to create an application using all their features – new features can be added when it feels that the user might benefit from it and provide them with enhanced experiences. Thus, PWAs enable developers with the ability to develop faster, resilient, and more engaging websites that can be accessed by billions of people around the world and enact faith with users and attain new capabilities as required. [14.] Mobile app features such as the ability to work offline, push notifications and accessing device hardware acknowledges PWAs to secure a spot in the user’s home screen. Additionally, a strong focus on performance with the utilization of modern web tech means that PWAs are fast for everyone [15].

Google has put forward a checklist to ensure a baseline PWA is created with best possible experiences which can be further enhanced into model PWA with delivering of additional eloquent offline experience, attaining interactive swifter in addition to taking consideration of many more essential factors. The baseline PWA should ensure these requirements:

- Hypertext Transfer Protocol Secure (HTTPS) protocol to be implemented while serving the site. This prevents unwanted invaders from altering the connections amongst the websites and their users’ browsers. It also prevents intruders from being able to passively listen to communications between websites and their users.
- Responsive pages for different device sizes which provide users with smoother and enhanced experiences.
- The user should be able to view at least minimum content in case of network unavailability but never an error page.
- Declaration of web app manifest file which contents all the information about the app and its behaviour to enable Add to Home screen function.
- Fast initial load even in slower internet connection to ensure that the app is performant irrespective of network status. This is to prevent abandonment

of web app as a 2016 Google study found that 53% of mobile sites are discarded when the site requires three seconds or more to load and additional 10% with each additional second of load time, according to BBC [16].

- Cross-browser operability of site so that the app feels the same on every browser.
- Transitions should feel swift while navigating around the app, even on a slow network. And in case there is a delay in the response from the network, the app should provide with the loading indicator.
- Individual pages should have URLs ensuring for deep linkable. This enables the shareability across social media platforms and each page to be launched and directly opened through a new browser window. [17.]

For checking these requirements, Google has also created a tool called Lighthouse which can evaluate the comprehensiveness of a web app in percentages [18]. With the implementation of numerous features, an app can be made progressive further, consequently leading to a better Lighthouse score. Nevertheless, it can only be used as an approximate indicator.

3.1 Advantages

PWAs, being loaded with the superpower of the web, also fulfil many expectations native apps deliver sometimes exceeding them as well.

3.1.1 Availability Regardless of Connection

PWAs are independent of the user’s internet connectivity, unlike conventional websites. As the user navigates to the site, the browser registers the service worker which can identify and respond to the user’s network status. This enables the user to have the same experience on the site whether s/he is online, offline or experiencing unstable connection. The users travelling in an area without a network can perform certain tasks in the PWA with confidence as those activities will be completed as they get back internet connection irrespective of whether the app is still open or closed. This level of reliability and trust was so far only received by native apps.

3.1.2 Fast Load

With the help of service workers, sites can be launched instantly irrespective of the user’s network connection similar to mobile apps downloaded from the app store. This is possible with the downloading of necessary assets, called caching and occurs during the first visit of the website by the user. Then, when the user reopens the application, instead of downloading the files again over the internet, they are simply retrieved from the user's device. However, this only works when the user revisits the application. For the initial visit, everything must still be downloaded. This situation is particularly precarious because when the user first visits the site, they are not yet sold on its value, and so, are likely to leave if the loading takes too long.

It needs to be ensured that the assets are as optimized as possible and that as little as possible is downloaded on that first visit so that the user stays around. In short, fast loading for the first visit, near-instant loading for every subsequent visit.

3.1.3 Re-engagement

Although users spend most of their time on mobile apps, the reach is significantly higher for web apps. A study by ComScore on top 500 sites versus top 500 apps shows that mobile web enjoys more than 2 times the number of visitors than on mobile apps. [19.] One of the significant advantages of mobile apps is the effortlessness with which the users re-engage with the app about updates and new contents, even when they are not using the app or looking at their phone. Combining the reach of the web and native app feature such as push notification can drive users for re-engagement in PWA significantly. With the help of service workers and various web APIs such as Push API for sending updates from the server to the app and Notification API for generating system notifications can aid user engagement when they are not using the browser. In 2016, Jumia, the leading e-commerce in Africa, started their PWA journey and by 2017 saw their traffic on PWA eclipse that of their native apps by more than 12 folds, 33% higher conversion rate and 50% decrease in bounce rate [20]. With the help of push notification, Lancôme, a luxury cosmetics brand witnessed a boost by 12% in conversion rates on recovered carts [21].

3.1.4 Home Screen Shortcut

Enabling users to access the app via app icons on their home screen and starting the apps in their environment which is effectively integrated with the core platform is an essential part of the app experience. But making a user download an app from the app store is a huge challenge. More than half of smartphone users in the United States downloaded zero new apps per month as per the report published in the first quarter of 2017 [22]. Gartner's research director Charles S. Golvin claims PWAs can address the commitment challenges that companies face with skittish customers. "The depth of connection to the brand does not have to be as much with a PWA; they do not have to commit to downloading an app when they are not quite ready for that connection," he adds. [5, 23.] Another reason for hesitation for users to download mobile apps is large storage required to download such apps. This might not be an issue with high-end devices, but a large fraction of smartphone users is faced with limited storage on their phone. A well-functioning PWA may make that connection more likely in the long term. Once a user has interacted sufficiently with the progressive web app, the browser will automatically suggest him for installation of the web app shortcut on his home screen whose experience is identical to any native app. PWA added this way has minimal storage footprint in comparison to mobile apps installed from the app store. Figure 2 illustrates the data gathered from PWA case studies by Google.

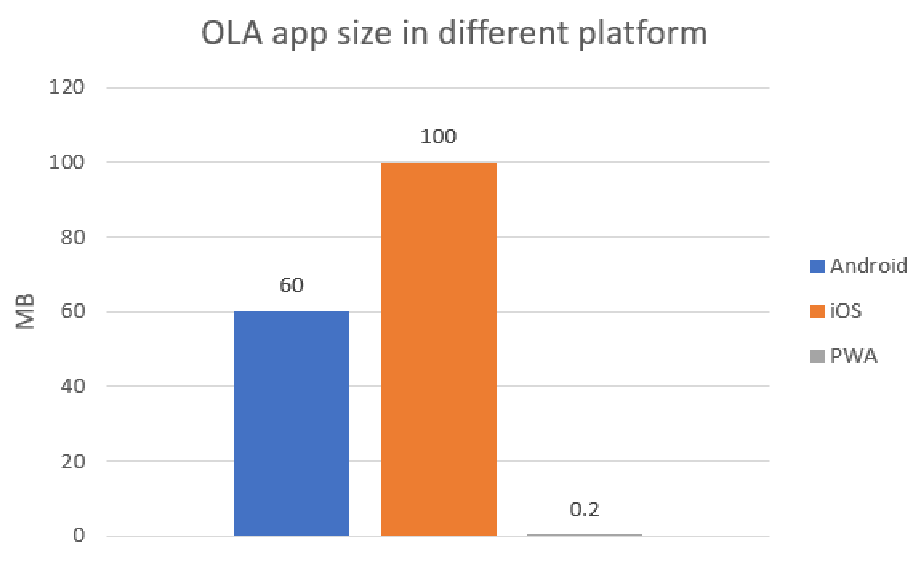


Figure 2. Data gathered from PWA case studies by Google (2017) [23]

Before the introduction of PWA app by OLA, a leading cab aggregator in India, their Android app needed 60MB storage space while iOS app needed much more at 100MB as shown in Figure 2. This resulted in users to be less apt to download their mobile apps. After building their PWA version which is at least 300 times tinier than their Android counterpart furthermore 500 times tinier than the iOS app, they saw their traffic grow by 68% in their Tier 2 and Tier 3 cities [23]. Similarly, George, a leading clothing brand in the UK, used a “Add to Home Screen” prompt that revealed consumer interactions across the site has surged by 28 per cent, completely providing a native app-like feel on the web [24].

3.1.5 Look of Native App

Progressive web apps opened from the home screen can attain a complete native app-like appearance. They can display a splash screen as they are starting up and gets loaded instantly. With the right configuration, they can launch in full-screen mode like a native app, which does not contain the browser or any phone UI around them. They can even lock themselves to a particular screen positioning. They also provide similar features as native apps such as app drawer, display icon on the home screen, push notification and integration with the system.

3.2 Core Tech of PWA

PWAs consists of three components: an app shell which is the nominal user interface that can be cached so it is available offline for successive visits; service workers that provide background sync and offline usage capability and an app manifest file which makes it possible to install the app from browser into the home screen of the device. [5,21.]

3.2.1 App Shell

The App shell theory is involved with loading a nominal user interface when the PWA is launched followed by caching the content, so it is available offline for successive visits. This ensures the loading of the UI from the cache immediately when a user visits the app next time from the same device and request of new contents from the server afterwards. This structure is fast and makes a user feel the app is fast as s/he sees something instantaneously, rather than a blank page or a loading icon. It enables the website to be offline reachable as well. With a service worker, it is possible to control what is requested from the server and what to retrieve from the cache.

This architecture enables a website to take advantage of most of the PWA features — it caches the app and requests the dynamic content which increases the performance. Besides, features like push notifications or Add to home screen can also be implemented if the user's browser supports them. This is possible with a progressive enhancement method of web app development.

3.2.2 Service Workers

Service workers are the main building block of PWA unlocking much of the functions that make PWA special. They make features such as offline access, periodic background sync and push notifications achievable on the web which normally require a native application. A service worker is a JavaScript code which the browser executes on a separate thread from the main JavaScript code of the webpage in the background and hence, does not have any access to the DOM. Thus, it does not block the working of the main JavaScript of the website while communicating between multiple settings. [25.]

Service worker, being a programmable network proxy, interposes the request that the application sends to the server to fetch any file. It then analyses the local cache to see if the file is already available. If it does not find the requested file, then allows the network to proceed the request to the server and when the file is received, saves in the cache for future use. [15.]

Because it is a powerful tool, a service worker can only be executed in secure environments using HTTPS. Following the progressive enhancement principle, features provided by service worker are only utilized if the browser supports them. For example, a service worker caches the app shell and relevant data of the app prompting them to be available even in the absence of the network connection. In case service workers are not supported, the files cached are not executed resulting in basic experience of the website. With the use of feature detection, it is possible to deliver progressive enhancement to the users and the features will not break in non-supported browsers.

The service worker life cycle

The service worker lifecycle starts when a user navigates to the website. If the service worker file is detected by the browser, the browser downloads and parses the file and then commences the execution. In case the execution fails, the registration gets declined which in turn ends the service worker registration procedure. While on the other hand, if the registration succeeds and the service worker gets resolved, its status changes into installed as shown in Figure 3. With **installation**, the static assets are cached and are ready to be served. Installation only occurs for the first time after registration. With the successful installation, the service worker gets **activated** and has full control of the website under its scope. [26]

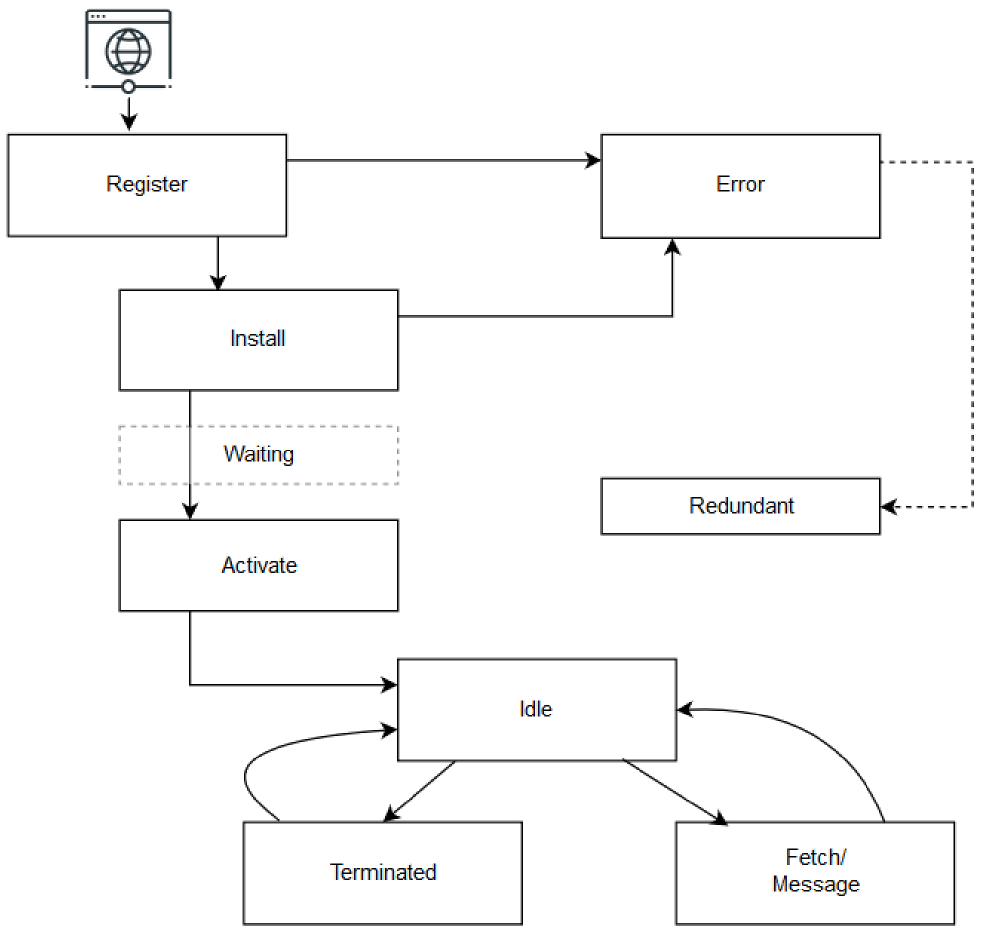


Figure 3. Service worker lifecycle modified from Hajian (2019) [26]

The scope refers to the contents that a specific service worker control. For example, the service worker file located in the origin of the app controls all the pages of the web app. On the other hand, '/offline/' means only access to pages under this scope. The service worker will get effective after the install and activate event completes without errors. Once the service worker gets activated, it will exist in one of the two states: the first one is to be in **terminated** state to free memory used by the application and another is to handle **fetch** and message events which take place once a message or network request event occurs from the web app [27].

A website without a service worker works normally as there is no service worker to handle the requests. As it gets installed and activated, it takes control of all the requests under its scope. But for the service worker to initiate its function, either the user needs to navigate to another page, or the page requires refreshing. [26.]

Finally, when the registered and activated service worker gets updated, the browser follows all the phases, as mentioned earlier. As the service worker is already in activated phase, the new service worker will only get installed. It will enter **waiting** state until all tabs with old service worker running are closed. This enables the outdated service worker to be terminated. When the webpage is reopened, the new service worker will take control of the site and the whole process repeats for subsequent service workers.

Caching

Caching is the process of storing web app information in the browser for a temporary state. It helps to reduce network requests and in case of PWA, support the web app for offline use. Since service worker resides between network and application, caching can be performed as part of service worker lifecycle as it intercepts network request and responds with the requested file, grabbing it from the cache instead of the server; thus, saving time and the user’s bandwidth. An app shell cached during the install event of service worker lifecycle is a powerful pattern as this offers significant performance boost in the form of instant loading for repeat visits of the web app and during offline as well. An example of caching mechanism during the install phase of service worker can be seen in Listing 1.

```
const cachedName = 'assets';
const allFilesToCache = [
  '/css/styles.css',
  '/js/script.js',
  '/imgs/logo.svg',
  '/',
  '/offlined.html',
];

self.addEventListener('install', evs => {
  evs.waitUntil(
    caches.open(cachedName)
    .then(c => {
      c.addAll(allFilesToCache);
    });
  );
});
```

Listing 1. Caching contents with a service worker modified from Google Developers [28]

Since the cache storage has a hard limit in each browser for a given origin, it may simply delete all data from that origin if it gets full, so it is best to store the bare minimum. Objects in a Cache needs to be explicitly requested to update and to delete as they do not expire. So, it is the developer's responsibility for maintaining cache entries periodically.

Re-engageable PWA with Notifications and Push APIs

The ability of the app to work offline with cached contents is a huge success as a web app. But this is not enough to provide users with elevated experiences of using PWA. Implementation of push notifications drive towards re-engagement and providing up-dated contents when available. Service workers not only help with serving files from the cache, but they also enable to deal with push notifications. Since Push API and Notifica-tions API work in a synchronous manner, using them together will facilitate to provide engaging functionality in PWA. Push API is used to receive new and updated contents from the server to the app and this is controlled by the service worker. Afterwards, the service worker uses Notifications API to display the new contents to the users or alert them about updated information.

3.2.3 Web App Manifest

It follows web app manifest specification and is written in a JSON format and specifies data about the application for example name, description, theme colour and icons as displayed in Listing 2. This file permits a web app to be installed on the user's home screen, customize the theme, splash screen setup and URL that opens when the app is launched.

```
{
      “name”: “MyPWA”,
      “short_name”: “MyPWA”,
      “description”: “My first PWA!”,
      “start_url”: “/”,
      “scope”: “.”,
      “theme_color”: “#453343”,
      “background_color”: “#453343”,
      “display”: “standalone”,
      “icons”: [{
            “src”: “/imgs/icon512x512.png”,
            “sizes”: “512x512”,
            “type”: “image/png”
      }]
}
```

Listing 2. A sample manifest.json file

Every property in the manifest file serve a definitive purpose which informs the browser data concerning the app's impression and appearance.

Adding to Home Screen

A reliable way for users to engage with mobile apps is with the app icon installed on the user's device. Integrating this functionality on web apps helps with user engagement and to do so users are able to install the web app to their home screen, which will appear like a regular app, with its icon. This feature is available in modern mobile browsers labelled as Add to Home Screen (A2HS) or web app install banner as displayed in Figure 4. It shows a prompt and when the user agrees to it, the app will be installed on the user's device app drawer. Once the user clicks on the app icon, it will start with a splash screen and run in full-screen mode, so it looks and feels like a regular app.

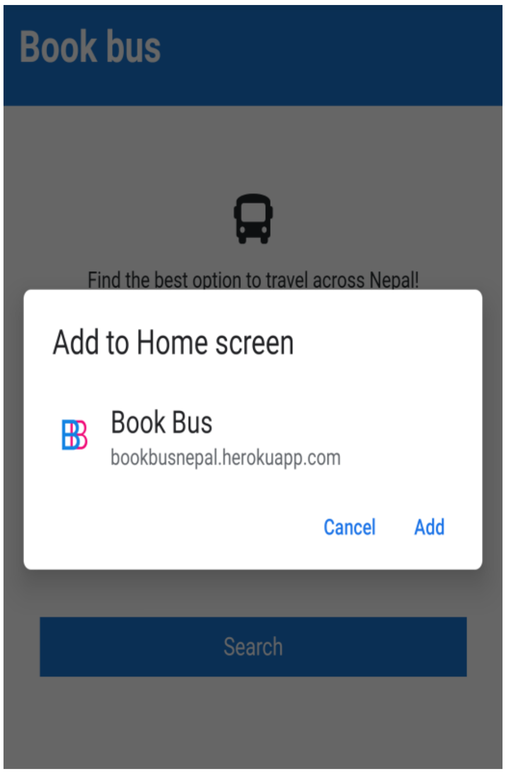

Book bus
Find the best option to travel across Nepal!
Add to Home screen
Book Bus
bookbusnepal.herokuapp.com
Cancel
Add
Search

Figure 4. Add to home screen banner for BookBus

There are some criteria for the web app that needs to be met for the A2HS prompt to be shown to the users:

- It should be served over HTTPS.
- It has a web app manifest file with correct fields, linked from HTML head.
- It has icon available of applicable size to display on the app drawer or home screen.
- An appropriate user engagement heuristic.
- It is not already installed.
- It has a registered service worker in case of Chrome. [29.]

When these criteria are met, the *beforeinstallprompt* event will be fired that can be used to prompt the user to add the Progressive Web App (PWA) and may show a mini-info bar. Listing 3 shows an intercepted *beforeinstallprompt* event and invoking a custom function to trigger the event for later use.

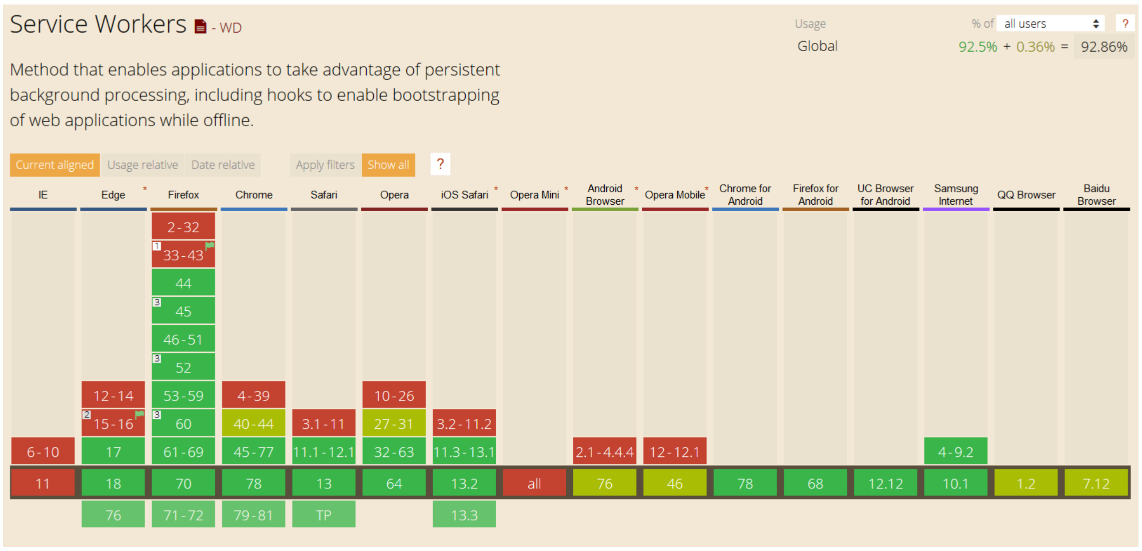
```
let delayedPrompt;

window.addEventListener('beforeinstallprompt', (e) => {
  delayedPrompt = e;
  showInstallPrompt();
});
```

Listing 3. Code for listening *beforeinstallprompt* and storing the information for later use to notify the user that the app can be installed [29]

Many different patterns can be used to notify the user about the app that can be installed thus promoting the installation, for instance, an element in the navigation, a button in the header or an element among the content section. These prompts must not interrupt users from what they are doing.

3.3 Browser Support

Developing PWA can only be obtained with the use of multiple technologies which ensure the delivery of the best web experiences. The main element necessary for PWAs is service worker support. Fortunately, it is supported by almost all browsers on mobile and desktop except Internet Explorer and Opera Mini which doesn’t support service worker yet as shown in Figure 5. Support for features like web app manifest, notification API, push API and web app install banner can also be found widely. At the time, the support for web app manifest and web app install banner are restricted in Safari while web push notifications have no support. So, the users need to install the web app manually by selecting the share button. It can be seen how about 93% of global browser users have access to service worker feature whereas the total stands at 83% for web app manifest file as stated in Figure 5 and 6 correspondingly.


Service Workers - WD

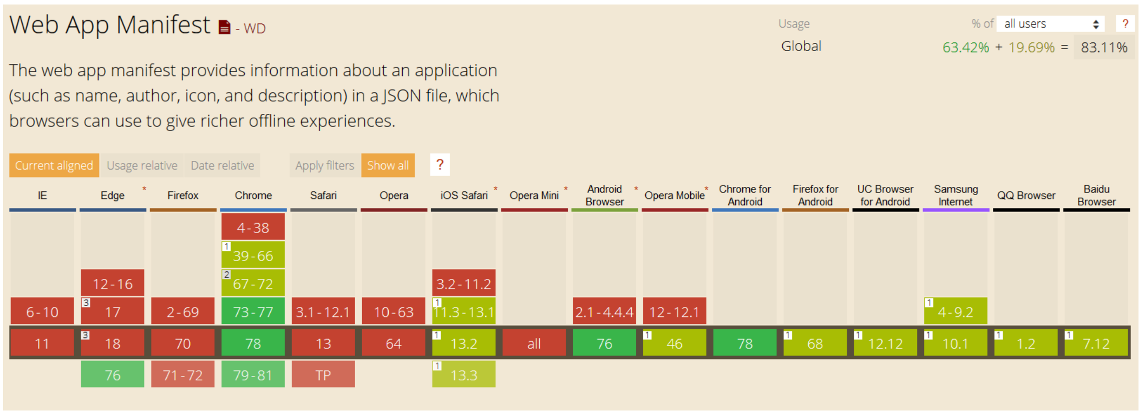
Usage: Global — % of all users: 92.5% + 0.36% = 92.86%

Method that enables applications to take advantage of persistent background processing, including hooks to enable bootstrapping of web applications while offline.

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser
		🟥 2-32													
		🟥 [1] 33-43													
		🟩 44													
		🟩 [3] 45													
		🟩 46-51													
		🟩 [3] 52													
	🟥 12-14	🟩 53-59	🟥 4-39		🟥 10-26										
	🟥 [2] 15-16	🟩 [3] 60	🟨 40-44	🟥 3.1-11	🟨 27-31	🟥 3.2-11.2									
🟥 6-10	🟩 17	🟩 61-69	🟩 45-77	🟩 11.1-12.1	🟩 32-63	🟩 11.3-13.1		🟥 2.1-4.4.4	🟥 12-12.1				🟩 4-9.2		
🟥 11	🟩 18	🟩 70	🟩 78	🟩 13	🟩 64	🟩 13.2	🟥 all	🟨 76	🟨 46	🟩 78	🟩 68	🟩 12.12	🟩 10.1	🟨 1.2	🟨 7.12
	🟩 76	🟩 71-72	🟩 79-81	🟩 TP		🟩 13.3									

Figure 5. Browser support for service workers and its global reach copied from caniuse.com [30]


Web App Manifest - WD

Usage: Global — % of all users: 63.42% + 19.69% = 83.11%

The web app manifest provides information about an application (such as name, author, icon, and description) in a JSON file, which browsers can use to give richer offline experiences.

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser
			🟥 4-38												
			🟨 [1] 39-66												
	🟥 12-16		🟨 [2] 67-72			🟥 3.2-11.2									
🟥 6-10	🟥 [3] 17	🟥 2-69	🟩 73-77	🟥 3.1-12.1	🟥 10-63	🟨 [1] 11.3-13.1		🟥 2.1-4.4.4	🟥 12-12.1				🟨 [1] 4-9.2		
🟥 11	🟥 [3] 18	🟥 70	🟩 78	🟥 13	🟥 64	🟨 [1] 13.2	🟥 all	🟩 76	🟨 [1] 46	🟩 78	🟨 [1] 68	🟨 [1] 12.12	🟨 [1] 10.1	🟨 [1] 1.2	🟨 [1] 7.12
	🟩 76	🟥 71-72	🟩 79-81	🟥 TP		🟨 [1] 13.3									

Figure 6. Browser support for manifest.json and its global reach copied from caniuse.com [31]

Some of the APIs for PWA is still in the experimental stage, with the documentation being in the draft but considering the achievement of many companies such as MakeMyTrip and Jumia should convince to apply some of the PWA traits in the web app already.

4 Design and Modelling

With such numerous advantages and benefits discussed previously about PWA, the aim of the study was to develop a bus booking system which allows users to search buses, book the ticket and receive the confirmation through email.

A good designing enables to develop the application smoothly, helps in following the schedule and minimizing the risks being able to foresee them. To understand how the application will be used, the use case diagram was designed first. This allows for developing a user-centric app.

4.1 Use Case Diagram

Figure 7 shows a use case diagram of the web app from the perspective of a user.

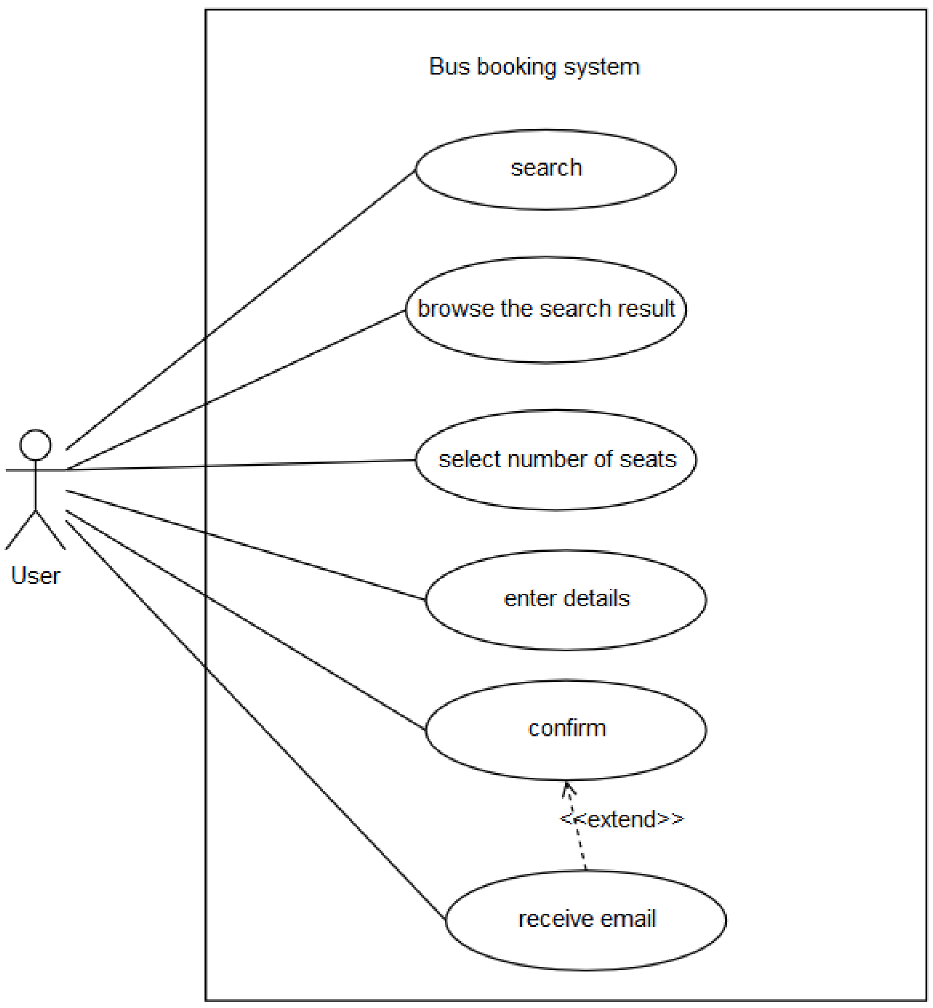


Figure 7. Use Case Diagram for BookBus system

The use case diagrams as presented in Figure 7 can be described by user stories. User stories provide a non-technical way of describing the outcome of the assignment thus enabling developers to better understand the feature in a user-friendly approach.

- As a user of the bus booking system, one would want to search for the buses available for the journey.
- As a user of the bus booking system, one would want to browse through the list of available buses before deciding.
- As a user of the bus booking system, one would want to choose the number of seats they would like to book.
- As a user of the bus booking system, one would want to enter details about them so that there is no problem when travelling.

- As a user of the bus booking system, one would want to make sure the information about the journey is correct by reviewing it.
- As a user of the bus booking system, one would want to confirm that they want to book the ticket for the journey.
- As a user of the bus booking system, one would want to receive an email confirming the booking.

With above-described user stories, it is clear what a user wants to achieve using the web app.

4.2 UI Design

To further simplify the development process, an outline of the user interface design was implemented. This helps in figuring out what kind of design is worthwhile from user interface/user experience (UI/UX) perspective resulting in significantly reduced development time. A wireframe of the search result page of the web app was created with draw.io as shown in Figure 8. This helps to figure out important aspects of the page and efficiently develop them.

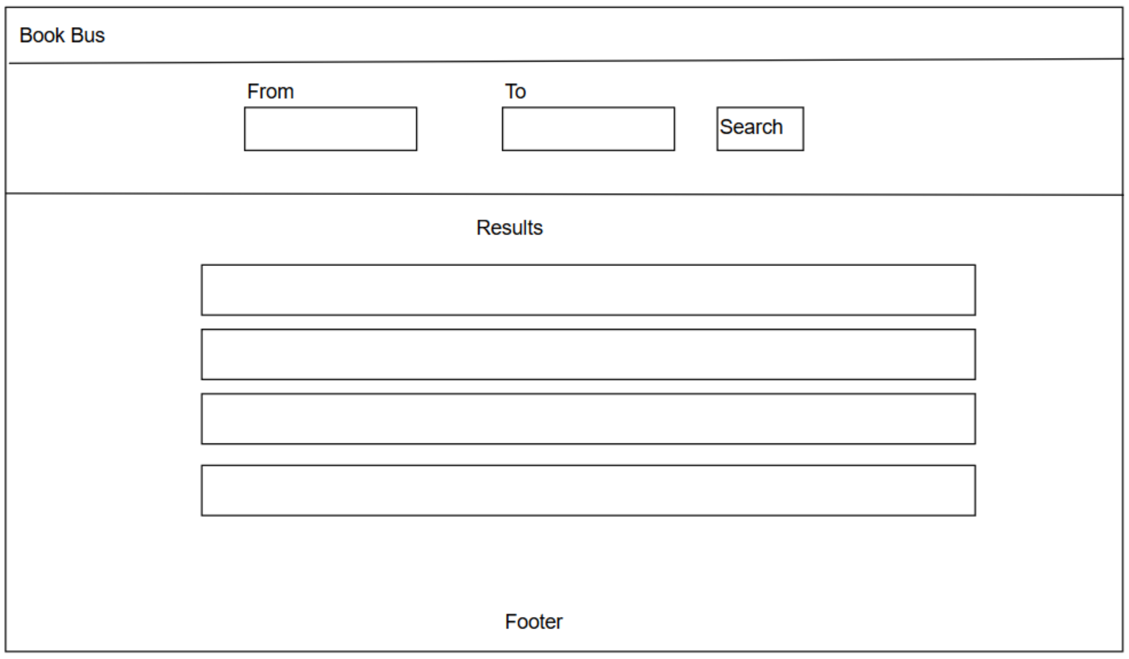


Figure 8. Wireframe of buses search result

In Figure 8, it can be seen that the search result page can be divided into two distinct sections with search fields filling the initial section of the page while search results getting displayed on another one. Search results include individual result component which can be developed separately. Overall, all these sections can be developed independently promoting a component-based architecture that ReactJS broadly advocates which is used for the development of front-end code.

5 Tools and Technologies

To develop BookBus as a full-stack web application, the tools and technologies described below were considered.

5.1 Database

Developing a full-stack application requires storing and accessing the information as the users interact with the app.

5.1.1 MongoDB

MongoDB is a document-oriented cross-platform database engine. It uses JSON-like documents called BSON to store data in key-value pair which classifies it as a NoSQL database program. SQL databases store data in tables and the schema definition is strict while NoSQL databases have flexible schema definition. This allows for lenient and faster data storage and retrieval mechanism which in turn provide the simplicity of design, finer control over availability and limiting object-relational impedance mismatch.

Mongoose

Mongoose is an object document mapper (ODM) for MongoDB database providing a clear-cut, schema-based answer for modelling the app records. It consists of validation, type casting, business logic hooks, query building and other features built-in to help interact with MongoDB.

5.2 Backend

The data access layer known as backend handles the business logic of the application and provides a safe spot for integration with the database.

5.2.1 Node.js

Node.js is a JavaScript runtime environment developed over Google’s V8 engine which accelerates the development process since the developer can use JavaScript knowledge to develop backend service. Thus, it combines web app development over a single programming language, instead of multiple programming languages for the development of client-side and server-side. Also, being event-driven, asynchronous and highly scalable, Node.js can handle a large quantity of input/output operations. With the comparative simplicity of installation and development across platforms, availability of abundant packages through node package modules (npm) registry Node.js was selected for BookBus application.

5.2.2 Express.js

Express.js is a lightweight web application framework for Node.js designed to build application programming interfaces (APIs). Due to its wide use, it is considered as the de-facto server framework to developing Node.js applications. It provides myriad sets of features for HTTP utilities and middleware. External middleware can also be integrated with Express.js thus further allowing to boost the development process. Therefore using Express.js, it was fast and easy to develop robust APIs for the BookBus application. Some of the external Node.js modules and middleware used for the development of the APIs include:

- body-parser: It parses the inbound request information in the middleware, accessible underneath the *req.body* property.
- compression: This middleware attempts compressing the response data in every request that pass across the middleware, predicated on various selections. However, it will not compress responses containing a Cache-Control header with the no-transform directive, as compressing them will alter the data.

- cors: This Node.js module provides an Express.js middleware which can be utilized for authorizing CORS with multiple selections.
- dotenv: It helps to load environment variables from .env file into process.env.
- helmet: It helps to secure the Express application by modifying various HTTP headers which prevent external prying eyes to know about the actual build process of the application.
- nodemailer: This module allows an easy approach to send email from the application which makes it possible to send email to users about their confirmed tickets.

The use of above mentioned Node.js modules and middleware along with Express.js enabled with building a healthy backend service that can withstand scaling and other issues that might occur as the application grows in size.

5.3 Frontend

The frontend is an abstraction of all the client-side or presentational layer of the application with a focus on developing a user-friendly interface.

5.3.1 ReactJS

ReactJS or simply React is a very popular JavaScript library developed by Facebook with declarative and component-based architecture for developing single-page applications (SPAs) [32]. It utilizes virtual DOM which is a JavaScript object. It creates an in-memory data structure cache called virtual DOM, compares the modifications between virtual DOM and modifies the resulting browser's DOM which is significantly faster rather than updating the entire DOM. It uses JSX or JavaScript XML for templating instead of regular JavaScript. It is type-safe; is significantly quicker as it carries out optimization during code compilation to JavaScript and the errors can be captured during compilation. And being similar in appearance to HTML, it is easier and faster to write templates in JSX. [33.]

JSX and modern JavaScript codes are not compatible with all the browsers available. So, to compile these codes to older JavaScript version so that current and older browsers can understand them, a tool called Babel is used. Babel uses plugins to convert syntax with a limited browser supports into a backwards-compatible version and polyfills to provide support for features that are missing entirely from target JavaScript environments.

To bundle all the modern JavaScript codes, assets, HTML and CSS files, a bundling tool such as Webpack can be used. Webpack is highly customizable but configuring it is a daunting and time-consuming task. Luckily, Facebook has created a tool called Create React App (CRA) which uses Webpack, Babel, ESLint, Jest and other amazing tools and includes pre-configured environments for development and production. It also supports PWA out of the box with the use of Workbox. Thus, the use of CRA streamlines and accelerates development process without the hassle and tedious task of configuring all those build tools. [34.] BookBus was initialized using CRA reflecting all the advantages it brings.

React is all about components. There are two techniques of creating components in React that is using class-based components (as known as stateful components) and functional components (also known as stateless components). With the advent of Hooks in React v16.8.0, it is now possible to have states in functional components [35]. BookBus was created using Hooks which makes it easier to reuse components across the application. Having encapsulated components with their own managed state permits to create complex UIs. This also makes codes more predictable, readable and easier to debug. But for complex applications, ReactJS is not enough as it requires the use of state management, routing and interaction with APIs.

5.3.2 Redux

Redux is a small JavaScript library to manage application state through limited API designed to be a predictable container. It maintains the state of an entire application in a single immutable object called state inside a single store, which cannot be changed directly. Instead of mutating the state directly, the desired mutation must be defined with plain objects called actions. Then these actions are passed through a special function called reducer where it is decided how the state of the entire application will be transformed based on every action. [36.] The detailed working mechanism of redux is discussed in Section 6.

5.3.3 React-router

Routings in SPAs are handled by client-side JavaScript instead of letting the browsers handle them. So, to obtain expected navigation features of normal web applications such as hotlinking and the back and forward button of the browser without sacrificing the sleekness of SPAs; routing should be handled correctly. React-router provides a collection of navigational components that mimic the navigational features provided by browsers and enables handling the routing decoratively. This approach enables managing the information in the application gracefully.

5.3.4 Bootstrap

Bootstrap is a CSS framework focused at responsive, mobile-first front-end web development. It uses a responsive grid system, contains massive prebuilt components and powerful plugins built on jQuery. With the utilization of a CSS framework, the development of frontend becomes faster and easier thus enabling developers to focus on the business logic of the application.

6 Application Architecture and Development

To develop a robust application, the application architecture needs to be solid to withstand possible failures. This starts with choosing the right technologies according to the need of the system build. Considering each tech stack has its strength and pitfalls, a careful analysis was performed for choosing the appropriate solutions. Then architecture of the application was constructed based on the selected tech stack as shown in Figure 9.

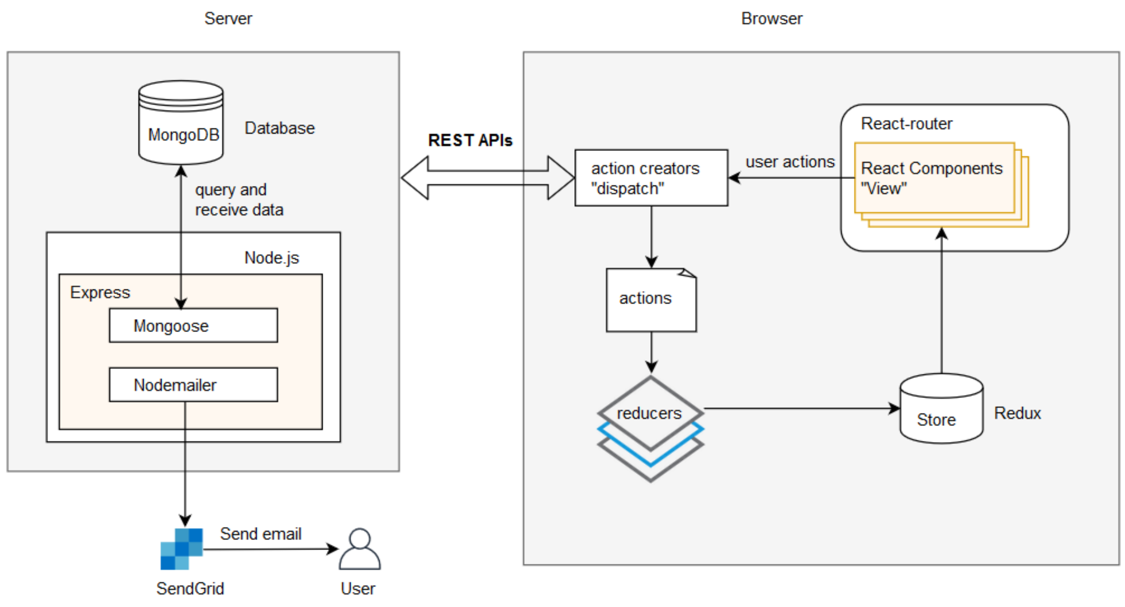
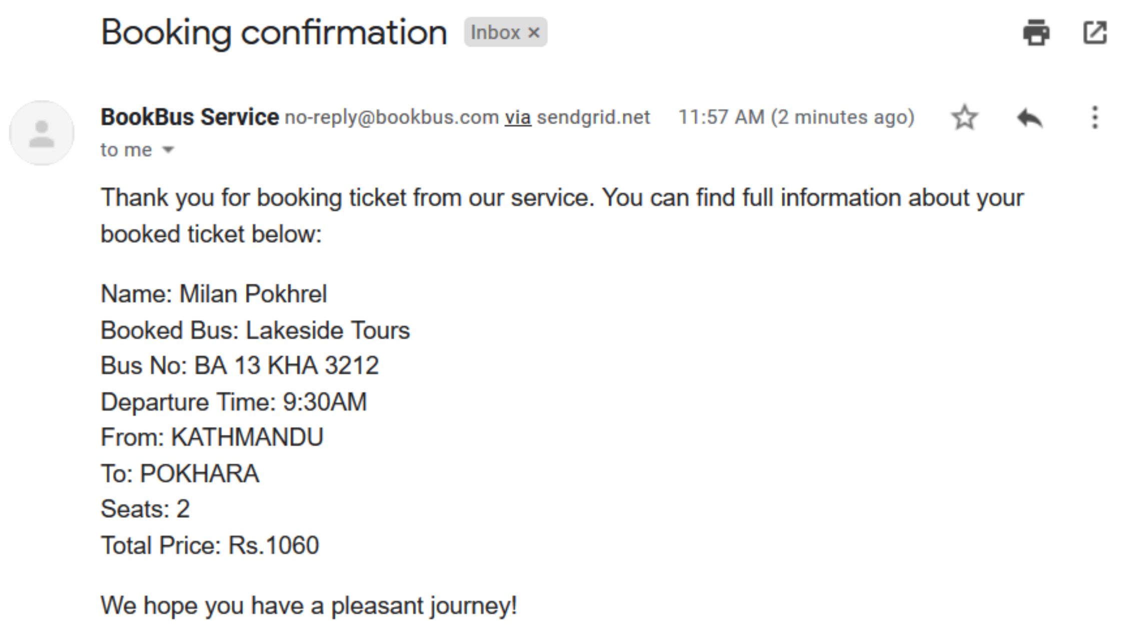


Figure 9. Application architecture overview of BookBus

As seen in Figure 9, the communication between the server and browser occurs through REST APIs. For the development of APIs, Node.js is used as a server-side runtime environment and Express.js as web framework maintaining a close relationship between the database and external service for mailing to users. The database used is MongoDB and to host the data, Mongo Atlas has been used as it is a cloud-based database service offered by MongoDB Inc so that there is no need to be concerned about database management. Mongoose acting as ODM connects the application with the database for querying, saving and retrieving the data about buses and users. For every route, user inputs have been validated and sanitized with the use of the *express-validator* library which minimizes the risk of possible attacks carried out to exploit the application. Once a user confirms the ticket booking, Nodemailer will send an email with full information regarding the booking to the user with the use of external mailing service, SendGrid. Figure 10 illustrates an email that the user receives after confirming the tickets.


Booking confirmation Inbox ×

BookBus Service no-reply@bookbus.com via sendgrid.net 11:57 AM (2 minutes ago)
to me

Thank you for booking ticket from our service. You can find full information about your booked ticket below:

Name: Milan Pokhrel
Booked Bus: Lakeside Tours
Bus No: BA 13 KHA 3212
Departure Time: 9:30AM
From: KATHMANDU
To: POKHARA
Seats: 2
Total Price: Rs.1060

We hope you have a pleasant journey!

Figure 10. Booking confirmation email received by the user

For the frontend segment, all the application state is managed by Redux in a store called redux store which enables using the same state in multiple React components. This streamlines the process of data sharing between components and during routing which is handled by react-router. Whenever a user performs certain tasks such as searching for buses, the action creator is called. Action creator is responsible for API handling from the frontend code as it makes a request to the server. Redux natively accepts actions as an object only and since API calls are asynchronous and have side effects, it is necessary to make redux handle them as well. Redux-thunk is used in BookBus application which allows redux to accepts functions as actions. The API calls are wrapped inside these functions and thus permitting for dispatching of appropriate actions based on the response from the server. As actions contain the payload of information and how it occurred and the only way to mutate state, these actions are then passed to the reducer which is a special function that handles how the application state will be modified. Once the reducer updates the state, the information is passed through the store and the updated information is reflected in the React components.

After the development of the web app, steps are followed to convert it into PWA. CRA has already provided some initial configuration for this purpose hence modifying these files allowed jumpstarting with transitioning to PWA. Changes were made to *manifest.json* and *index.html* file to make sure they hold the right definitions to reflect the app launched from home icon. As mentioned earlier, CRA uses Workbox to cache assets and make the web app work offline, no changes were made in the service worker file.

7 Result and Discussion

A MERN stack was used to develop a full-stack application with mobile-first design implementation. It was later converted into PWA with the implementation of service workers and manifest.json file to provide the offline capability and installation in the home screen. The application is still in the development phase but is hosted in Heroku for general access and testing purpose. Using Lighthouse to measure the performance and PWA status, as per Figure 11 BookBus scores exceptionally high and thus it can be imparted that the development of PWA was a success while being able to fully understand the working mechanism of PWA.

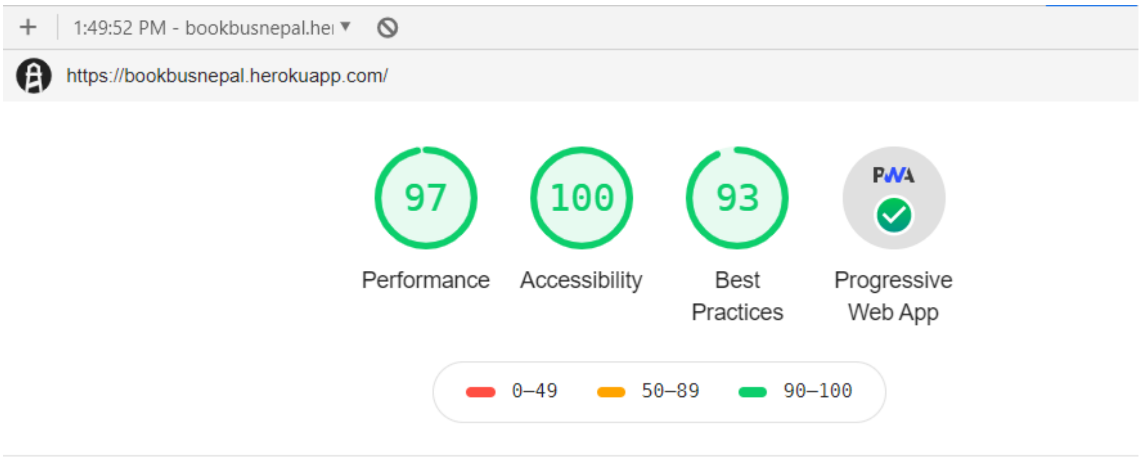


Figure 11. Lighthouse audit result for BookBus (mobile version)

There is a lot of room for development and improvement of the application so that it can be launched as a full-fledged application. For starters, authentication can be implemented with enabling users to keep track of their booking for future use. Developing of an admin panel would allow the management of buses to be handled gracefully. A calendar can be added in the search section so that users can choose to book from different dates. Integration of a payment system would enable users to experience full online service of the application while the pdf format of the confirmed ticket would allow them to have a more secure sensation of buying the ticket online. A notification feature can be put into operation so the user will be notified about the travel. The UI can be further improved to support edge cases of the peculiar screen size of some mobile devices.

8 Conclusion

A detailed study of PWA was carried out analysing the positive aspects it brings to the web development and current limitations it endures. An application ‘BookBus’ was developed to explore the ideas of PWA and how its features can be integrated into a normal web app. Developing a PWA is an interesting approach to developing web apps. With browsers pushing forward to support PWA functionalities, it can be said that the future of PWA looks bright enough. All major app stores now support the submission of PWA in their platform so that users can download them as native apps. This demonstrates the growth of PWA and the opportunity it holds. Google, Microsoft and many others are embracing PWAs to build their core applications as well. It will not be wrong to say that PWA will revolutionize web development aspect and put it head to head with native applications.

References

1 Internet History One-Page Summary - How Invented, Created [Internet]. Livinginternet.com. [cited 10 August 2019]. Available from: https://www.livinginternet.com/i/ii_summary.htm

2 Apple Reinvents the Phone with iPhone [Internet]. Apple Newsroom. January 2007 [cited 19 August 2019]. Available from: https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/

3 Share of mobile internet traffic in global regions 2019 [Internet]. Statista. 2019 [cited 12 October 2019]. Available from: https://www.statista.com/statistics/306528/share-of-mobile-internet-traffic-in-global-regions/

4 1 Billion More Phones Than People in The World! [Internet]. BankMyCell. 2019 [cited 10 October 2019]. Available from: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world

5 Kho Nancy D. Everything you need to know about Progressive Web Apps. EContent; Wilton Spring 2018;41(2):20-24.

6 Marcotte E. Responsive Web Design [Internet]. A List Apart. 2010 [cited 12 September 2019]. Available from: https://alistapart.com/article/responsive-web-design/

7 Gustafson A. Understanding Progressive Enhancement [Internet]. A List Apart. 2008 [cited 12 September 2019]. Available from: https://alistapart.com/article/understandingprogressiveenhancement/

8 Eric Schmidt at Mobile World Congress [Internet]. YouTube. 2010 [cited 13 September 2019]. Available from: https://www.youtube.com/watch?v=ClkQA2Lb_iE

9 Clement J. App stores: number of apps in leading app stores 2019 [Internet]. Statista. 2019 [cited 15 October 2019]. Available from: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

10 Smartphone Addiction & Cell Phone Usage Statistics in 2019 [Internet]. BankMyCell. 2019 [cited 16 October 2019]. Available from: https://www.bankmycell.com/blog/smartphone-addiction/

11 Chaffey D. Mobile marketing statistics compilation [Internet]. Smart Insights. 2018 [cited 3 October 2019]. Available from: https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

12 Richter F. Infographic: App Users Spend 77% of Their Time on Their Top 3 Apps [Internet]. Statista Infographics. 2017 [cited 18 September 2019]. Available from: https://www.statista.com/chart/3835/top-10-app-usage/

13 Ater T. Building Progressive Web Apps. 1st ed. California: O'Reilly Media, Inc.; 2017.

14 Hume D. Progressive Web Apps. 1st ed. New York: Manning Publications; 2017.

15 Domes S. Progressive Web Apps with React. 1st ed. Birmingham: Packt Publishing Ltd.; 2017.

16 Wagner, J. Why Performance Matters. [Internet] Google Developers. [cited 7 October 2019] Available at: https://developers.google.com/web/fundamentals/performance/why-performance-matters/

17 Progressive Web App Checklist [Internet]. Google Developers. [cited 15 October 2019]. Available from: https://developers.google.com/web/progressive-web-apps/checklist

18 Lighthouse [Internet]. Google Developers. [cited 16 October 2019]. Available from: https://developers.google.com/web/tools/lighthouse/

19 2017 U.S. Mobile App Report [Internet]. Slideshare.net. 2017 [cited 18 October 2019]. Available from: https://www.slideshare.net/comScoremarcom/2017-us-mobile-app-report

20 Jumia sees 33% increase in conversion rate, 12X more users on PWA [Internet]. Google Developers. 2017 [cited 20 October 2019]. Available from: https://developers.google.com/web/showcase/2017/jumia

21 Lancôme rebuilds their mobile website as a PWA, increases conversions 17% [Internet]. Google Developers. 2017 [cited 20 October 2019]. Available from: https://developers.google.com/web/showcase/2017/lancome

22 Monthly app downloads of U.S. smartphone users 2017 [Internet]. Statista. 2017 [cited 20 October 2019]. Available from: https://www.statista.com/statistics/325926/monthly-app-downloads-of-us-smartphone-users/

23 Ola drives mobility for a billion Indians with Progressive Web App [Internet]. Google Developers. 2017 [cited 20 October 2019]. Available from: https://developers.google.com/web/showcase/2017/ola

24 George.com enhances the mobile customer experience with new Progressive Web App [Internet]. Google Developers. 2018 [cited 20 October 2019]. Available from: https://developers.google.com/web/showcase/2018/asda-george

25 Making PWAs work offline with Service workers [Internet]. MDN Web Docs. 2019 [cited 22 October 2019]. Available from: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Offline_Service_workers

26 Hajian M. Progressive Web Apps with Angular: Create Responsive, Fast and Reliable PWAs Using Angular. 1st ed. Oslo: Apress; 2019.

27 Gaunt M. Service Workers: An Introduction [Internet]. Google Developers. [cited 23 October 2019]. Available from: https://developers.google.com/web/fundamentals/primers/service-workers

28 Osmani A. The App Shell Model [Internet]. Google Developers. [cited 25 October 2019]. Available from: https://developers.google.com/web/fundamentals/architecture/app-shell

29 LePage P. Add to Home Screen [Internet]. Google Developers. [cited 25 October 2019]. Available from: https://developers.google.com/web/fundamentals/app-install-banners

30 Can I use... Support tables for HTML5, CSS3, etc [Internet]. Caniuse.com. 2019 [cited 5 November 2019]. Available from: https://www.caniuse.com/#search=service%20workers

31 Can I use... Support tables for HTML5, CSS3, etc [Internet]. Caniuse.com. 2019 [cited 5 November 2019]. Available from: https://www.caniuse.com/#search=manifest

32 React – A JavaScript library for building user interfaces [Internet]. Reactjs.org. [cited 6 November 2019]. Available from: https://reactjs.org/

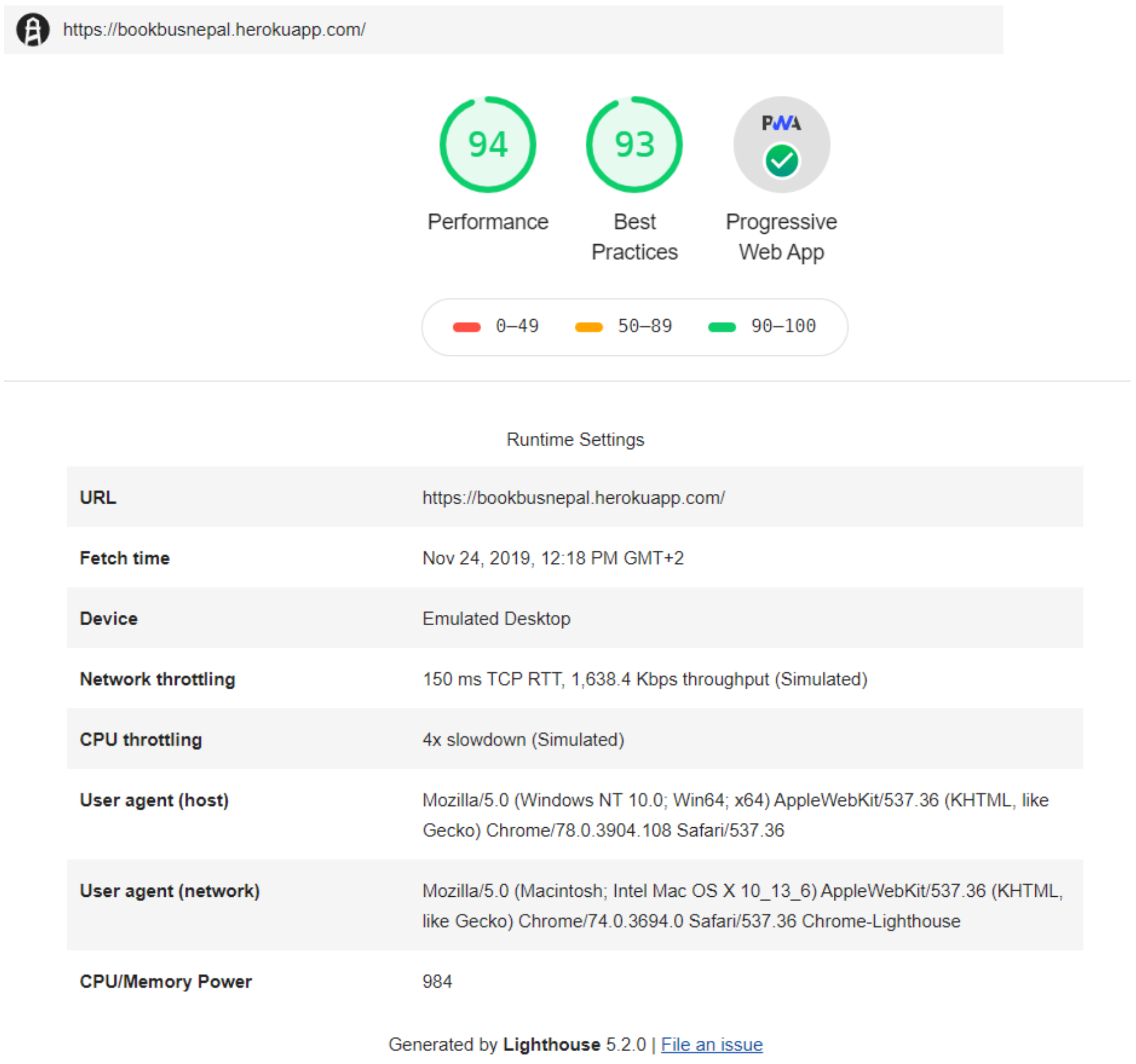
33 Introducing JSX – React [Internet]. Reactjs.org. [cited 6 November 2019]. Available from: https://reactjs.org/docs/introducing-jsx.html

34 Create React App [Internet]. Create React App. [cited 6 November 2019]. Available from: https://create-react-app.dev/

35 Introducing Hooks [Internet]. Reactjs.org. [cited 6 November 2019]. Available from: https://reactjs.org/docs/hooks-intro.html

36 Getting Started with Redux [Internet]. Redux.js.org. [cited 7 November 2019]. Available from: https://redux.js.org/introduction/getting-started

Appendix 1: Lighthouse report for BookBus application with emulated runtime settings (desktop version)

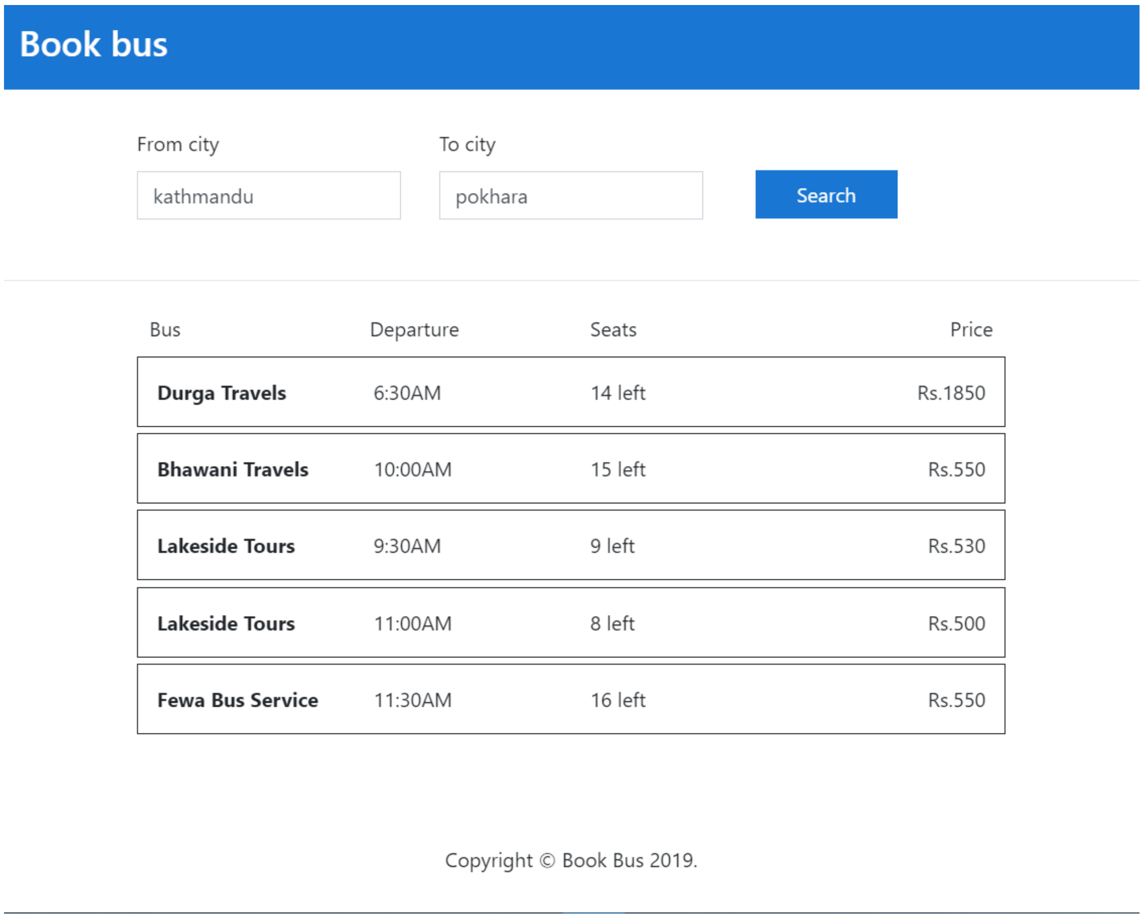


Runtime Settings

URL	https://bookbusnepal.herokuapp.com/
Fetch time	Nov 24, 2019, 12:18 PM GMT+2
Device	Emulated Desktop
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	984

Generated by **Lighthouse** 5.2.0 | File an issue

Appendix 2: Search Result Page for Desktop



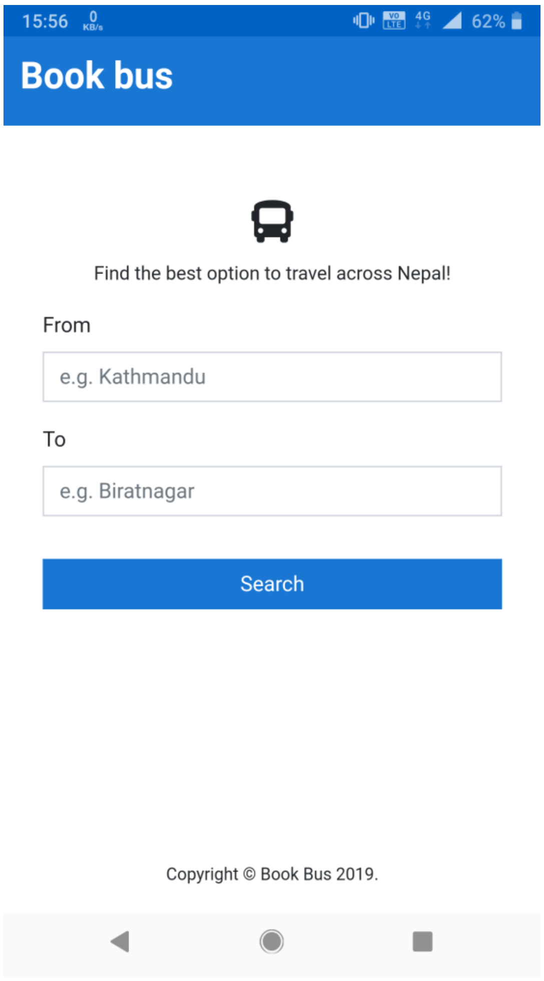
Book bus

From city	To city	
kathmandu	pokhara	Search

Bus	Departure	Seats	Price
Durga Travels	6:30AM	14 left	Rs.1850
Bhawani Travels	10:00AM	15 left	Rs.550
Lakeside Tours	9:30AM	9 left	Rs.530
Lakeside Tours	11:00AM	8 left	Rs.500
Fewa Bus Service	11:30AM	16 left	Rs.550

Copyright © Book Bus 2019.

Appendix 3: BookBus launched from the user’s home screen


15:56
62%
Book bus
Find the best option to travel across Nepal!
From
e.g. Kathmandu
To
e.g. Biratnagar
Search
Copyright © Book Bus 2019.