SNAPQUEST

By

HARDIK SHAH (2241131)

HIMANSHI AGARWAL (2241132)

SUMIT ROBERTS EMMANUEL (2241160)

Under the supervision of

Dr. SHARMILA B

Mobile Application CIA Project Report Submitted in complete fulfillment of the requirements, CHRIST (Deemed To Be University)

October - 2024

ACKNOWLEDGEMENT

First of all, we thank God almighty for his immense grace and blessings showered on us at every stage of this work. We are grateful to our respectable Head, Department of Computer Science, CHRIST (Deemed to be University), **Dr Ashok Immanuel V**, for providing the opportunity to take up this project as part of my curriculum.

We also pay our gratitude to the Coordinator, Department of Computer Science, CHRIST (Deemedto be University) **Dr Sagaya Aurelia P** for their support throughout.

We are grateful to our guide, Assistant Professor, Department of Computer Science, CHRIST (Deemed to be University), **Dr Sharmila B**, whose insightful leadership and knowledge helped us to complete this project successfully. Thank you so much for your continuous support and presence whenever needed.

We express our sincere thanks to all faculty members and staff of the Department of Computer Science, CHRIST (Deemed to be University), for their valuable suggestions during the course of this project. Their critical suggestions helped us to improve the project work.

Last but not least, we would like to thank everyone who is involved in the project directly or indirectly.

Table of Contents

1. INTRODUCTION

1.1 Overview of the system

SnapQuest is an exciting mobile game that combines creativity, observation, and speed. Each day, players receive a random photo task, such as "Take a picture of a red bicycle," with a 5-minute countdown starting as soon as the task is revealed. The challenge is quickly locating and photographing the specified object before time runs out.

Points are awarded based on how fast players complete the task, with quicker submissions earning more points. Failure to finish within the time limit results in the deduction of points. A daily leaderboard ranks players by their performance, fostering a competitive environment and motivating them to improve.

SnapQuest transforms everyday surroundings into a playground for exploration and creativity, offering a unique blend of photography and time-based challenges. It’s a fun and fast-paced way to push users to observe their world more closely while enjoying a thrilling daily adventure.

1.2 Goals

1. **Foster Creativity and Observation**: Encourage players to engage with their surroundings and think creatively by completing unique daily photo tasks.

2. **Enhance Time Management Skills**: Challenge users to act swiftly and efficiently, improving their ability to manage time and make quick decisions under pressure.

3. **Incentivize Consistency and Daily Participation**: Motivate players to return daily by offering new challenges, rewarding consistency, and promoting habitual participation.

4. **Build a Competitive Community**: Provide a dynamic and competitive experience through a daily leaderboard, driving users to continuously improve their performance.

5. **Create a Fun, Engaging Experience**: Offer a fast-paced, enjoyable game that blends photography with time-based challenges, keeping players entertained and engaged.

1.4 Boundaries of the Application

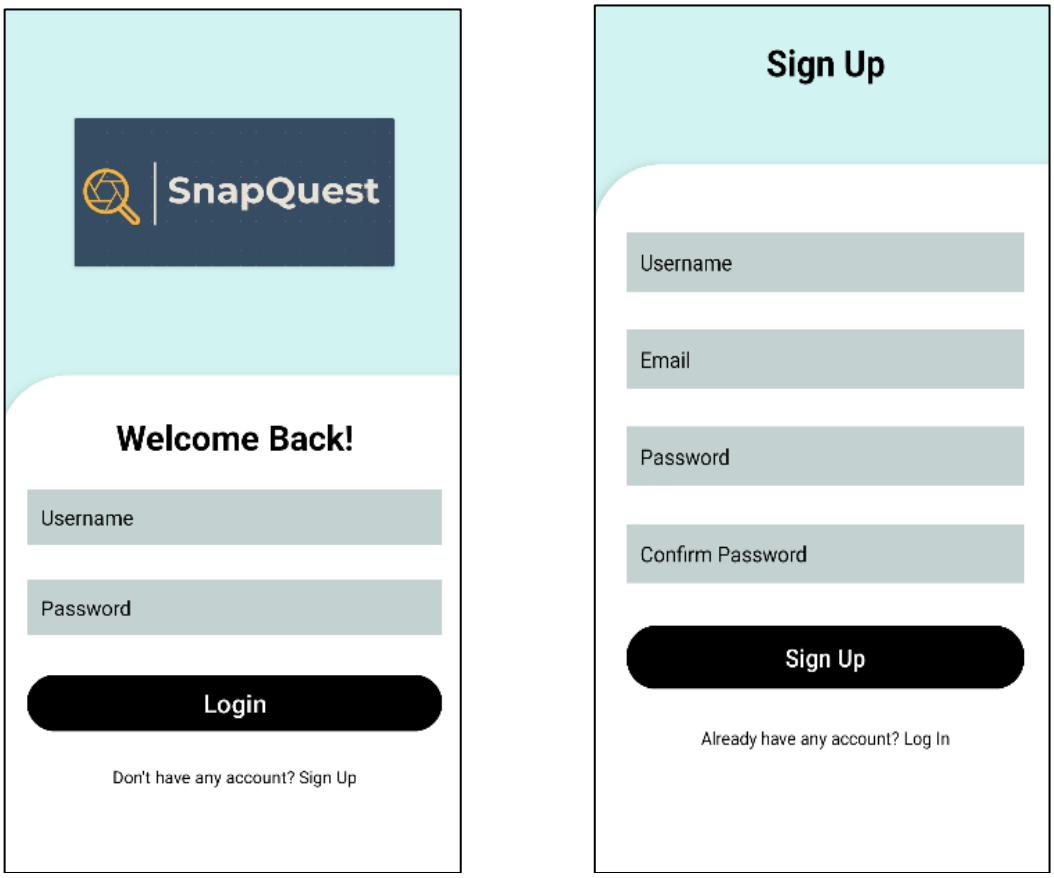
- **Features**:
    - **Daily Photo Challenge**: Players receive a unique photo task each day, encouraging exploration and creativity.
    - **Countdown Timer**: A time limit is set for each task, pushing users to complete the challenge quickly.

- **Points System**: Points are awarded based on task completion speed and accuracy, with penalties for failure or incorrect submissions.
- **Photo Validation**: Submitted photos are validated via an API to ensure they match the task, determining points earned or deducted.
- **Leaderboard**: A daily ranking system that highlights top players based on performance, fostering competition.

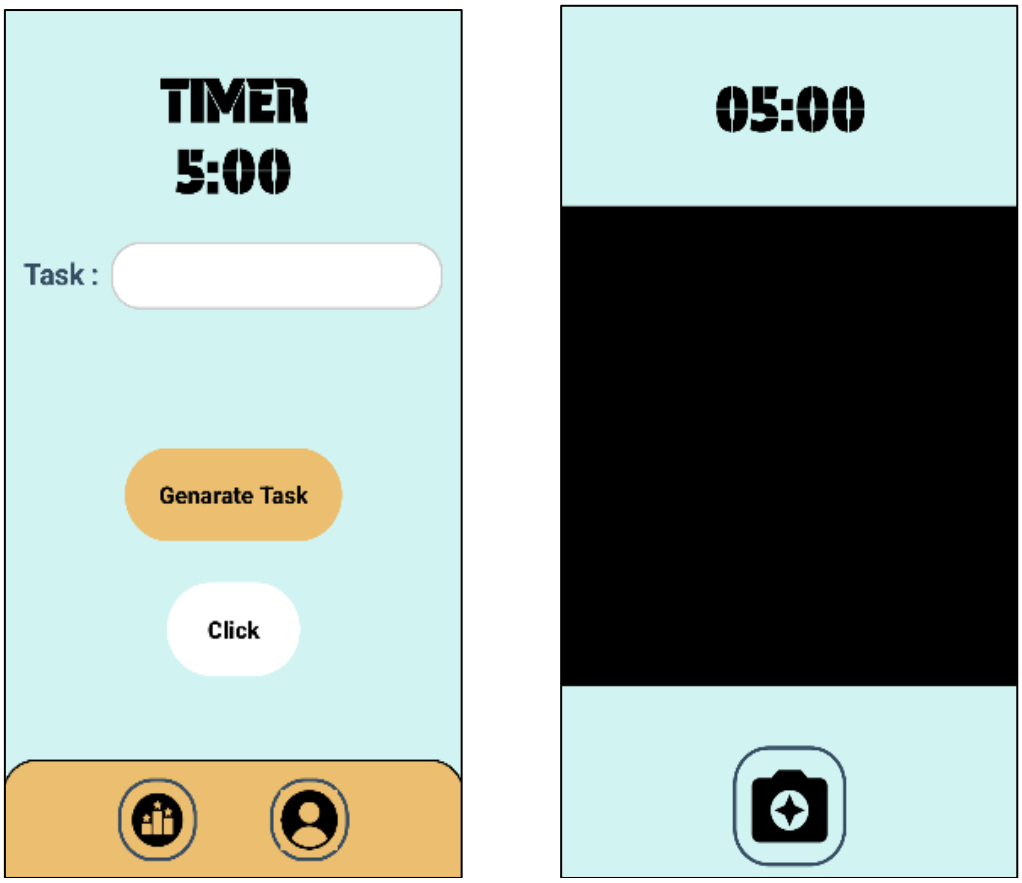
- **Target Audience**: Primarily Photography Enthusiasts, Family and Groups, Urban Explorers who enjoy taking photos and love to explore their surroundings and discover new things.

2. DESIGN AND DEVELOPEMNT

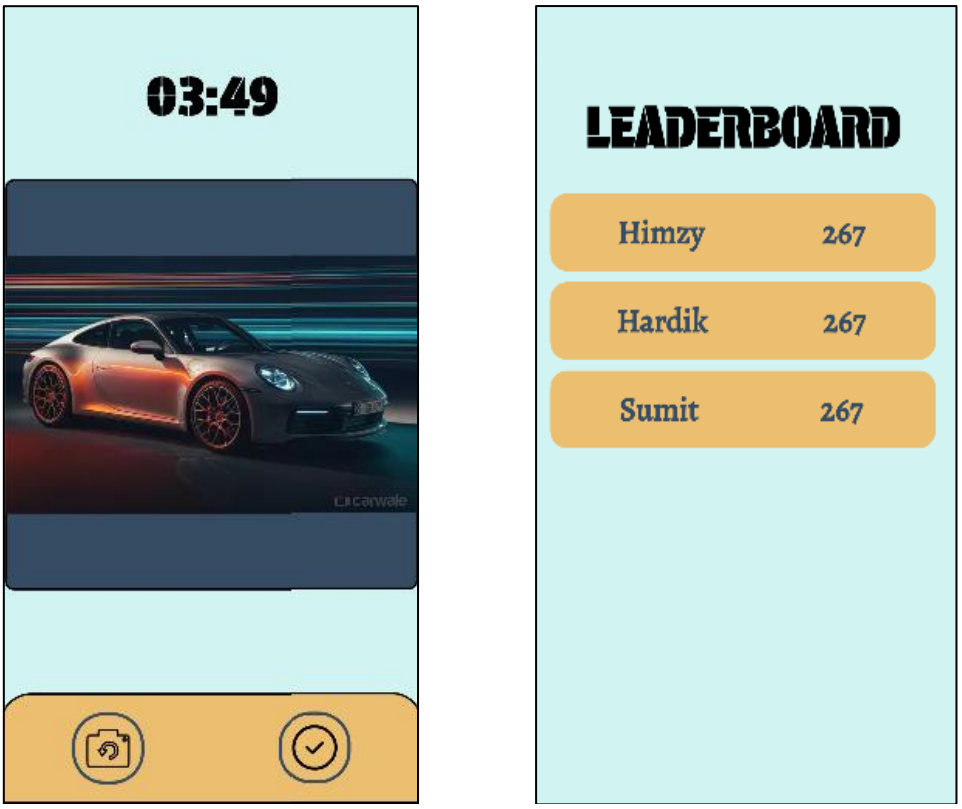
2.1 Screenshots



Login Page and Signup Page of SnapQuest



Task Page and Camera of SnapQuest



Preview Page and Leaderboard Page of SnapQuest

2.2 Development Environment

The **SnapQuest** mobile application was developed using the following tools and technologies:

- **Platform**: Android
- **IDE**: Android Studio
- **Programming Language**: Kotlin (for app development)
- **Database**: Firebase Firestore (for storing tasks, user clicked images, points etc.)
- **Design Tools**: XML (for designing UI layouts)
- **Libraries/Frameworks**:
    - **Firebase Authentication**: For secure user login and registration.

2.3 Process Flow

1. **Planning and Requirement Gathering**: The initial step involved understanding the app's core functionalities, such as daily photo challenges, countdown timers, photo validation via API, a points system, and leaderboards. The user experience was designed to be engaging and competitive, ensuring a seamless flow from task initiation to photo validation and point tracking.

2. **Designing the User Interface**:
    - Designed using XML layouts in Android Studio, focusing on a clean and intuitive interface.
    - The main UI components include:
        - **Login/Registration screens**: For secure user access.
        - **Daily Task Page:** Displays the current task along with the countdown timer.
        - **Camera Interface:** Enables users to capture photos in real-time for the daily challenge.
        - **Leaderboard Screen**: Shows daily rankings, motivating players through competition.
        - **Preview Tasks Page**: Allows users to review their completed tasks and their corresponding points.
3. **Setting Up the Database**:
    - Firestore Database was chosen to store user profiles, daily tasks, photos, points, and leaderboard rankings.
    - Each user has their own collection, where photos and points are stored, categorized by date and task completion status.
4. **Development of Core Features**:
    - **User Authentication**: Firebase Authentication was integrated to manage user sign-ups and logins.
    - **Daily Photo Challenge**: A system to assign a unique photo task to each user daily, with a countdown timer for task completion.
    - **Photo Capture and Validation**: The app utilizes the device camera for task-related photo capture. A third-party API validates the image to ensure it matches the task, awarding or deducting points accordingly.
    - **Points System**: Points are awarded based on how quickly and accurately the task is completed. Incorrect or late submissions lead to point deductions.
    - **Leaderboard**: A dynamic leaderboard that ranks players based on points, encouraging daily participation and competition.
5. **Testing**: The application was rigorously tested across various Android devices to ensure responsive UI, functional camera integration, accurate photo validation, and reliable points assignment. Bugs related to photo validation, timer functionality, and user data synchronization were identified and resolved.

6. **Deployment**: Once development was completed, the app was packaged and signed in Android Studio, ready for release on the Google Play Store.

3. TESTING AND IMPLEMENTATION

3.1 Device Testing:

- The app was installed on various Android smartphones with different screen sizes, resolutions, and Android OS versions to check compatibility and responsiveness.
- Focus was placed on how the UI adapts to different screen sizes and resolutions.

3.2 Functionality Tests:

- Login and Registration: Verified that Firebase Authentication correctly allowed users to create accounts and log in.
- Task generation and display Test: Ensure that a unique photo task is generated and displayed for each user daily. Confirm that the task description is clear and corresponds with the challenge.
- Countdown Timer Test: Validate that the countdown timer starts when the task is revealed. Ensure the timer stops once the task is completed or when the time runs out. Test the behavior when the app is in the background or reopened.
- Camera and Photo Capture Test: Check the image quality and compatibility across different devices. Ensure the captured photo is correctly stored in the database for validation.
- Photo Validation via API: Test that the API accurately validates the captured image against the task criteria. Verify that points are correctly awarded or deducted based on validation results. Check for error handling in case the API fails or returns incorrect results.
- Points System Test: Ensure that points are awarded based on task completion speed and accuracy. Validate point deductions for incorrect submissions or task failures. Test the cumulative points calculation across multiple tasks.

3.3 User Experience Testing:

- The flow between screens was tested to ensure a smooth, user-friendly experience.
- App speed and performance were tested to avoid lags or delays when capturing pictures and leaderboard score updates.

3.4 Debugging: Logged errors and crashes were monitored using Logcat in Android Studio, and necessary code adjustments were made to ensure stability.

3.5 Source Code

3.5.1 Login

```
private fun login() {
    val name = username.text.toString().trim()
    val pass = password.text.toString().trim()
    if (name.isEmpty() || pass.isEmpty()) {
        Toast.makeText( context: this,  text: "Fill all Fields!", Toast.LENGTH_LONG).show()
        return
    }
    db.collection( collectionPath: "Users").document(name).get()
        .addOnSuccessListener { document ->
            if (document.exists()) {
                val storedPassword = document.getString( field: "password")

                if (storedPassword == pass) {
                    Toast.makeText( context: this,  text: "Login Successful!", Toast.LENGTH_SHORT).show()
                    saveSession(name)
                    val intent = Intent( packageContext: this, TaskActivity::class.java)
                    startActivity(intent)
                }
                else {
                    Toast.makeText( context: this,  text: "Incorrect Password", Toast.LENGTH_SHORT).show()
                }
            }
            else {
                Toast.makeText( context: this,  text: "Username does not exist", Toast.LENGTH_SHORT).show()
            }
```

3.5.2 Signup

```
signupButton.setOnClickListener { it: View!
    val name = userName.text.toString().trim()
    val email = email.text.toString().trim()
    val password = password.text.toString().trim()
    val confirmPassword = confirmPassword.text.toString().trim()

    if (name.isBlank() || email.isBlank() || password.isBlank() || confirmPassword.isBlank()) {
        Toast.makeText( context: this, text: "All fields are required", Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }

    if (password != confirmPassword) {
        Toast.makeText( context: this, text: "Passwords do not match", Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }

    // Check if username already exists
    db.collection( collectionPath: "Users").document(name).get()
        .addOnSuccessListener { documentSnapshot ->
            if (documentSnapshot.exists()) {

                Toast.makeText( context: this, text: "Username already exists", Toast.LENGTH_SHORT).show()
            } else {
```

```
            val user = hashMapOf(
                "username" to name,
                "email" to email,
                "password" to password,
                "points" to 0
            )

            db.collection( collectionPath: "Users").document(name) DocumentReference
                .set(user) Task<Void!>
                .addOnSuccessListener { it: Void!
                    Toast.makeText( context: this, text: "Signup successful", Toast.LENGTH_SHORT).show()

                    startActivity(Intent( packageContext: this, LoginActivity::cla
                    finish()
                }
                .addOnFailureListener { e ->
                    Toast.makeText( context: this, text: "Error signing up: ${e.message}", Toast.LENGTH_SHORT).show
                }
        }
    }
    .addOnFailureListener { e ->
        Toast.makeText( context: this, text: "Error checking username: ${e.message}", Toast.LENGTH_SHORT).show()
    }
```

Fetching Documentation...

3.5.3 Task and Timer

```
private fun generateNewTask() {
    val randomTaskId = "2" //(1..10).random().toString()
    sharedPreferences.edit().putString("taskId", randomTaskId).apply()


    db.collection( collectionPath: "Tasks").document(randomTaskId).get()
        .addOnSuccessListener { document ->
            if (document.exists()) {
                val taskText = document.getString( field: "task")
                task.text = taskText

                click.isEnabled = true
                click.visibility = View.VISIBLE

                saveTaskGenerationTime()

                startTimer(TASK_DURATION_MILLIS)
            } else {
                Toast.makeText( context: this,  text: "Task not found", Toast.LENGTH_SHORT).show()
            }
        }
        .addOnFailureListener { e ->
            Toast.makeText( context: this,  text: "Error loading task: ${e.message}", Toast.LENGTH_SHORT).show(
        }
}
```

```
private fun startTimer(durationMillis: Long) {
    timer?.cancel() // Cancel any previous timer if running

    timer = object : CountDownTimer(durationMillis,  countDownInterval: 1000) {
        override fun onTick(millisUntilFinished: Long) {
            val minutes = TimeUnit.MILLISECONDS.toMinutes(millisUntilFinished)
            val seconds = TimeUnit.MILLISECONDS.toSeconds(millisUntilFinished) % 60
            timerView.text = String.format(Locale.getDefault(),  format: "%02d:%02d", minutes, seconds)

            // Save remaining time to SharedPreferences to persist the timer state
            sharedPreferences.edit().putLong("timer_remaining", millisUntilFinished).apply()
        }

        override fun onFinish() {
            sharedPreferences.edit().remove("timer_remaining").apply() // Clear timer on finish
            val intent = Intent( packageContext: this@TaskActivity, FailedActivity::class.java)
            startActivity(intent)
            finish()
        }
    }.start()
}
```

3.5.4 Camera and Photo

```
private fun startCamera() {                                    11 3 1
    val cameraProviderFuture = ProcessCameraProvider.getInstance( context: this)

    cameraProviderFuture.addListener({
        val cameraProvider = cameraProviderFuture.get()

        val preview = Preview.Builder().build().also { it: Preview
            it.setSurfaceProvider(cameraPreview.surfaceProvider)
        }

        val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

        imageCapture = ImageCapture.Builder().build()

        try {
            cameraProvider.unbindAll()
            cameraProvider.bindToLifecycle( lifecycleOwner: this, cameraSelector, preview, imageCapture)

        } catch (exc: Exception) {
            // Log or handle any errors
        }

    }, ContextCompat.getMainExecutor( context: this))
}
```

```
private fun takePhoto() {
    val outputOptions = ImageCapture.OutputFileOptions.Builder(
        contentResolver,
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        ContentValues()
    ).build()

    imageCapture.takePicture(
        outputOptions,
        Executors.newSingleThreadExecutor(),
        object : ImageCapture.OnImageSavedCallback {
            override fun onError(exc: ImageCaptureException) {
                // Handle the error
            }

            override fun onImageSaved(outputFileResults: ImageCapture.OutputFileResults) {
                // You can retrieve the image path via outputFileResults.savedUri
                val image=outputFileResults.savedUri
                if (image != null) {
                    uploadImageToFirebase(image)
                }
```

3.5.5 Preview

```
private fun loadCapturedImage() {
    val sharedPreferences = getSharedPreferences( name: "task_prefs", MODE_PRIVATE)
    val encodedImage = sharedPreferences.getString("captured_image", null)
    encodedImage?.let{ it: String
        val imageUri=Uri.parse(it)
        clickImage.setImageURI(imageUri)
    }
}
```

3.6 Main Features

1. User Authentication:

- Secure user login and registration using Firebase Authentication.
- Personal account management, allowing users to track their own expenses and budgets individually.

2. Daily Task with Timer:

- Users receive a unique photo task each day. A countdown timer begins immediately, pushing them to complete the task within the given time frame.

3. Capture Photo:

- Players must locate and capture the specified object or scene using their device's camera, adding an interactive and creative challenge.

4. Photo Validation with API:

- Once a photo is submitted, the app uses an API to validate whether the captured image matches the daily task. Points are awarded or deducted based on the accuracy of the photo submission.

5. Leaderboard:

- A daily leaderboard ranks players based on their task completions and the speed of their submissions, promoting a competitive environment.

6. Data Storage and Syncing:

- All user data stored in Firebase Firestore is securely saved and synced across

devices.

4. RESULTS AND CONCLUSION

4.1 Summary

SnapQuest is a fast-paced mobile game that challenges players to complete daily photo tasks within a limited time. Each day, users receive a unique task, such as capturing a specific object, with a countdown timer adding urgency. Photos are validated using an API to ensure they meet the task requirements, and points are awarded or deducted based on accuracy and speed. Players can track their progress through a leaderboard that ranks them against others, fostering competition. The app combines creativity, exploration, and time management in a fun, engaging way that encourages daily participation.

4.2 Future Scope

1. **Advanced Photo Recognition:** Enhancing the photo validation API to recognize more complex objects, scenes, or even specific brands, improving task diversity and challenge difficulty.
2. **Task Customization:**
   Allowing users to create and share their own photo challenges, fostering community engagement and creativity.
3. **Cross-Platform Availability:**
   Expand the app to iOS and create a web-based dashboard where users can view and manage their expenses across devices**.**