PIN POT

**(Recording and Reporting Incident on Damaged Road)**

**A Project Report**

*Submitted in partial fulfilment for the award of the degree*

*of*

**Master of Technology**

***in***

**Information Technology**

*by*

**NIKITA SAHOO**

**15MIN0356**

*Under the guidance of*

**Mr. Vinod S.E.**

**Assistant Professor**

****



**School of Information Technology and Engineering**

August, 2019

****

**School of Information Technology and Engineering**

**DECLARATION BY THE CANDIDATE**

I hereby declare that the thesis entitled **“PIN POT (RECORDING AND REPORTING INCIDENT ON DAMAGED ROAD)”** submitted by me to Vellore Institute of Technology, Vellore, in partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** is a record of bonafide project work carried out by me under the supervision of **Mr. Vinod S.E, Assistant Professor**. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

**Place**: Bangalore

**Date**: 17/08/2019 **Signature of the Candidate**

****

**School of Information Technology and Engineering**

**BONAFIDE CERTIFICATE**

This is to certify that the project work entitled “**PIN POT (RECORDING AND REPORTING INCIDENT ON DAMAGED ROAD)”** by **NIKITA SAHOO** (**15MIN0356),** to Vellore Institute of Technology, Vellore, in partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology**, is a project bonafide work carried out by him/her under my supervision. The project fulfills the requirement as per the regulations of this Institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this Institute or any other Institute or University.

**Mr.Vinod S.E.**

**Internal Supervisor**

**Assistant. Professor VIT University**

**Internal Examiner(s) External Examiner(s)**

**ABSTRACT**

As part of this project using the mobile app, one can identify and intelligently report incidents regarding damaged roads. As per the current situation, most of the cities in our country it’s evident that we do not have an intelligent system which can be used by any person to proactively report incident of road damage or a pothole when found.

Using this mobile app one can quickly register an incident about damaged road. It’s an easy and effective way to let authorities know where the incident is and how much is the impact. User can tag the incident location on the map along with the picture.

Benefits of Pin Pot Mobile app- Intelligent reporting of road damage incident. Quick and easy interface to capture impact of the damage. Precise details about the road damage/pot hole which can be easily accessed by the authorities. Platform to implement reporting and analytics based on the road damage data. Enables every individual to raise their voice against road quality.

**ACKNOWLEDGEMENT**

This major project would not have been possible without the assistance of many people to whom we are indebted. The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who had a major contribution to it. These are the people who made it possible with their constant guidance and encouragement. My utmost gratitude goes to **Mr. Vinod S.E (Assistant Professor)** for his constant guidance and support which helped this project come to fruition. I would also like to express my gratitude towards **Vellore Institute of Technology and Wipro Technologies** for providing the opportunity to study in the **WiSTA Program** and the encouragement and inspiration given throughout the course of study. I extend my deep sense of gratefulness to The Management and the Faculty members of Wipro WISTA program for their support. A special word of thanks to my classmates who were always there with their moral support. Finally, I would like to thank my family members and all my dear friends towards this endeavor, which in turn helped me accomplish this feat.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | iii |
|  | **ACKNOWLEDGEMENT** | iv |
|  | **LIST OF TABLES** | vii |
|  | **LIST OF FIGURES** | viii |
|  | **LIST OF ABBREVIATIONS** | ix |
|  |  |  |
| 1. | **INTRODUCTION** | 1 |
|  | 1.1PURPOSE | 1 |
| 2. | **LITERATURE SURVEY** | 2 |
| 3. | **PROPOSED SYSTEM** | 4 |
|  | 3.1 EXISTING SYSTEM | 4 |
|  | 3.2 PROPOSED SYSTEM | 4 |
| 4. | **DETAILED DESIGN OF THE PROJECT** | 5 |
|  | 4.1 SOFTWARE REQUIREMENT SPECIFICATION | 5 |
|  | 4.2 FUNCTIONAL REQUIREMENTS | 5 |
|  | 4.2.1 REPORTING A NEW INCIDENT | 5 |
|  | 4.2.2 ENHANCED SYSTEM FUNCTIONALITIES | 6 |
|  | 4.2.3 SHOWING A REPORTED INCIDENT | 6 |
|  | 4.2.4 AUTHENTICATION | 6 |
|  | 4.3 NON-FUNCTIONAL REQUIREMENTS | 7 |
|  | 4.3.1 PERFORMANCE | 7 |
|  | 4.3.2 MULTI-PLATFORM SUPPORT | 7 |
|  | 4.3.3 REUSABILITY | 7 |
|  | 4.3.4 AVAILABILITY | 7 |
|  | 4.4 METHODOLOGY | 8 |
|  | 4.4.1 AGILE METHODOLOGY | 8 |
|  | 4.4.2 SCRUM | 10 |
|  | 4.5 REQUIREMENTS MATRIX | 11 |
|  | 4.5.1 SOFTWARE REQUIREMENTS | 11 |
|  | 4.5.2 HARDWARE REQUIREMENTS | 11 |
|  | 4.6 PLATFORM SPECIFICATION | 12 |
|  | 4.7 DETAILED DESIGN OF THE PROJECT | 19 |
|  | 4.7.1 USER INTERFACE | 19 |
|  | 4.7.2 PRODUCT FUNCTIONALITY | 23 |
|  | 4.7.3 USE CASES | 24 |
|  | 4.7.4 SEQUENCE DIAGRAMS | 30 |
| 5. | **IMPLEMENTATION OF SYSTEM / METHODOLOGY** | 31 |
| 6. | **RESULTS AND DISCUSSION** | 67 |
|  | * 1. TEST PLAN | 69 |
|  | * 1. TEST DESIGN | 70 |
|  | * 1. TEST REPORT | 71 |
| 7. | **CONCLUSION AND FUTURE WORK** | 72 |
| 8. | **REFERENCES** | 74 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Table No** | **Title** | **Page No** |
| **1.0** | **List of current Technology used in Mobile App Development** | 18 |
| **4.7.3.1** | **UC\_Firsttime\_UserLogin** | 24 |
| **4.7.3.2** | **UC\_UserLogin** | 25 |
| **4.7.3.3** | **UC\_Record\_incident** | 26 |
| **4.7.3.4** | **UC\_Viewincidents\_byList** | 27 |
| **4.7.3.5** | **UC\_Viewincidents\_byMap** | 28 |
| **4.7.3.6** | **UC\_View Profile** | 28 |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No** | **Title** | **Page No** |
| 1 | Agile methodology | 9 |
| 2 | Native mobile application development features | 13 |
| 3 | HTML5 mobile application development features | 14 |
| 4 | Hybrid mobile application development features | 17 |
| 5 | Screen1 Map view | 19 |
| 6 | List of Incidents | 20 |
| 7 | New incident capture image | 20 |
| 8 | New incident address input | 21 |
| 9 | New incident address input 2 | 21 |
| 10 | Check the severity | 21 |
| 11 | Incident review all | 21 |
| 12 | Incident for Cracking | 22 |
| 13 | Incident for Pothole | 22 |
| 14 | Incident for Breakage | 22 |
| 15 | User Profile | 22 |
| 16 | User Flow Diagram | 23 |
| 17 | Use case Diagram | 29 |
| 18 | Sequence Diagram-Authentication | 30 |
| 19 | Sequence Diagram – Report an incident | 30 |

**LIST OF Abbreviations**

|  |  |
| --- | --- |
| **ACRONYM** | **EXPANSION** |
| **DFD** | Data Flow Diagram |
| **CFD** | Control Flow Diagrams |
| **UI** | User Interface |
| **API** | Application Program Interface |
| **REST** | Representational State Transfer |
| **iOS** | iPhone Operating System |
| **IDE** | Integrated Development Environment |

1. **INTRODUCTION**

This document is designed to be a reference for any person wishing to implement or any person interested in the architecture of the **PIN POT App** application. This document describes overview of software requirements; each, application’s architecture and sub-architecture along with their associated interfaces, database schemas, and the motivations behind the chosen design. Both high-level and low-level designs are included in this document.

This document should be read by an individual with a technical background and has experience reading data flow diagrams (DFDs), control flow diagrams (CFDs), interface designs, and development experience in object-oriented programming, testing and event driven programming.

This design document has an accompanying specification document and design document.

**1.1PURPOSE**

A software requirement and design is a written description of a software product, that a software designer writes to give a software development team overall guidance to the architecture of the software project, development procedure and proof. Most of the incident are not reported by the people because there is not proper communication channel between the people and the department. Whoever are putting a complain about the rough path all are based on paper work. Therefore, this problem can be solved if everything can be computerized to help the users to raise a concern to the department and it is helpful for the organisation as well to keep track of the road problem in the affected area.

1. **LITERATURE SURVEY**

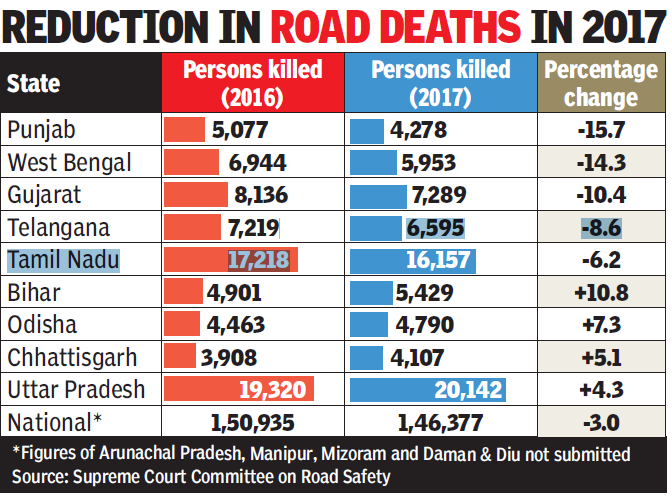
As per the current situation, most of the cities in our country it’s evident that we do not have an intelligent system which can be used by any person to proactively report incident of road damage or a pothole when found. PIN POT application can help reduce the road accidents by allowing the users to report the road damage and various locations.

The World Health Organisation (WHO) has conducted a survey on road accidents around the world. As per the survey, almost 1.35 million (13.5 lakh) die each year due to road casualties and a good majority of them are aged between five and 29 years. What is even more concerning is that India accounts for around 1.5 lakh road deaths every year.

There is a serious need to reduce the growing rate of accidents in India, as well as the world. Road traffic injuries continue to be one of the leading causes of death, disabilities and hospitalization in the country despite our commitment and efforts to bring down the number of road accidents and facilities.

The report for the calendar year 2017 reveals that there were 4,64,910 unfortunate incidences of road accidents during 2017 which claimed 1,47,913 lives and caused injuries to 4,70,975 persons.

The year 2017 shows some improvement over 2016. The number of road accidents was down by 3.3% and the number of injuries by 4.8% in 2017 as compared to 2016. The number of deaths on account of road accidents in 2017 has also been less as compared to the previous year by 1.9%.



By this PIN POT application, we can help reduce the percentage of accidents and can see our roads in a well condition and stop road accidents due to damaged roads.

**3.PROPOSED SYSTEM:**

Before we go to the proposed system and its importance in details, we must understand why the existing system is not cost effective and feasible to work with.

**3.1EXISTING SYSTEM**

Existing system doesn’t provide a user to raise any concern easily to the road authority. Or there is no such application to raise the road damage concern which is leading to many road accident, hence we should focus on the computerized system to update about the damaged path.

**3.2 PROPOSED SYSTEM**

The Pin Pot App is user-friendly application. This automated system makes all functionality easier for the user. It is very simple to use. The system requirements are very low.

The system proposed has many advantages.

1. **User-Friendly**: It will have user-friendly interface. Thus the users will feel very easy to work on it. The software provides accuracy along with a pleasant interface. Make the present manual system more interactive, speedy and user friendly.
2. **Security**: The proposed system is highly secured, because for login the system it requires the username and password which is different for each person which providing department a different view of the customer information.
3. **Report maintainability**: It maintains all the raised incident by the user. Manages users information separately for all raised complain in different location.
4. **Storage**: Stores information about user and locality.
5. **Accuracy**: It provides the user a quick response to select the problem and allow the user to upload picture and describe the problem.

6. **Maintenance Effective**: Reduces the cost of maintenance.

**4.DETAILED DESIGN OF THE PROJECT:**

**4.1 SOFTWARE REQUIREMENT SPECIFICATION:**

The aim of this section is to gather and analyse and give an in-depth insight of the complete **PIN POT app** by defining the requirements in detail. Nevertheless, it also concentrates on the capabilities required by stakeholders and their needs while defining high-level product features.

As part of this project using the mobile app, one can identify and intelligently report incidents regarding damaged roads. As per the current situation, most of the cities in our country it’s evident that we do not have an intelligent system which can be used by any person to proactively report incident of road damage or a pothole when found.

**4.2 FUNCTIONAL REQUIREMENTS:**

This subsection contains the requirements for the e-store. These requirements are organized by the features. Features from vision documents are then refined into use case diagrams and to sequence diagram to best capture the functional requirements of the system. All these functional requirements can be traced using tractability matrix.



**4.2.1.Reporting a new incident**

The system should allow users to report a road damage incident.

The system should allow users to report an incident along with the exact location of road damage.

The system should allow users to provide information severity of incident.

The system should allow users to capture photo of the incident.

The system should allow the users to submit an incident along with personal details.

**4.2.2.Enhanced System Functionalities**

The system should track current location of device.

The system should update the geo location as per current status.

System should fetch the current location offers and send communicational notification to user.

System should handle all user data in single user interface.

System should have capability keep the data submitted by users in a local capacity.

System should able to interact with the backend servers to save the incident details.

**4.2.3.Showing a reported incident**

The system should be able to show reported incident on a map view.

The system should be able to show the incident reported in a list view.

**4.2.4.Authentication**

System should be able to authenticate users before using the app.

System should allow user to login using Facebook or Google.

Once authenticated for the first-time system should be able to remember the user until the user logs out.

System should allow users to logout from the app.

* 1. **NON-FUNCTIONAL REQUIREMENTS:**

**4.3.1. Performance**

The mobile application will be on a hybrid mobile development platform and will use RESTful services to talk to backend servers and data

The system must be interactive and the delays involved must be less. So, in every action-response of the system, there are no immediate delays.

To decrease the delay time in interactions the system uses local or offline storage functionalities.

While reporting an incident until the incident it submitted the system will store all the data locally. Once the report is submitted and once connection is available to server the system should update the new incident information.

**4.3.2.Multi-platform support**

The system is developed in a “write once and run anywhere” fashion. Hence the app can be used in any mobile platforms including iOS, android

**4.3.3.Reusability**

This project should be reusable with modifications if functionality changes. This product should have the capability of being used in government organizations or any other public service authorities

**4.3.4.Availability**

The system can be available on app store for public use on all platforms

* 1. **METHODOLOGY:**

To achieve goals and planned results within a defined schedule and a budget, a manager uses a project. Regardless of which field or which trade, there are assortments of methodologies to help managers at every stage of a project from the initiation to implementation to the closure. In this document, we will try to discuss the most commonly used project management methodologies.

A methodology is a model, which project managers employ for the design, planning, implementation and achievement of their project objectives. There are different project management methodologies to benefit different projects.

In this project I have followed Agile Methodology.

**4.4.1. Agile Methodology**

Agile methodology emphasizes a more bottom-up, collaborative process which includes reactive planning, and working in short, time-boxed increments. The flexibility of agile project management makes it a good fit for product-focused teams whose work is constantly subject to changing customer requirements, such as software and technology development. Roughly 76 percent of developers are using or plan to use an “Agile” project management tool. In the past several years, other project-focused industries have also taken an interest in Agile planning, from construction to human resources to marketing.

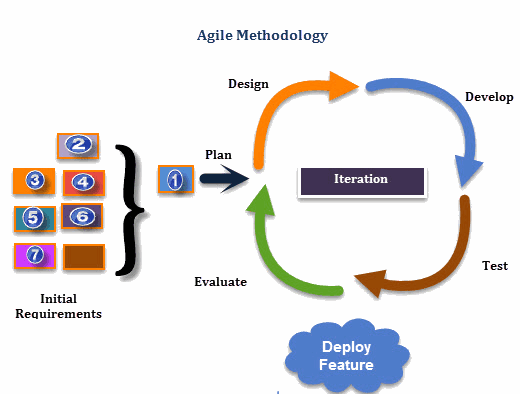


Figure 1 : Agile methodology

Agile software development methodology is for a project that needs extreme agility in requirements. The key features of agile are its short-termed delivery cycles (sprints), agile requirements, dynamic team culture, less restrictive project control and emphasis on real-time communication.

**4.4.2. Scrum**

This is an agile methodology. The main goal of this methodology is to improve team productivity dramatically by removing every possible burden. Scrum projects are managed by a Scrum master.

SCRUM and Agile Methodologies provide the developer with a methodology to produce and show results faster with clients. SCRUM and Agile methodologies provide a way to incorporate new functionality quickly into the product.

**Product Backlog:**The Product Backlog provides a list of the system requirements. It may be prioritized, to show which requirements take presence over others. Normally a Product Manager works with the business owner to prioritize and record functionality features which will be built into the product at a later date.

**Sprint Planning Meetings:**Sprints are planned with the development team. Sprints are often 1 - 4 weeks in length, and have a set number of functionality to be built into that Sprint. Sprints often incorporate development and testing. Multiple sprints (2 - 3 Sprints) will make up 1 release of a shippable product.

**Sprint Backlog / Front Burner:**The front burner often describes the items which are planned for development during the course of a sprint. These are the User Stories which are broken down into Sub Tasks and assigned to individuals of the team who then in turn own that story / functionality.

**Daily SCRUM Meetings:** Daily SCRUM calls are facilitated among the team. 15 minutes is allotted for a quick stand up, asking:

* What have you accomplished?
* What are you working on?
* What do you plan to work?
* What are your blockers?"

**Review:** Sprint reviews are done following each Sprint. The Agile team comes together and determines what went right, wrong, and how to improve their processes going forward.

* 1. **REQUIREMENTS MATRIX:**

**4.5.1 SOFTWARE REQUIREMENTS**

1.Operating system : Windows 7 or Higher

2. Web browser : Internet explorer or Google Chrome

3. Markup Language : HTML, CSS

4. Scripting Language : Angular, JavaScript

5. Framework: Ionic/Cordova

**4.5.2 HARDWARE REQUIREMENTS**

1. Processor : Intel(R) Core (TM) 13 CPU

2. Processor speed : 2.40GHz

3. RAM size : 512MB or Higher

4. Hard disk size : 500 GB or higher

* 1. **PLATFORM SPECIFICATION:**

**4.6.1. NATIVE APPLICATIONS**

In a nutshell, native apps provide the best usability, the best features, and the best overall mobile experience. There are some things you only get with native apps:

* **Multi touch** - double taps, pinch-spread, and other compound UI gestures
* **Fast graphics API** - the native platform gives you the fastest graphics, which may not be a big deal if you’re showing a static screen with only a few elements, or a very big deal if you’re using a lot of data and require a fast refresh.
* **Fluid animation** - related to the fast graphics API is the ability to have fluid animation. This is especially important in gaming, highly interactive reporting, or intensely computational algorithms for transforming photos and sounds.
* **Built-in components** - The camera, address book, geolocation, and other features native to the device can be seamlessly integrated into mobile apps. Another important built-in components is encrypted storage, but more about that later.
* **Ease of use** - The native platform is what people are accustomed to, and so when you add that familiarity with all of the native features they expect, you have an app that’s just plain easier to use.
* **Documentation** - There are over 2500 books alone for iOS and Android development, with many more articles, blog posts, and detailed technical threads on sites like StackOverflow.



Figure 2 : Native mobile application development features

Native apps are usually developed using an integrated development environment (IDE). IDEs provide tools for building debugging, project management, version control, and other tools professional developers need. While iOS and Android apps are developed using different IDEs and languages, there’s a lot of parity in the development environments, and there’s not much reason to delve into the differences. Simply put, you use the tools required by the device.

You need these tools because native apps are more difficult to develop. Likewise, the level of experience required is higher than other development scenarios, you don’t just cut and paste Objective-C and expect it to work. Indeed, the technological know-how of your development team is an important consideration. If you’re a professional developer, you don’t have to be sold on proven APIs and frameworks, painless special effects through established components, or the benefits of having your code all in one place. Let’s face it, today a skilled native iOS or Android developer is a rock star, and can make rock star demands.

While we’ve touched on native apps from a development perspective, there’s also the more important perspective: the end user. When you’re looking for an app, you’ll find it in the store. When you start the app, it fires up immediately. When you use the app, you get fast performance, consistent platform look and feel. When your app needs an update, it tells you so. Native apps give you everything you’d expect from the company that built your device, as if it were simply meant to be.

**4.6.2.HTML 5 MOBILE APPLICATIONS**

If you’re new to mobile app development, you’re late to the party. However, for mobile Web-based apps, we’re still partying like it’s 1999! Sure, browsers have gotten better in the past umpteen years, but the underlying technology isn’t that much different than when you feared the Y2K bug.

But that can be a good thing. An HTML5 mobile app is basically a web page, or series of web pages, that are designed to work on a tiny screen. As such, HTML5 apps are device agnostic and can be opened with any modern mobile browser. And because your content is on the web, it's searchable, which can be a huge benefit depending on the app (shopping, for example).



Figure 3 : HTML5 mobile application development features

If you have experience developing Web apps, you'll take to HTML5 like a duck to water. If you're new to Web development, the technological bar is lower; it's easier to get started here than in native or hybrid development. Unfortunately, every mobile device seems to have their own idea of what constitutes usable screen size and resolution, and so there's an additional burden of testing on different devices. Browser incompatibility is especially rife on Android devices, so browser beware.

An important part of the "write-once-run-anywhere" HTML5 methodology is that distribution and support is much easier than for native apps. Need to make a bug fix or add features? Done and deployed for all users. For a native app, there are longer development and testing cycles, after which the consumer typically must log into a store and download a new version to get the latest fix.

In the last year, HTML5 has emerged as a very popular way for building mobile applications. Multiple UI frameworks are available for solving some of the most complex problems that no developer wants to reinvent. iScroll does a phenomenal job of emulating momentum style scrolling. JQuery Mobile and Sencha Touch provide elegant mobile components, with hundreds if not thousands of plugins that offer everything from carousels to super elaborate controls.

So if HTML5 apps are easier to develop, easier to support, and can reach the widest range of devices, where do these apps lose out? We already reviewed the major benefits of native development, so we'll just reiterate that you can't access native features on the device. Users won’t have the familiarity of the native look and feel, or be able to use compound gestures they are familiar with. But strides are being made on all fronts, and more and more functionality is supported by browsers all the time.

The latest batch of browsers support hardware accelerated CSS3 animation properties, providing smooth motion for sliding panels as well transitions between screens, but even that can’t match the power and flexibility of native apps. Today, it’s simply not possible to capture multi-touch input events (determining when more than one finger is on the screen) or create path-style elegance with spinout buttons and photos that hover, then drop into the right place.

However, significant limitations, especially for enterprise mobile, are offline storage and security. While you can implement a semblance of offline capability by caching files on the device, it just isn't a very good solution. Although the underlying database might be encrypted, it’s not as well segmented as a native keychain encryption that protects each app with a developer certificate. Also, if a web app with authentication is launched from the desktop, it will require users to enter their credentials every time the app it is sent to the background. This is a lousy experience for the user. In general, implementing even trivial security measures on a native platform can be complex tasks for a mobile Web developer. Therefore, if security is of the utmost importance, it can be the deciding factor on which mobile technology you choose.

**4.6.3. HYBRID MOBILE APPLICATION**

Hybrid development combines the best (or worst) of both the native and HTML5 worlds. We define hybrid as a web app, primarily built using HTML5 and JavaScript, that is then wrapped inside a thin native container that provides access to native platform features. PhoneGap is an example of the most popular container for creating hybrid mobile apps.

For the most part, hybrid apps provide the best of both worlds. Existing web developers that have become gurus at optimizing JavaScript, pushing CSS to create beautiful layouts, and writing compliant HTML code that works on any platform can now create sophisticated mobile applications that don’t sacrifice the cool native capabilities. In certain circumstances, native developers can write plugins for tasks like image processing, but in cases like this, the devil is in the details.

On iOS, the embedded web browser or the UIWebView is not identical to the Safari browser. While the differences are minor, they can cause debugging headaches. That’s why it pays off to invest in popular frameworks that have addressed all of the limitations.

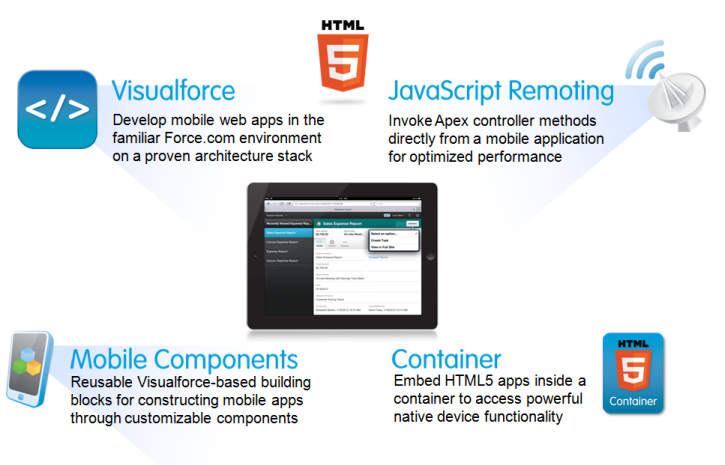


Figure 4 : Hybrid mobile application development features

You know that native apps are installed on the device, while HTML5 apps reside on a Web server, so you might be wondering if hybrid apps store their files on the device or on a server? Yes. In fact there are two ways to implement a hybrid app.

* **Local** - You can package HTML and JavaScript code inside the mobile application binary, in a manner similar to the structure of a native application. In this scenario you use REST APIs to move data back and forth between the device and the cloud.
* **Server** - Alternatively you can implement the full web application from the server (with optional caching for better performance), simply using the container as a thin shell over the UI Webview.

Netflix has a really cool app that uses the same code base for running the UI on all devices: tablets, phones, smart TVs, DVD players, refrigerators, and cars. While most people have no idea, nor care, how the app is implemented, you’ll be interested to know they can change the interface on the fly or conduct A/B testing to determine the optimal user interactions. The guts of decoding and streaming videos are delegated to the native layer for best performance, so it’s a fast, seemingly native app, that really does provide the best of both worlds.

**Conclusion**

Mobile development is a constantly moving target. Every six months, there’s a new mobile operating system, with unique features only accessible with native APIs. The containers bring those to hybrid apps soon thereafter, with the web making tremendous leaps every few years. Let's sum those up in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Native** | **HTML5** | **Hybrid** |
| **App Features** |  |  |  |
| Graphics | Native APIs | HTML, Canvas, SVG | HTML, Canvas, SVG |
| Performance | Fast | Slow | Slow |
| Native look and feel | Native | Emulated | Emulated |
| Distribution | Appstore | Web | Appstore |
| **Device Access** |  |  |  |
| Camera | Yes | No | Yes |
| Notifications | Yes | No | Yes |
| Contacts, calendar | Yes | No | Yes |
| Offline storage | Secure file storage | Shared SQL | Secure file system, shared SQL |
| Geolocation | Yes | Yes | Yes |
| **Gestures** |  |  |  |
| Swipe | Yes | Yes | Yes |
| Pinch, spread | Yes | No | Yes |
| **Connectivity** | Online and offline | Mostly online | Online and offline |
| **Development skills** | ObjectiveC, Java | HTML5, CSS, Javascript | HTML5, CSS, JavaScript |

* 1. **DETAILED DESIGN OF THE PROJECT:**

**4.7.1.USER INTERFACE**

The application will have a state-of-the-art mobile user interface which will focus mainly on ease to use and clean visual design.

The user interface will have 3 main menu options respectively as mentioned below

Map view – to show reported incidents on a map

List view - to show reported incidents on a list

User profile – to let users to manage their profile

Along with the main menu the system will have a dedicated button to start reporting



Figure 5: Screen1 Map view

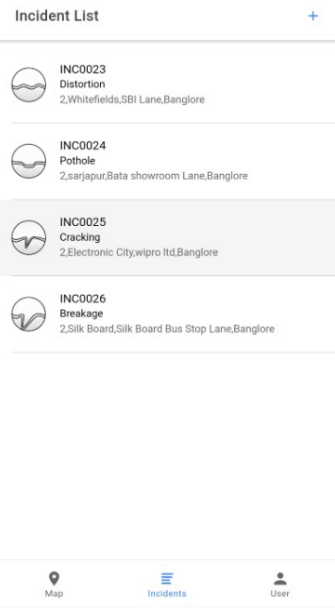
 

Figure 6 : List of Incidents Figure 7: New incident capture image

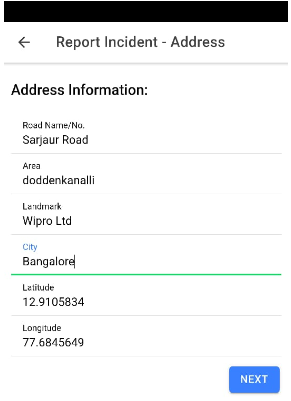
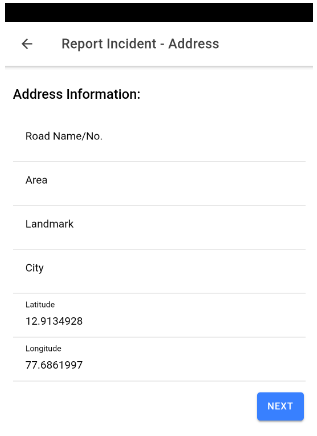


Figure 8: New incident address input Figure 9: New incident address input 2

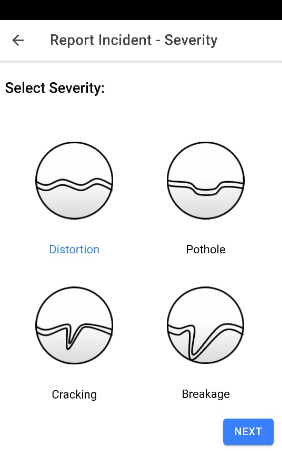
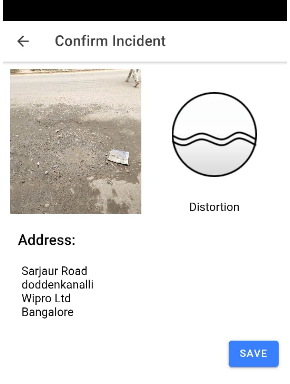
 

Figure 10: Check the severity Figure 11: Incident review all

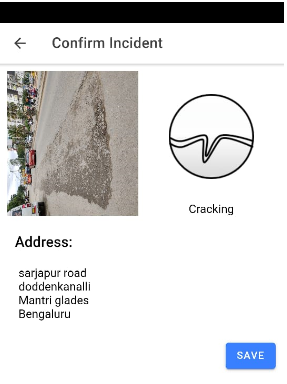
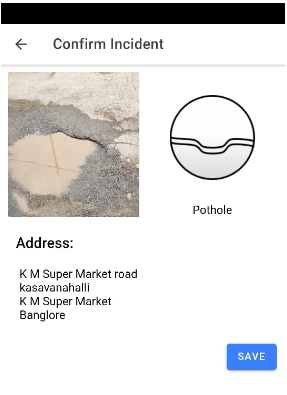
 

Figure 12: Incident for Cracking Figure 13: Incident for Pothole

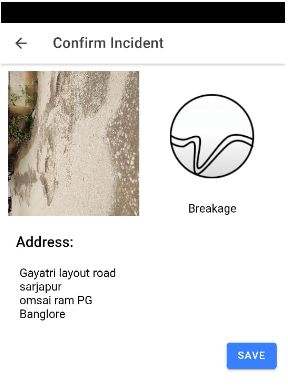
 

Figure 14: Incident for Breakage Figure 15: User Profile

**4.7.2. PRODUCT FUNCTIONALITY**

This user flow chart shows product functionality and how the user interacts with the product

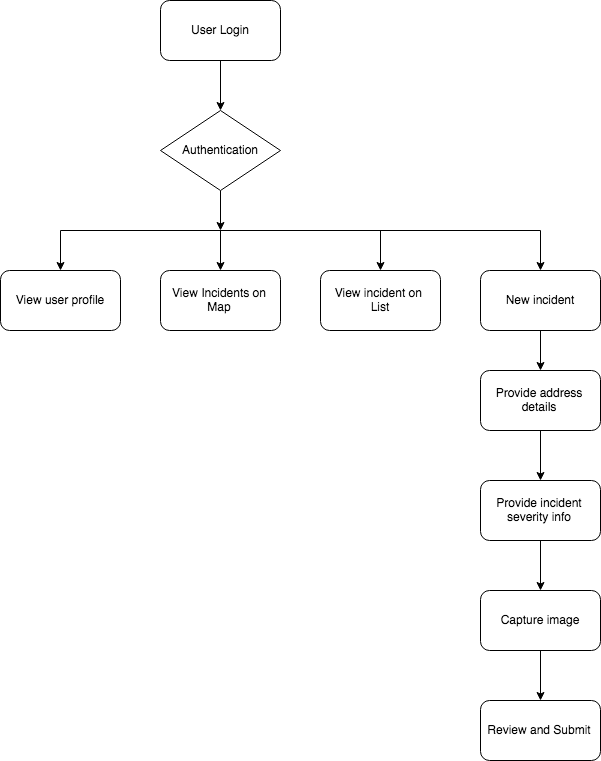


Figure 16 : User Flow Diagram

* + 1. **USE CASES**

**4.7.3.1. UC\_Firsttime\_UserLogin**

|  |  |
| --- | --- |
| **Name** | **UC\_Firsttime\_UserLogin** |
| **Input** | **User will lunch the application** |
| **Output** | **User profile gets fetched** |
| **Pre-condition** | **App must be installed in user’s device.** |
| **Steps** | **User will launch the app** |
| **Post-condition** | **User logs into the app and he will be redirected to landing screen** |
| **Exception** | 1. **App will show exception in case of no user profile found** 2. **Exception will be thrown in case of no internet** |

**4.7.3.2. UC\_UserLogin**

|  |  |
| --- | --- |
| **Name** | **UC\_UserLogin** |
| **Goal** | **To skip the authentication if already linked with Google or Facebook accounts** |
| **Input** | **User account details fetched from local storage of the app** |
| **Output** | **User profile gets fetched** |
| **Pre-condition** | **App must be installed in user’s device.** |
| **Steps** | 1. **App will check for existing user profiles in local storage** 2. **If user profile already exists and user is not doing login for the first time, user will be redirected to Landing screen.** |
| **Post-condition** | **User logs into the app and he will be redirected to landing screen** |
| **Exception** | 1. **Exception will be thrown in case of no internet** |

**4.7.3.3. UC\_Record\_incident**

|  |  |
| --- | --- |
| **Name** | **UC\_Record\_incident** |
| **Goal** | **To capture the road damage incident picture and start recording an incident.** |
| **Input** | **User takes picture from mobile camera provided in the app landing page.** |
| **Output** | **The picture clicked by user is saved temporarily** |
| **Pre-condition** | **The application must have desired permission to use camera hardware in mobile phone.** |
| **Steps** | 1. **User captures the picture of the incident** 2. **The picture saved temporarily allowing user to delete or alter the taken picture** 3. **App will continue to ask next set of inputs from user as part of recording incident.** 4. **Application will allow user to provide location details and incident severity details** |
| **Post-condition** | **The incident along with the picture and severity details will be saved in backend** |
| **Exception** | **Exception will be thrown in case proper permissions are not provided to use the mobile camera feature.** |

**4.7.3.4. UC\_Viewincidents\_byList**

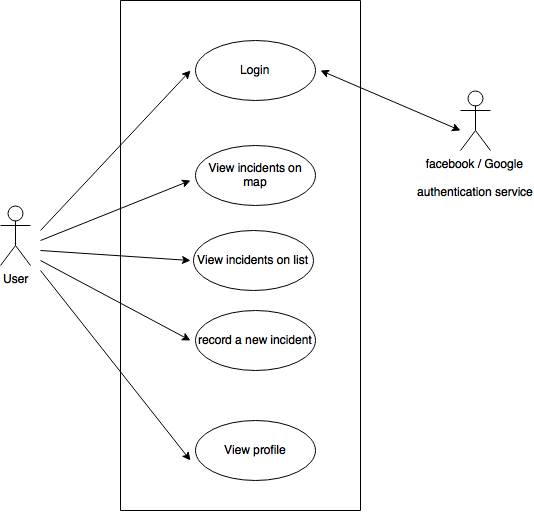
|  |  |
| --- | --- |
| **Name** | **UC\_Viewincidents\_byList** |
| **Goal** | **To show user list of incidents in current area** |
| **Input** | **Geolocation** |
| **Output** | **List of incidents reported in that location** |
| **Pre-condition** | **User must have logged in to app.** |
| **Steps** | **User selects the list view option in the main menu and the app will display all the list of incident in that area.** |
| **Post-condition** | **NA** |
| **Exception** | **Exception is thrown in case app is not able to load user profile** |

**4.7.3.5. UC\_Viewincidents\_byMap**

|  |  |
| --- | --- |
| **Name** | **UC\_Viewincidents\_byMap** |
| **Goal** | **To show user list of incidents in current area** |
| **Input** | **Geolocation** |
| **Output** | **List of incidents reported in that location** |
| **Pre-condition** | **User must have logged in to app.** |
| **Steps** | **User selects the list view option in the main menu and the app will display all the incidents on map** |
| **Post-condition** | **NA** |
| **Exception** | **Exception is thrown in case app is not able to load user profile** |

**4.7.3.6. UC\_View Profile**

|  |  |
| --- | --- |
| **Name** | **UC\_ViewProfile** |
| **Goal** | **To show user profile and no. of incidents reported by user** |
| **Input** | **Geolocation** |
| **Output** | **User profile details and no. of incidents reported by that user** |
| **Pre-condition** | **User must have logged in to app.** |
| **Steps** | **User selects profile option on the main menu and able to see his profile details along with the no. of incidents raised by user** |
| **Post-condition** | **NA** |
| **Exception** | **Exception is thrown in case app is not able to load user profile** |



` Figure 17: Use case diagram

* + 1. **SEQUENCE DIAGRAMS**

**4.7.4.1. Authentication**

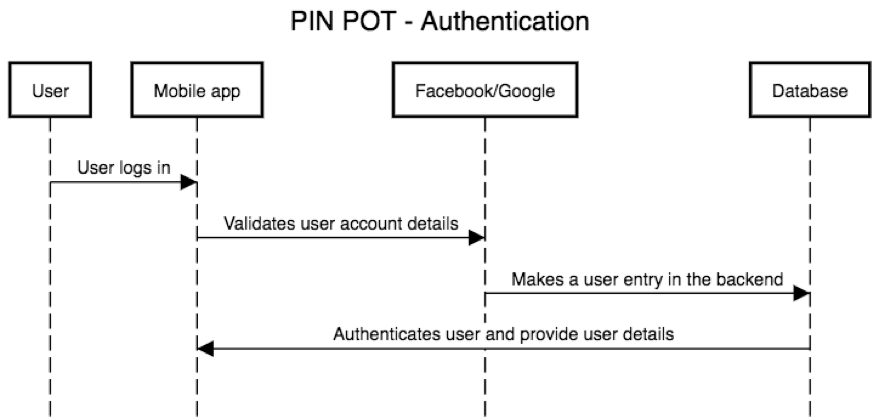


Figure 18: Sequence Diagram-Authentication

**4.7.4.2. Report an Incident**

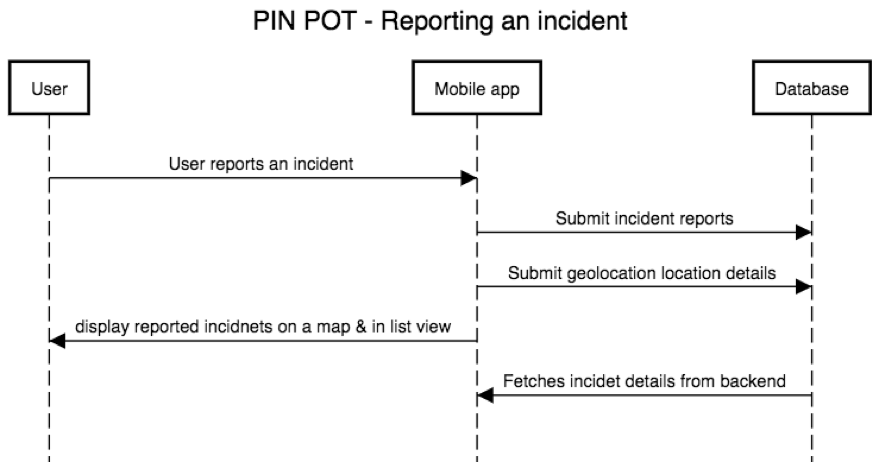


Figure 19: Sequence diagram – Report an incident

**5. IMPLEMENTATION OF** **SYSTEM:**

**HTML file:**

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<title>Ionic App</title>

<base href="/" />

<meta name="viewport" content="viewport-fit=cover, width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0, user-scalable=no" />

<meta name="format-detection" content="telephone=no" />

<meta name="msapplication-tap-highlight" content="no" />

<link rel="icon" type="image/png" href="assets/icon/favicon.png" />

<!-- add to homescreen for ios -->

<meta name="apple-mobile-web-app-capable" content="yes" />

<meta name="apple-mobile-web-app-status-bar-style" content="black" />

</head>

<body>

<app-root></app-root>

</body>

</html>

Home.html

<ion-tabs>

<ion-tab-bar slot="bottom">

<ion-tab-button tab="map">

<ion-icon name="pin"></ion-icon>

<ion-label>Map</ion-label>

</ion-tab-button>

<ion-tab-button tab="incidents">

<ion-icon name="list"></ion-icon>

<ion-label>Incidents</ion-label>

</ion-tab-button>

<ion-tab-button tab="user">

<ion-icon name="person"></ion-icon>

<ion-label>User</ion-label>

</ion-tab-button>

</ion-tab-bar>

</ion-tabs>

Incident.html

<ion-header>

<ion-toolbar>

<ion-title>Incident List</ion-title>

<ion-buttons slot="primary">

<ion-button color="primary" id="btn-add-incident" [routerLink]="['/new-incident']">

<ion-icon slot="start" name="add"></ion-icon>

</ion-button>

</ion-buttons>

</ion-toolbar>

</ion-header>

<ion-content>

<ion-list>

<ion-item \*ngFor="let incident of incidents" [routerLink]="['./',incident.id]">

<ion-avatar>

<img [src]="incident.severity.imageUrl"/>

</ion-avatar>

<ion-label margin>

<h2>{{incident.id}}</h2>

<h4>{{incident.severity.text}}</h4>

<p>{{

incident.address.road + "," +

incident.address.area + "," +

incident.address.landmark + "," +

incident.address.city

}}</p>

</ion-label>

</ion-item>

</ion-list>

</ion-content>

Incident Details.html

<ion-header>

<ion-toolbar>

<ion-buttons slot="start">

<ion-back-button></ion-back-button>

</ion-buttons>

<ion-title>Incident Details - {{incident.id}}</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<ion-grid>

<ion-row>

<ion-col>

<img style="width:200px;height:200px;" alt="No Image Available" [src]="incident.image"/>

</ion-col>

<ion-col text-center>

<ion-img padding margin [src]="incident.severity.imageUrl"></ion-img>

<ion-label>{{incident.severity.text}}</ion-label>

</ion-col>

</ion-row>

<h4 margin>Address:</h4>

<ion-row margin>

<ion-col>

<ion-text>

{{incident.address.road}}<br/>

{{incident.address.area}}<br/>

{{incident.address.landmark}}<br/>

{{incident.address.city}}

</ion-text>

</ion-col>

</ion-row>

<ion-row>

<ion-col>

<!-- <ion-button (click)="edit();"><ion-icon name="create" slot="start"></ion-icon> Edit</ion-button> -->

<ion-button (click)="delete();" color="danger"><ion-icon name="trash" slot="start"></ion-icon> Delete</ion-button>

</ion-col>

</ion-row>

</ion-grid>

</ion-content>

Map.HTML

<ion-header>

<ion-toolbar>

<ion-title>Map</ion-title>

<ion-buttons slot="primary">

<ion-button color="primary" id="btn-add-incident" [routerLink]="['/new-incident']">

<ion-icon slot="start" name="add"></ion-icon>

</ion-button>

</ion-buttons>

</ion-toolbar>

</ion-header>

<ion-content>

<div #mapElement id='mapElement' style='height:100%;'></div>

</ion-content>

New Incident.html

<ion-header>

<ion-toolbar>

<ion-buttons slot="start">

<ion-back-button></ion-back-button>

</ion-buttons>

<ion-title>Confirm Incident</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<ion-grid>

<ion-row>

<ion-col>

<img style="width:200px;height:200px;" alt="No Image Available" [src]="incident.image"/>

</ion-col>

<ion-col text-center>

<ion-img padding margin [src]="incident.severity.imageUrl"></ion-img>

<ion-label>{{incident.severity.text}}</ion-label>

</ion-col>

</ion-row>

<h4 margin>Address:</h4>

<ion-row margin>

<ion-col>

<ion-text>

{{incident.address.road}}<br/>

{{incident.address.area}}<br/>

{{incident.address.landmark}}<br/>

{{incident.address.city}}

</ion-text>

</ion-col>

</ion-row>

<ion-row>

<ion-col>

<ion-button class="ion-float-right" (click)="save();">Save</ion-button>

</ion-col>

</ion-row>

</ion-grid>

</ion-content>

Address.html

<ion-header>

<ion-toolbar>

<ion-buttons slot="start">

<ion-back-button></ion-back-button>

</ion-buttons>

<ion-title>Report Incident - Address</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<ion-grid>

<ion-row>

<ion-col>

<h4>Address Information: </h4>

</ion-col>

</ion-row>

<ion-row>

<ion-col>

<ion-item>

<ion-label position='floating'>Road Name/No.</ion-label>

<ion-input [(ngModel)]="address.road"></ion-input>

</ion-item>

<ion-item>

<ion-label position='floating'>Area</ion-label>

<ion-input [(ngModel)]="address.area"></ion-input>

</ion-item>

<ion-item>

<ion-label position='floating'>Landmark</ion-label>

<ion-input [(ngModel)]="address.landmark"></ion-input>

</ion-item>

<ion-item>

<ion-label position='floating'>City</ion-label>

<ion-input [(ngModel)]="address.city"></ion-input>

</ion-item>

<ion-item>

<ion-label position='floating'>Latitude</ion-label>

<ion-input [(ngModel)]="location.latitude"></ion-input>

</ion-item>

<ion-item>

<ion-label position='floating'>Longitude</ion-label>

<ion-input [(ngModel)]="location.longitude"></ion-input>

</ion-item>

</ion-col>

</ion-row>

<ion-row>

<ion-col>

<ion-button class="ion-float-right" (click)="next();">Next</ion-button>

</ion-col>

</ion-row>

</ion-grid>

</ion-content>

Camera.html

<ion-header>

<ion-toolbar>

<ion-buttons slot="start">

<ion-back-button></ion-back-button>

</ion-buttons>

<ion-title>Report Incident - Image</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<ion-grid>

<ion-row>

<ion-col>

<h4>Capture Image</h4>

</ion-col>

</ion-row>

<ion-row>

<ion-col>

<!--camera-->

<ion-img [src]="imageData" class="captured-image"></ion-img>

</ion-col>

</ion-row>

<ion-row>

<ion-col>

<ion-button class="ion-float-right" (click)="next();">Next</ion-button>

<ion-button class="ion-float-right" (click)="takePicture();" color="light">Retake</ion-button>

</ion-col>

</ion-row>

</ion-grid>

</ion-content>

Severity.html

<ion-header>

<ion-toolbar>

<ion-buttons slot="start">

<ion-back-button></ion-back-button>

</ion-buttons>

<ion-title>Report Incident - Severity</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<ion-grid>

<ion-row>

<ion-col>

<h4>Select Severity: </h4>

</ion-col>

</ion-row>

<ion-row>

<ion-col>

<ion-grid>

<ion-row \*ngFor="let severityGroup of groupedSeverities">

<ion-col \*ngFor="let severity of severityGroup.items" text-center (click)="selectedServerity=severity">

<ion-img padding margin [src]="severity.imageUrl"></ion-img>

<ion-label [ngClass]="{'active-severity':selectedServerity==severity}">{{severity.text}}</ion-label>

</ion-col>

</ion-row>

</ion-grid>

</ion-col>

</ion-row>

<ion-row>

<ion-col>

<ion-button class="ion-float-right" (click)="next();">Next</ion-button>

</ion-col>

</ion-row>

</ion-grid>

</ion-content

User.html

<ion-header>

<ion-toolbar>

<ion-title>User</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<ion-card>

<img [src]="user.thumbnail"/>

<ion-card-header>

<ion-card-title>{{user.name}}</ion-card-title>

<ion-card-subtitle>{{incidents.length}} incidents reported</ion-card-subtitle>

</ion-card-header>

</ion-card>

</ion-content>

Login.html

<ion-header>

<ion-toolbar>

<ion-title>Pinpot</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<form #form="ngForm" (ngSubmit)="login(form)">

<ion-grid>

<ion-row color="primary">

<ion-col>

<div text-center>

<h3>Login</h3>

</div>

<div padding>

<ion-item>

<ion-input [(ngModel)]="email" name="email" type="email" placeholder="your@email.com" required></ion-input>

</ion-item>

<ion-item>

<ion-input [(ngModel)]="password" name="password" type="password" placeholder="Password" required></ion-input>

</ion-item>

</div>

<div padding>

<ion-button type="submit" [disabled]="form.invalid" expand="block">Login</ion-button>

</div>

</ion-col>

</ion-row>

</ion-grid>

</form>

</ion-content>

Register.html

<ion-header>

<ion-toolbar>

<ion-title>Pinpot</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<form #form="ngForm" (ngSubmit)="register();">

<ion-grid>

<ion-row>

<ion-col>

<div text-center>

<h3>Register</h3>

</div>

<div padding>

<ion-item>

<ion-input [(ngModel)]="user.name" name="username" type="text" placeholder="Name" required></ion-input>

</ion-item>

<ion-item>

<ion-input [(ngModel)]="user.email" name="useremail" type="email" placeholder="your@email.com" required></ion-input>

</ion-item>

<ion-item>

<ion-input [(ngModel)]="user.password" name="password" type="password" placeholder="Password" required></ion-input>

</ion-item>

<ion-item>

<ion-input [(ngModel)]="confirm" name="confirm" type="password" placeholder="Password again" required></ion-input>

</ion-item>

<ion-radio-group name="gender" [(ngModel)]="user.gender">

<ion-list-header>

<ion-label>Gender</ion-label>

</ion-list-header>

<ion-item>

<ion-label>Male</ion-label>

<ion-radio slot="start" color="primary" value="M" checked></ion-radio>

</ion-item>

<ion-item>

<ion-label>Female</ion-label>

<ion-radio slot="start" color="primary" value="F"></ion-radio>

</ion-item>

</ion-radio-group>

</div>

<div padding>

<ion-button type="submit" [disabled]="form.invalid" expand="block">Register</ion-button>

</div>

</ion-col>

</ion-row>

</ion-grid>

</form>

</ion-content>

**TS Files:**

App.module.ts

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { RouteReuseStrategy } from '@angular/router';

import { IonicModule, IonicRouteStrategy } from '@ionic/angular';

import { SplashScreen } from '@ionic-native/splash-screen/ngx';

import { StatusBar } from '@ionic-native/status-bar/ngx';

import { Camera } from '@ionic-native/camera/ngx';

import { Network } from '@ionic-native/network/ngx';

import { Geolocation } from '@ionic-native/geolocation/ngx';

import { GoogleMaps } from '@ionic-native/google-maps/ngx';

import { AppComponent } from './app.component';

import { AppRoutingModule } from './app-routing.module';

@NgModule({

declarations: [AppComponent],

entryComponents: [],

imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule],

providers: [

StatusBar,

SplashScreen,

{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy },

Camera,

Network,

Geolocation,

GoogleMaps

],

bootstrap: [AppComponent]

})

export class AppModule {}

Home.ts

import { Component } from '@angular/core';

@Component({

selector: 'app-home',

templateUrl: 'home.page.html',

styleUrls: ['home.page.scss'],

})

export class HomePage {

subscription: any;

platform: any;

constructor() {}

ionViewDidEnter(){

this.subscription = this.platform.backButton.subscribe(()=>{

navigator['app'].exitApp();

});

}

ionViewWillLeave(){

this.subscription.unsubscribe();

}

}

Incident.ts

import { Component, OnInit } from '@angular/core';

import { IncidentService } from './incident.service';

import { Incident } from './incident.model';

@Component({

selector: 'app-incident',

templateUrl: './incident.page.html',

styleUrls: ['./incident.page.scss'],

})

export class IncidentPage implements OnInit {

incidents: Incident[];

constructor(private incidentService: IncidentService) { }

ngOnInit() {

this.incidentService.getAllIncidents().subscribe((incidents: Incident[]) => {

this.incidents = incidents;

});

}

}

Incident Details.ts

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute, Router } from '@angular/router';

import { Incident } from '../incident.model';

import { IncidentService } from '../incident.service';

import { AlertController, NavController } from '@ionic/angular';

@Component({

selector: 'app-incident-detail',

templateUrl: './incident-detail.page.html',

styleUrls: ['./incident-detail.page.scss'],

})

export class IncidentDetailPage implements OnInit {

incident: Incident;

constructor(private activateRoute: ActivatedRoute, private router: Router,

private incidentService: IncidentService, private alertController: AlertController,

private navController: NavController) { }

ngOnInit() {

this.activateRoute.paramMap.subscribe(paramMap => {

if (!paramMap.has('incidentId')) {

this.router.navigate(['/']); return;

} const incidentId = paramMap.get('incidentId');

this.incident = this.incidentService.getIncident(incidentId);

if (!this.incident) {

this.router.navigate(['/', 'tabs', 'incidents']);

return; }

});

} edit() {

} async confirmDeleteAlert() {

const alert = await this.alertController.create({

header: 'Are you sure?',

message: 'Do you want to

del delete the incident?',

buttons: [

{ text: 'Okay',

handler: () => {

ete the incident?',

buttons: [

{ text: 'Okay',

handler: () => {

this.incidentService.deleteIncident(this.incident.id);

}

},

'Cancel'

]

});

await alert.present();

}

delete() {

this.confirmDeleteAlert();

}

}

Map.ts

import { Component, OnInit } from '@angular/core';

import { Address, Location } from '../../incident.model';

import { Router, NavigationExtras, ActivatedRoute } from '@angular/router';

import { Geolocation } from '@ionic-native/geolocation/ngx';

import { AlertController } from '@ionic/angular';

import { IncidentService } from '../../incident.service';

@Component({

selector: 'app-address',

templateUrl: './address.page.html',

styleUrls: ['./address.page.scss'],

})

export class AddressPage implements OnInit {

address: Address = {

road: '',

area: '',

landmark: '',

city: ''

};

state: any;

location: Location = {

latitude: null,

longitude: null

};

constructor(private router: Router, private route: ActivatedRoute,

private alertController: AlertController, private geolocation: Geolocation,

private incidentService: IncidentService) {

this.route.queryParams.subscribe(params => {

this.state = this.router.getCurrentNavigation().extras.state;

if (!this.state) {

this.router.navigate(['/']);

}

});

}

ngOnInit() {

this.geolocation.getCurrentPosition().then((position) => {

this.location = {

latitude: position.coords.latitude,

longitude: position.coords.longitude

};

});

}

async invalidAddressAlert() {

const alert = await this.alertController.create({

header: 'Invalid Address',

message: 'Please enter correct address, all fields are required.',

buttons: ['Okay']

});

await alert.present();

}

next() {

if (this.address.area === '' ||

this.address.city === '' ||

this.address.landmark === '' ||

this.address.road === '') {

this.invalidAddressAlert();

return;

}

const extra: NavigationExtras = {

state: {

...this.state , address: this.address, location: this.location

}

};

this.router.navigate(['severity'], extra);

}}

New Incident.ts

import { Component, OnInit } from '@angular/core';

import { Router, NavigationExtras, ActivatedRoute } from '@angular/router';

import { AlertController } from '@ionic/angular';

import { Camera, CameraOptions } from '@ionic-native/camera/ngx';

import { IncidentService } from '../../incident.service';

@Component({

selector: 'app-camera',

templateUrl: './camera.page.html',

styleUrls: ['./camera.page.scss'],

})

export class CameraPage implements OnInit {

imageData: string;

constructor(private router: Router, private activatedRoute: ActivatedRoute,

private alertController: AlertController, private camera: Camera,

private incidentService: IncidentService) { }

ngOnInit() {

this.activatedRoute.paramMap.subscribe(paramMap => {

if (paramMap.has('incidentId')) {

const incidentId = paramMap.get('incidentId');

this.imageData = this.incidentService.getIncident(incidentId).image;

} else {

this.takePicture();

}

});

}

async cameraErrorAlert(err) {

const alert = await this.alertController.create({

header: 'Issue with camera',

message: 'Unable to take picture. Error ' + err,

buttons: [{text: 'Okay', handler: () => this.router.navigate(['/'])}]

});

await alert.present();

}

takePicture() {

const options: CameraOptions = {

quality: 100,

destinationType: this.camera.DestinationType.DATA\_URL,

encodingType: this.camera.EncodingType.JPEG,

mediaType: this.camera.MediaType.PICTURE

};

this.camera.getPicture(options).then((data) => {

this.imageData = 'data:image/jpeg;base64,' + data;

}, (err) => {

// this.imageData=' ';

this.router.navigate(['address'], extra);

}

}

Address.ts

import { Component, OnInit } from '@angular/core';

import { IncidentService } from '../../incident.service';

import { Severity } from '../../incident.model';

import { NavigationExtras, ActivatedRoute, Router } from '@angular/router';

import { AlertController } from '@ionic/angular';

@Component({

selector: 'app-severity',

templateUrl: './severity.page.html',

styleUrls: ['./severity.page.scss'],

})

export class SeverityPage implements OnInit {

selectedServerity: Severity;

severities: Severity[];

groupedSeverities: any;

state: any;

constructor(private incidentService: IncidentService,

private route: ActivatedRoute,

private router: Router,

private alertController: AlertController) { }

ngOnInit() {

this.severities = this.incidentService.getAllSeverities();

this.groupedSeverities = [];

for (let i = 0; i < this.severities.length; i += 2) {

this.groupedSeverities.push({items: this.severities.slice(i, i + 2)});

}

this.route.queryParams.subscribe(params => {

console.log (this.router.routerState);

this.state = this.router.getCurrentNavigation().extras.state;

if (!this.state) {

this.router.navigate(['/']);

}

});

}

async selectSeverityAlert() {

const alert = await this.alertController.create({

header: 'No Severity Selected',

message: 'Please select severity before continuing',

buttons: ['Okay']

});

await alert.present();

}

next() {

if (!this.selectedServerity) {

this.selectSeverityAlert();

return;

}

const extra: NavigationExtras = {

state: {

...this.state , severity: this.selectedServerity

}

};

this.router.navigate(['confirm-incident'], extra);

}

}

import { Component, OnInit } from '@angular/core';

import { User } from '../user.model';

import { UserService } from '../user.service';

import { AlertController } from '@ionic/angular';

import { Router } from '@angular/router';

@Component({

selector: 'app-login',

export class LoginPage implements OnInit {

email: string;

password: string;

users: User[];

constructor(private userService: UserService, private alertController: AlertController, private router: Router) { }

ngOnInit() {

this.users = this.userService.getUsers();

}

async invalidUserAlert() {

const alert = await this.alertController.create({

header: 'Invalid Email or Password',

message: 'Username or Password does is not valid.',

buttons: ['Okay']

});

await alert.present();

}

login(form: { email: string; password: string; }) {

console.log (this.email, this.password);

const loginUser = this.users.find((user) => {

return user.email === this.email && user.password === this.password;

});

if (!loginUser) {

this.invalidUserAlert();

return;

}

this.router.navigate(['./', 'home']);

}

}

Camera.ts

import { Component, OnInit } from '@angular/core';

import { User } from '../user.model';

import { UserService } from '../user.service';

import { AlertController } from '@ionic/angular';

import { Router } from '@angular/router';

@Component({

selector: 'app-register',

templateUrl: './register.page.html',

styleUrls: ['./register.page.scss'],

})

export class RegisterPage implements OnInit {

user: User = {

name: '',

email: '',

password: '',

gender: 'M',

id: 'U1'

};

confirm: string;

constructor(private userService: UserService, private alertController: AlertController, private router: Router) { }

ngOnInit() {

}

async passwordDoesNotMatchAlert() {

const alert = await this.alertController.create({

header: 'Password does not match',

message: 'Password provided does not match.',

buttons: ['Okay']

});

await alert.present();

}

register() {

if (this.confirm !== this.user.password) {

this.passwordDoesNotMatchAlert();

return;

}

this.router.navigate(['/tabs/maps']);

}

}

User.ts

import { Injectable } from '@angular/core';

})

export class UserService {

private users: User[] = [{

id: 'User 1',

name: 'Nikita',

gender: 'F',

password: 'abcd',

email: 'abcd@gmail.com'

}];

user: User;

constructor() { }

private userThumbnail(user: User): string {

if ( user.thumbnail ) {

return user.thumbnail;

}

if ( user.gender === 'F' ) {

return '/assets/images/users/female.png';

}

return '/assets/images/users/male.png';

}

getUsers(): User[] {

const users = this.users.map((u) => ({...u, thumbnail: this.userThumbnail(u)}));

return users;

}

getUserDetails(): User {

if (!this.user) {

this.user = this.users[0];

}

const user = {...this.user, thumbnail: this.userThumbnail(this.user)};

return user;

}

register(user: User) {

this.user = user;

}

}

**6. RESULTS AND DISCUSSION:**

Testing is the process of running a system with the intention of finding errors. Testing enhances the integrity of a system by detecting deviations in design and errors in the system. Testing aims at detecting error-prone areas. This helps in the prevention of errors in a system. Testing also adds value to the product by conforming to the user requirements.

The main purpose of testing is to detect errors and error-prone areas in a system. Testing must be thorough and well-planned. A partially tested system is as bad as an untested system. And the price of an untested and under-tested system is high.

The implementation is the final and important phase. It involves user-training, system testing to ensure successful running of the proposed system. The user tests the system and changes are made according to their needs. The testing involves the testing of the developed system using various kinds of data. While testing, errors are noted and correctness is the mode.

The objectives of testing are:

* Testing is a process of executing a program with the intent of finding errors.
* A Successful test case is one that uncovers an as- yet-undiscovered error.

System testing is a stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as per the user need, before the live operation commences.

Testing is vital to the success of a system. System testing makes a logical assumption that if all parts of the as system are correct, the goal will be successfully achieved. A series of tests are performed before the system is ready for the user acceptance test.

Testing in Salesforce.com:

* Check that configuration and code is functional
* Ensure that the system’s initial build meets the agreed requirements
* Help to control the project scope
* Confirm that the finished system can support the client’s business processes
* Gain client’s approval to release new functionality for general use
  1. **TEST PLAN**

Testing should not be used for:

* Making changes to business processes
* Introducing additional requirements outside of scope
* Making significant cosmetic changes to page layouts and user interface
* Training users

Plan:

* Unit Testing (code developers)
* System Testing
* User Acceptance Testing (aka Functional Testing)
* Production Testing
* Regression Testing
  1. **TEST DESIGN**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Test Case Name | Step Name | Step Description | Expected Result |
| TC 01 | Login | 1 | Launch PinPot | App should launch |
| TC 02 | Map view | 1 | Log into app | User should be able to view all nearby incidents on the map along with menu items / tab bar |
| TC 03 | List view | 1 | Log into app | User should be able to view incidents on map along with menu items/ tab bar |
|  |  | 2 | Click on list view icon in main menu | User should be able to view incidents in list view |
| TC 04 | Detect location | 1 | Log into app | User should be able to view his location in the home screen. |
| TC 05 | Submit incident | 1 | Clicks on new incident icon in home screen | Redirect to new incident screen User should be able to start creating new incident. |
|  |  | 2 | Fill incident location details | User should be able to fill location details and click next |
|  |  | 3 | Provide incident severity details | User should be able to select incident severity and click next |
|  |  | 4 | Capture picture of the incident | User should be able to capture a picture |
|  |  | 5 | Review and submit | Review the provided details and submit the incident |
|  |  | 6 | Cancel the incident | User should be able to cancel the new incident at any screen by clicking cancel. Redirect to map view screen. |
| TC 06 | View user profile | 1 | Clicks on user profile in home screen | User should be able to view his profile |
|  |  |  |  | User should be able to no. of contributed incident |

* 1. **TEST REPORT**

This section lists the results that were produced by running the test cases. Below Table lists the test cases that were used while testing the interface along with the expected result and the actual results for each test case.

|  |  |  |
| --- | --- | --- |
| Test Case Number | Test case name | Actual Result |
| TC 01 | Login | Pass |
| TC 02 | Map view | Pass |
| TC 03 | List view | Pass |
| TC 04 | Detect location | Pass |
| TC 05 | Submit incident | Pass |
| TC 06 | View user profile | Pass |
|  | | |

**7. CONCLUSION AND FUTURE WORK:**

**Conclusion**

The objective of “Pin Pot App” project is to build a program for logging a complain about damaged road. The system developed is able to meet all the basic requirements. The management of the records will be also benefited by the proposed system, as it will automate the whole procedure, which will reduce the workload. The security of the system is also one of the prime concerns. There is always a room for improvement in any software, however efficient the system may be. The important thing is that the system should be flexible enough for future modifications. The system has been factored into different modules to make system adapt to the further changes. Every effort has been made to cover all user requirements and make it user friendly.

Goal achieved: The System is able provide the interface to the user so that he can replicate his desired data.

User friendliness: Though the most part of the system is supposed to act in the background, efforts have been made to make the foreground interaction with authorized team, making them understand about the real time problem as smooth as possible. Which will help to reduce the road accident and easy travel of the people. Also the integration of the existing system with the project has been kept in mind throughout the development phase.

**Future Enhancements**

**Limitations and Scope for Future Enhancements:**

* System works in all platforms and its compatible environments.
* Advanced techniques are not used to check the authorization when logged in from multiple system for a same user.

**Future Enhancements:**

The project has been developed in a very short period of time and all efforts have been taken so that this project is very efficient in its execution there still exists some scope of improvement in the project. The following lists some of the enhancement that can be added incorporate into the project.

Application of the project can be done more attractively. Database management and stock maintenance module can be added which helps the administrator. This project can also be modified into a client server application. More security measures can be taken.

User requirements keep changing as the system is being used. Some of the future enhancements that can be done to this system are:

* As the technology emerges, it is possible to upgrade the system and can be adaptable to desired environment.
* Based on the future security issues, security can be improved.
* Based on the customer experience the UI can be upgraded.
* Notification to email/mobile can be implemented when got any update from the road authority team.

**REFERENCES:**

* <https://www.w3schools.com/>
* <https://material.io/guidelines/>
* <https://angular.io/>tutorial
* <https://ionicframework.com/>
* <https://getbootstrap.com/>
* <https://www.draw.io>
* <https://angular-maps.com/guides>
* <https://developers.google.com/maps/documentation/javascript/get-api-key>
* <https://developers.google.com/places/android-sdk>