

ΑΤΟΜΙΚΗ ΕΡΓΑΣΙΑ ΤΟΥ ΦΟΙΤΗΤΗ

ΙΩΑΝΝΗ – ΑΛΕΞΑΝΔΡΟΥ ΕΥΑΓΓΕΛΟΥ (mpsp2210)

Στο μάθημα

ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ KΙΝΗΤΕΣ ΣΥΣΚΕΥΕΣ

Πίνακας περιεχομένων

[Σύνοψη 3](#_Toc140684916)

[Τρόπος Υλοποίησης 3](#_Toc140684917)

[Πολίτες 3](#_Toc140684918)

[Υπάλληλοι Πολιτικής Προστασίας 3](#_Toc140684919)

[Ανάλυση Απαιτήσεων 4](#_Toc140684920)

[Υλοποίηση 5](#_Toc140684921)

[Authentication – Logging In - Registering 5](#_Toc140684922)

[Κύριες Λειτουργίες Εφαρμογής 10](#_Toc140684923)

[Firebase Realtime Database 10](#_Toc140684924)

[Αναφορά Συμβάντος από Πολίτη 12](#_Toc140684925)

[Αctivity προβολής στατιστικών 19](#_Toc140684926)

[Εμφάνιση ειδοποιήσεων 21](#_Toc140684927)

[Λειτουργίες υπαλλήλου πολιτικής προστασίας 26](#_Toc140684928)

[Εμφάνιση υποβληθέντων περιστατικών 26](#_Toc140684929)

[Εμφάνιση λεπτομερειών συμβάντος 29](#_Toc140684930)

# Σύνοψη

Σκοπός της παρούσας εργασίας είναι η κατασκευή μίας εφαρμογής με τίτλο “SmartAlert”, άμεσης ειδοποίησης πολιτών σε έκτακτες περιστάσεις υψηλού κινδύνου.

## Τρόπος Υλοποίησης

Οι χρήστες της εφαρμογής χωρίζονται σε δύο ρόλους:

* Τους πολίτες
* Τους υπαλλήλους πολιτικής προστασίας

### Πολίτες

Οι πολίτες έχουν το δικαίωμα να αναφέρουν κάποια έκτακτη κατάσταση κινδύνου από τις κάτωθι κατηγορίες:

* Πυρκαγιά
* Πλημμύρα
* Σεισμό
* Άλλη έκτακτη κατάσταση κινδύνου

Κατά την αναφορά της έκτακτης κατάστασης κινδύνου μπορούν να προσθέσουν μια σύντομη περιγραφή της έκτακτης κατάστασης κινδύνου αλλά και να ανεβάσουν φωτογραφία που απαθανατίζει το συμβάν.

### Υπάλληλοι Πολιτικής Προστασίας

Οι υπάλληλοι της πολιτικής προστασίας είναι υπεύθυνοι να αξιολογούν τη σοβαρότητα των συμβάντων και να αξιολογούν αν για τα υποβληθέντα από τους χρήστες συμβάντα θα πρέπει να σταλεί μία γενικότερη ειδοποίηση στους πολίτες που έχουν εγκατεστημένη την εφαρμογή και βρίσκονται κοντά. Σε περίπτωση που ο υπάλληλος κρίνει ότι το συμβάν δεν είναι αξιόπιστο ή δεν πληροί τις προϋποθέσεις για να χαρακτηριστεί «κατάσταση ανάγκης», μπορεί να απορρίψει τη συγκεκριμένη αναφορά ή να συνεχίσει να διερευνά το συμβάν έως ότου αποφασίσει για την έγκριση ή απόρριψη του.

# Ανάλυση Απαιτήσεων

1. Τα συμβάντα θα πρέπει να υποβάλλονται σε ένα backend module ώστε να διαμοιράζονται τόσο μεταξύ των χρηστών αλλά και μεταξύ των χρηστών και των υπαλλήλων της πολιτικής προστασίας.
2. Η υποβολή του συμβάντος θα πρέπει να συνοδεύεται από σχόλιο του χρήστη, την τοποθεσία του η οποία θα λαμβάνεται αυτόματα από το κινητό, από χρονοσφραγίδα την ώρα υποβολής του συμβάντος καθώς και από πιθανή επισυναπτόμενη φωτογραφία.
3. Το backend module πρέπει να παρέχει στον υπάλληλο της πολιτικής προστασίας μία «σύσταση» ως προς την κατάταξη του κάθε υποβληθέντος συμβάντος με γνώμονα την εγκυρότητα που παρουσιάζει. Οι παράγοντες εγκυρότητας είναι οι ακόλουθοι:
   1. Το πλήθος των χρηστών που έχουν υποβάλλει ένα συγκεκριμένο περιστατικό κινδύνου
   2. Το χρονικό περιθώριο υποβολής μεταξύ των περιστατικών των διαφόρων χρηστών
   3. Η απόσταση μεταξύ των υποβληθέντων περιστατικών
4. Μετά την έγκριση του συμβάντος από τον υπάλληλο της πολιτικής προστασίας, οι χρήστες που βρίσκονται εγγύς του συμβάντος θα πρέπει να ειδοποιούνται για το περιστατικό, λαμβάνοντας πληροφορίες επί του περιστατικού και οδηγίες από την πολιτική προστασία.
5. Η εφαρμογή πρέπει να υποστηρίζει την αγγλική και την ελληνική γλώσσα και να χρησιμοποιεί την αντίστοιχη με αυτήν που έχει επιλέξει για τη συσκευή του ο χρήστης.
6. Οι χρήστες πρέπει να εγγράφονται στην εφαρμογή. Οι τρόποι που επιλέχθηκαν ήταν μέσω Google Authentication και η δημιουργία νέου χρήστη με email.
7. Η ύπαρξη ενός Activity προβολής στατιστικών, όπου κάθε χρήστης μπορεί να δει πόσες φορές έχει ενεργοποιηθεί συμβάν κινδύνου. Οι πληροφορίες εμφανίζονται βάσει κατηγορίας συμβάντος.
8. Δημιουργία εφαρμογής η οποία θα είναι  responsive και aesthetically pleasing.
   1. Δόθηκε δηλαδή βαρύτητα τόσο στην ομαλή απόκριση της εφαρμογής με σκοπό την ταχύτητα και την ενημέρωση του χρήστη σε κάθε περίπτωση ότι κάποια διεργασία βρίσκεται σε εξέλιξη αλλά και στη δημιουργία ενός «όμορφου» αισθητικά UI το οποίο περιλαμβάνει «φωτεινό» και «σκούρο» θέμα το οποίο εναλλάσσεται ανάλογα με την επιλογή που έχει κάνει ο χρήστης για τη συσκευή του ώστε να υπάρχει μία συνοχή.
   2. Πρόβλεψη για χρήση της εφαρμογής σε tablet ώστε να μην «αλλοιώνεται».
9. Έμφαση στο UX ώστε η εφαρμογή να είναι απλή κατά την χρήση αλλά και να καθοδηγεί τον χρήστη σε κάθε περίπτωση.
10. Να παρέχεται στο χρήστη η δυνατότητα εμφάνισης του συμβάντος εκτάκτου ανάγκης πάνω στο χάρτη μαζί με την τρέχουσα θέση του σε περίπτωση που εκδοθεί αντίστοιχη ειδοποίση.

# Υλοποίηση

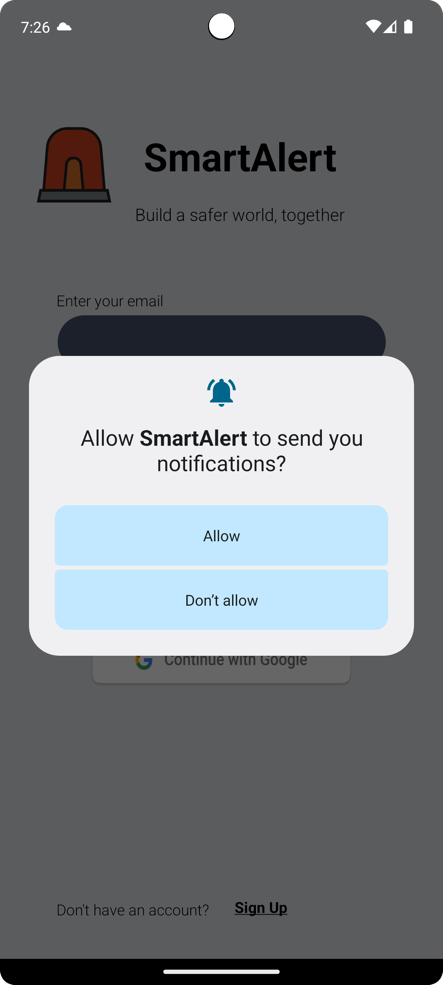
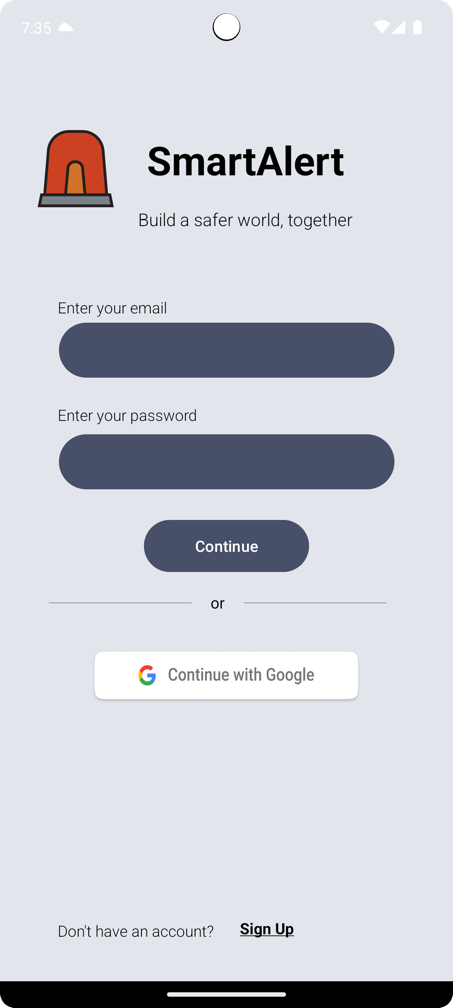
## Authentication – Logging In - Registering

Για το authentication επέλεξα τη χρήση του Firebase Authentication δίνοντας στους χρήστες τη δυνατότητα σύνδεσης και δημιουργίας νέου λογαριασμού χρησιμοποιώντας:

* Google λογαριασμό
* Email – password

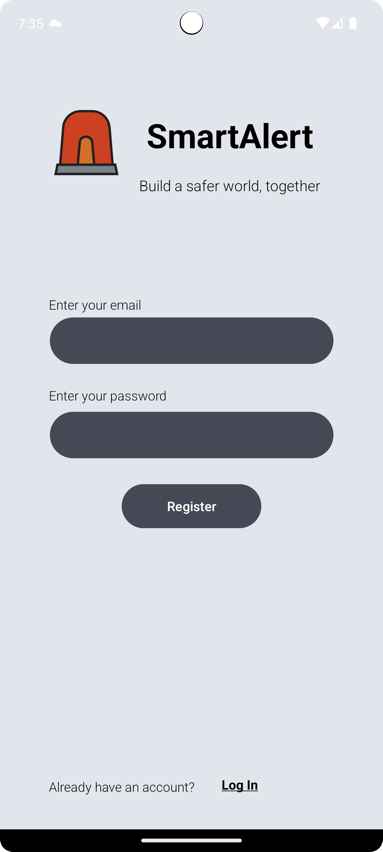
Κατά την εμφάνιση του Log In Activity εκτελώ επίσης τις ακόλουθες διεργασίες:

* Ελέγχεται εάν ο χρήστης έχει παραχωρήσει δικαίωμα εμφάνισης ειδοποιήσεων (απαιτείται σε API >= 33). Σε περίπτωση που ο χρήστης δεν έχει παραχωρήσει τη συγκεκριμένη άδεια, εμφανίζεται το ανάλογο alert ώστε να ζητήσει την έγκριση του χρήστη.
* Ελέγχεται εάν ο χρήστης έχει ήδη συνδεθεί ώστε να μεταφερθεί αυτόματα στο Main Activity της εφαρμογής.

Εικόνα 1: Εμφάνιση alert για έγκριση Εικόνα 2: Log In Activity - light theme

δικαιώματος εμφάνισης ειδοποιήσεων



Εικόνα 3: Register Activity - Light theme

class LoginActivity : AppCompatActivity() {  
  
 private var auth: FirebaseAuth = FirebaseAuth.getInstance()  
 lateinit var mGoogleSignInClient: GoogleSignInClient private var email: String = "" private var password: String = "" val Req\_Code:Int=123 private val NOTIFICATION\_PERMISSION\_REQUEST\_CODE = 1  
  
 @RequiresApi(Build.VERSION\_CODES.O)  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.login\_activity)  
  
  
 val email = findViewById<EditText>(R.id.emailText)  
 val password = findViewById<EditText>(R.id.passwordText)  
 val loginBtn = findViewById<Button>(R.id.continueBtn)  
 val signUpBtn = findViewById<Button>(R.id.createAccount)  
  
 val animationView = findViewById<LottieAnimationView>(R.id.lottie\_animation\_view)  
 // Apply the appropriate theme  
 AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE\_NIGHT\_FOLLOW\_SYSTEM)  
  
 //Sign in with Google :  
 val gso = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT\_SIGN\_IN)  
 .requestIdToken(getString(R.string.default\_client))  
 .requestEmail()  
 .build()  
 // getting the value of gso inside the GoogleSigninClient  
 mGoogleSignInClient= GoogleSignIn.getClient(this,gso)  
 val googleBtn = findViewById<ImageButton>(R.id.googleBtn)  
 googleBtn.setOnClickListener {  
 animationView.visibility = View.VISIBLE  
 signInGoogle()  
 }  
 //Log in with email/ password  
 loginBtn.setOnClickListener {  
 animationView.visibility = View.VISIBLE  
 Log.d("Login", "logging in with email")  
 val emailText = email.text.toString()  
 val passwordText = password.text.toString()  
  
  
 Log.d("Login", "Email: $emailText and Password: $passwordText")  
 //spinner.visibility = View.VISIBLE  
 logIn(email.text.toString(), password.text.toString())  
 }  
  
 //Go to Register Page  
 signUpBtn.setOnClickListener {  
 val intent = Intent(this, RegisterActivity::class.java)  
 startActivity(intent)  
 }  
 if (Build.VERSION.SDK\_INT >= 33) {  
 if (ContextCompat.checkSelfPermission(  
 this,  
 android.Manifest.permission.POST\_NOTIFICATIONS  
 ) == PackageManager.PERMISSION\_DENIED  
 ) {  
 ActivityCompat.requestPermissions(  
 this,  
 arrayOf(android.Manifest.permission.POST\_NOTIFICATIONS),  
 1  
 )  
 }  
 }  
 }  
  
 override fun onStart() {  
 super.onStart()  
 val currentUser = auth.currentUser  
 if (currentUser != null){  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 }  
 }  
 private fun signInGoogle(){  
 //spinner.visibility = View.VISIBLE  
 Log.d("Login", "Logging with Google")  
 val signInIntent: Intent = mGoogleSignInClient.signInIntent  
 startActivityForResult(signInIntent,Req\_Code)  
 }  
 // onActivityResult() function : this is where we provide the task and data for the Google Account  
 override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
 super.onActivityResult(requestCode, resultCode, data)  
 if(requestCode==Req\_Code){  
 val task: Task<GoogleSignInAccount> = GoogleSignIn.getSignedInAccountFromIntent(data)  
 handleResult(task)  
 }  
 else if (requestCode == NOTIFICATION\_PERMISSION\_REQUEST\_CODE) {  
 // Check if the user granted notification permission  
 val notificationManager = getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager  
 val areNotificationsEnabled = if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.S) {  
 notificationManager.areBubblesAllowed()  
 } else {  
 NotificationManagerCompat.from(this).areNotificationsEnabled()  
 }  
  
 if (areNotificationsEnabled) {  
 // Notification permission granted  
 } else {  
 if (Build.VERSION.SDK\_INT >= 33) {  
 if (ContextCompat.checkSelfPermission(this, android.Manifest.permission.POST\_NOTIFICATIONS) == PackageManager.PERMISSION\_DENIED) {  
 ActivityCompat.requestPermissions(this, arrayOf(android.Manifest.permission.POST\_NOTIFICATIONS), 1)  
 }  
 }  
 }  
 }  
 }  
 // handleResult() function - this is where we update the UI after Google signin takes place  
 private fun handleResult(completedTask: Task<GoogleSignInAccount>){  
 try {  
 val account: GoogleSignInAccount? =completedTask.getResult(ApiException::class.java)  
 if (account != null) {  
 UpdateUI(account)  
 }  
 } catch (e: ApiException){  
 Toast.makeText(applicationContext, e.localizedMessage, Toast.LENGTH\_LONG).show()  
 val animationView = findViewById<LottieAnimationView>(R.id.lottie\_animation\_view)  
 animationView.visibility = View.GONE  
 }  
 }  
 private fun UpdateUI(account: GoogleSignInAccount){  
 val credential= GoogleAuthProvider.getCredential(account.idToken,null)  
 auth.signInWithCredential(credential).addOnCompleteListener {task-> if(task.isSuccessful) {  
 val intent = Intent(this, MainActivity::class.java)  
 val animationView = findViewById<LottieAnimationView>(R.id.lottie\_animation\_view)  
 animationView.visibility = View.GONE  
 startActivity(intent)  
 Log.d("Logging in", "Successfully logged in with Google")  
 }  
 }  
 }  
 private fun logIn(email:String, password: String) {  
 if(email != "" && password != ""){  
 auth.signInWithEmailAndPassword(email,password).addOnCompleteListener { task -> if(task.isSuccessful){  
 val intent = Intent(this, MainActivity::class.java)  
 val animationView = findViewById<LottieAnimationView>(R.id.lottie\_animation\_view)  
 animationView.visibility = View.GONE  
 startActivity(intent)  
 }  
 }.addOnFailureListener { exception -> Toast.makeText(applicationContext, exception.localizedMessage, Toast.LENGTH\_LONG).show()  
 val animationView = findViewById<LottieAnimationView>(R.id.lottie\_animation\_view)  
 animationView.visibility = View.GONE  
 }  
 }else{  
 showAlert("Credentials", "Please enter your credentials")  
 }  
  
  
 }  
 private fun showAlert(title: String, message: String) {  
 val builder = AlertDialog.Builder(this)  
 builder.setTitle(title)  
 builder.setMessage(message)  
 builder.setPositiveButton("OK") { dialog, \_ ->

dialog.dismiss()  
 }  
 val dialog = builder.create()  
 dialog.show()  
 }  
}

class RegisterActivity : AppCompatActivity() {  
  
 private var auth: FirebaseAuth = FirebaseAuth.getInstance()  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_register)  
  
 val loginBtn = findViewById<Button>(R.id.loginBtn)  
 val registerBtn = findViewById<Button>(R.id.registerBtn)  
 val email = findViewById<EditText>(R.id.emailText)  
 val password = findViewById<EditText>(R.id.passwordText)  
  
 loginBtn.setOnClickListener {  
 val intent = Intent(this, LoginActivity::class.java)  
 startActivity(intent)  
 }  
  
 registerBtn.setOnClickListener {  
 if(email.text.toString() != "" && password.text.toString() != ""){  
 auth.createUserWithEmailAndPassword(email.text.toString(), password.text.toString())  
 .addOnCompleteListener { task -> if (task.isSuccessful) {  
 // Sign in success, update UI with the signed-in user's information  
 val user = auth.currentUser  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 } else {  
 // If sign in fails, display a message to the user.  
 println("An error occured")  
 }  
 }.addOnFailureListener { exception -> exception.localizedMessage?.let { it1 -> showAlert("Error", it1) }  
 }  
 }else{  
 showAlert("Credentials", "Please enter your credentials")  
 }  
 }  
  
 }  
 private fun showAlert(title: String, message: String) {  
 val builder = AlertDialog.Builder(this)  
 builder.setTitle(title)  
 builder.setMessage(message)  
 builder.setPositiveButton("OK") { dialog, \_ -> dialog.dismiss()  
 }  
 val dialog = builder.create()  
 dialog.show()  
 }  
}

## Κύριες Λειτουργίες Εφαρμογής

Στη συγκεκριμένη ενότητα θα αναφερθώ στον τρόπο που υλοποιήθηκε το σύνολο των δυνατοτήτων της εφαρμογής. Η εφαρμογή χρησιμοποιεί τη Firebase Realtime database για την αποθήκευση των συμβάντων και την ανάγνωσή τους από άλλους χρήστες.

### Firebase Realtime Database

Database Schema



Εικόνα 4: Σχήμα βάσης δεδομένων

Η βάση δεδομένων χωρίζεται στα παρακάτω children:

* **Admin:** κρατά αποθηκευμένα τα uid των υπαλλήλων της πολιτικής προστασίας ώστε να εκτελείται έλεγχος για το ρόλο του κάθε χρήστη και να προσδίδει επιπλέον δυνατότητες στους υπαλλήλους της πολιτικής προστασίας.
* **Notification:** αποθηκεύονται πληροφορίες για συμβάντα που υποβάλλονται από χρήστες ώστε να ενημερώνεται ο υπάλληλος πολιτικής προστασίας μέσω ειδοποίησης κατά την υποβολή αυτών των συμβάντων.
* **Pending:** αποθηκεύονται τα συμβάντα που έχουν υποβληθεί από τους χρήστες και δεν έχουν ακόμα αξιολογηθεί από τους υπαλλήλους της πολιτικής προστασίας ώστε να δοθεί έγκριση ή να απορριφθεί το συμβάν.
* **Statistics:** αποθηκεύονται οι καταστάσεις εκτάκτου κινδύνου που έχουν εγκριθεί από τους υπαλλήλους πολιτικής προστασίας ώστε να εμφανίζονται στο ανάλογο Activity προβολής στατιστικών.
* **Active:** αποθηκεύονται οι καταστάσεις εκτάκτου κινδύνου που έχουν εγκριθεί από τους υπαλλήλους πολιτικής προστασίας ώστε να ενημερώνονται οι πολίτες που βρίσκονται κοντά στο συμβάν μέσω άμεσης ειδοποίησης.

Database Rules

Κατά τη δημιουργία της βάσης δεδομένων, δόθηκε ιδιαίτερη έμφαση στους κανόνες που πρέπει να τη διέπουν. Συγκεκριμένα, πρέπει τα δεδομένα να προστατεύονται έτσι ώστε:

* Τα δεδομένα να μην είναι προσβάσιμα σε άτομα που δε χρειάζονται να διαβάσουν την πληροφορία.
* Η προστασία των δεδομένων από τους χρήστες έτσι ώστε να μην αλλοιώνονται από άτομα που δε θα έπρεπε να έχουν πρόσβαση σε αυτά.

1. {

2. "rules": {

3. "pending": {

4. ".read": "auth != null",

5. ".write": "auth != null"

6. },

7. "active": {

8. ".read": "auth != null",

9. "$uid": {

10. ".write": "auth.uid == 'BYkSwwU7JsRmmNezWUcOYqYEdaS2'"

11. }

12. },

13. "admin": {

14. ".read": "auth !=null",

15. ".write": "false"

16. },

17. "notification": {

18. ".read": "auth.uid == 'BYkSwwU7JsRmmNezWUcOYqYEdaS2'",

19. ".write": "auth != null"

20. },

21. "statistics": {

22. ".read": "auth != null",

23. ".write" : "auth != null"

24. }

25. }

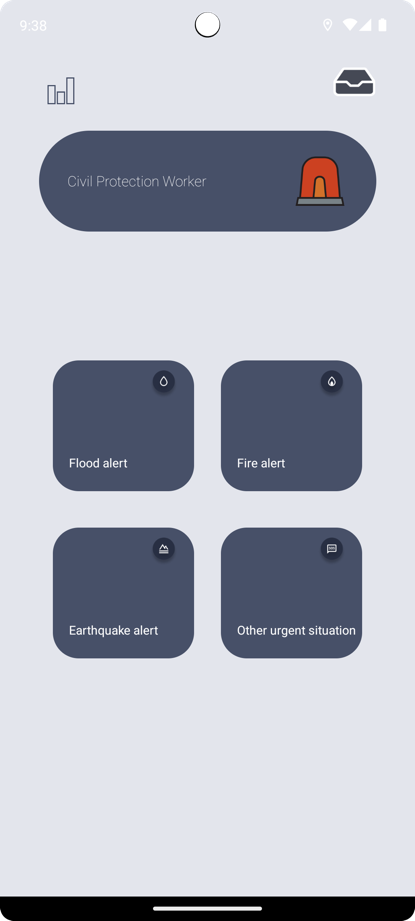
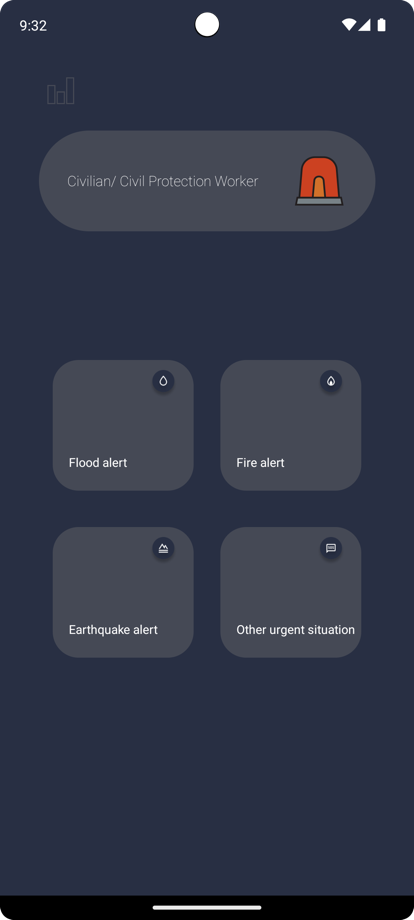
26. }

27.

Σε κάθε περίπτωση, για την πρόσβαση στα δεδομένα απαιτείται ο χρήστης να έχει συνδεθεί στην εφαρμογή. Επιπλέον αυτού, στο ‘active’ child μόνο ο υπάλληλος της πολιτικής προστασίας έχει τη δυνατότητα να εγγράψει πληροφορία, ενώ στο ‘admin’ κανένας δεν μπορεί να εγγράψει πληροφορία (νέους υπαλλήλους πολιτικής προστασίας) καθώς αυτοί προστίθεται μόνο από τον super admin. Τα ‘notification’ δύναται να διαβαστούν μόνο από τον υπάλληλο της πολιτικής προστασίας.

### Αναφορά Συμβάντος από Πολίτη

Η αναφορά των επιμέρους συμβάντων από τους πολίτες εκτελούνται μέσω του Main Activity.

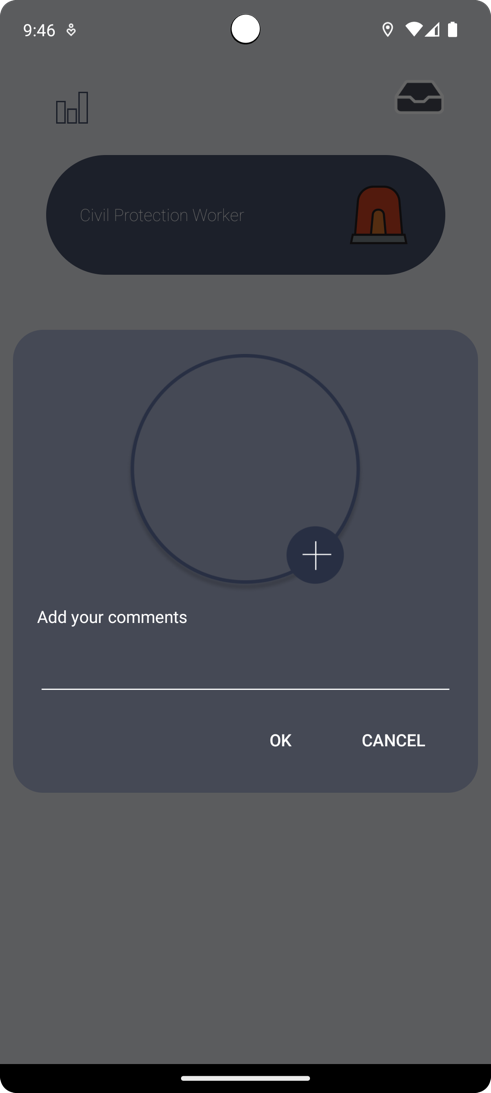
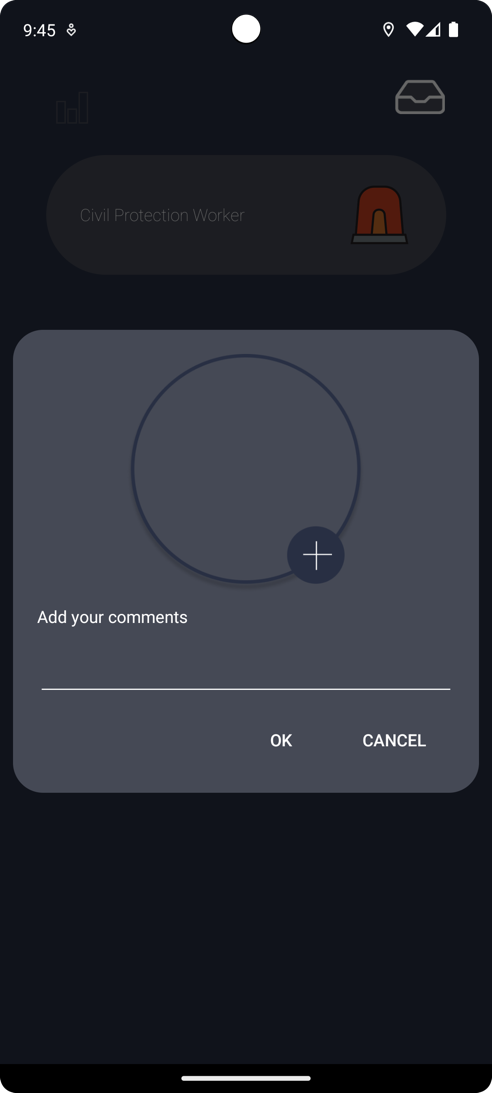
Εικόνα 5: Main Activity - light theme Εικόνα 6: Main Activity - dark theme

Όπως βλέπουμε στα ανωτέρω screenshot, τα συμβάντα που μπορεί να αναφέρει ο πολίτης έχουν ήδη καθοριστεί από την εφαρμογή και χωρίζονται στις ακόλουθες κατηγορίες:

* Πυρκαγιά
* Πλημμύρα
* Σεισμός
* Άλλη κατάσταση ανάγκης

Άνωθεν των συμβάντων υπάρχει ένα banner που ενημερώνει τον χρήστη για το ρόλο που έχει στην εφαρμογή. Οι ρόλος που μπορεί να έχει ο χρήστης είναι είτε πολίτης είτε υπάλληλος πολιτικής προστασίας.

Άνωθεν αυτού, υπάρχουν δύο button. Το αριστερό μεταφέρει τον χρήστη στο Activity προβολής των στατιστικών η οποία είναι προσβάσιμη και στους δύο χρήστες. Το δεξί button εμφανίζεται μόνο σε χρήστη που έχει ρόλο υπαλλήλου πολιτικής προστασίας και χρησιμεύει ως «inbox» ώστε να μπορεί να προβάλλει όλες τις καταστάσεις κινδύνου που έχουν αναφερθεί από τους πολίτες. Εάν υπάρχει κάποιο συμβάν το οποίο βρίσκεται σε αναμονή (pending), τότε το button γίνεται κόκκινο για να τραβήξει την προσοχή του υπαλλήλου πολιτικής προστασίας.

Εικόνα 7: Αναφορά συμβάντος - light theme Εικόνα 8: Αναφορά συμβάντος - dark theme

Με τη χρήση κάθε κομβίου αναφοράς κατάστασης κινδύνου, η εφαρμογή λαμβάνει αυτόματα τη θέσει του χρήστη και εμφανίζει το ανωτέρω alert ώστε αυτός να προσθέσει πιθανή εικόνα και κάποια σχόλια σχετικά με το συμβάν.

Αφού ο χρήστης αναφέρει επιτυχώς το συμβάν, εμφανίζεται ένα μήνυμα που τον ενημερώνει.



Εικόνα 9: Επιτυχής αναφορά συμβάντος

Επίσης, στην ανωτέρω εικόνα μπορούμε να δούμε και την αλλαγή στο χρώμα του inbox button η οποία αναφέρθηκε προηγουμένως. Όπως βλέπουμε στον κώδικα που παρατίθεται, ελέγχουμε αρχικά εάν ο χρήστης έχει επιτρέψει την πρόσβαση στην τοποθεσία του και στη συνέχεια η εφαρμογή μας λαμβάνει την τοποθεσία του. Εάν εντός 5 δευτερολέπτων η εφαρμογή δεν καταφέρει να λάβει την τοποθεσία της συσκευής, εμφανίζεται ένα μήνυμα στο χρήστη.

private fun getLocation(view: View, database: DatabaseReference, path: String, animationView: LottieAnimationView) {  
 if (ActivityCompat.checkSelfPermission(  
 this,  
 android.Manifest.permission.ACCESS\_FINE\_LOCATION ) != PackageManager.PERMISSION\_GRANTED && ActivityCompat.checkSelfPermission(  
 this,  
 android.Manifest.permission.ACCESS\_COARSE\_LOCATION ) != PackageManager.PERMISSION\_GRANTED ) {  
 // Request permission for location  
 ActivityCompat.requestPermissions(  
 this,  
 arrayOf(Manifest.permission.ACCESS\_FINE\_LOCATION),  
 LOCATION\_PERMISSION\_REQUEST\_CODE  
 )  
 return  
 }  
  
 val locationRequest = LocationRequest.create().apply {  
 priority = LocationRequest.PRIORITY\_HIGH\_ACCURACY  
 interval = 500 // Update interval in milliseconds  
 fastestInterval = 5000 // Fastest update interval in milliseconds  
 }  
  
 val locationCallback = object : LocationCallback() {  
 override fun onLocationResult(p0: LocationResult) {  
 super.onLocationResult(p0)  
 fusedLocationClient.removeLocationUpdates(this) // Stop receiving location updates  
  
 if (p0.locations.isNotEmpty()) {  
 val location = p0.locations.first()  
 userLocation = location  
  
 countryCityPair = getAddress(userLocation!!)  
  
 loadData(database.child("pending").child(path).child(countryCityPair.first!!).child(countryCityPair.second!!), view, path, animationView)  
 } else {  
 // Throw an error or show a message indicating that the location could not be obtained  
 Toast.makeText(this@MainActivity, "Unable to get current location", Toast.LENGTH\_SHORT).show()  
 }  
 }  
 }  
  
 // Timer to stop the function if 5 seconds have passed  
 timeoutHandler = Handler(Looper.getMainLooper())  
 timeoutRunnable = Runnable {  
 // Throw an error or show a message indicating that the location could not be obtained within the timeout  
 Toast.makeText(this@MainActivity, "Timeout: Unable to get current location", Toast.LENGTH\_SHORT).show()  
 fusedLocationClient.removeLocationUpdates(locationCallback) // Stop location updates  
 return@Runnable  
 }  
 timeoutHandler.postDelayed(timeoutRunnable, 5000) // 5 seconds timeout  
  
 fusedLocationClient.requestLocationUpdates(locationRequest, locationCallback, Looper.getMainLooper())  
}

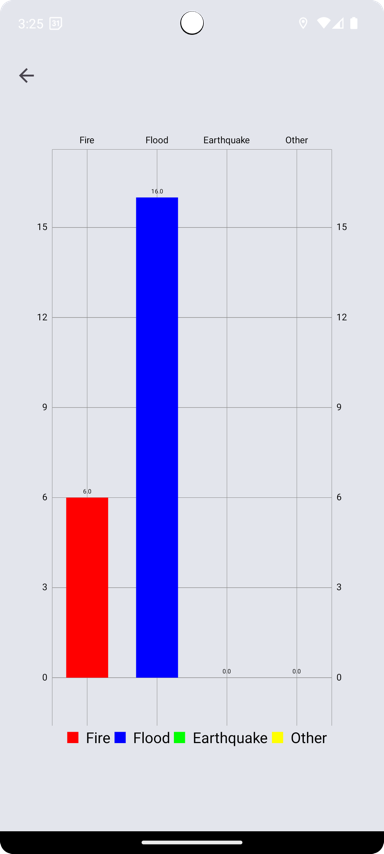
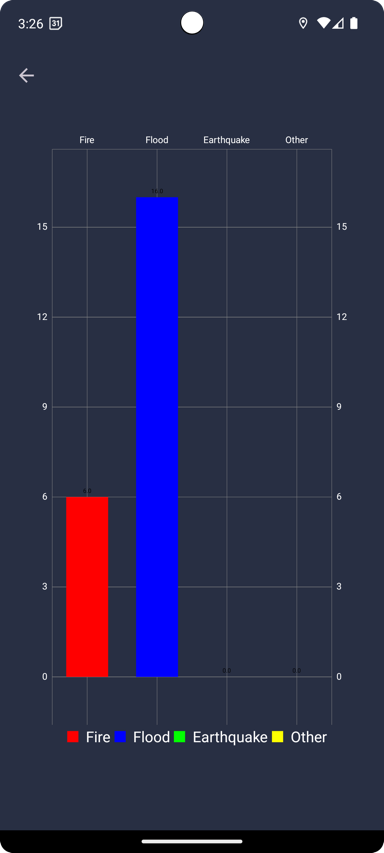
Αφού ληφθεί η τοποθεσία, βρίσκουμε σε ποια πόλη και χώρα βρίσκεται ο χρήστης ώστε να έχουμε έναν καλύτερο καταμερισμό και οργάνωση στη βάση δεδομένων μας. Στη συνέχεια, εκτελείται ένα function ώστε να ελέγξουμε εάν στη βάση δεδομένων υπάρχει κάποιο συμβάν που έχει ήδη αναφερθεί σε συγκεκριμένη απόσταση και χρόνο από το νέο περιστατικό που επιθυμεί ο χρήστης να αναφέρει. Ενδεικτικά, χρησιμοποιήθηκε μία απόσταση 2000μ και χρόνος 2 ωρών από την πρώτη αναφορά. Αυτές οι παραμέτροι θα μπορούσαν να αλλάξουν αναλόγως των απαιτήσεων της πολιτικής προστασίας είτε για όλα τα περιστατικά είτε για καθένα ξεχωριστά.

private fun loadData(databaseRef: DatabaseReference, view: View, path:String, animationView: LottieAnimationView) {  
 var nearestDistance = Double.MAX\_VALUE var nearestChildSnapshot: DataSnapshot? = null timeoutHandler.removeCallbacks(timeoutRunnable)  
 databaseRef.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
 for (childSnapshot in dataSnapshot.children) {  
 val childKey: String = childSnapshot.key ?: "" val childValue: Any? = childSnapshot.value  
  
 val latitude: Double? = childSnapshot.child("latitude").getValue(Double::class.java)  
 val longitude: Double? = childSnapshot.child("longitude").getValue(Double::class.java)  
 val time: String? = childSnapshot.child("timestamp").getValue(String::class.java)  
  
 val position = LatLng(latitude!!, longitude!!)  
 val distance = calculateDistance(userLocation!!, position)  
  
 if (distance < 2000 && calculateTimeDifference(time!!) < 7.2e+6) {  
 if (distance < nearestDistance) {  
 nearestDistance = distance.toDouble()  
 nearestChildSnapshot = childSnapshot }  
 }  
  
 Log.d("Distance", distance.toString())  
 Log.d("Time Difference", calculateTimeDifference(time!!).toString())  
 }  
  
 // Invoke showAlertDialogButtonClicked with the nearest child snapshot, if available  
 if (nearestChildSnapshot != null) {  
 snapKey = nearestChildSnapshot!!.key ?: ""  
 snapCount = nearestChildSnapshot!!.child("count").getValue(Long::class.java)  
 Log.d("SnapKey", snapKey)  
  
 animationView.visibility = View.GONE  
 showAlertDialogButtonClicked(view, database, snapKey, path)  
 }else{  
 animationView.visibility = View.GONE  
 showAlertDialogButtonClicked(view, database, "", path)  
 }  
 }  
  
 override fun onCancelled(databaseError: DatabaseError) {  
 // Handle any errors that occur  
 Log.e("Database Error", databaseError.message)  
 }  
 })  
}

Μετά από αυτή τη διεργασία, εμφανίζεται το παράθυρο εισαγωγής πληροφοριών από τον χρήστη το οποίο λαμβάνει τις πληροφορίες. Εάν υπάρχει ήδη αναφορά του συγκεκριμένου συμβάντος, προσθέτει τις επιπλέον πληροφορίες σε αυτό ενώ εάν πρόκειται για καινούριο συμβάντος, δημιουργείται ένα νέο child που περιέχει τις πληροφορίες από τον χρήστη.

fun showAlertDialogButtonClicked(view: View?, database: DatabaseReference, snapKey: String, path:String) {  
 // Inflate the custom alert layout  
 val inflater = layoutInflater  
 val customLayout = inflater.inflate(R.layout.custom\_alert, null)  
 val positiveBtn = customLayout.findViewById<Button>(R.id.positiveBtn)  
 val negativeBtn = customLayout.findViewById<Button>(R.id.negativeBtn)  
 val descriptionText = customLayout.findViewById<EditText>(R.id.commentsEditText)  
 imageBtn = customLayout.findViewById<ImageButton>(R.id.imageBtn)  
 imageUploaded = customLayout.findViewById(R.id.imageUploaded)  
  
 // Create and show the alert dialog  
 val alertDialogBuilder = AlertDialog.Builder(this)  
 alertDialogBuilder.setView(customLayout)  
 alertDialogBuilder.setCancelable(false)  
 val alertDialog = alertDialogBuilder.create()  
  
 alertDialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))  
 positiveBtn.setOnClickListener {  
  
 val timestamp = Date() // Get the current timestamp as a Date object  
  
 // Format the timestamp as a human-readable string  
 val dateFormat = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault())  
 val timestampString = dateFormat.format(timestamp)  
  
 val data = HashMap<String, Any>()  
 val notificationData = HashMap<String, Any>()  
 if (userLocation != null && snapKey == "") {  
 data["longitude"] = userLocation!!.longitude  
 data["latitude"] = userLocation!!.latitude  
 data["description"] = descriptionText.text.toString()  
 data["count"] = 1  
 data["timestamp"] = timestampString  
  
 val dataRef =  
 database.child("pending").child(path).child(countryCityPair.first!!)  
 .child(countryCityPair.second!!).push()  
 dataRef.setValue(data)  
  
 notificationData["longitude"] = userLocation!!.longitude  
 notificationData["latitude"] = userLocation!!.latitude  
 notificationData["description"] = descriptionText.text.toString()  
 notificationData["count"] = 1  
 notificationData["timestamp"] = timestampString  
 notificationData["country"] = countryCityPair.first!!  
 notificationData["city"] = countryCityPair.second!!  
 notificationData["type"] = path  
 val notificationRef = database.child("notification").push()  
 notificationRef.setValue(notificationData)  
  
  
 notificationRef.child("description").setValue(data["description"])  
  
 dataRef.child("description").push().setValue(data["description"])  
 dataRef.child("key").setValue(notificationRef.key)  
 if (imageUrl.isNotEmpty()) {  
 dataRef.child("photos").push().setValue(imageUrl)  
  
 notificationRef.child("photos").push().setValue(imageUrl)  
 } else if (imageUrl.isEmpty()) {  
  
 }  
 sendDialogDataToActivity(getString(R.string.uploaded\_successfully))  
 alertDialog.dismiss()  
 }else if (userLocation != null && snapKey.isNotEmpty() && descriptionText.text.toString() != ""){  
 snapCount = snapCount?.plus(1) ?: 1  
  
 database.child("pending").child(path).child(countryCityPair.first!!).child(countryCityPair.second!!)  
 .child(snapKey)  
 .child("description").push().setValue(descriptionText.text.toString())  
 database.child("pending").child(path).child(countryCityPair.first!!).child(countryCityPair.second!!)  
 .child(snapKey)  
 .child("count").setValue(snapCount)  
 if (imageUrl.isNotEmpty()){  
 database.child("pending").child(path).child(countryCityPair.first!!).child(countryCityPair.second!!)  
 .child(snapKey)  
 .child("photos").setValue(imageUrl)  
 }  
 alertDialog.dismiss()  
 sendDialogDataToActivity("Uploaded successfully!")  
  
 }  
 else {  
 Toast.makeText(this, "Please try again later", Toast.LENGTH\_LONG).show()  
 }  
 }  
  
  
 negativeBtn.setOnClickListener {  
 alertDialog.dismiss()  
 }  
 imageBtn.setOnClickListener {  
 uploadPhoto(view!!)  
 }  
 // Set the background of imageBtn  
 if (imageUri != null) {  
 val bitmap = MediaStore.Images.Media.getBitmap(contentResolver, imageUri)  
 val drawable = BitmapDrawable(resources, bitmap)  
 imageBtn.background = drawable  
 }  
  
 alertDialog.show()  
  
}

### Αctivity προβολής στατιστικών

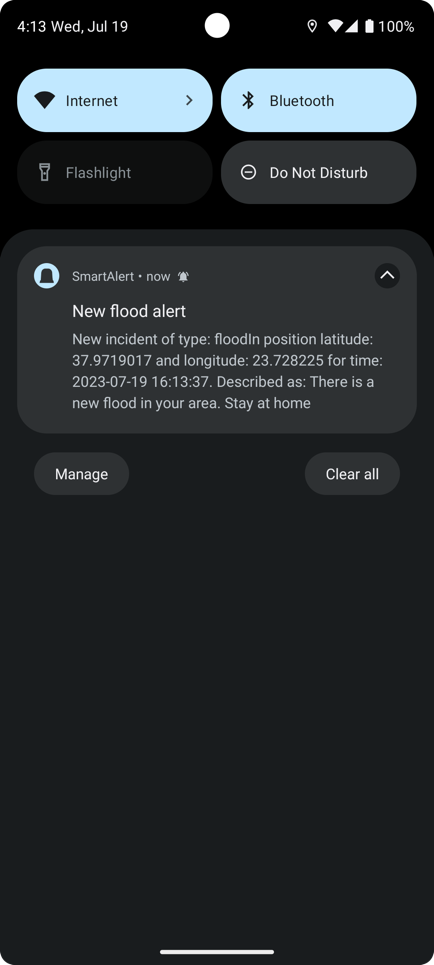
Εικόνα 0: Προβολή στατιστικών - dark theme Εικόνα 11: Προβολή στατιστικών - light theme

Στο συγκεκριμένο Activity γίνεται εμφάνιση στατιστικών για τις ειδοποιήσεις που έχουν σταλεί από την πολιτική προστασία βάσει κατηγορίας συμβάντος.

Οι πληροφορίες αυτές λαμβάνονται από ένα child στατιστικών που περιέχει πληροφορίες σχετικά με κάθε συμβάν για το οποίο έχει δημιουργηθεί ειδοποίηση από τον υπάλληλο της πολιτικής προστασίας.

class StatisticsActivity : AppCompatActivity() {  
 private lateinit var barChart: BarChart private lateinit var legend: Legend private lateinit var databaseReference: DatabaseReference  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_statistics)  
  
 val toolbar: Toolbar = findViewById(R.id.toolbarStat)  
 setSupportActionBar(toolbar)  
 supportActionBar?.title = "" supportActionBar?.setDisplayHomeAsUpEnabled(true)  
  
 barChart = findViewById(R.id.idBarChart)  
 legend = barChart.legend legend.isEnabled = true legend.textSize = 16f legend.formSize = 12f legend.formToTextSpace = 8f legend.horizontalAlignment = Legend.LegendHorizontalAlignment.CENTER legend.verticalAlignment = Legend.LegendVerticalAlignment.BOTTOM legend.orientation = Legend.LegendOrientation.HORIZONTAL  
  
 val entries = listOf("Fire", "Flood", "Earthquake", "Other")  
 val colors = listOf(Color.RED, Color.BLUE, Color.GREEN, Color.YELLOW)  
  
 val legendEntries = ArrayList<LegendEntry>()  
 for (i in entries.indices) {  
 val entry = LegendEntry()  
 entry.label = entries[i]  
 entry.formColor = colors[i]  
 legendEntries.add(entry)  
 }  
  
 legend.setCustom(legendEntries)  
  
 databaseReference = FirebaseDatabase.getInstance().reference.child("statistics")  
  
 val barEntriesList = ArrayList<BarEntry>()  
  
 val childValueEventListener = object : ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
 var fireCount = 0 var floodCount = 0 var earthquakeCount = 0 var otherCount = 0  
  
 for (childSnapshot in dataSnapshot.children) {  
 val type: String? = childSnapshot.child("type").getValue(String::class.java)  
  
 if (type == "fire") {  
 fireCount += 1 } else if (type == "flood") {  
 floodCount += 1 } else if (type == "earthquake") {  
 earthquakeCount += 1 } else {  
 otherCount += 1 }  
 }  
  
 barEntriesList.add(BarEntry(0f, fireCount.toFloat()))  
 barEntriesList.add(BarEntry(1f, floodCount.toFloat()))  
 barEntriesList.add(BarEntry(2f, earthquakeCount.toFloat()))  
 barEntriesList.add(BarEntry(3f, otherCount.toFloat()))  
  
 val barDataSet = BarDataSet(barEntriesList, "Disaster Types")  
 barDataSet.setColors(colors)  
  
 val barData = BarData(barDataSet)  
 barData.barWidth = 0.6f // Set X and Y label colors to white  
 if(isDarkModeEnabled(applicationContext)) {  
 barChart.xAxis.textColor = Color.WHITE  
 barChart.axisLeft.textColor = Color.WHITE  
 barChart.axisRight.textColor = Color.WHITE  
 legend.textColor = Color.WHITE  
 }else{  
 barChart.xAxis.textColor = Color.BLACK  
 barChart.axisLeft.textColor = Color.BLACK  
 barChart.axisRight.textColor = Color.BLACK  
 legend.textColor = Color.BLACK  
 }  
  
  
 barChart.data = barData  
 barChart.setFitBars(true)  
 barChart.xAxis.valueFormatter = IndexAxisValueFormatter(entries)  
 barChart.xAxis.granularity = 1f  
 barChart.description.isEnabled = false  
 barChart.invalidate()  
 }  
  
 override fun onCancelled(databaseError: DatabaseError) {  
 Log.e("ChildValueListener", "Error: ${databaseError.message}")  
 }  
 }  
  
 databaseReference.addListenerForSingleValueEvent(childValueEventListener)  
 }  
  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
 when (item.itemId) {  
 android.R.id.home -> {  
 val intent = Intent(this, MainActivity::class.java)  
 startActivity(intent)  
 }  
 }  
 return super.onOptionsItemSelected(item)  
 }  
 private fun getColorFromAttribute(attrId: Int): Int {  
 val typedValue = TypedValue()  
 theme.resolveAttribute(attrId, typedValue, true)  
 return typedValue.data  
 }  
 fun isDarkModeEnabled(context: Context): Boolean {  
 val currentNightMode = context.resources.configuration.uiMode and Configuration.UI\_MODE\_NIGHT\_MASK  
 return currentNightMode == Configuration.UI\_MODE\_NIGHT\_YES  
 }  
  
}

### Εμφάνιση ειδοποιήσεων



Εικόνα 1: Εμφάνιση ειδοποίησης σε χρήστη

Αφού δημιουργηθεί μία ειδοποίηση από τον χρήστη της πολιτικής προστασίας, αυτή αποστέλλεται αυτόματα στους χρήστες που βρίσκονται κοντά στο περιστατικό με πληροφορίες όπως φαίνονται στην παραπάνω εικόνα. Οι πληροφορίες που εισάγει ο υπάλληλος είναι η περιγραφή και πληροφορίες για συμβάν τόσο στα ελληνικά όσο και στα αγγλικά καθώς και πιθανή εικόνα. Οι λοιπές πληροφορίες λαμβάνονται αυτόματα από το σύστημα.

class UserListener : Service() {  
  
 private lateinit var databaseReference: DatabaseReference private lateinit var childEventListener: ChildEventListener private var isInitialDataRetrieved = false private lateinit var locationManager: LocationManager private lateinit var userLocationListener: UserLocationListener  
  
  
 override fun onCreate() {  
 super.onCreate()  
 val currentLocale = getCurrentLocale()  
 databaseReference = FirebaseDatabase.getInstance().reference.child("active")  
 // Create the location manager and listener  
 locationManager = getSystemService(Context.LOCATION\_SERVICE) as LocationManager  
 userLocationListener = UserLocationListener()  
  
 // Request location updates  
 if (ActivityCompat.checkSelfPermission(  
 this,  
 Manifest.permission.ACCESS\_FINE\_LOCATION  
 ) == PackageManager.PERMISSION\_GRANTED  
 ) {  
 locationManager.requestLocationUpdates(  
 LocationManager.GPS\_PROVIDER,  
 0L,  
 0F,  
 userLocationListener  
 )  
 }  
  
 childEventListener = object : ChildEventListener {  
 override fun onChildAdded(dataSnapshot: DataSnapshot, previousChildName: String?) {  
 Log.d("User", "started listening")  
 if (isInitialDataRetrieved) {  
 Log.d("Snapshot", dataSnapshot.toString())  
 var type: String? = dataSnapshot.child("type").getValue(String::class.java)  
 val latitude: Double? = dataSnapshot.child("latitude").getValue(Double::class.java)  
 val longitude: Double? = dataSnapshot.child("longitude").getValue(Double::class.java)  
 val time: String? = dataSnapshot.child("timestamp").getValue(String::class.java)  
 val description: String?  
 val position = LatLng(latitude!!, longitude!!)  
 val userLocation = getUserLocation()  
 val distance = calculateDistance(userLocation!!, position)  
 Log.d("User", "Distance : $distance")  
 if(distance < 10000) {  
 if (currentLocale.language == "el"){  
 if (type == "fire"){  
 type = "φωτιάς"  
 }else if (type == "flood"){  
 type = "πλημμύρας"  
 }else if (type == "earthquake"){  
 type = "σεισμού"  
 }else{  
 type = "άλλη"  
 }  
  
 description = dataSnapshot.child("GR").getValue(String::class.java)  
 val message: String = "Νέα κατάσταση κινδύνου $type" +  
 " σε θέση πλάτους: $latitude και μήκους: $longitude για χρόνο: $time. Με περιγραφή: $description"  
  
 Log.d("LATlon", latitude.toString() + longitude.toString() +LatLng(latitude, longitude).toString() )  
 createNotification(applicationContext, message, "Νέα προειδοποίηση $type", LatLng(latitude, longitude), LatLng(userLocation.latitude, userLocation.longitude))  
  
 }else{  
 description = dataSnapshot.child("EN").getValue(String::class.java)  
 val message: String = "New incident of type: $type" +  
 " In position latitude: $latitude and longitude: $longitude for time: $time. Described as: $description"  
 Log.d("LATlon", message)  
  
 createNotification(applicationContext, message, "New $type alert", LatLng(latitude, longitude), LatLng(userLocation.latitude, userLocation.longitude))  
 }  
 }  
 }  
 }  
  
 override fun onChildChanged(snapshot: DataSnapshot, previousChildName: String?) {  
 // Handle child changed event  
 }  
  
 override fun onChildRemoved(snapshot: DataSnapshot) {  
 // Handle child removed event  
 }  
  
 override fun onChildMoved(snapshot: DataSnapshot, previousChildName: String?) {  
 // Handle child moved event  
 }  
  
 override fun onCancelled(error: DatabaseError) {  
 // Handle database error  
 }  
 }  
  
 databaseReference.addChildEventListener(childEventListener)  
  
 // Attach a ValueEventListener to listen for initial data retrieval  
 val valueEventListener = object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 // Set the flag to indicate initial data retrieval is complete  
 isInitialDataRetrieved = true  
 }  
  
 override fun onCancelled(error: DatabaseError) {  
 // Handle database error  
 }  
 }  
  
 databaseReference.addListenerForSingleValueEvent(valueEventListener)  
  
 }  
  
 override fun onBind(intent: Intent): IBinder? {  
 return null  
 }  
  
 override fun onDestroy() {  
 super.onDestroy()  
 // Remove the value event listener when the service is destroyed  
 databaseReference.removeEventListener(childEventListener)  
 // Stop receiving location updates  
 locationManager.removeUpdates(userLocationListener)  
 }  
 fun createNotification(context: Context, message: String, title: String, incidentLatLng: LatLng, userPosition: LatLng) {  
 val channelId = "your\_channel\_id"  
 val notificationId = 1  
  
 // Create an Intent  
 val intent = Intent(context, MapsActivity::class.java)  
 intent.putExtra("incidentLatLng", incidentLatLng)  
 intent.putExtra("userPosition", userPosition)  
 intent.putExtra("message", message)  
 intent.flags = Intent.FLAG\_ACTIVITY\_NEW\_TASK or Intent.FLAG\_ACTIVITY\_CLEAR\_TASK  
 // Add any extras to the intent if needed  
  
 // Create a PendingIntent with FLAG\_UPDATE\_CURRENT  
 val pendingIntent = PendingIntent.getActivity(  
 context,  
 0,  
 intent,  
 PendingIntent.FLAG\_IMMUTABLE  
 )  
  
 // Create a notification channel for Android Oreo and above  
 if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {  
 val channel = NotificationChannel(  
 channelId,  
 "Channel Name",  
 NotificationManager.IMPORTANCE\_HIGH  
 )  
 val notificationManager =  
 context.getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager  
 notificationManager.createNotificationChannel(channel)  
 }  
  
 // Build the notification with BigTextStyle and set the content intent  
 val notificationBuilder = NotificationCompat.Builder(context, channelId)  
 .setSmallIcon(R.drawable.alert\_svgrepo\_com)  
 .setContentTitle(title)  
 .setStyle(NotificationCompat.BigTextStyle().bigText(message))  
 .setPriority(NotificationCompat.PRIORITY\_HIGH)  
 .setContentIntent(pendingIntent)  
 .setAutoCancel(true)  
  
 // Show the notification  
 val notificationManager =  
 context.getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager  
 notificationManager.notify(notificationId, notificationBuilder.build())  
 }  
  
 private fun getCurrentLocale(): Locale {  
 val configuration = resources.configuration  
 return if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.N) {  
 configuration.locales[0]  
 } else {  
 configuration.locale  
 }  
 }  
 fun getUserLocation(): Location? {  
 Log.d("Position", "getting user position")  
 return userLocationListener.getUserLocation()  
 }  
 // Function to calculate the distance between two positions  
 private fun calculateDistance(userLocation: Location, position: LatLng): Float {  
 val targetLocation = Location("")  
 targetLocation.latitude = position.latitude  
 targetLocation.longitude = position.longitude  
  
 return userLocation.distanceTo(targetLocation)  
 }  
}

Ο μηχανισμός λήψης των ειδοποιήσεων έχει δημιουργηθεί ως Service για να λειτουργεί και όταν η εφαρμογή βρίσκεται στο background. Εντός αυτού του service υπάρχει έλεγχος ώστε να υπολογίζεται η απόσταση μεταξύ του συμβάντος και της τρέχουσας τοποθεσίας του χρήστη ώστε να ενημερώνονται μόνο αυτοί που βρίσκονται πλησίον. Επίσης, εάν ο χρήστης πατήσει πάνω στην ειδοποίηση, μεταφέρεται σε ένα Map Activity που δείχνει τη θέση του καθώς και τη θέση του συμβάντος που βρίσκεται σε εξέλιξη.



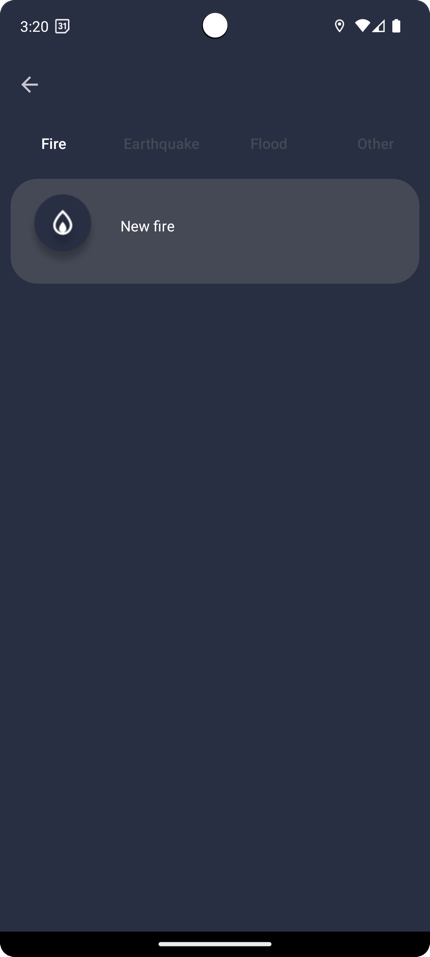
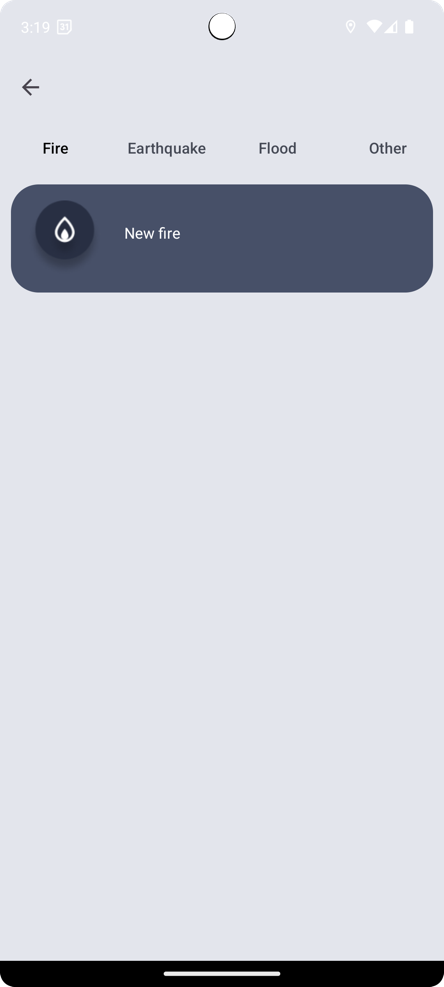
Εικόνα 1: Θέση του χρήστη και θέση συμβάντος

## Λειτουργίες υπαλλήλου πολιτικής προστασίας

### Εμφάνιση υποβληθέντων περιστατικών

Η εφαρμογή δίνει τη δυνατότητα στον υπάλληλο πολιτικής προστασίας να ενημερώνεται επί των συμβάντων που έχουν υποβληθεί από τους πολίτες, ανά κατηγορία συμβάντος. Επίσης, πατώντας σε κάθε συμβάν, μπορεί να δει όλες τις πληροφορίες για το συμβάν με σκοπό να αξιολογήσει τη σοβαρότητα και την αξιοπιστία του και να προχωρήσει είτε σε εκπομπή ειδοποίησης είτε σε απόρριψη του συγκεκριμένου συμβάντος. Η υλοποίηση έγινε μέσω ενός Recycler view.

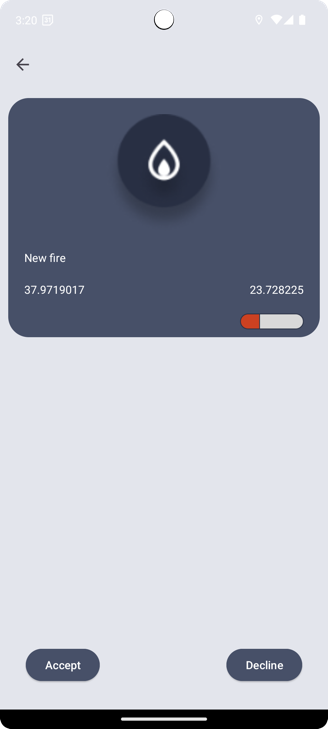
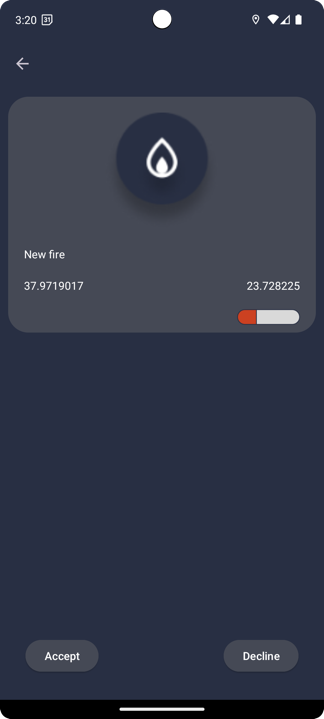
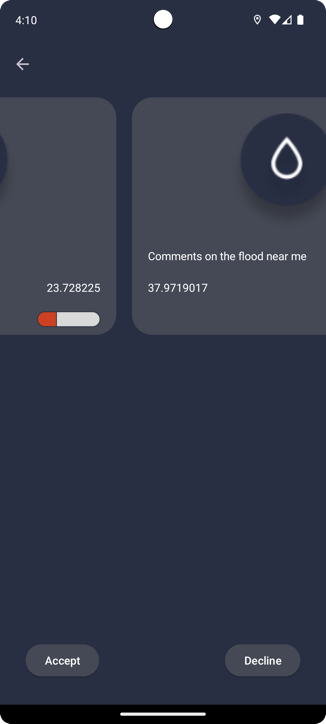
class PlaceholderFragment : Fragment() {  
  
 private lateinit var pageViewModel: PageViewModel private var \_binding: FragmentAdminBinding? = null private val fireDataList: MutableList<FirebaseData> = mutableListOf()  
 private val fireDrawable = drawable.fire\_button private val floodDrawable = drawable.flood\_button private val earthquakeDrawable = drawable.earthquake\_button private val otherDrawable = drawable.other\_button  
  
 // This property is only valid between onCreateView and  
 // onDestroyView.  
 private val binding get() = \_binding!!  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 pageViewModel = ViewModelProvider(this).get(PageViewModel::class.java).apply {  
 setIndex(arguments?.getInt(ARG\_SECTION\_NUMBER) ?: 1)  
 }  
  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
  
 \_binding = FragmentAdminBinding.inflate(inflater, container, false)  
 val root = binding.root  
 fireDataList.clear()  
  
 val recyclerView: RecyclerView = binding.recyclerView  
 val sectionNumber = arguments?.getInt(ARG\_SECTION\_NUMBER) ?: 1  
 // Inside PlaceholderFragment's onCreateView method  
  
 // Create a LinearLayoutManager  
 val layoutManager = LinearLayoutManager(requireContext())  
  
 // Set the layout manager for the RecyclerView  
 recyclerView.layoutManager = layoutManager  
 // Create an instance of your RecyclerViewAdapter with the retrieved item list  
 val firebaseAdapter = FiredataAdapter(fireDataList)  
  
 // Set the adapter for the RecyclerView  
 recyclerView.adapter = firebaseAdapter  
  
 when (sectionNumber) {  
 1 -> getData(firebaseAdapter, fireDataList, fireDrawable, "fire")  
 2 -> getData(firebaseAdapter, fireDataList, earthquakeDrawable, "earthquake")  
 3 -> getData(firebaseAdapter, fireDataList, floodDrawable, "flood")  
 // Add more cases for additional tabs if needed  
 else -> getData(firebaseAdapter, fireDataList, otherDrawable, "other")  
 }  
 return root  
 }  
  
 companion object {  
 /\*\*  
 \* The fragment argument representing the section number for this  
 \* fragment.  
 \*/  
 private const val ARG\_SECTION\_NUMBER = "section\_number"  
  
 /\*\*  
 \* Returns a new instance of this fragment for the given section  
 \* number.  
 \*/  
 @JvmStatic  
 fun newInstance(sectionNumber: Int): PlaceholderFragment {  
 return PlaceholderFragment().apply {  
 arguments = Bundle().apply {  
 putInt(ARG\_SECTION\_NUMBER, sectionNumber)  
 }  
 }  
 }  
 }  
  
 override fun onDestroyView() {  
 super.onDestroyView()  
 \_binding = null  
 }  
}  
  
private fun getData(recyclerViewAdapter: FiredataAdapter, fireDataList: MutableList<FirebaseData>, imageDrawable: Int, path: String) {  
 val database = FirebaseDatabase.getInstance().reference  
 fireDataList.clear() // Clear the list before populating  
 database.child("pending").child(path).addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
  
  
 for (countrySnapshot in dataSnapshot.children) {  
 Log.d("Country Snapshot", countrySnapshot.toString())  
 for (citySnapshot in countrySnapshot.children) {  
 Log.d("City Snapshot", citySnapshot.toString())  
 for (fireSnapshot in citySnapshot.children) {  
 Log.d("Fire Snapshot", fireSnapshot.toString())  
  
//// Parse the values from the fireSnapshot  
 val count = fireSnapshot.child("count").getValue(Long::class.java)  
 val latitude = fireSnapshot.child("latitude").getValue(Double::class.java)  
 val longitude = fireSnapshot.child("longitude").getValue(Double::class.java)  
 val timestamp = fireSnapshot.child("timestamp").getValue(String::class.java)  
 val descriptionList: MutableList<String> = mutableListOf()  
 val photosList: MutableList<String> = mutableListOf()  
 val country = countrySnapshot.key  
 val city = citySnapshot.key  
 val key = fireSnapshot.key  
  
 for (childSnapshot in fireSnapshot.child("description").children) {  
 val description = childSnapshot.getValue(String::class.java)  
 description?.let {  
 descriptionList.add(it)  
 }  
 }  
// // Create a FireData object and add it to the list  
 val fireData = FirebaseData(count, descriptionList, latitude, longitude, timestamp, imageDrawable, photosList, country, city, key, path)  
 fireDataList.add(fireData)  
 Log.d("Fire Data", fireData.toString())  
 }  
 }  
 }  
  
  
 // Log the populated fireDataList  
 for (fireData in fireDataList) {  
 Log.d("Fire Data", fireData.toString())  
 }  
  
 // Notify the RecyclerView adapter that the data has changed  
 recyclerViewAdapter.notifyDataSetChanged()  
 }  
  
 override fun onCancelled(error: DatabaseError) {  
 Log.e("Firebase Error", error.message)  
 }  
 })  
}

Εικόνα 14: Inbox Activity - dark theme Εικόνα 15: Inbox Activity - light theme

### Εμφάνιση λεπτομερειών συμβάντος

Αφού ο υπάλληλος της πολιτικής προστασίας πατήσει σε κάποιο συμβάν, μπορεί να δει τις λεπτομέρειες αυτού και να πράξει όπως καθορίστηκε ανωτέρω.

Εικόνα 16: Εμφάνιση πληροφοριών Εικόνα 17: Εμφάνιση πληροφοριών Εικόνα 18: Εμφάνιση

- light theme - dark theme πολλαπλών

πληροφοριών – dark theme

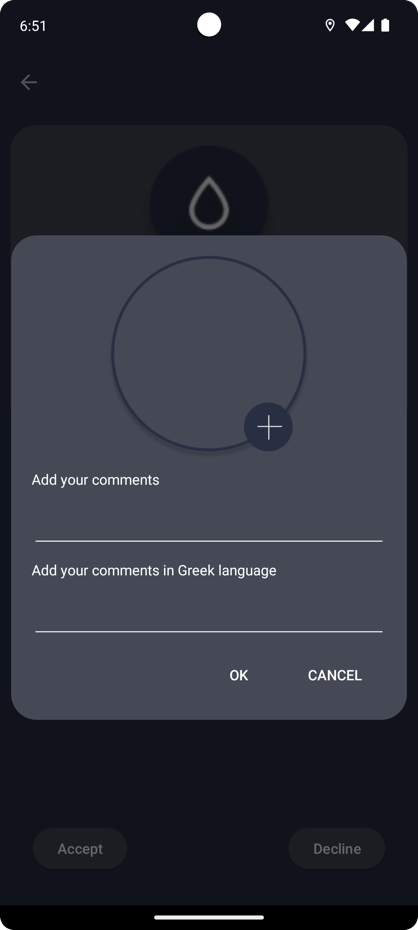
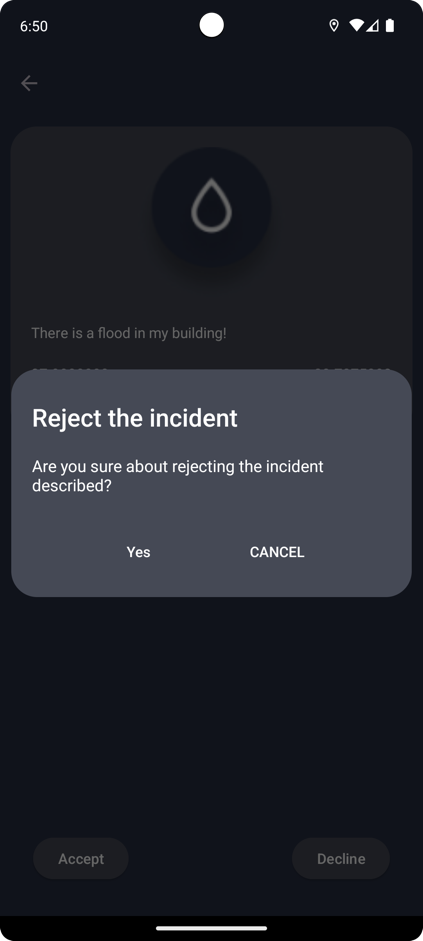
Όπως βλέπουμε, ο υπάλληλος της πολιτικής προστασίας αφού εξετάσει τις πληροφορίες που έχει στη διάθεση του μπορεί να επιβεβαιώσει την κατάσταση εκτάκτου κινδύνου ή να την απορρίψει. Επίσης, μπορεί να μεταφερθεί στην προηγούμενη οθόνη αφήνοντας με αυτόν τον τρόπο το συμβάν σε αναμονή. Επίσης, διατίθεται από το backend σύστημα μας μία πρόταση ως προς την αξιοπιστία και σοβαρότητα του υποβληθέντος συμβάντος με μορφή μπάρας που γεμίζει και μπορεί να είναι είτε κόκκινη είτε κίτρινη είτε πράσινη.

Εικόνα 19: Προτάσεις αξιοπιστίας και σοβαρότητας

### Διαχείριση συμβάντος

Όπως έχουμε ήδη αναφέρει ο υπάλληλος της πολιτικής προστασίας μπορεί είτε να απορρίψει το συμβάν είτε να το επιβεβαιώσει στέλνοντας έτσι μια ειδοποίηση στους πολίτες που βρίσκονται πλησίον αυτού.

Εικόνα 20: Επιβεβαίωση συμβάντος Εικόνα 21: Απόρριψη συμβάντος

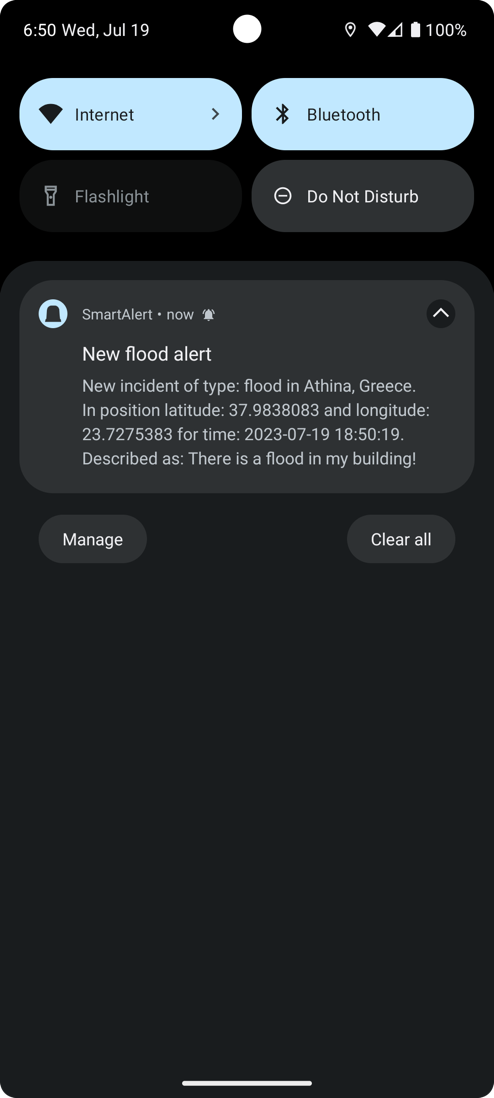
και εισαγωγή στοιχείων

Κατά την απόρριψη του συμβάντος αυτό διαγράφεται από τη βάση δεδομένων μας.

class InfoActivity : AppCompatActivity() {  
  
 private lateinit var infoRecycler: RecyclerView private lateinit var recyclerViewAdapter: InfoAdapter private lateinit var fireDataList: MutableList<InfoData> private val database = FirebaseDatabase.getInstance().reference private var imageUrl: String = "" private var imageUri: Uri? = null private lateinit var imageBtn: ImageButton private lateinit var imageUploaded: RoundedImageView private val PICK\_IMAGE\_REQUEST = 1 private lateinit var storageRef: StorageReference  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_info)  
  
 infoRecycler = findViewById(R.id.infoRecycler)  
 infoRecycler.layoutManager = LinearLayoutManager(this, LinearLayoutManager.HORIZONTAL, false)  
 fireDataList = mutableListOf()  
 recyclerViewAdapter = InfoAdapter(fireDataList)  
 infoRecycler.adapter = recyclerViewAdapter val toolbar: Toolbar = findViewById(R.id.toolbar\_info)  
 setSupportActionBar(toolbar)  
 supportActionBar?.setDisplayHomeAsUpEnabled(true)  
 supportActionBar?.title = ""  
  
  
  
 val acceptBtn = findViewById<Button>(R.id.acceptBtn)  
 val declineBtn = findViewById<Button>(R.id.declineBtn)  
 val country = intent.getStringExtra("country")  
 val city = intent.getStringExtra("city")  
 val key = intent.getStringExtra("key")  
 val path = intent.getStringExtra("path")  
 val dataRef = database.child("pending").child(path!!).child(country!!).child(city!!).child(key!!)  
 val notificationRef = database.child("notification")  
 storageRef = FirebaseStorage.getInstance().reference getData(recyclerViewAdapter, fireDataList, R.drawable.fire\_alert, path!!, country!!, city!!, key!!)  
  
  
 acceptBtn.setOnClickListener {  
 showAlertDialogButtonClickedAccept(this, it, dataRef, path, notificationRef)  
  
 }  
 declineBtn.setOnClickListener {  
 showAlertDialogButtonClickedDecline(dataRef, notificationRef)  
 }  
  
 }  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
 when (item.itemId) {  
 android.R.id.home -> {  
 val intent = Intent(this, AdminActivity::class.java)  
 startActivity(intent)  
 }  
 }  
 return super.onOptionsItemSelected(item)  
 }  
 fun showAlertDialogButtonClickedAccept(context: Context, view: View, dataRef: DatabaseReference, typeOfEvent: String, notificationRef: DatabaseReference) {  
 // Inflate the custom alert layout  
 val inflater = layoutInflater  
 val customLayout = inflater.inflate(R.layout.alert\_accept, null)  
 val positiveBtn = customLayout.findViewById<Button>(R.id.alert\_acceptBtn)  
 val negativeBtn = customLayout.findViewById<Button>(R.id.alert\_dismissBtn)  
 val descriptionTextEn : EditText = customLayout.findViewById(R.id.acceptCommentEng)  
 val descriptionTextGr : EditText = customLayout.findViewById(R.id.acceptCommentGr)  
 imageBtn = customLayout.findViewById<ImageButton>(R.id.acceptImgBtn)  
 imageUploaded = customLayout.findViewById(R.id.acceptImgUploaded)  
  
 // Create and show the alert dialog  
 val alertDialogBuilder = AlertDialog.Builder(this)  
 alertDialogBuilder.setView(customLayout)  
 alertDialogBuilder.setCancelable(false)  
 val alertDialog = alertDialogBuilder.create()  
  
 alertDialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))  
 positiveBtn.setOnClickListener {  
  
 val timestamp = Date() // Get the current timestamp as a Date object  
  
 // Format the timestamp as a human-readable string  
 val dateFormat = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault())  
 val timestampString = dateFormat.format(timestamp)  
  
 dataRef.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
  
 val latitude: Double? = dataSnapshot.child("latitude").value as? Double?  
 val longitude: Double? = dataSnapshot.child("longitude").value as? Double?  
 val notificationKey: String = dataSnapshot.child("key").value as String  
 val data = HashMap<String, Any>()  
 data["GR"] = descriptionTextGr.text.toString()  
 data["EN"] = descriptionTextEn.text.toString()  
 data["timestamp"] = timestampString  
 data["latitude"] = latitude!!.toDouble()  
 data["longitude"] = longitude!!.toDouble()  
 data["timestamp"] = timestampString  
 data["type"] = typeOfEvent  
  
 val dataRefActive = database.child("active").push()  
 val statisticsRef = database.child("statistics").push()  
 dataRefActive.setValue(data)  
 statisticsRef.setValue(data)  
 if (imageUrl.isNotEmpty()){  
 dataRefActive.child("photos").setValue(imageUrl)  
 }else if(imageUrl.isEmpty()){  
  
 }  
 notificationRef.child(notificationKey).removeValue().addOnSuccessListener {  
 dataRef.removeValue().addOnSuccessListener {  
 }  
 }  
  
 Toast.makeText(context, "Successfully created the event", Toast.LENGTH\_LONG).show()  
 val intent = Intent(context, AdminActivity::class.java)  
 startActivity(intent)  
 alertDialog.dismiss()  
 }  
  
 override fun onCancelled(databaseError: DatabaseError) {  
 // Handle any errors that occur  
 Log.e("Database Error", databaseError.message)  
 }  
 })  
 }  
 negativeBtn.setOnClickListener {  
 alertDialog.dismiss()  
 }  
 imageBtn.setOnClickListener {  
 uploadPhoto(view!!)  
 }  
 // Set the background of imageBtn  
 if (imageUri != null) {  
 val bitmap = MediaStore.Images.Media.getBitmap(contentResolver, imageUri)  
 val drawable = BitmapDrawable(resources, bitmap)  
 imageBtn.background = drawable  
 }  
  
 alertDialog.show()  
  
 }  
  
  
 fun showAlertDialogButtonClickedDecline(database: DatabaseReference, notificationRef: DatabaseReference) {  
 // Inflate the custom alert layout  
 val inflater = layoutInflater  
 val customLayout = inflater.inflate(R.layout.alert\_decline, null)  
 val alertReject = customLayout.findViewById<Button>(R.id.alert\_rejectBtn)  
 val alertCancel = customLayout.findViewById<Button>(R.id.alert\_cancelBtn)  
  
 // Create and show the alert dialog  
 val alertDialogBuilder = AlertDialog.Builder(this)  
 alertDialogBuilder.setView(customLayout)  
 alertDialogBuilder.setCancelable(false)  
 val alertDialog = alertDialogBuilder.create()  
  
 alertDialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))  
 alertReject.setOnClickListener {  
 database.removeValue()  
 .addOnSuccessListener {  
 notificationRef.child(fireDataList[0].notificationKey!!).removeValue().addOnSuccessListener {  
 alertDialog.dismiss()  
 Toast.makeText(this, "Successfully discarded the event", Toast.LENGTH\_LONG).show()  
 val intent = Intent(this, AdminActivity::class.java)  
 startActivity(intent)  
 }  
 }  
 .addOnFailureListener { error -> Toast.makeText(this, "Error while trying to discard incident", Toast.LENGTH\_LONG).show()  
 }  
  
 }  
  
  
 alertCancel.setOnClickListener {  
 alertDialog.dismiss()  
 }  
  
 alertDialog.show()  
  
 }  
 fun uploadPhoto(view: View) {  
 // Create an intent to pick an image from the gallery  
 val intent = Intent(Intent.ACTION\_PICK, MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI)  
 startActivityForResult(intent, PICK\_IMAGE\_REQUEST)  
 }  
 override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
 super.onActivityResult(requestCode, resultCode, data)  
 if (requestCode == PICK\_IMAGE\_REQUEST && resultCode == RESULT\_OK && data != null) {  
 val imageUri: Uri? = data.data  
 if (imageUri != null) {  
 // Upload the image to Firebase Storage  
 val fileRef = storageRef.child("images/${imageUri.lastPathSegment}")  
 fileRef.putFile(imageUri)  
 .addOnSuccessListener { taskSnapshot -> // Image upload successful  
 // Get the download URL of the uploaded image  
 fileRef.downloadUrl.addOnSuccessListener { downloadUri -> // Handle the download URL (e.g., save it in the database)  
 imageUrl = downloadUri.toString()  
 // Save the image URL in your Firebase Realtime Database  
 // Set the background of imageBtn as a circular shape  
 val bitmap = MediaStore.Images.Media.getBitmap(contentResolver, imageUri)  
 val drawable = BitmapDrawable(resources, bitmap)  
 imageUploaded.background = drawable  
 imageBtn.visibility = View.GONE  
 imageUploaded.visibility = View.VISIBLE  
  
 }  
  
 }  
 .addOnFailureListener { exception -> Toast.makeText(this, "Failed to upload photo", Toast.LENGTH\_LONG).show()  
 }  
 }  
 }  
 }  
}  
  
private fun getData(recyclerViewAdapter: InfoAdapter, fireDataList: MutableList<InfoData>, imageDrawable: Int, path: String, country: String, city: String, key:String) {  
 val database = FirebaseDatabase.getInstance().reference  
 fireDataList.clear() // Clear the list before populating  
 database.child("pending").child(path).child(country).child(city).child(key).addListenerForSingleValueEvent(object :  
 ValueEventListener {  
 override fun onDataChange(dataSnapshot: DataSnapshot) {  
  
//// Parse the values from the fireSnapshot  
 val count = dataSnapshot.child("count").getValue(Long::class.java)  
 val latitude = dataSnapshot.child("latitude").getValue(Double::class.java)  
 val longitude = dataSnapshot.child("longitude").getValue(Double::class.java)  
 val timestamp = dataSnapshot.child("timestamp").getValue(String::class.java)  
 val notificationKey = dataSnapshot.child("key").getValue(String::class.java)  
 val descriptionList: MutableList<String> = mutableListOf()  
 val photosList: MutableList<String> = mutableListOf()  
 for (childSnapshot in dataSnapshot.child("description").children) {  
 val description = childSnapshot.getValue(String::class.java)  
 description?.let {  
 descriptionList.add(it)  
 }  
 }  
 for (childSnapshot in dataSnapshot.child("photos").children) {  
 if(childSnapshot.exists()){  
 val description = childSnapshot.getValue(String::class.java)  
 description?.let {  
 photosList.add(it)  
 }  
 }else{  
 photosList.add(R.drawable.fire\_alert.toString())  
 }  
  
 }  
 for (i in descriptionList.indices) {  
 val description = descriptionList[i]  
 var photo: String  
 if(i < photosList.count()){  
 photo = photosList[i]  
 val fireData = InfoData(count, description, latitude, longitude, timestamp, imageDrawable, photo, path = path, notificationKey = notificationKey)  
 fireDataList.add(fireData)  
  
 }else{  
 //val photo = "0"  
 val fireData = InfoData(count, description, latitude, longitude, timestamp, imageDrawable, path = path, notificationKey = notificationKey)  
 fireDataList.add(fireData)  
 }  
 }  
  
  
 recyclerViewAdapter.notifyDataSetChanged()  
 }  
  
 // Notify the RecyclerView adapter that the data has changed  
  
  
 override fun onCancelled(error: DatabaseError) {  
 Log.e("Firebase Error", error.message)  
 }  
  
 })  
}

### Ειδοποίηση υπαλλήλου πολιτικής προστασίας

Κατά τη δημιουργία ενός νέου συμβάντος από κάποιον πολίτη, αποστέλλεται ειδοποίηση στον υπάλληλο της πολιτικής προστασίας ώστε να ερευνήσει το περιστατικό. Εάν ο υπάλληλος της πολιτικής προστασίας πατήσει την ειδοποίηση, μεταφέρεται αυτόματα στην οθόνη προβολής των περιστατικών έκτακτης ανάγκης.



Εικόνα 22: Ειδοποίηση υπαλλήλου πολιτικής προστασίας

Η υλοποίηση των ειδοποιήσεων έγινε με ανάλογο τρόπο με αυτόν που χρησιμοποιούν οι ειδοποιήσεις των πολιτών. Δηλαδή με ένα Service.

class AdminListener : Service() {  
  
 private lateinit var databaseReference: DatabaseReference private lateinit var childEventListener: ChildEventListener private var isInitialDataRetrieved = false  
  
  
 override fun onCreate() {  
 super.onCreate()  
 Log.d("Admin", "Admin Listening")  
 val currentLocale = getCurrentLocale()  
 val databaseReference = FirebaseDatabase.getInstance().reference.child("notification")  
 val childEventListener = object : ChildEventListener {  
 override fun onChildAdded(dataSnapshot: DataSnapshot, previousChildName: String?) {  
 if (isInitialDataRetrieved) {  
 Log.d("Child Added", dataSnapshot.toString())  
 var type: String? = dataSnapshot.child("type").getValue(String::class.java)  
 val city: String? = dataSnapshot.child("city").getValue(String::class.java)  
 val country: String? = dataSnapshot.child("country").getValue(String::class.java)  
 val latitude: Double? = dataSnapshot.child("latitude").getValue(Double::class.java)  
 val longitude: Double? = dataSnapshot.child("longitude").getValue(Double::class.java)  
 val time: String? = dataSnapshot.child("timestamp").getValue(String::class.java)  
 val description: String? = dataSnapshot.child("description").getValue(String::class.java)  
 if (currentLocale.language == "el"){  
 if (type == "fire"){  
 type = "φωτιάς" }else if (type == "flood"){  
 type = "πλημμύρας" }else if (type == "earthquake"){  
 type = "σεισμού" }else{  
 type = "άλλη" }  
 val message: String = "Νέα κατάσταση κινδύνου $type στην $city, $country. " +  
 "Σε θέση πλάτους: $latitude και μήκους: $longitude για χρόνο: $time. Με περιγραφή: $description" createNotification(applicationContext, message, "Νέα προειδοποίηση $type")  
 }else{  
 val message: String = "New incident of type: $type in $city, $country. " +  
 "In position latitude: $latitude and longitude: $longitude for time: $time. Described as: $description" createNotification(applicationContext, message, "New $type alert")  
 }  
  
  
  
 }  
 }  
  
 override fun onChildChanged(snapshot: DataSnapshot, previousChildName: String?) {  
 // Handle child changed event  
 }  
  
 override fun onChildRemoved(snapshot: DataSnapshot) {  
 // Handle child removed event  
 }  
  
 override fun onChildMoved(snapshot: DataSnapshot, previousChildName: String?) {  
 // Handle child moved event  
 }  
  
 override fun onCancelled(error: DatabaseError) {  
 // Handle database error  
 }  
 }  
  
 databaseReference.addChildEventListener(childEventListener)  
  
 // Attach a ValueEventListener to listen for initial data retrieval  
 val valueEventListener = object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 // Set the flag to indicate initial data retrieval is complete  
 isInitialDataRetrieved = true  
 }  
  
 override fun onCancelled(error: DatabaseError) {  
 // Handle database error  
 }  
 }  
  
 databaseReference.addListenerForSingleValueEvent(valueEventListener)  
  
 }  
  
 override fun onBind(intent: Intent): IBinder? {  
 return null  
 }  
  
 override fun onDestroy() {  
 super.onDestroy()  
  
 // Remove the value event listener when the service is destroyed  
 databaseReference.removeEventListener(childEventListener)  
 }  
 fun createNotification(context: Context, message: String, title: String) {  
 val channelId = "your\_channel\_id"  
 val notificationId = 1  
  
 // Create an Intent  
 val intent = Intent(context, AdminActivity::class.java)  
 intent.flags = Intent.FLAG\_ACTIVITY\_NEW\_TASK or Intent.FLAG\_ACTIVITY\_CLEAR\_TASK  
 // Add any extras to the intent if needed  
  
 // Create a PendingIntent with FLAG\_UPDATE\_CURRENT  
 val pendingIntent = PendingIntent.getActivity(  
 context,  
 0,  
 intent,  
 PendingIntent.FLAG\_IMMUTABLE  
 )  
  
 // Create a notification channel for Android Oreo and above  
 if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {  
 val channel = NotificationChannel(  
 channelId,  
 "Channel Name",  
 NotificationManager.IMPORTANCE\_HIGH  
 )  
 val notificationManager =  
 context.getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager  
 notificationManager.createNotificationChannel(channel)  
 }  
  
 // Build the notification with BigTextStyle and set the content intent  
 val notificationBuilder = NotificationCompat.Builder(context, channelId)  
 .setSmallIcon(R.drawable.alert\_svgrepo\_com)  
 .setContentTitle(title)  
 .setStyle(NotificationCompat.BigTextStyle().bigText(message))  
 .setPriority(NotificationCompat.PRIORITY\_HIGH)  
 .setContentIntent(pendingIntent)  
 .setAutoCancel(true)  
  
 // Show the notification  
 val notificationManager =  
 context.getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager  
 notificationManager.notify(notificationId, notificationBuilder.build())  
 }  
  
 private fun getCurrentLocale(): Locale {  
 val configuration = resources.configuration  
 return if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.N) {  
 configuration.locales[0]  
 } else {  
 configuration.locale  
 }  
 }  
}