**Υλοποίηση online συστήματος αποθήκευσης και διαχείρισης φωτογραφιών**

**Ονοματεπώνυμο:**

**ΜΑΡΤΙΟΣ 2023**

**Περίληψη**

Αυτή η εργασία παρουσιάζει μια desktop εφαρμογή που αναπτύχθηκε σε Java και επιτρέπει στους χρήστες να ανεβάζουν, να αποθηκεύουν και να παρουσιάζουν εικόνες χρησιμοποιώντας το Firebase Storage. Η εφαρμογή χρησιμοποιεί το Firebase Admin SDK και τη βιβλιοθήκη Google Cloud Storage για να αλληλεπιδράσει με το Firebase και να εκτελέσει λειτουργίες αρχείων. Η διεπαφή χρήστη έχει δημιουργηθεί χρησιμοποιώντας το Swing, παρέχοντας στους χρήστες την επιλογή να επιλέξουν φωτογραφίες από το τοπικό τους σύστημα, να τις δουν σε ένα grid με δυνατότητα αλλαγής μεγέθους και να τις ανεβάσουν στο Firebase Storage.

Η διπλωματική εργασία περιλαμβάνει το σχεδιασμό και την υλοποίηση της εφαρμογής, καθώς και την ενσωμάτωσή της με το Firebase Storage. Η εργασία επεξεργάζεται επίσης τον αλγόριθμο κλιμάκωσης εικόνας που χρησιμοποιείται για να διασφαλίσει ότι οι εικόνες πληρούν τις απαιτήσεις μεγέθους του Firebase Storage.

Αυτή η εφαρμογή παρουσιάζει μια απλή και αποτελεσματική προσέγγιση για την αποθήκευση και τη διαχείριση εικόνων στο cloud, καλύπτοντας τις απαιτήσεις διαχείρισης εικόνας τόσο ατόμων όσο και επιχειρήσεων. Η εργασία υπογραμμίζει τα οφέλη από τη χρήση του Firebase Storage ως λύση αποθήκευσης cloud, συμπεριλαμβανομένης της ανθεκτικότητας, της αξιοπιστίας και της επεκτασιμότητας του. Τονίζει επίσης τα πλεονεκτήματα της Java και του Swing στην ανάπτυξη εφαρμογών για επιτραπέζιους υπολογιστές, συμπεριλαμβανομένων των φιλικών προς το χρήστη γραφικών στοιχείων διεπαφής χρήστη και της συμβατότητας μεταξύ πλατφορμών.

H παρούσα πτυχιακή εργασία προσφέρει μια ρεαλιστική λύση για την αποθήκευση και διαχείριση εικόνων στο cloud μέσω μιας desktop εφαρμογής που βασίζεται σε Java και του Firebase Storage. Η διεπαφή χρήστη είναι εύκολη στη χρήση και απλή, καθιστώντας την προσβάσιμη σε άτομα με διαφορετική τεχνική τεχνογνωσία. Η εργασία παρέχει μια ολοκληρωμένη επισκόπηση του σχεδιασμού και της υλοποίησης της εφαρμογής, καθιστώντας την πολύτιμη πηγή για τους προγραμματιστές που φιλοδοξούν να εφαρμόσουν παρόμοιες εφαρμογές.

**Abstract**

This thesis introduces a desktop application developed in Java that allows users to upload, store, and showcase images using Firebase Storage. The application utilizes the Firebase Admin SDK and the Google Cloud Storage library to interact with Firebase and execute file operations. The user interface is built using Swing, providing users with an option to select pictures from their local system, exhibit them in a resizable container, and upload them to Firebase Storage.

The thesis includes the application's design and implementation, along with its integration with Firebase Storage. The thesis also elaborates on the image scaling algorithm utilized to ensure the pictures meet Firebase Storage's size requirements.

This application presents a simple and efficient approach to storing and managing images on the cloud, catering to the image management requirements of individuals and businesses alike. The thesis highlights the benefits of using Firebase Storage as a cloud storage solution, including its durability, reliability, and scalability. It also emphasizes the advantages of Java and Swing in developing desktop applications, including their user-friendly graphical user interface elements and cross-platform compatibility.

In conclusion, this thesis offers a pragmatic solution to cloud image storage and management through a Java-based desktop application and Firebase Storage. The user interface is easy to use and straightforward, rendering it accessible to individuals with varying technical expertise. The thesis provides a comprehensive overview of the application's design and implementation, making it a valuable resource for developers aspiring to implement similar applications.

**Περιεχόμενα**

[**1 Εισαγωγή 5**](#_2et92p0)

[1.1 Background 5](#_azkq2fnzx5kv)

[1.2 Δήλωση Προβλήματος 6](#_rzlroqsdnsac)

[1.3 Στόχοι 7](#_s9btixsu0gie)

[1.4 Σημαντικότητα της εργασίας 7](#_l9eofe5k7i34)

[1.5 Επισκόπηση της εργασίας 8](#_rld7yatad7fv)

[**2 Ανασκόπηση τεχνολογιών 10**](#_bu62z1eij1eo)

[2.1 Εισαγωγή στο Firebase Storage 10](#_ostjz4lpaudc)

[2.2 Εισαγωγή στο Firebase Cloud Firestore 12](#_4vffmwjlztnj)

[2.3 Η γλώσσα προγραμματισμού Java 14](#_9ucsxwhgv03l)

[2.4 Επισκόπηση των Swing και JavaFX UI Frameworks 15](#_dr1mez1ordhd)

[2.4.1 Swing Framework 15](#_f0tas6jekbij)

[2.4.2 JavaFX FrameWork 16](#_56bbbsb8jxje)

[2.4.3 Σύγκριση Swing και JavaFX 16](#_yvzu6h3lt9kg)

[**3 Μεθοδολογία 17**](#_pdktd1m4hfye)

[3.1 Maven και βιβλιοθήκες 17](#_s1pmmmta4wzt)

[3.2 Firebase Project 19](#_v7y5kmyeyj3s)

[3.3 Συναρτήσεις επικοινωνίας με το backend 22](#_etts32mjk5it)

[3.3.1 uploadImageToFirebaseStorage 22](#_pgfdo1dkj8m9)

[3.3.2 saveImageDetailsToFirestore 23](#_nvcf0l8s11ve)

[3.3.3 loadImagesFromFirebaseStorage 26](#_fusdsvo90ywe)

[3.4 Ανέβασμα Φωτογραφίας 27](#_gu2z48pp8xx)

[3.5 Φόρτωση Φωτογραφιών 35](#_4lfjc1c83chd)

[3.6 Προβολή Στοιχείων 36](#_1wogzkfiiv8b)

[3.7 Φίλτρα 42](#_3vmzpgkwte6u)

[3.7.1 YearFilterWidget 43](#_yogugvecht0h)

[3.7.2 PeopleFilterWidget 44](#_u5lxe3au1dl6)

[3.7.3 LocationFilterWidget 45](#_t96j364jz2cb)

[3.7.4 Εφαρμογή Φίλτρων 46](#_tj5p1j15awf0)

[**4 Συμπεράσματα και μελλοντική εργασία 52**](#_wycn4joqpmiu)

[**5 Αναφορές 54**](#_8rkt631lyroj)

# **Εισαγωγή**

Το εισαγωγικό αυτό κεφάλαιο θέτει τη βάση για την εργασία, παρέχοντας μια επισκόπηση υψηλού επιπέδου του τι μπορεί να περιμένει ο αναγνώστης να μάθει. Σε αυτό το κεφάλαιο, θα συζητήσουμε το υπόβαθρο του έργου, τη δήλωση προβλήματος που στοχεύει να λύσει αυτή η εφαρμογή και τους στόχους που τέθηκαν. Επιπλέον, θα περιγράψουμε το εύρος και τους περιορισμούς της εφαρμογής, τη σημασία της μελέτης και θα παρέχουμε μια επισκόπηση της δομής της εργασία. Διαβάζοντας αυτό το κεφάλαιο, οι αναγνώστες θα αποκτήσουν μια σαφή κατανόηση του σκοπού και του πεδίου αυτής της εργασίας.

## 1.1 Background

Η ταχεία ανάπτυξη της ψηφιακής τεχνολογίας έχει επιφέρει σημαντικές αλλαγές στον τρόπο με τον οποίο οι άνθρωποι αποθηκεύουν και διαχειρίζονται τα δεδομένα τους, συμπεριλαμβανομένων των εικόνων. Με την αυξανόμενη δημοτικότητα της αποθήκευσης cloud [1] , οι χρήστες μπορούν πλέον να αποθηκεύουν και να έχουν πρόσβαση στα δεδομένα τους από οπουδήποτε στον κόσμο. Οι λύσεις αποθήκευσης cloud προσφέρουν πολλά πλεονεκτήματα, όπως επεκτασιμότητα, ευελιξία και οικονομική αποδοτικότητα, καθιστώντας τις μια δημοφιλή επιλογή τόσο για ιδιώτες όσο και για επιχειρήσεις.

Το Firebase Storage [2] είναι μια τέτοια λύση αποθήκευσης cloud που προσφέρει έναν απλό και αποτελεσματικό τρόπο αποθήκευσης και διαχείρισης δεδομένων, συμπεριλαμβανομένων των εικόνων. Το Firebase είναι μια πλατφόρμα ανάπτυξης εφαρμογών για κινητές συσκευές και web που παρέχει διάφορες δυνατότητες, όπως έλεγχο ταυτότητας, βάση δεδομένων σε πραγματικό χρόνο, ανταλλαγή μηνυμάτων στο cloud και αποθήκευση στο cloud. Το Firebase Storage είναι μια λύση αποθήκευσης cloud που επιτρέπει στους χρήστες να αποθηκεύουν και να ανακτούν αρχεία, όπως εικόνες, βίντεο και έγγραφα.

Η Java [3] είναι μια δημοφιλής γλώσσα προγραμματισμού που χρησιμοποιείται για την ανάπτυξη εφαρμογών για επιτραπέζιους υπολογιστές, ιστούς και κινητές συσκευές. Η Java παρέχει διάφορες δυνατότητες, όπως ανεξαρτησία πλατφόρμας, αντικειμενοστραφή προγραμματισμό και αυτόματη διαχείριση μνήμης, καθιστώντας την μια δημοφιλή επιλογή για προγραμματιστές. Το Swing [4] είναι μια εργαλειοθήκη γραφικού περιβάλλοντος χρήστη (GUI) για Java που παρέχει διάφορα στοιχεία, όπως κουμπιά, ετικέτες και κοντέινερ, για τη δημιουργία desktop εφαρμογών.

Ο συνδυασμός του Firebase Storage και των εφαρμογών επιτραπέζιου υπολογιστή που βασίζονται σε Java παρέχει έναν απλό και αποτελεσματικό τρόπο αποθήκευσης και διαχείρισης εικόνων στο cloud. Αυτή η εργασία παρουσιάζει μια εφαρμογή επιτραπέζιου υπολογιστή που βασίζεται σε Java που επιτρέπει στους χρήστες να ανεβάζουν, να αποθηκεύουν και να εμφανίζουν εικόνες χρησιμοποιώντας το Firebase Storage. Η εφαρμογή χρησιμοποιεί το Firebase Admin SDK και τη βιβλιοθήκη Google Cloud Storage για να αλληλεπιδράσει με το Firebase και να εκτελέσει λειτουργίες αρχείων. Η γραφική διεπαφή χρήστη υλοποιείται χρησιμοποιώντας το Swing, επιτρέποντας στους χρήστες να επιλέξουν εικόνες από τον τοπικό τους υπολογιστή, να τις εμφανίσουν σε ένα κοντέινερ με δυνατότητα αλλαγής μεγέθους και να τις ανεβάσουν στο Firebase Storage. Η εφαρμογή παρέχει επίσης λειτουργία κλιμάκωσης εικόνας για να διασφαλίσει ότι οι μεταφορτωμένες εικόνες πληρούν τις απαιτήσεις μεγέθους του Firebase Storage.

## 1.2 Δήλωση Προβλήματος

Η αποθήκευση και η διαχείριση εικόνων στο cloud γίνεται όλο και πιο σημαντική στη σημερινή ψηφιακή εποχή. Με την άνοδο του cloud computing, οι επιχειρήσεις και τα άτομα βασίζονται περισσότερο σε λύσεις αποθήκευσης cloud για την αποθήκευση και τη διαχείριση των δεδομένων τους. Ωστόσο, υπάρχουν διάφορες προκλήσεις που σχετίζονται με τη χρήση λύσεων αποθήκευσης cloud, όπως η ασφάλεια, η αξιοπιστία και η επεκτασιμότητα.

Μία από τις μεγαλύτερες προκλήσεις είναι η εύρεση μιας λύσης αποθήκευσης cloud που μπορεί να αποθηκεύει και να διαχειρίζεται αποτελεσματικά μεγάλο αριθμό εικόνων, διασφαλίζοντας ταυτόχρονα την ασφάλεια και την προσβασιμότητά τους. Μια άλλη πρόκληση είναι να διασφαλιστεί ότι οι μεταφορτωμένες εικόνες πληρούν τις απαιτήσεις μεγέθους και ποιότητας της λύσης αποθήκευσης cloud, κάτι που μπορεί να είναι χρονοβόρο και απαιτεί χειροκίνητη παρέμβαση.

Επομένως, το πρόβλημα που αντιμετωπίζεται σε αυτή την εργασία είναι η ανάπτυξη μιας εφαρμογής επιτραπέζιου υπολογιστή που βασίζεται σε Java που μπορεί να ανεβάζει, να αποθηκεύει και να διαχειρίζεται εικόνες στο Firebase Storage, μια δημοφιλή λύση αποθήκευσης cloud που παρέχεται από την Google. Η εφαρμογή θα περιλαμβάνει επίσης λειτουργία κλιμάκωσης εικόνας για αυτόματη αλλαγή του μεγέθους των εικόνων ώστε να πληρούν τις απαιτήσεις μεγέθους του Firebase Storage. Ο κύριος στόχος είναι να παρέχουμε έναν απλό και αποτελεσματικό τρόπο για άτομα ή επιχειρήσεις να αποθηκεύουν και να διαχειρίζονται εικόνες στο cloud χωρίς την ανάγκη χειροκίνητης παρέμβασης.

## 1.3 Στόχοι

Ο πρωταρχικός στόχος αυτής της εργασία είναι να αναπτύξει μια εφαρμογή επιτραπέζιου υπολογιστή βασισμένη σε Java που επιτρέπει στους χρήστες να ανεβάζουν, να αποθηκεύουν και να εμφανίζουν εικόνες χρησιμοποιώντας το Firebase Storage. Για την επίτευξη αυτού του πρωταρχικού στόχου, θα επιτευχθούν οι ακόλουθοι επιμέρους στόχοι:

1. Να σχεδιαστεί μια διαισθητική και φιλική προς το χρήστη γραφική διεπαφή χρήστη που επιτρέπει στους χρήστες να επιλέγουν εικόνες από τον τοπικό τους υπολογιστή, να εμφανίζουν τις εικόνες σε ένα κοντέινερ με δυνατότητα αλλαγής μεγέθους και να τις ανεβάζουν στο Firebase Storage.
2. Να εφαρμοστεί μια λειτουργία κλιμάκωσης εικόνας για να διασφαλιστεί ότι οι μεταφορτωμένες εικόνες πληρούν τις απαιτήσεις μεγέθους του Firebase Storage.
3. Να ενσωματωθεί το Firebase Admin SDK και τη βιβλιοθήκη Google Cloud Storage στην εφαρμογή για αλληλεπίδραση με το Firebase και εκτέλεση λειτουργιών αρχείων.

Οι παραπάνω στόχοι θα επιτευχθούν μέσω του σχεδιασμού, της υλοποίησης και της δοκιμής της εφαρμογής επιτραπέζιου υπολογιστή που βασίζεται σε Java. Η υλοποίηση των στόχων θα έχει ως αποτέλεσμα μια εφαρμογή απλή, αποτελεσματική και αξιόπιστη στη διαχείριση και την αποθήκευση εικόνων στο cloud χρησιμοποιώντας το Firebase Storage. Η εφαρμογή θα είναι επίσης επεκτάσιμη, καθιστώντας την χρήσιμη για άτομα και επιχειρήσεις με διαφορετικές ανάγκες διαχείρισης εικόνας.

## 1.4 Σημαντικότητα της εργασίας

Η σημασία αυτής της εργασίας έγκειται στην ικανότητά της να παρέχει έναν απλό και αποτελεσματικό τρόπο αποθήκευσης και διαχείρισης εικόνων στο cloud σε άτομα και επιχειρήσεις. Με την αυξανόμενη ζήτηση για λύσεις αποθήκευσης που βασίζονται σε cloud, υπάρχει ανάγκη για πρακτικές εφαρμογές που μπορούν να καλύψουν αποτελεσματικά αυτές τις ανάγκες. Αυτή η εργασία προσφέρει μια λύση σε αυτό το πρόβλημα, χρησιμοποιώντας το Firebase Storage ως μια αξιόπιστη και επεκτάσιμη λύση αποθήκευσης που βασίζεται στο cloud.

Επιπλέον, η μελέτη παρέχει πληροφορίες σχετικά με τη χρήση της Java και του Swing για την υλοποίηση εφαρμογών επιτραπέζιου υπολογιστή, κάτι που αποτελεί πολύτιμη συμβολή στον τομέα της ανάπτυξης λογισμικού. Το μοτίβο αρχιτεκτονικής Model-View-Controller (MVC) που χρησιμοποιείται στο σχεδιασμό της εφαρμογής παρέχει επίσης μια πρακτική προσέγγιση για την ανάπτυξη επεκτάσιμων και συντηρήσιμων εφαρμογών.

Η εφαρμογή παρέχει λειτουργία κλιμάκωσης εικόνας για να διασφαλίσει ότι οι μεταφορτωμένες εικόνες πληρούν τις απαιτήσεις μεγέθους του Firebase Storage, το οποίο είναι ένα κρίσιμο χαρακτηριστικό για χρήστες με περιορισμένο χώρο αποθήκευσης. Η εφαρμογή προσφέρει επίσης μια φιλική προς το χρήστη διεπαφή που είναι εύκολη στη χρήση και στην πλοήγηση, καθιστώντας την προσβάσιμη σε χρήστες με διαφορετικά επίπεδα τεχνικής εξειδίκευσης.

Η σημασία αυτής της μελέτης έγκειται στη συμβολή της στην ανάπτυξη εφαρμογών που βασίζονται στο cloud, στη χρήση Java και Swing για ανάπτυξη εφαρμογών επιτραπέζιου υπολογιστή και στην παροχή μιας πρακτικής λύσης για την αποθήκευση και διαχείριση εικόνων στο cloud. Τα ευρήματα αυτής της μελέτης θα είναι πολύτιμα για τους προγραμματιστές λογισμικού, τις επιχειρήσεις και τα άτομα που επιδιώκουν να διαχειρίζονται τις εικόνες τους αποτελεσματικά και αποτελεσματικά στο cloud.

## 1.5 Επισκόπηση της εργασίας

Η εργασία ξεκινά με μια εισαγωγή που παρέχει το υπόβαθρο και τη δήλωση του προβλήματος, ακολουθούμενη από τους στόχους, το εύρος και τους περιορισμούς και τη σημασία της μελέτης. Το Κεφάλαιο 2 παρέχει μια βιβλιογραφική ανασκόπηση των λύσεων αποθήκευσης cloud, της γλώσσας προγραμματισμού Java και της ανάπτυξης εφαρμογών για υπολογιστές. Το Κεφάλαιο 3 παρουσιάζει τη μεθοδολογία που χρησιμοποιείται για το σχεδιασμό και την υλοποίηση της εφαρμογής, συμπεριλαμβανομένου του αρχιτεκτονικού μοτίβου, του σχεδιασμού της διεπαφής χρήστη και του αλγόριθμου κλιμάκωσης εικόνας.

Τέλος, το Κεφάλαιο 4 παρέχει τα συμπεράσματα και τη μελλοντική εργασία, συνοψίζοντας τις συνεισφορές αυτής της διατριβής και συζητώντας πιθανούς τομείς για μελλοντική έρευνα και ανάπτυξη. Συνοπτικά, αυτή η εργασία παρέχει μια πρακτική λύση για την αποθήκευση και τη διαχείριση εικόνων στο cloud με χρήση Firebase Storage και επιτραπέζιου υπολογιστή που βασίζεται σε Java, καθιστώντας την προσβάσιμη σε χρήστες με διαφορετικά επίπεδα τεχνικής εξειδίκευσης.

# **Ανασκόπηση τεχνολογιών**

Το Κεφάλαιο 2 αυτής της εργασίας παρέχει μια περιεκτική βιβλιογραφική ανασκόπηση των σχετικών τεχνολογιών και πλαισίων που χρησιμοποιούνται στην ανάπτυξη της εφαρμογής επιτραπέζιου υπολογιστή που βασίζεται σε Java για διαχείριση εικόνας στο Firebase Storage. Αυτό το κεφάλαιο στοχεύει να δημιουργήσει μια στέρεη βάση γνώσης που ενημερώνει την ανάπτυξη της εφαρμογής, με έμφαση στο Firebase Storage, την επεξεργασία εικόνας σε Java και τα πλαίσια διεπαφής χρήστη. Διερευνώντας τη βιβλιογραφία σχετικά με αυτά τα θέματα, αυτό το κεφάλαιο παρέχει μια κριτική ανάλυση των πλεονεκτημάτων και των περιορισμών κάθε τεχνολογίας και πλαισίου, επιτρέποντας στον αναγνώστη να κατανοήσει βαθύτερα τις υποκείμενες έννοιες και το σκεπτικό πίσω από το σχεδιασμό και την υλοποίηση της εφαρμογής.

## 2.1 Εισαγωγή στο Firebase Storage

Το Firebase Storage είναι μια υπηρεσία αποθήκευσης cloud που παρέχεται από την Google για την αποθήκευση και την εξυπηρέτηση περιεχομένου που δημιουργείται από τους χρήστες, όπως εικόνες, βίντεο και αρχεία ήχου. Αποτελεί μέρος της σουίτας εργαλείων και υπηρεσιών Firebase που στοχεύουν να βοηθήσουν τους προγραμματιστές να δημιουργήσουν και να διαχειριστούν εφαρμογές για κινητά και web υψηλής ποιότητας. Το Firebase Storage επιτρέπει στους προγραμματιστές να αποθηκεύουν και να ανακτούν αρχεία από το cloud με ευκολία, καθιστώντας το ιδανική λύση για προγραμματιστές που πρέπει να διαχειρίζονται μεγάλες ποσότητες δεδομένων.

Το Firebase Storage [5] παρέχει ένα απλό και διαισθητικό API που μπορεί εύκολα να ενσωματωθεί σε οποιαδήποτε εφαρμογή. Η υπηρεσία έχει σχεδιαστεί για να είναι επεκτάσιμη και αξιόπιστη, με ενσωματωμένους μηχανισμούς πλεονασμού και ανακατεύθυνσης για να διασφαλίζεται ότι τα δεδομένα είναι πάντα διαθέσιμα. Το Firebase Storage περιλαμβάνει επίσης λειτουργίες όπως έλεγχος πρόσβασης, διαχείριση μεταδεδομένων και έκδοση εκδόσεων, καθιστώντας το ένα ισχυρό εργαλείο για τη διαχείριση περιεχομένου στο cloud.

Ένα παράδειγμα στο οποίο ανεβάζουμε ένα αρχείο στο Firebase Storage χρησιμοποιώντας τη γλώσσα προγραμματισμού Java, είναι το παρακάτω:

| // Initialize Firebase Storage  FirebaseOptions options = new FirebaseOptions.Builder()  .setCredentials(GoogleCredentials.fromStream(serviceAccount))  .setStorageBucket("your-bucket-name.appspot.com")  .build();  FirebaseApp.initializeApp(options);  // Create a reference to the file you want to upload  File file = new File("path/to/image.jpg");  String fileName = file.getName();  StorageReference fileRef = FirebaseStorage.getInstance().getReference().child("images/" + fileName);  // Upload the file to Firebase Storage  UploadTask uploadTask = fileRef.putFile(Uri.fromFile(file));  Task<Uri> urlTask = uploadTask.continueWithTask(task -> {  if (!task.isSuccessful()) {  throw task.getException();  }  // Get the download URL for the uploaded file  return fileRef.getDownloadUrl();  }).addOnCompleteListener(task -> {  if (task.isSuccessful()) {  Uri downloadUri = task.getResult();  System.out.println("File uploaded successfully: " + downloadUri);  } else {  System.out.println("File upload failed");  }  }); |
| --- |

Σε αυτό το παράδειγμα, αρχικοποιούμε πρώτα το Firebase Storage με τα κατάλληλα διαπιστευτήρια και τον κάδο αποθήκευσης (bucket). Στη συνέχεια, δημιουργούμε ένα αντικείμενο StorageReference που δείχνει τη θέση στο Firebase Storage όπου θέλουμε να ανεβάσουμε το αρχείο μας.

Στη συνέχεια χρησιμοποιούμε τη μέθοδο putFile για να ανεβάσουμε το αρχείο στο Firebase Storage. Αυτή η μέθοδος επιστρέφει ένα αντικείμενο UploadTask που μπορούμε να χρησιμοποιήσουμε για να παρακολουθήσουμε την πρόοδο της μεταφόρτωσης.

Μόλις ολοκληρωθεί η μεταφόρτωση, χρησιμοποιούμε τη μέθοδο getDownloadUrl για να λάβουμε τη διεύθυνση URL λήψης για το μεταφορτωμένο αρχείο. Στη συνέχεια, μπορούμε να χρησιμοποιήσουμε αυτήν τη διεύθυνση URL για να αποκτήσουμε πρόσβαση στο αρχείο από την εφαρμογή μας ή να το μοιραστούμε με άλλους.

## 2.2 Εισαγωγή στο Firebase Cloud Firestore

Τα τελευταία χρόνια, το cloud computing έχει γίνει όλο και πιο δημοφιλές για αποθήκευση, επεξεργασία και φιλοξενία εφαρμογών. Το Firebase είναι μια πλατφόρμα που παρέχει μια ποικιλία υπηρεσιών cloud, συμπεριλαμβανομένου του Firebase Cloud Firestore, το οποίο είναι μια ευέλικτη, επεκτάσιμη και σε πραγματικό χρόνο βάση δεδομένων για κινητά, web και ανάπτυξη διακομιστών.

Το Firestore [6] είναι μια βάση δεδομένων εγγράφων NoSQL που αποθηκεύει δεδομένα με τη μορφή εγγράφων, τα οποία είναι παρόμοια με αντικείμενα JSON. Παρέχει ισχυρές δυνατότητες αναζήτησης δεδομένων και ευρετηρίασης και το μοντέλο δεδομένων του είναι βελτιστοποιημένο για ευέλικτη και επεκτάσιμη πρόσβαση σε δεδομένα. Το Firestore επιτρέπει στους χρήστες να αποθηκεύουν και να ανακτούν δεδομένα γρήγορα και αποτελεσματικά από το cloud, καθιστώντας το ιδανική λύση βάσης δεδομένων για εφαρμογές που απαιτούν συγχρονισμό δεδομένων σε πραγματικό χρόνο και πρόσβαση σε δεδομένα εκτός σύνδεσης.

Το Firestore παρέχει μια απλή και εύχρηστη διεπαφή για πρόσβαση στη βάση δεδομένων από εφαρμογές που βασίζονται σε Java. Το Firebase Admin SDK παρέχει ένα Java API για αλληλεπίδραση με το Firestore, το οποίο επιτρέπει στους χρήστες να εκτελούν λειτουργίες όπως ανάγνωση, εγγραφή, ενημέρωση και διαγραφή δεδομένων. Το Firebase Admin SDK παρέχει επίσης δυνατότητες ελέγχου ταυτότητας και ασφάλειας, οι οποίες επιτρέπουν στους χρήστες να περιορίζουν την πρόσβαση στη βάση δεδομένων και να διασφαλίζουν το απόρρητο και την ασφάλεια των δεδομένων.

Για να χρησιμοποιηθεί το Firestore σε μια εφαρμογή που βασίζεται σε Java, οι χρήστες πρέπει πρώτα να δημιουργήσουν ένα Firebase project και να ενεργοποιήσουν την υπηρεσία Firestore. Στη συνέχεια, μπορούν να κατεβάσουν και να διαμορφώσουν το Firebase Admin SDK για Java, το οποίο παρέχει τα απαραίτητα API και τα διαπιστευτήρια ελέγχου ταυτότητας για σύνδεση στη βάση δεδομένων Firestore. Μόλις διαμορφωθεί το SDK, οι χρήστες μπορούν να αρχίσουν να χρησιμοποιούν το API για να εκτελούν λειτουργίες βάσης δεδομένων και να χειρίζονται συμβάντα όπως αλλαγές εγγράφων, διαγραφές και δημιουργίες.

Το Firestore είναι μια ισχυρή και ευέλικτη λύση βάσης δεδομένων που χρησιμοποιείται ευρέως από προγραμματιστές για μια ποικιλία εφαρμογών. Η ενσωμάτωσή του με εφαρμογές που βασίζονται σε Java μέσω του Firebase Admin SDK το καθιστά δημοφιλή επιλογή για προγραμματιστές που απαιτούν συγχρονισμό δεδομένων σε πραγματικό χρόνο και πρόσβαση σε δεδομένα εκτός σύνδεσης. Οι επόμενες ενότητες αυτού του κεφαλαίου θα παρέχουν περισσότερες λεπτομέρειες σχετικά με τις δυνατότητες και τις δυνατότητες του Firestore και πώς μπορεί να χρησιμοποιηθεί σε εφαρμογές που βασίζονται σε Java.

Ένα παράδειγμα στο οποίο προσθέτουμε ένα record σε μία βάση του cloud firestore είναι το παρακάτω:

| // First, initialize the Firestore database  Firestore db = FirestoreClient.getFirestore();  // Then, create a new document in the "users" collection with a generated ID  DocumentReference docRef = db.collection("users").document();  String userId = docRef.getId();  // Define a new user object to be added to the database  User user = new User(userId, "John", "Doe", "johndoe@example.com");  // Add the user object as a new document in the "users" collection  ApiFuture<WriteResult> future = docRef.set(user);  // Wait for the write operation to complete and handle any errors  try {  future.get();  System.out.println("User added to Firestore database with ID: " + userId);  } catch (InterruptedException | ExecutionException e) {  System.err.println("Error adding user to Firestore database: " + e.getMessage());  } |
| --- |

Σε αυτό το παράδειγμα, αρχικοποιούμε πρώτα τη βάση δεδομένων Firestore καλώντας το FirestoreClient.getFirestore(). Στη συνέχεια, δημιουργούμε μια νέα αναφορά εγγράφου στη συλλογή "users" καλώντας db.collection("users").document(). Αυτό δημιουργεί αυτόματα ένα νέο μοναδικό αναγνωριστικό για το έγγραφο.

Στη συνέχεια, ορίζουμε ένα νέο αντικείμενο χρήστη με κάποια δεδομένα που θέλουμε να προσθέσουμε στη βάση δεδομένων Firestore. Σε αυτήν την περίπτωση, το αντικείμενο User έχει τέσσερα πεδία: userId, firstName, lastName και email.

Τέλος, προσθέτουμε το αντικείμενο User στη βάση δεδομένων Firestore καλώντας το docRef.set(user). Αυτό επιστρέφει ένα αντικείμενο ApiFuture<WriteResult>, το οποίο μπορούμε να περιμένουμε χρησιμοποιώντας τη μέθοδο future.get(). Εάν δεν υπάρχουν σφάλματα, εκτυπώνουμε ένα μήνυμα επιτυχίας στην κονσόλα. Εάν υπάρχουν σφάλματα, εκτυπώνουμε ένα μήνυμα σφάλματος.

## 2.3 Η γλώσσα προγραμματισμού Java

Η Java [7] είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού υψηλού επιπέδου που χρησιμοποιείται ευρέως για την ανάπτυξη επιτραπέζιων και διαδικτυακών εφαρμογών. Παρουσιάστηκε για πρώτη φορά από τη Sun Microsystems το 1995 και επί του παρόντος συντηρείται από την Oracle Corporation. Η Java έχει σχεδιαστεί για να είναι ανεξάρτητη από την πλατφόρμα (crossplatform) , πράγμα που σημαίνει ότι ο κώδικας που είναι γραμμένος σε Java μπορεί να εκτελεστεί σε οποιαδήποτε πλατφόρμα που υποστηρίζει την Java Virtual Machine (JVM).

Ένα από τα βασικά χαρακτηριστικά της Java είναι ο ισχυρός μηχανισμός ελέγχου τύπου, ο οποίος βοηθά στην αποφυγή σφαλμάτων προγραμματισμού διασφαλίζοντας ότι οι τύποι τιμών που χρησιμοποιούνται στο πρόγραμμα είναι συνεπείς. Η Java περιλαμβάνει επίσης αυτόματη διαχείριση μνήμης, η οποία μειώνει την ανάγκη για μη αυτόματη εκχώρηση και εκχώρηση μνήμης. Αυτή η δυνατότητα, μαζί με άλλα χαρακτηριστικά ασφαλείας, καθιστά την Java δημοφιλή γλώσσα για την ανάπτυξη ασφαλών εφαρμογών.

Η Java διαθέτει μια τεράστια τυπική βιβλιοθήκη που περιλαμβάνει κλάσεις και διεπαφές για διάφορες εργασίες προγραμματισμού, όπως δικτύωση, συνδεσιμότητα βάσεων δεδομένων και ανάπτυξη γραφικών διεπαφών χρήστη (GUI). Η τυπική βιβλιοθήκη χωρίζεται σε διαφορετικά πακέτα, το καθένα από τα οποία περιέχει σχετικές κλάσεις και διεπαφές.

Η Java υποστηρίζει ένα ευρύ φάσμα παραδειγμάτων προγραμματισμού, συμπεριλαμβανομένου διαδικαστικού, λειτουργικού και αντικειμενοστρεφούς προγραμματισμού. Ο αντικειμενοστραφής προγραμματισμός είναι ένα πρότυπο προγραμματισμού που εστιάζει στη χρήση αντικειμένων, που είναι περιπτώσεις κλάσεων, για την αναπαράσταση οντοτήτων και εννοιών του πραγματικού κόσμου. Η Java αναφέρεται συχνά ως μία από τις πιο δημοφιλείς αντικειμενοστρεφείς γλώσσες προγραμματισμού, με πολλές εφαρμογές μεγάλης κλίμακας για επιχειρήσεις να δημιουργούνται χρησιμοποιώντας Java.

Συνοπτικά, η Java είναι μια ευρέως χρησιμοποιούμενη, ανεξάρτητη από πλατφόρμα γλώσσα προγραμματισμού με πολλά χαρακτηριστικά που την καθιστούν κατάλληλη για την ανάπτυξη μιας ποικιλίας εφαρμογών. Ο ισχυρός μηχανισμός ελέγχου τύπου, η αυτόματη διαχείριση μνήμης και η τεράστια τυπική βιβλιοθήκη το καθιστούν δημοφιλή επιλογή για την ανάπτυξη ασφαλών και επεκτάσιμων εφαρμογών. Η υποστήριξή της για διαφορετικά παραδείγματα προγραμματισμού, συμπεριλαμβανομένου του αντικειμενοστρεφούς προγραμματισμού, την καθιστά μια ευέλικτη γλώσσα που μπορεί να χρησιμοποιηθεί για διάφορες εργασίες προγραμματισμού.

## 2.4 Επισκόπηση των Swing και JavaFX UI Frameworks

Οι γραφικές διεπαφές χρήστη (GUI) διαδραματίζουν ουσιαστικό ρόλο στη βελτίωση της εμπειρίας χρήστη των εφαρμογών λογισμικού. Η Java είναι μια δημοφιλής γλώσσα προγραμματισμού για την ανάπτυξη εφαρμογών επιτραπέζιου υπολογιστή, χάρη στην ανεξαρτησία της πλατφόρμας και το πλούσιο σύνολο βιβλιοθηκών. Οι τυπικές βιβλιοθήκες Java περιλαμβάνουν δύο βασικά πλαίσια για τη δημιουργία γραφικών διεπαφής επιτραπέζιων υπολογιστών: Swing και JavaFX. Αυτό το κεφάλαιο παρέχει μια επισκόπηση αυτών των δύο πλαισίων και των χαρακτηριστικών τους.

### 2.4.1 Swing Framework

Το Swing είναι μια ώριμη και ευρέως χρησιμοποιούμενη εργαλειοθήκη Java GUI που εισήχθη στις τάξεις Java Foundation (JFC) το 1997. Το Swing παρέχει ένα πλούσιο σύνολο στοιχείων διεπαφής χρήστη, όπως κουμπιά, ετικέτες, πεδία κειμένου, σύνθετα πλαίσια και πίνακες, μεταξύ των οι υπολοιποι. Περιλαμβάνει επίσης πολλούς διαχειριστές διάταξης για την οργάνωση αυτών των στοιχείων σε ένα γραφικό παράθυρο. Το Swing είναι μέρος της πλατφόρμας Java SE, καθιστώντας το ευρέως διαθέσιμο και εύκολο στη χρήση για προγραμματιστές Java.

Το Swing βασίζεται σε μια αρχιτεκτονική Model-View-Controller (MVC), όπου το μοντέλο αντιπροσωπεύει τα δεδομένα, η προβολή αντιπροσωπεύει την οπτική αναπαράσταση των δεδομένων και ο ελεγκτής χειρίζεται την αλληλεπίδραση του χρήστη με την προβολή. Αυτός ο διαχωρισμός ανησυχιών καθιστά τις εφαρμογές Swing πιο διατηρήσιμες και επεκτάσιμες. Το Swing είναι επίσης εξαιρετικά προσαρμόσιμο, επιτρέποντας στους προγραμματιστές να τροποποιούν την εμφάνιση και την αίσθηση των εφαρμογών τους.

### 2.4.2 JavaFX FrameWork

Το JavaFX είναι ένα σύγχρονο και πιο προηγμένο κιτ εργαλείων GUI που παρουσιάστηκε το 2008. Είναι διάδοχος του Swing και παρέχει ένα πλουσιότερο σύνολο στοιχείων διεπαφής χρήστη, συμπεριλαμβανομένων τρισδιάστατων γραφικών, κινούμενων εικόνων και υποστήριξης πολυμέσων. Το JavaFX υποστηρίζει γραφικά με επιτάχυνση υλικού, καθιστώντας το πιο αποκριτικό και αποτελεσματικό από το Swing. Περιλαμβάνει επίσης ένα API γραφήματος σκηνής για την οργάνωση και το χειρισμό γραφικών στοιχείων.

Το JavaFX βασίζεται σε μια αρχιτεκτονική γραφήματος σκηνής, όπου τα γραφικά στοιχεία αναπαρίστανται ως κόμβοι σε μια ιεραρχική δομή. Αυτό επιτρέπει τον ευκολότερο χειρισμό και την κίνηση των γραφικών στοιχείων. Το JavaFX υποστηρίζει επίσης την αρχιτεκτονική Model-View-ViewModel (MVVM), η οποία διαχωρίζει τη λογική της παρουσίασης από την επιχειρηματική λογική. Αυτό διευκολύνει τη συντήρηση και τη δοκιμή εφαρμογών JavaFX.

### 2.4.3 Σύγκριση Swing και JavaFX

Τόσο το Swing όσο και το JavaFX είναι ισχυρά και ώριμα frameworks για τη δημιουργία γραφικών διεπαφής επιτραπέζιων υπολογιστών σε Java. Ενώ το Swing είναι παλαιότερο και πιο καθιερωμένο, το JavaFX παρέχει πιο προηγμένες λειτουργίες και καλύτερη απόδοση. Το Swing είναι πιο κατάλληλο για την ανάπτυξη παραδοσιακών επιχειρηματικών εφαρμογών με απλούστερες διεπαφές χρήστη, ενώ το JavaFX είναι πιο κατάλληλο για τη δημιουργία πλούσιων και σύγχρονων διεπαφών χρήστη με γραφικά 3D και υποστήριξη πολυμέσων.

Tο Swing και το JavaFX είναι και τα δύο πολύτιμα πλαίσια για τη δημιουργία γραφικών διεπαφής επιτραπέζιων υπολογιστών σε Java. Οι προγραμματιστές μπορούν να επιλέξουν το πλαίσιο που ταιριάζει καλύτερα στις ανάγκες τους με βάση την πολυπλοκότητα της εφαρμογής τους, τις απαιτούμενες δυνατότητες και την επιθυμητή απόδοση.

# **Μεθοδολογία**

Το Κεφάλαιο 3 αυτής της εργασίας παρέχει μια επισκόπηση της μεθοδολογίας που χρησιμοποιείται για το σχεδιασμό και την ανάπτυξη της desktop εφαρμογής που βασίζεται σε Java για τη διαχείριση εικόνων στο Firebase Storage. Το κεφάλαιο περιγράφει λεπτομερώς την αρχιτεκτονική του συστήματος, τα εργαλεία και τις βιβλιοθήκες που χρησιμοποιούνται, τις λεπτομέρειες υλοποίησης και τη σχεδίαση της διεπαφής χρήστη. Επιπλέον, το κεφάλαιο εξηγεί τη διαδικασία δοκιμής και αξιολόγησης που χρησιμοποιείται για να διασφαλιστεί ότι η εφαρμογή πληροί τις απαιτούμενες προδιαγραφές και τις ανάγκες των χρηστών. Αυτό το κεφάλαιο παρέχει μια ολοκληρωμένη κατανόηση της διαδικασίας δημιουργίας της εφαρμογής και των εκτιμήσεων που έγιναν κατά την ανάπτυξη.

## 3.1 Maven και βιβλιοθήκες

Το Maven [8] είναι ένα ισχυρό εργαλείο αυτοματισμού κατασκευής για έργα Java, σχεδιασμένο να διαχειρίζεται εξαρτήσεις και να δημιουργεί διαδικασίες. Σε αυτό το project, το Maven χρησιμοποιείται για τη διαχείριση εξαρτήσεων και τη δημιουργία της εφαρμογής. Το αρχείο pom.xml του έργου ορίζει τα μεταδεδομένα του έργου, όπως το αναγνωριστικό ομάδας, το αναγνωριστικό τεχνουργήματος και τον αριθμό έκδοσης. Απαριθμεί επίσης τις εξαρτήσεις που απαιτούνται από το έργο.

Σε αυτό το έργο, χρησιμοποιούνται τρεις κύριες εξαρτήσεις: το Firebase SDK για Java, το Google Cloud Firestore και η βιβλιοθήκη Google Cloud Storage. Το Firebase SDK για Java επιτρέπει στο έργο να αλληλεπιδρά με υπηρεσίες Firebase, όπως ο έλεγχος ταυτότητας Firebase και το Firebase Cloud Firestore. Η εξάρτηση Google Cloud Firestore παρέχει τις απαραίτητες βιβλιοθήκες για εργασία με τη βάση δεδομένων Firestore. Τέλος, η βιβλιοθήκη Google Cloud Storage παρέχει τα εργαλεία αλληλεπίδρασης με την υπηρεσία Google Cloud Storage. Επιπλέον, η εξάρτηση logback-classic χρησιμοποιείται για την καταγραφή.

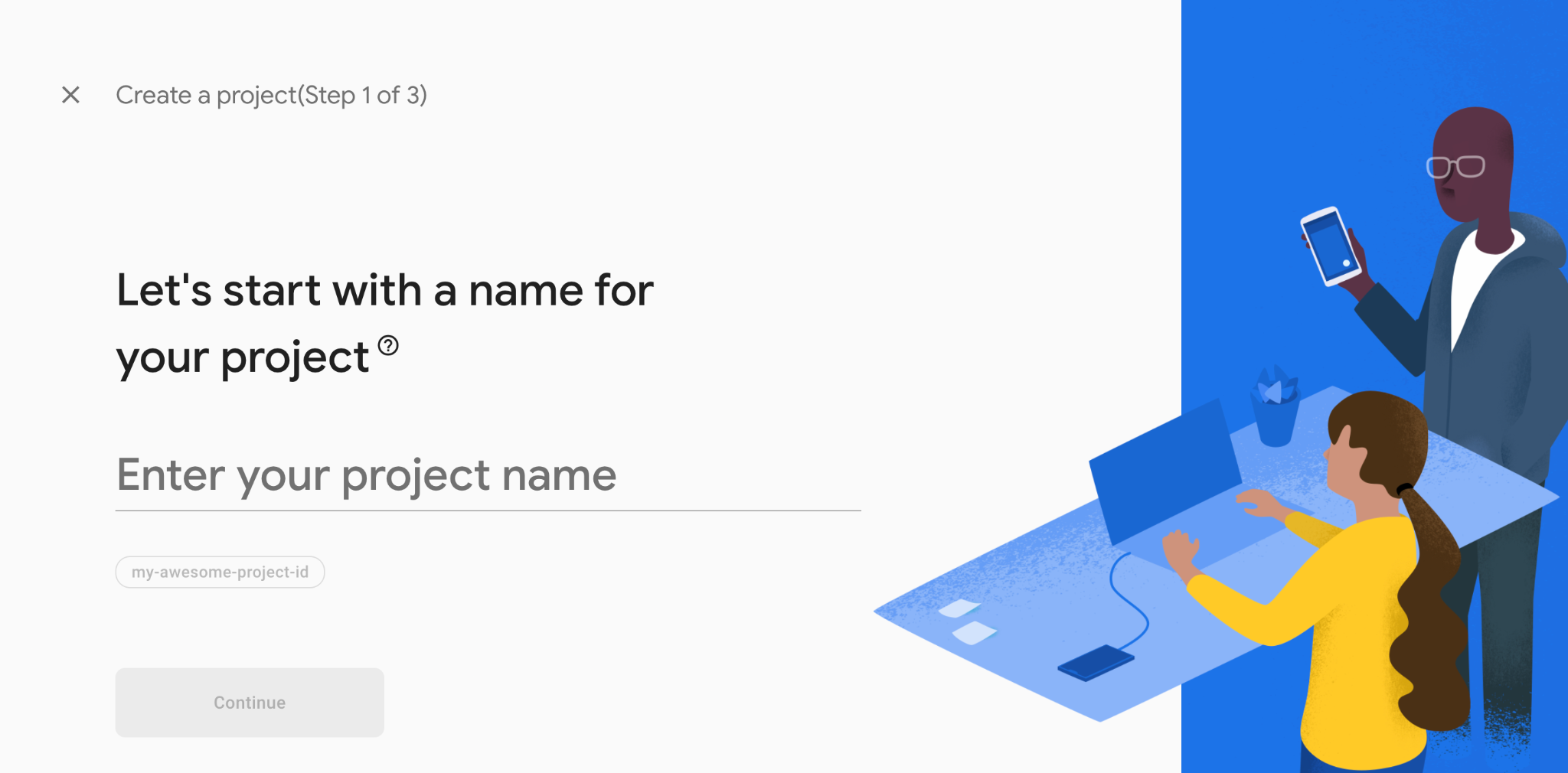
Χρησιμοποιώντας το Maven, οι προγραμματιστές μπορούν εύκολα να διαχειριστούν τις εξαρτήσεις, να δημιουργήσουν την εφαρμογή και να την αναπτύξουν σε διαφορετικά περιβάλλοντα. Χρησιμοποιώντας το Maven, οι προγραμματιστές μπορούν να επικεντρωθούν στη σύνταξη κώδικα χωρίς να ανησυχούν για τη διαχείριση της διαδικασίας κατασκευής ή την αντιμετώπιση προβλημάτων εξάρτησης.

To pom αρχείο αυτού του project είναι το παρακάτω:

| <project>  <modelVersion>4.0.0</modelVersion>  <groupId>Gallery</groupId>  <artifactId>Gallery</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>Photography</name>    <dependencies>  <!-- Firebase SDK for Java -->  <dependency>  <groupId>com.google.firebase</groupId>  <artifactId>firebase-admin</artifactId>  <version>8.0.0</version>  </dependency>    <dependency>  <groupId>com.google.cloud</groupId>  <artifactId>google-cloud-firestore</artifactId>  <version>3.8.1</version>  </dependency>  <!-- Google Cloud Storage library -->  <dependency>  <groupId>com.google.cloud</groupId>  <artifactId>google-cloud-storage</artifactId>  <version>1.107.0</version>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  <version>1.2.3</version>  </dependency>    </dependencies>  </project> |
| --- |

## 3.2 Firebase Project

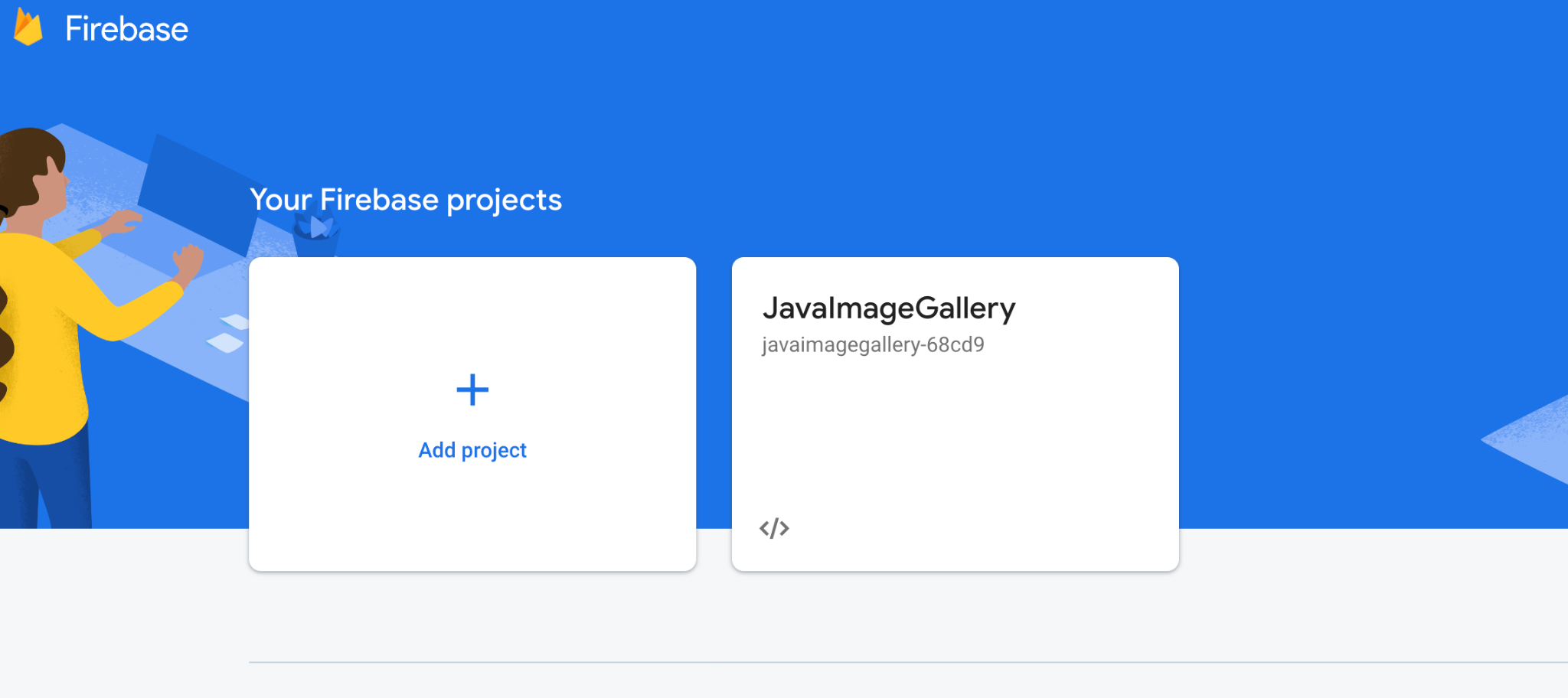
Για να δημιουργήσουμε ένα έργο Firebase, πήγαμε στον ιστότοπο του Firebase Console και κάνουμε κλικ στο "Προσθήκη έργου". Δώσαμε όνομα στο έργο μας και πατήσαμε «Δημιουργία έργου». Στη συνέχεια, κάναμε κλικ στο εικονίδιο «Ρυθμίσεις» και πήγαμε στην καρτέλα «Λογαριασμοί υπηρεσίας». Από εκεί, κάναμε κλικ στο "Δημιουργία νέου ιδιωτικού κλειδιού" για να δημιουργήσουμε ένα νέο αρχείο serviceAccountKey.json.



Εικόνα 3.1: Δημιουργία ενός νέου project στο firebase.

Για να συνδέσουμε την εφαρμογή Java στο έργο Firebase, προσθέσαμε την εξάρτηση firebase-admin στο αρχείο pom.xml. Στη συνέχεια, τοποθετήσαμε το αρχείο serviceAccountKey.json στον ριζικό κατάλογο του έργου Java. Τέλος, αρχικοποιήσαμε το FirebaseApp με το αρχείο serviceAccountKey.json χρησιμοποιώντας τον ακόλουθο κώδικα:

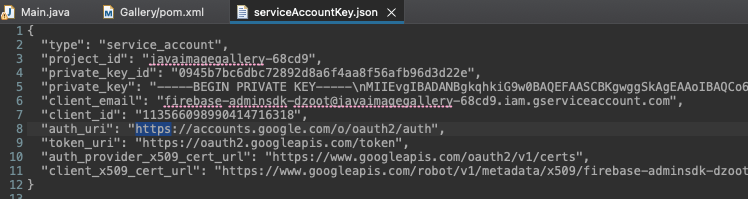
| FileInputStream serviceAccount =  new FileInputStream("path/to/serviceAccountKey.json");  FirebaseOptions options = new FirebaseOptions.Builder()  .setCredentials(GoogleCredentials.fromStream(serviceAccount))  .build();  FirebaseApp.initializeApp(options); |
| --- |



Εικόνα 3.2: Το Firebase project που δημιουργήσαμε

Μετά τη δημιουργία του Firebase project και τη δημιουργία του κλειδιού λογαριασμού υπηρεσίας, προχωρήσαμε στη δημιουργία μιας βάσης δεδομένων Cloud Firestore και ενός κάδου αποθήκευσης Firebase εντός του έργου. Επιλέξαμε το Cloud Firestore ως βάση δεδομένων μας επειδή παρέχει συγχρονισμό σε πραγματικό χρόνο και αυτόματη κλιμάκωση, καθιστώντας το κατάλληλο για την εφαρμογή συλλογής φωτογραφιών μας.

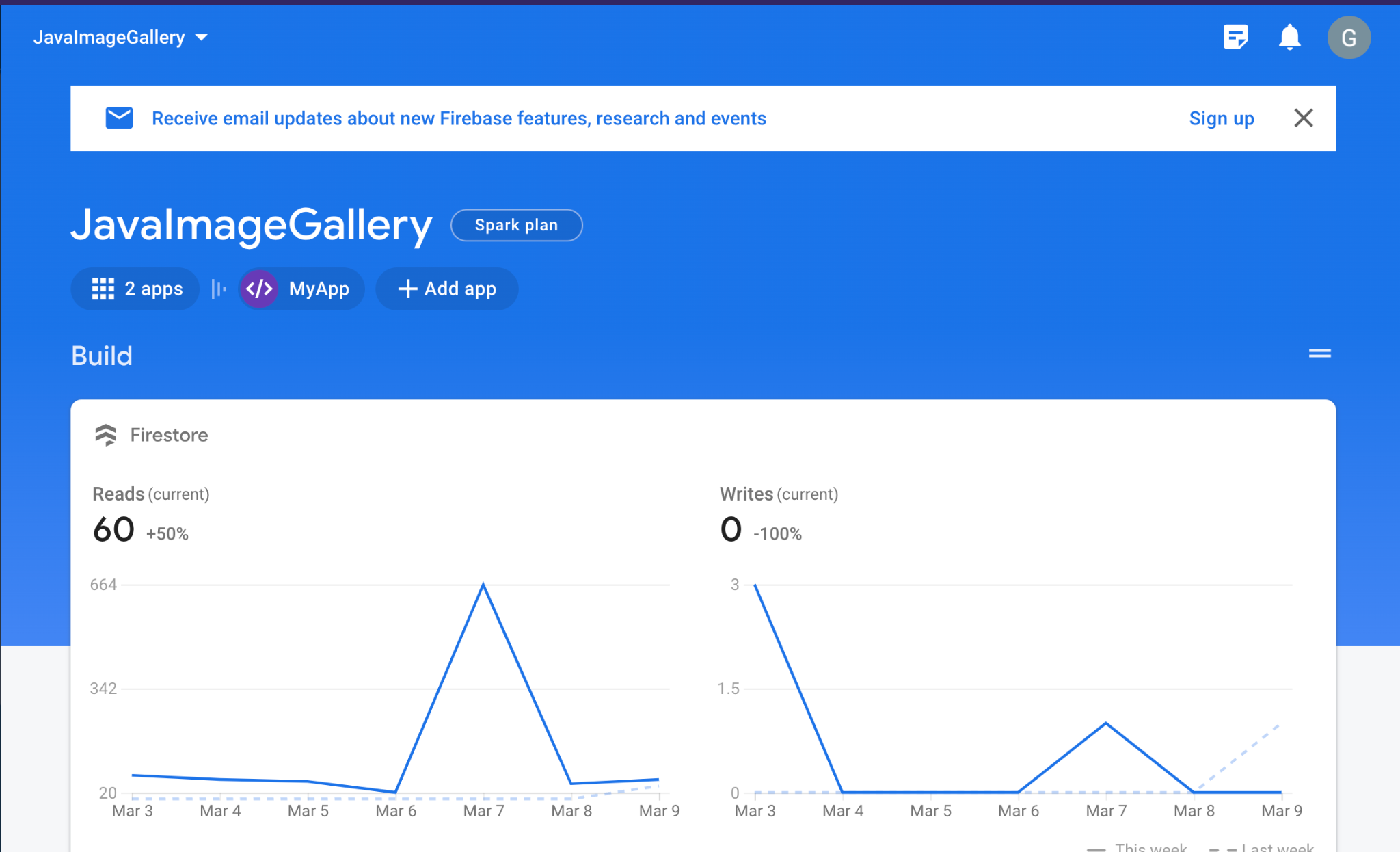
Για να δημιουργήσουμε τη βάση δεδομένων του Cloud Firestore, πήγαμε στην κονσόλα Firebase και πλοηγηθήκαμε στην καρτέλα "Βάση δεδομένων". Επιλέξαμε «Δημιουργία βάσης δεδομένων» και επιλέξαμε «Έναρξη σε δοκιμαστική λειτουργία» αφού μόλις αναπτύξαμε την εφαρμογή. Στη συνέχεια επιλέξαμε τη θέση της βάσης δεδομένων και προχωρήσαμε στη δημιουργία της.



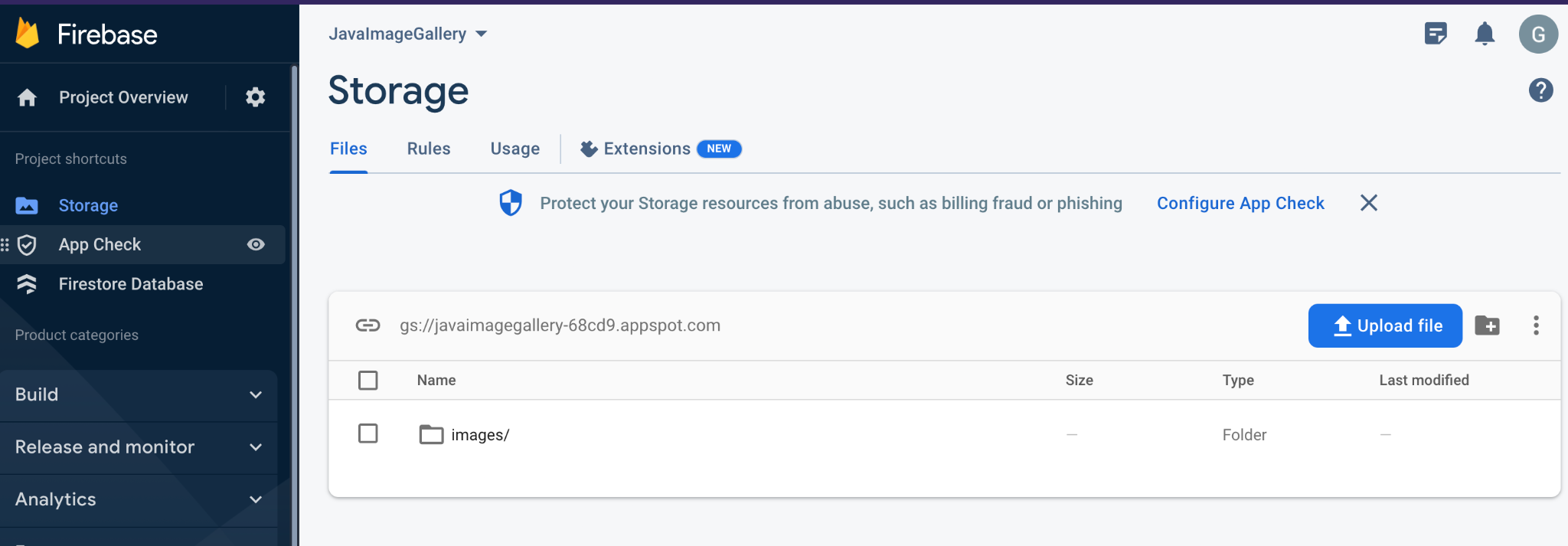
Εικόνα 3.3: Το αρχείο serviceAccountKey.json από το firebase

Για το Firebase Storage, πήγαμε στην κονσόλα Firebase και πλοηγηθήκαμε στην καρτέλα "Storage". Επιλέξαμε «Ξεκινήστε» και προχωρήσαμε στη δημιουργία νέου κάδου. Επιλέξαμε μια θέση για τον κάδο και επιλέξαμε την προεπιλεγμένη κλάση αποθήκευσης. Τέλος, ελέγξαμε τις ρυθμίσεις και δημιουργήσαμε τον κάδο.

Αυτά τα βήματα μας επέτρεψαν να δημιουργήσουμε τόσο μια βάση δεδομένων όσο και μια λύση αποθήκευσης για την εφαρμογή συλλογής φωτογραφιών στο πλαίσιο του Firebase project.



Εικόνα 3.4: Η αρχική σελίδα του project στο Firebase



Εικόνα 3.5: Το bucket στο Firebase Storage

Τα στοιχεία από το firebase ορίστηκαν σαν μεταβλητές στην αρχή της main στο πρόγραμμά μας:

| private static final String PROJECT\_ID = "javaimagegallery-68cd9";  private static final String BUCKET\_NAME = "javaimagegallery-68cd9.appspot.com";  private static final String SERVICE\_ACCOUNT\_KEY\_PATH = "src/main/java/serviceAccountKey.json"; |
| --- |

## 3.3 Συναρτήσεις επικοινωνίας με το backend

Στο σημείο αυτό θα παρουσιαστούν κάποιες γενικές συναρτήσεις που υλοποιήθηκαν οι οποίες αποτελούν σημαντικό κομμάτι του συστήματος. Οι συναρτήσεις αυτές αφορούν την επικοινωνία με το backend, δηλαδή με το Firebase Storage και στο Cloud Firestore.

### 3.3.1 uploadImageToFirebaseStorage

Η μέθοδος uploadImageToFirebaseStorage είναι μια μέθοδος που ανεβάζει μια εικόνα στο Firebase Storage χρησιμοποιώντας το Firebase SDK για Java. Η συνάρτηση λαμβάνει ως παράμετρο ένα αντικείμενο αρχείου που αντιπροσωπεύει το αρχείο εικόνας που πρέπει να φορτωθεί. Η συνάρτηση δημιουργεί ένα μοναδικό όνομα για την εικόνα χρησιμοποιώντας τη μέθοδο UUID.randomUUID().toString() και προσθέτοντας .jpg στο τέλος της συμβολοσειράς.

Μετά τη δημιουργία ενός μοναδικού ονόματος εικόνας, η συνάρτηση δημιουργεί ένα αντικείμενο BlobId το οποίο καθορίζει τη θέση στον κάδο αποθήκευσης Firebase όπου θα μεταφορτωθεί η εικόνα. Η σταθερά BUCKET\_NAME χρησιμοποιείται για τον καθορισμό του ονόματος του κάδου αποθήκευσης Firebase.

Στη συνέχεια, η συνάρτηση δημιουργεί ένα αντικείμενο BlobInfo που καθορίζει τα μεταδεδομένα για το αρχείο εικόνας. Τα μεταδεδομένα περιλαμβάνουν το αντικείμενο blobId που δημιουργήθηκε νωρίτερα, καθώς και τον τύπο περιεχομένου του αρχείου εικόνας, το οποίο έχει οριστεί σε image/jpeg.

Τέλος, το αρχείο εικόνας μεταφορτώνεται στο Firebase Storage χρησιμοποιώντας τη μέθοδο δημιουργίας του αντικειμένου αποθήκευσης, η οποία είναι μια παρουσία της κλάσης Storage. Η μέθοδος δημιουργίας λαμβάνει το αντικείμενο blobInfo και μια ροή εισόδου του αρχείου εικόνας, η οποία λαμβάνεται χρησιμοποιώντας την κλάση FileInputStream. Εάν η εικόνα μεταφορτωθεί με επιτυχία, η συνάρτηση επιστρέφει το μοναδικό όνομα εικόνας. Διαφορετικά, επιστρέφει μηδενικό.

| private String uploadImageToFirebaseStorage(File file) {  String imageName = UUID.randomUUID().toString() + ".jpg"; // generate a unique image name  System.out.println(imageName);  // create a bucket reference and upload the file to Firebase Storage  BlobId blobId = BlobId.of(BUCKET\_NAME, "images/" + imageName);  BlobInfo blobInfo = BlobInfo.newBuilder(blobId).setContentType("image/jpeg").build();  try {  storage.create(blobInfo, new FileInputStream(file));  System.out.println("Image uploaded successfully to Firebase Storage.");  } catch (FileNotFoundException e) {  e.printStackTrace();  }  return imageName;  } |
| --- |

### 3.3.2 saveImageDetailsToFirestore

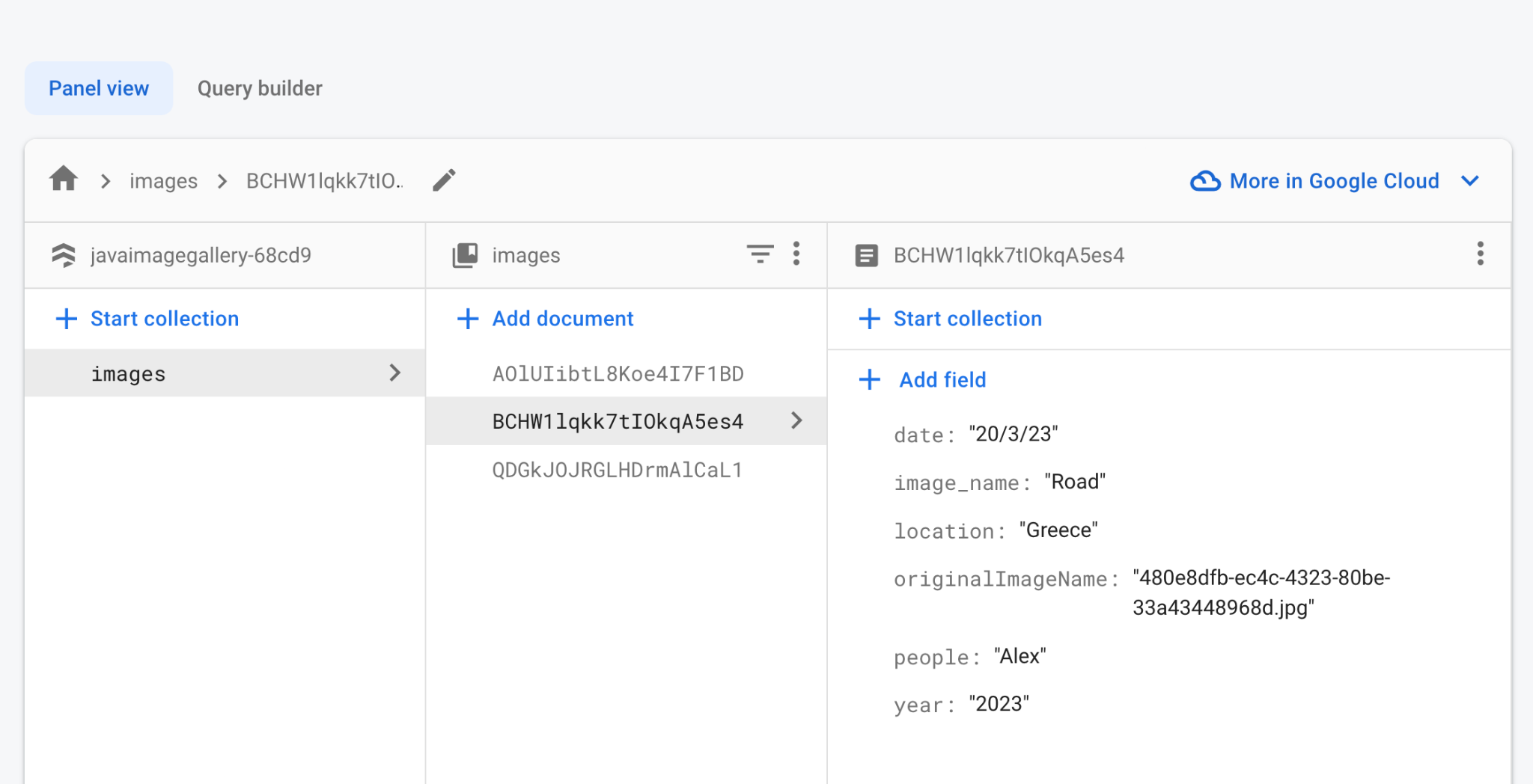
Η συνάρτηση saveImageDetailsToFirestore είναι υπεύθυνη για την αποθήκευση των λεπτομερειών μιας εικόνας σε μια βάση δεδομένων του Cloud Firestore. Η συνάρτηση λαμβάνει πολλές παραμέτρους όπως το όνομα εικόνας, το έτος, την τοποθεσία, την ημερομηνία, τα άτομα και το originalImageName. Αυτές οι παράμετροι αντιπροσωπεύουν τα μεταδεδομένα που σχετίζονται με την εικόνα.

Η συνάρτηση αρχικά αρχικοποιεί το FirebaseApp με τα διαπιστευτήρια λογαριασμού υπηρεσίας στο αρχείο SERVICE\_ACCOUNT\_KEY\_PATH. Στη συνέχεια, λαμβάνει μια παρουσία της βάσης δεδομένων του Cloud Firestore χρησιμοποιώντας το FirestoreClient.getFirestore().

Η συνάρτηση δημιουργεί ένα νέο έγγραφο στη συλλογή εικόνων με μοναδικό αναγνωριστικό. Στη συνέχεια δημιουργεί ένα αντικείμενο χάρτη που ονομάζεται δεδομένα που περιέχει τα μεταδεδομένα εικόνας. Τα μεταδεδομένα προστίθενται στο έγγραφο χρησιμοποιώντας τη μέθοδο set() του αντικειμένου DocumentReference που επιστρέφεται από την κλήση db.collection("images").document().

Η συνάρτηση χρησιμοποιεί το ApiFuture για την ασύγχρονη εγγραφή των δεδομένων στο έγγραφο. Η μέθοδος result.get() περιμένει να ολοκληρωθεί η εγγραφή πριν επιστρέψει. Εάν η εγγραφή είναι επιτυχής, η συνάρτηση εξάγει ένα μήνυμα στην κονσόλα που υποδεικνύει ότι οι λεπτομέρειες της εικόνας αποθηκεύτηκαν στο Firestore. Εάν υπάρχει σφάλμα, εκτυπώνεται ένα μήνυμα σφάλματος στην κονσόλα.

Aυτή η λειτουργία παίζει σημαντικό ρόλο στην αποθήκευση μεταδεδομένων εικόνας στη βάση δεδομένων του Cloud Firestore, επιτρέποντας στους χρήστες να αναζητούν και να ανακτούν εύκολα εικόνες με βάση διαφορετικά κριτήρια, όπως τοποθεσία, έτος και άτομα.



Εικόνα 3.6: Η βάση δεδομένων στο Cloud Firestore

| private void saveImageDetailsToFirestore(String imageName, String year, String location, String date, String people, String originalImageName) throws IOException {  // Create a new document with a unique ID  FileInputStream serviceAccount = new FileInputStream(SERVICE\_ACCOUNT\_KEY\_PATH);  FirebaseOptions options = new FirebaseOptions.Builder()  .setCredentials(GoogleCredentials.fromStream(serviceAccount))  .build();    try {  FirebaseApp.initializeApp(options);  } catch (Exception aa) {  System.out.println("Already Exists");  }    db = FirestoreClient.getFirestore();  DocumentReference docRef = db.collection("images").document();  Map<String, Object> data = new HashMap<String, Object>();  data.put("image\_name", imageName);  data.put("year", year);  data.put("location", location);  data.put("date", date);  data.put("people", people);  data.put("originalImageName", originalImageName);  // Add the data to the document  ApiFuture<WriteResult> result = docRef.set(data);  try {  // Wait for the write to complete  result.get();  System.out.println("Image details saved to Firestore");  } catch (InterruptedException e) {  System.err.println("Error saving image details to Firestore: " + e.getMessage());  } catch (ExecutionException e) {  System.err.println("Error saving image details to Firestore: " + e.getMessage());  }  } |
| --- |

### 3.3.3 loadImagesFromFirebaseStorage

Αυτή η συνάρτηση φορτώνει εικόνες από το Firebase Storage και τις εμφανίζει σε ένα Java Swing GUI (Graphical User Interface).

Η συνάρτηση λαμβάνει μια παράμετρο τοποθεσίας που πλέον δεν χρησιμοποιείται κάπου. Αρχικά λαμβάνει μια αναφορά στον κάδο αποθήκευσης Firebase χρησιμοποιώντας τη μέθοδο storage.get(). Στη συνέχεια ανακτά μια σελίδα με blobs (δηλαδή αρχεία εικόνας) από το φάκελο "images/" στον κάδο χρησιμοποιώντας τη μέθοδο bucket.list(), με τις επιλογές BlobListOption.prefix() και BlobListOption.currentDirectory() για περιορισμό της αναζήτησης στο φάκελο "εικόνες/".

Δημιουργείται ένας νέος πίνακας με διάταξη πλέγματος (grid) για τη συγκράτηση των εικόνων και, στη συνέχεια, η συνάρτηση επαναλαμβάνεται χρησιμοποιώντας έναν βρόχο for, προσθέτοντας κάθε εικόνα στον πίνακα. Για κάθε blob, η συνάρτηση λαμβάνει τη διεύθυνση URL λήψης χρησιμοποιώντας τη μέθοδο blob.getMediaLink() και δημιουργεί ένα εικονίδιο εικόνας με αυτήν τη διεύθυνση URL και μέγεθος 250x250 pixel χρησιμοποιώντας τη μέθοδο createScaledImageIcon(). Στη συνέχεια, δημιουργεί μια ετικέτα για να κρατήσει το εικονίδιο της εικόνας, προσθέτει έναν listener ποντικιού στην ετικέτα και, τέλος, προσθέτει την ετικέτα στον πίνακα.

Αφού προστεθούν όλες οι εικόνες στον πίνακα, το παράθυρο περιεχομένου του GUI ενημερώνεται για να εμφανίζει τις εικόνες και προστίθεται ένα νέο κουμπί "Upload Image" με ένα ActionListener που καλεί τη συνάρτηση onUploadImageLogic() όταν πατηθεί. Προστίθεται επίσης ένα κουμπί "Load Images", το οποίο όταν πατηθεί καλεί τη συνάρτηση loadImagesFromFirebaseStorage() με την παράμετρο "All" για να φορτώσει ξανά όλες τις εικόνες.

Στη συνέχεια, η συνάρτηση δημιουργεί ένα πλαίσιο για τα φίλτρα, ρυθμίζει τη βάση δεδομένων Firestore χρησιμοποιώντας το SDK διαχείρισης Firebase και δημιουργεί γραφικά στοιχεία για φιλτράρισμα εικόνων ανά έτος, άτομο και τοποθεσία. Το φίλτρο θέσης αρχικοποιείται με την παράμετρο θέσης που μεταβιβάζεται στη συνάρτηση. Τα γραφικά στοιχεία προστίθενται στον πίνακα φίλτρων, μαζί με ένα κουμπί "Apply" που εφαρμόζει τα επιλεγμένα φίλτρα στις εικόνες.

Όταν κάνουμε κλικ στο κουμπί "Apply", η συνάρτηση ανακτά τις επιλεγμένες τιμές φίλτρου, δημιουργεί ένα ερώτημα για το φιλτράρισμα των εικόνων με βάση τις επιλεγμένες τιμές και ανακτά τις αντίστοιχες εικόνες από τη βάση δεδομένων Firestore. Εάν δεν έχουν επιλεγεί φίλτρα, ανακτά όλες τις εικόνες. Στη συνέχεια, οι εικόνες που ταιριάζουν εμφανίζονται στο GUI σε ένα νέο παράθυρο όπως θα δούμε στο 3.7.

## 3.4 Ανέβασμα Φωτογραφίας

Η συνάρτηση Main τρέχει τις συναρτήσεις initialize() και initFirebase().

| private void initFirebase() {  try {  FileInputStream serviceAccount = new FileInputStream(SERVICE\_ACCOUNT\_KEY\_PATH);    storage = StorageOptions.newBuilder().setProjectId(PROJECT\_ID)  .setCredentials(GoogleCredentials.fromStream(serviceAccount)).build().getService();        } catch (IOException e) {  e.printStackTrace();  }  } |
| --- |

Η initFirebase προετοιμάζει ένα Firebase Storage instance για ένα δεδομένο project ID χρησιμοποιώντας το service account key file που βρίσκεται σε μια συγκεκριμένη διαδρομή στο σύστημα αρχείων.

Η συνάρτηση επιχειρεί πρώτα να διαβάσει το αρχείο κλειδιού λογαριασμού υπηρεσίας χρησιμοποιώντας ένα αντικείμενο FileInputStream. Η διαδρομή προς το αρχείο καθορίζεται από τη σταθερά SERVICE\_ACCOUNT\_KEY\_PATH.

Στη συνέχεια, η συνάρτηση δημιουργεί ένα νέο αντικείμενο StorageOptions χρησιμοποιώντας τη μέθοδο StorageOptions.newBuilder(). Ορίζει το αναγνωριστικό έργου για το αντικείμενο επιλογών χρησιμοποιώντας τη μέθοδο setProjectId(PROJECT\_ID), όπου το PROJECT\_ID είναι μια σταθερά που αντιπροσωπεύει το αναγνωριστικό έργου.

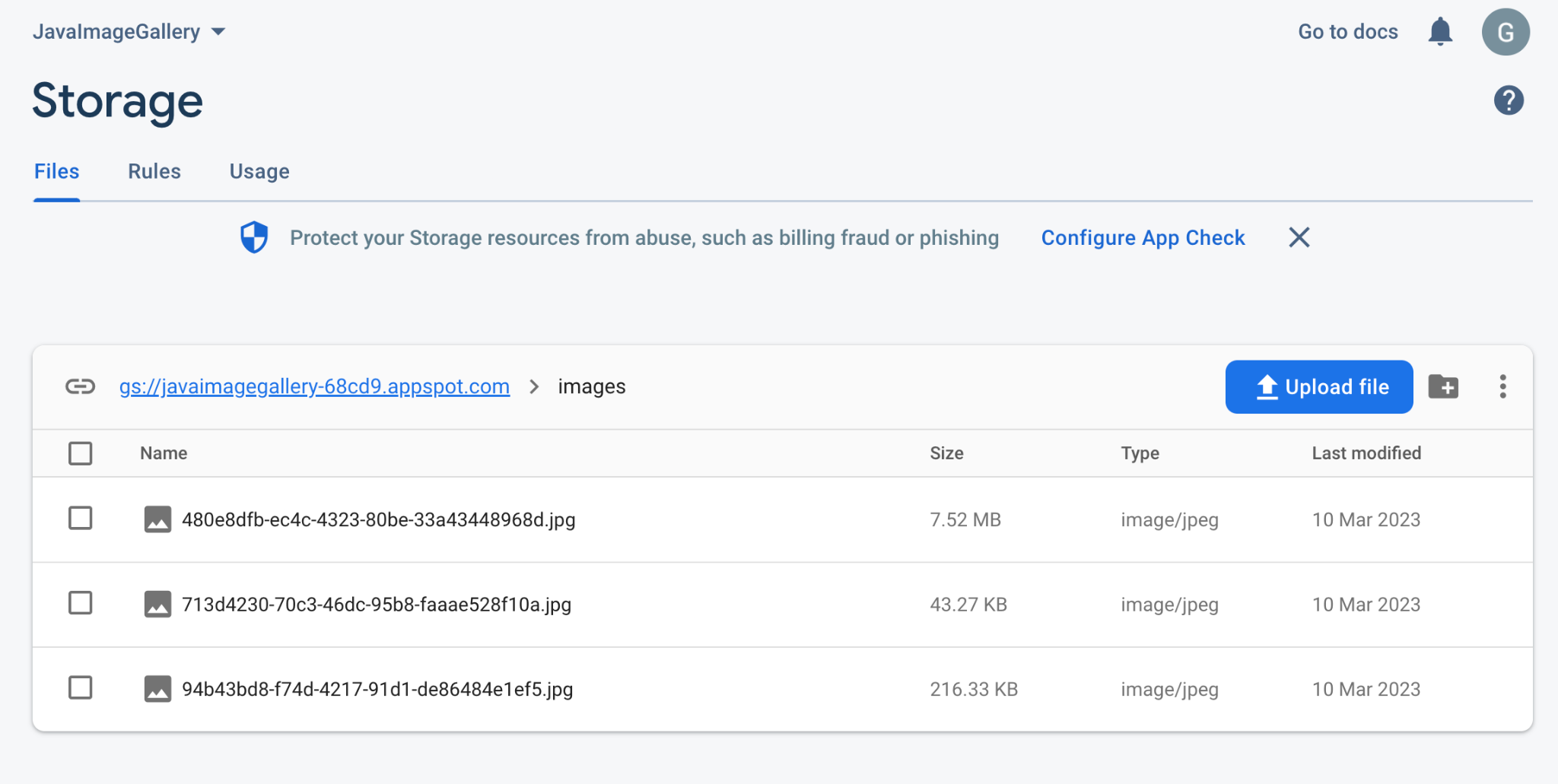
Στη συνέχεια, η συνάρτηση ορίζει τα διαπιστευτήρια για το αντικείμενο StorageOptions χρησιμοποιώντας τη μέθοδο setCredentials(), μεταβιβάζοντας ένα αντικείμενο GoogleCredentials που δημιουργήθηκε από το αρχείο κλειδιού λογαριασμού υπηρεσίας χρησιμοποιώντας τη μέθοδο GoogleCredentials.fromStream().

Τέλος, η συνάρτηση καλεί τη μέθοδο getService() στο αντικείμενο StorageOptions για να ανακτήσει την παρουσία του Firebase Storage και να την εκχωρήσει σε ένα πεδίο αποθήκευσης στην κλάση. Εάν υπάρχουν σφάλματα κατά τη διάρκεια αυτής της διαδικασίας, η συνάρτηση θα εκτυπώσει το ίχνος στοίβας για την εξαίρεση.

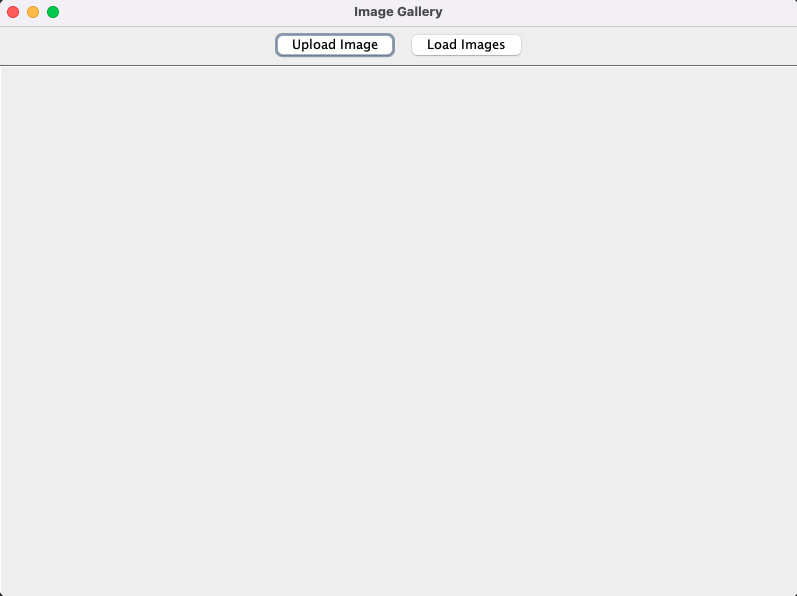
| private void initialize() {  frame = new JFrame();  frame.setTitle("Image Gallery");  frame.setBounds(100, 100, 800, 600);  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  contentPane = new JPanel();  contentPane.setLayout(new BorderLayout(0, 0));  frame.setContentPane(contentPane);  JPanel panelButtons = new JPanel();  panelButtons.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  contentPane.add(panelButtons, BorderLayout.NORTH);  btnUpload = new JButton("Upload Image");  btnUpload.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  onUploadImageLogic();  }  });  panelButtons.add(btnUpload);  btnLoadImages = new JButton("Load Images");  btnLoadImages.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  loadImagesFromFirebaseStorage("All");  }  });  panelButtons.add(btnLoadImages);        panelImages = new JPanel();  JScrollPane scrollPane = new JScrollPane(panelImages);  contentPane.add(scrollPane, BorderLayout.CENTER);  //contentPane.add(filtersPanel, BorderLayout.CENTER);    } |
| --- |

Η συνάρτηση initialize προετοιμάζει μια γραφική διεπαφή χρήστη (GUI) για την εφαρμογή. Ακολουθεί μια ανάλυση του τι κάνει η συνάρτηση:

* Δημιουργεί ένα νέο JFrame και ορίζει τον τίτλο, το μέγεθος και τη λειτουργία κλεισίματος.
* Δημιουργεί ένα νέο JPanel που ονομάζεται contentPane και ορίζει τη διάταξή του σε BorderLayout.
* Δημιουργεί ένα νέο JPanel που ονομάζεται panelButtons και ορίζει τη διάταξή του σε FlowLayout. Αυτός ο πίνακας θα κρατήσει τα δύο κουμπιά για τη μεταφόρτωση και τη φόρτωση εικόνων.
* Δημιουργεί ένα νέο JButton που ονομάζεται btnUpload με το κείμενο "Upload Image" και προσθέτει ένα ActionListener σε αυτό. Όταν κάνουμε κλικ στο κουμπί, θα καλέσει τη μέθοδο onUploadImageLogic().
* Προσθέτει το κουμπί btnUpload στον πίνακα panelButtons.
* Δημιουργεί ένα νέο JButton που ονομάζεται btnLoadImages με το κείμενο "Load Images" και προσθέτει ένα ActionListener σε αυτό. Όταν κάνουμε κλικ στο κουμπί, θα καλεστεί η μέθοδος loadImagesFromFirebaseStorage().
* Προσθέτει το κουμπί btnLoadImages στον πίνακα panelButtons.
* Δημιουργεί ένα νέο JPanel που ονομάζεται panelImages. Αυτός ο πίνακας θα κρατήσει τις φορτωμένες εικόνες.
* Δημιουργεί ένα νέο JScrollPane και προσθέτει το panelImages σε αυτό. Το JScrollPane προστίθεται στο contentPane χρησιμοποιώντας το BorderLayout.CENTER.



Εικόνα 3.7: Οι φωτογραφίες που έχουν ανέβει στο Firebase Storage.



Εικόνα 3.8: Η αρχική οθόνη της εφαρμογής

Το κουμπί “Upload Image” καλεί τη συνάρτηση onUploadImageLogic η οποία φαίνεται παρακάτω:

| private void onUploadImageLogic() {  // create a new JDialog  final JDialog uploadDialog = new JDialog(frame, "Upload Image Details", true);  uploadDialog.setLayout(new GridLayout(0, 2, 5, 5));  uploadDialog.setDefaultCloseOperation(JDialog.DISPOSE\_ON\_CLOSE);  uploadDialog.setSize(400, 200);  // create the form labels and fields  JLabel nameLabel = new JLabel("Image Name:");  final JTextField nameField = new JTextField();  JLabel yearLabel = new JLabel("Year:");  final JTextField yearField = new JTextField();  JLabel locationLabel = new JLabel("Location:");  final JTextField locationField = new JTextField();  JLabel dateLabel = new JLabel("Date:");  final JTextField dateField = new JTextField();  JLabel peopleLabel = new JLabel("People:");  final JTextField peopleField = new JTextField();  // add the labels and fields to the dialog  uploadDialog.add(nameLabel);  uploadDialog.add(nameField);  uploadDialog.add(yearLabel);  uploadDialog.add(yearField);  uploadDialog.add(locationLabel);  uploadDialog.add(locationField);  uploadDialog.add(dateLabel);  uploadDialog.add(dateField);  uploadDialog.add(peopleLabel);  uploadDialog.add(peopleField);  // create the submit button  JButton submitButton = new JButton("Submit");  submitButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  // save the image details to Firestore  String imageName = nameField.getText();  String year = yearField.getText();  String location = locationField.getText();  String date = dateField.getText();  String people = peopleField.getText();    // close the dialog and show the file chooser  uploadDialog.dispose();  String originalName = showFileChooser();  try {  saveImageDetailsToFirestore(imageName, year, location, date, people, originalName);  } catch (IOException e1) {  // TODO Auto-generated catch block  e1.printStackTrace();  }  }  });  uploadDialog.add(submitButton);  // show the upload dialog  uploadDialog.setVisible(true);  } |
| --- |

Η συνάρτηση onUploadImageLogic() χειρίζεται τη λογική για τη μεταφόρτωση μιας εικόνας και τις σχετικές λεπτομέρειες στη βάση δεδομένων Firebase Firestore.

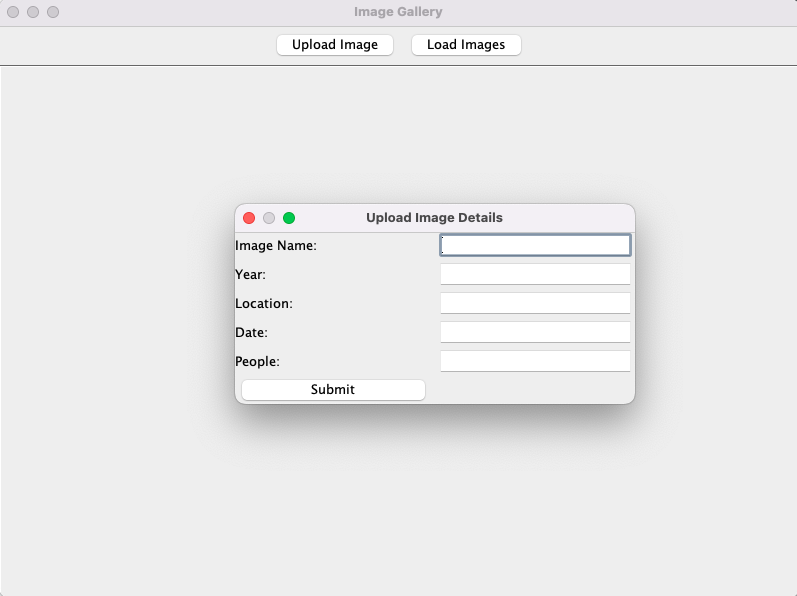
Η συνάρτηση δημιουργεί πρώτα ένα νέο JDialog, το οποίο είναι ένα ελαφρύ παράθυρο που χρησιμοποιείται για την εμφάνιση μιας φόρμας που πρέπει να συμπληρώσει ο χρήστης με τις λεπτομέρειες της εικόνας. Η φόρμα περιλαμβάνει πεδία για το όνομα της εικόνας, το έτος, την τοποθεσία, την ημερομηνία και τα άτομα.

Μόλις ο χρήστης συμπληρώσει τη φόρμα και κάνει κλικ στο κουμπί "Υποβολή", η συνάρτηση ανακτά τις τιμές από κάθε πεδίο και τις αποθηκεύει στο Firestore χρησιμοποιώντας τη συνάρτηση saveImageDetailsToFirestore(). Αυτή η συνάρτηση λαμβάνει τις λεπτομέρειες της εικόνας ως ορίσματα και χρησιμοποιεί το Firebase Firestore SDK για να αποθηκεύσει τα δεδομένα στη βάση δεδομένων.

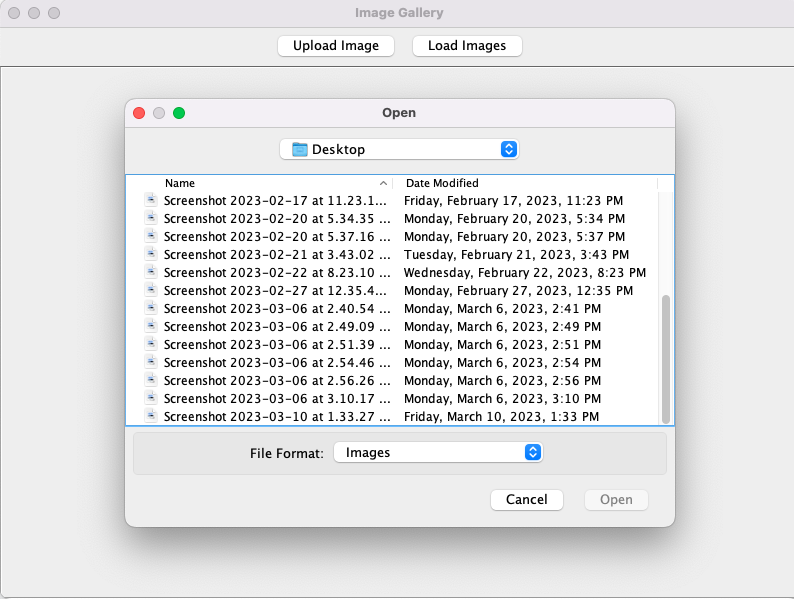
Στη συνέχεια, η συνάρτηση κλείνει το JDialog και καλεί τη συνάρτηση showFileChooser() για να επιτρέψει στον χρήστη να επιλέξει το αρχείο εικόνας για αποστολή. Μόλις ο χρήστης επιλέξει το αρχείο, η συνάρτηση καλεί ξανά το saveImageDetailsToFirestore(), μεταβιβάζοντας το αρχικό όνομα αρχείου της εικόνας ως πρόσθετο όρισμα.

Εάν παρουσιαστούν σφάλματα κατά τη διαδικασία μεταφόρτωσης, καταγράφονται και εκτυπώνονται στην κονσόλα χρησιμοποιώντας τη συνάρτηση printStackTrace().

Αυτή η συνάρτηση είναι υπεύθυνη για τη συλλογή των λεπτομερειών της εικόνας από τον χρήστη, την αποθήκευση τους στη βάση δεδομένων Firebase Firestore και, στη συνέχεια, την έναρξη της διαδικασίας αποστολής αρχείων.



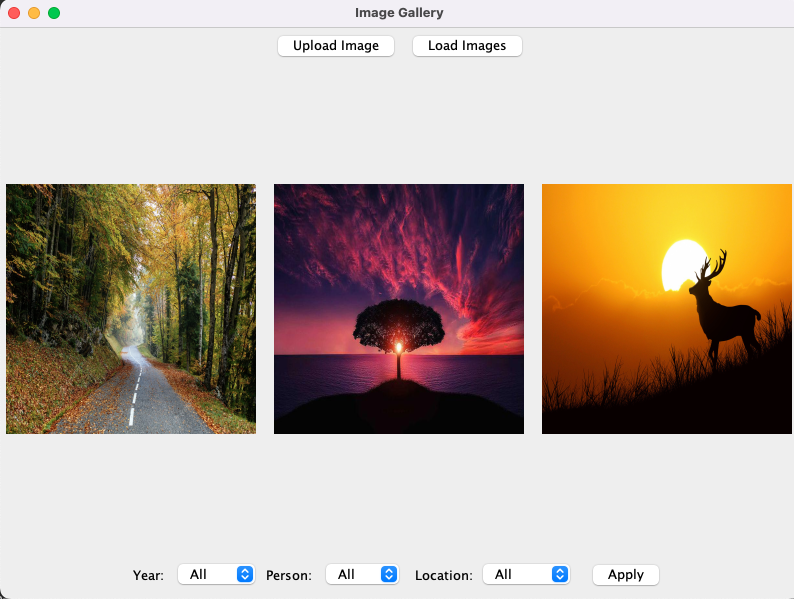
Εικόνα 3.9: Το popup εισαγωγής στοιχείων για το ανέβασμα μιας νέας εικόνας



Εικόνα 3.10: Το popup επιλογής αρχείου από το file system.

## 3.5 Φόρτωση Φωτογραφιών

Για να γίνει φόρτωση των φωτογραφιών και αυτές να εμφανιστούν στο Grid χρησιμοποιούμε τη συνάρτηση loadImagesFromFirebaseStorage που περιγράφηκε προηγουμένως.



Εικόνα 3.11: Η οθόνη όπου εμφανίζονται οι εικόνες

Η συνάρτηση αυτή εμφανίζει τις εικόνες που έχουν αποθηκευτεί στη βάση και εμφανίζει και το μενού με τα φίλτρα το οποίο αναλύεται στο 3.7.

## 3.6 Προβολή Στοιχείων

Στη loadImagesFromFirebaseStorage, για κάθε εικόνα υλοποιείται ένας listener ο οποίος ενεργοποιείται στο κλικ πάνω σε κάθε εικόνα. Το action αυτό εμφανίζει ένα νέο παράθυρο με την εικόνα καθώς και τις πληροφορίες για αυτή από το Cloud Firestore. Στο παράθυρο αυτό υπάρχει επιπλέον και ένα κουμπί “Download” το οποίο επιτρέπει το κατέβασμα μιας εικόνας.

Ο κώδικας για τον listener καθώς και για το Download button φαίνεται παρακάτω:

| label.addMouseListener(new MouseListener() {  public void mouseClicked(java.awt.event.MouseEvent e) {  final JFrame imageFrame = new JFrame();  imageFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);  imageFrame.setSize(500, 500);  // create a new label to hold the image in the bigger frame  JLabel imageLabel = new JLabel(icon);  imageLabel.setHorizontalAlignment(JLabel.CENTER);  imageLabel.setVerticalAlignment(JLabel.CENTER);  // add the label to the bigger frame and show it  imageFrame.getContentPane().add(imageLabel);  // create a download button  JButton downloadButton = new JButton("Download");  downloadButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  try {  // open a connection to the image URL  URL url = new URL(downloadUrl);  HttpURLConnection conn = (HttpURLConnection) url.openConnection();  conn.setRequestMethod("GET");  conn.connect();  // get the input stream for the connection  InputStream in = conn.getInputStream();  // create a file chooser to save the image  JFileChooser fileChooser = new JFileChooser();  fileChooser.setDialogTitle("Save Image");  fileChooser.setSelectedFile(new File("image.jpg"));  // show the save dialog and get the selected file  int result = fileChooser.showSaveDialog(imageFrame);  if (result == JFileChooser.APPROVE\_OPTION) {  File file = fileChooser.getSelectedFile();  // write the input stream to the selected file  OutputStream out = new FileOutputStream(file);  byte[] buffer = new byte[4096];  int bytesRead;  while ((bytesRead = in.read(buffer)) != -1) {  out.write(buffer, 0, bytesRead);  }  out.close();  in.close();  }  } catch (IOException ex) {  ex.printStackTrace();  }  }  });    // show the image details  String originalImageName = getImageOriginalNameFromDownloadUrl(downloadUrl).replace("images%2F","");  System.out.println("\*\*\*\*\*\*");  System.out.println(originalImageName);  FileInputStream serviceAccount = null;  try {  serviceAccount = new FileInputStream(SERVICE\_ACCOUNT\_KEY\_PATH);  } catch (FileNotFoundException e2) {  // TODO Auto-generated catch block  e2.printStackTrace();  }  FirebaseOptions options = null;  try {  options = new FirebaseOptions.Builder()  .setCredentials(GoogleCredentials.fromStream(serviceAccount))  .build();  } catch (IOException e2) {  // TODO Auto-generated catch block  e2.printStackTrace();  }  try {  FirebaseApp.initializeApp(options);  } catch (Exception aa) {  System.out.println("Already Exists");  }  db = FirestoreClient.getFirestore();  CollectionReference imagesCollectionRef = db.collection("images");  Query query = imagesCollectionRef.whereEqualTo("originalImageName", originalImageName);  ApiFuture<QuerySnapshot> querySnapshot = query.get();  QuerySnapshot snapshot = null;  try {  snapshot = querySnapshot.get();  } catch (InterruptedException e1) {  // TODO Auto-generated catch block  e1.printStackTrace();  } catch (ExecutionException e1) {  // TODO Auto-generated catch block  e1.printStackTrace();  }  if (!snapshot.isEmpty()) {  DocumentSnapshot document = snapshot.getDocuments().get(0);  String imageName = document.getString("image\_name");  String year = document.getString("year");  String location = document.getString("location");  String date = document.getString("date");  String people = document.getString("people");  JLabel nameLabel = new JLabel("Name: " + imageName);  JLabel yearLabel = new JLabel("Year: " + year);  JLabel locationLabel = new JLabel("Location: " + location);  JLabel dateLabel = new JLabel("Date: " + date);  JLabel peopleLabel = new JLabel("People: " + people);  JPanel detailsPanel = new JPanel(new GridLayout(5, 1));  detailsPanel.add(nameLabel);  detailsPanel.add(yearLabel);  detailsPanel.add(locationLabel);  detailsPanel.add(dateLabel);  detailsPanel.add(peopleLabel);  imageFrame.getContentPane().add(detailsPanel, BorderLayout.NORTH);  }  // create a panel to hold the download button  JPanel buttonPanel = new JPanel();  buttonPanel.add(downloadButton);  imageFrame.getContentPane().add(buttonPanel, BorderLayout.SOUTH);  imageFrame.setVisible(true);  }  public void mousePressed(java.awt.event.MouseEvent e) {  // TODO Auto-generated method stub    }  public void mouseReleased(java.awt.event.MouseEvent e) {  // TODO Auto-generated method stub    }  public void mouseEntered(java.awt.event.MouseEvent e) {  // TODO Auto-generated method stub    }  public void mouseExited(java.awt.event.MouseEvent e) {  // TODO Auto-generated method stub    }  }); |
| --- |

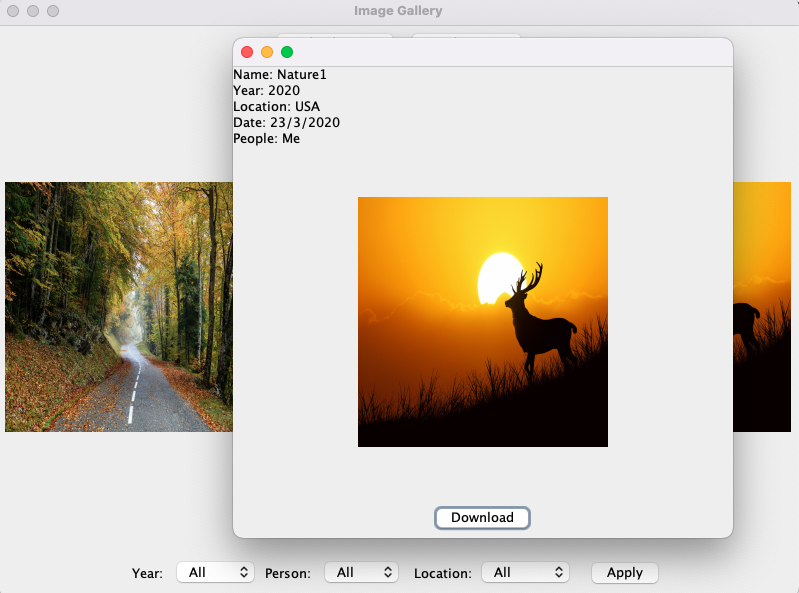
Ακολουθεί μια αναλυτική ανάλυση βήμα προς βήμα για το τι κάνει ο κώδικας:

* Προσθέτει έναν listener ποντικιού στην ετικέτα (δηλαδή στην εικόνα) . Όταν γίνεται κλικ στην ετικέτα, ενεργοποιεί τη μέθοδο mouseClicked() της διεπαφής MouseListener.
* Δημιουργεί ένα νέο JFrame για την εμφάνιση της εικόνας και ορίζει τις ιδιότητές του, όπως το μέγεθος και την προεπιλεγμένη λειτουργία κλεισίματος.
* Δημιουργεί ένα νέο JLabel για να κρατά την εικόνα στο μεγαλύτερο πλαίσιο και ορίζει την ευθυγράμμισή της.
* Προσθέτει την ετικέτα στο μεγαλύτερο πλαίσιο και τη δείχνει.
* Δημιουργεί ένα κουμπί "Download" και προσθέτει ένα ActionListener σε αυτό. Όταν κάνετε κλικ στο κουμπί, πραγματοποιεί λήψη της εικόνας στο τοπικό σύστημα αρχείων.
* Εμφανίζει τις λεπτομέρειες της εικόνας, όπως όνομα εικόνας, έτος, τοποθεσία, ημερομηνία και άτομα που σχετίζονται με την εικόνα. Ανακτά αυτές τις πληροφορίες από μια βάση δεδομένων Firestore.
* Δημιουργεί ένα JPanel για να κρατήσει πατημένο το κουμπί λήψης και το προσθέτει στο κάτω μέρος του μεγαλύτερου πλαισίου.
* Κάνει ορατό το μεγαλύτερο πλαίσιο.

Ο listener ποντικιού σε αυτόν τον κώδικα παρέχει έναν τρόπο εμφάνισης μιας εικόνας μαζί με τις λεπτομέρειες της και ένα κουμπί λήψης. Μπορεί να είναι χρήσιμο σε γκαλερί εικόνων ή άλλες εφαρμογές όπου οι χρήστες πρέπει να δουν και να κατεβάσουν εικόνες.

O listener του download button λειτουργεί ως εξής:

* Όταν κάνουμε κλικ στο κουμπί Download, καλείται η μέθοδος actionPerformed.
* Ο κώδικας δημιουργεί ένα νέο αντικείμενο URL με το URL της εικόνας προς λήψη.
* Στη συνέχεια δημιουργεί ένα αντικείμενο HttpURLConnection για σύνδεση με τη διεύθυνση URL και ορίζει τη μέθοδο αιτήματος σε "GET".
* Ο κώδικας συνδέεται με τη διεύθυνση URL και ανακτά το InputStream για τη σύνδεση.
* Δημιουργείται ένα παράθυρο διαλόγου JFileChooser που επιτρέπει στον χρήστη να επιλέξει πού θα αποθηκεύσει το αρχείο εικόνας.
* Εάν ο χρήστης επιλέξει μια θέση αρχείου και κάνει κλικ στο "Αποθήκευση", ο κωδικός λαμβάνει το επιλεγμένο αρχείο και δημιουργεί μια OutputStream για να γράψει τα δεδομένα εικόνας σε αυτό.
* Ο κώδικας διαβάζει το InputStream σε κομμάτια των 4096 byte και γράφει κάθε κομμάτι στη OutputStream μέχρι να φτάσει στο τέλος του InputStream.
* Στη συνέχεια, το OutputStream και το InputStream κλείνουν και το αρχείο εικόνας έχει ληφθεί και αποθηκευτεί στην επιλεγμένη τοποθεσία του χρήστη.
* Εάν παρουσιαστεί ένα IOException σε οποιοδήποτε σημείο κατά τη διάρκεια αυτής της διαδικασίας, το ίχνος στοίβας εκτυπώνεται στην κονσόλα.



Εικόνα 3.12: Επιλέγοντας μία εικόνα μπορούμε να δούμε τις πληροφορίες για αυτή σε ένα νέο παράθυρο.

## 3.7 Φίλτρα

Τα φίλτρα "year", "person" και "location" είναι βασικά φίλτρα που επιτρέπουν στους χρήστες να φιλτράρουν και να αναζητούν φωτογραφίες με βάση συγκεκριμένα κριτήρια. Το φίλτρο "year" δίνει τη δυνατότητα στους χρήστες να προβάλλουν φωτογραφίες που τραβήχτηκαν μέσα σε ένα συγκεκριμένο έτος ή εύρος ετών. Το φίλτρο "person" επιτρέπει στους χρήστες να φιλτράρουν τις φωτογραφίες με βάση τα άτομα σε αυτές, είτε πρόκειται για ένα συγκεκριμένο άτομο είτε για ομάδα ατόμων. Τέλος, το φίλτρο "location" επιτρέπει στους χρήστες να προβάλλουν φωτογραφίες που τραβήχτηκαν σε μια συγκεκριμένη τοποθεσία ή μια ομάδα τοποθεσιών. Αυτά τα φίλτρα παρέχουν έναν βολικό τρόπο στους χρήστες να βρίσκουν και να προβάλλουν γρήγορα τις φωτογραφίες που τους ενδιαφέρουν, κάνοντας τη συνολική εμπειρία διαχείρισης φωτογραφιών πιο αποτελεσματική και φιλική προς το χρήστη.

Για την υλοποίηση των φίλτρων χρησιμοποιούμε ξεχωριστά widgets τα οποία βρίσκονται σε ξεχωριστές κλάσεις.

### 3.7.1 YearFilterWidget

Η κλάση YearFilterWidget είναι ένα widget JComboBox που αντιπροσωπεύει μια αναπτυσσόμενη λίστα (dropdown) ετών που χρησιμοποιείται ως φίλτρο σε μια εφαρμογή Java Swing. Η μεταβλητή παρουσίας ετών είναι μια λίστα συμβολοσειρών που περιέχουν μοναδικά έτη που λαμβάνονται από μια βάση δεδομένων Firestore. Ο κατασκευαστής της κλάσης παίρνει ένα αντικείμενο βάσης δεδομένων Firestore ως είσοδο και ρωτά τη συλλογή εικόνων για όλα τα μοναδικά έτη, τα οποία στη συνέχεια προστίθενται στη λίστα ετών. Το αντικείμενο DefaultComboBoxModel χρησιμοποιείται για τη συμπλήρωση της αναπτυσσόμενης λίστας με τα έτη από τη λίστα ετών. Η μέθοδος getSelectedYear επιστρέφει το τρέχον επιλεγμένο έτος ως συμβολοσειρά. Αυτή η κλάση χρησιμοποιείται για τη δημιουργία ενός γραφικού στοιχείου φίλτρου που επιτρέπει στους χρήστες να φιλτράρουν τις εικόνες στην εφαρμογή με βάση το έτος λήψης.

| public class YearFilterWidget extends JComboBox<String> {  private List<String> years;  public DefaultComboBoxModel<String> model;  public YearFilterWidget(Firestore db) {  super();  this.years = new ArrayList<String>();  this.years.add("All");  // Query the Firestore collection for all unique years  ApiFuture<QuerySnapshot> future = db.collection("images").orderBy("date").get();  try {  QuerySnapshot documents = future.get();  for (QueryDocumentSnapshot document : documents) {  String year = document.getString("year");  if (!years.contains(year)) {  years.add(year);  }  }  } catch (Exception e) {  System.err.println("Error retrieving years from Firestore: " + e.getMessage());  }  // Add the years to the dropdown  String[] yearArray = new String[years.size()];  years.toArray(yearArray);  model = new DefaultComboBoxModel<String>(yearArray);  this.setModel(model);  }  public String getSelectedYear() {  return (String) this.getSelectedItem();  }  } |
| --- |

### 3.7.2 PeopleFilterWidget

Η κλάση PeopleFilterWidget επεκτείνει την κλάση JComboBox και παρέχει ένα αναπτυσσόμενο μενού για φιλτράρισμα εικόνων με βάση τα άτομα που εμφανίζονται σε αυτές. Ο κατασκευαστής αυτής της κλάσης λαμβάνει ένα αντικείμενο Firestore ως είσοδο και εκτελεί ένα ερώτημα για να ανακτήσει όλα τα μοναδικά άτομα από τη συλλογή "images". Τα ανακτηθέντα άτομα προστίθενται σε μια ArrayList και η επιλογή "Όλα" περιλαμβάνεται επίσης στη λίστα. Στη συνέχεια, τα άτομα προστίθενται στο αναπτυσσόμενο μενού χρησιμοποιώντας ένα DefaultComboBoxModel. Τέλος, παρέχεται η μέθοδος getSelectedPerson() για την ανάκτηση του επιλεγμένου ατόμου από το αναπτυσσόμενο μενού.

| public class PeopleFilterWidget extends JComboBox<String> {  private List<String> people;  private DefaultComboBoxModel<String> model;  public PeopleFilterWidget(Firestore db) {  super();  this.people = new ArrayList<String>();  this.people.add("All");  // Query the Firestore collection for all unique people  ApiFuture<QuerySnapshot> future = db.collection("images").get();  try {  QuerySnapshot documents = future.get();  for (QueryDocumentSnapshot document : documents) {  String peopleString = document.getString("people");  if (peopleString != null && !peopleString.isEmpty()) {  String[] peopleArray = peopleString.split(",\\s\*");  for (String person : peopleArray) {  if (!people.contains(person)) {  people.add(person);  }  }  }  }  } catch (Exception e) {  System.err.println("Error retrieving people from Firestore: " + e.getMessage());  }  // Add the people to the dropdown  String[] peopleArray = new String[people.size()];  people.toArray(peopleArray);  model = new DefaultComboBoxModel<String>(peopleArray);  this.setModel(model);  }  public String getSelectedPerson() {  return (String) this.getSelectedItem();  }  } |
| --- |

### 3.7.3 LocationFilterWidget

Η παρακάτω κλάση ονομάζεται LocationFilterWidget και χρησιμοποιείται για τη δημιουργία ενός JPanel που περιέχει ένα JComboBox για φιλτράρισμα εικόνων ανά τοποθεσία. Λαμβάνει μια παρουσία Firestore και μια προεπιλεγμένη θέση ως παραμέτρους.

Μέσα στον κατασκευαστή, προστίθεται μια ετικέτα "Location:" στον πίνακα και αρχικοποιείται ένα κενό ArrayList και JComboBox. Στη συνέχεια, γίνεται ένα ερώτημα στη συλλογή "images" του Firestore για την ανάκτηση όλων των εγγράφων και των τιμών του πεδίου "location”. Τα αποτελέσματα χρησιμοποιούνται για τη συμπλήρωση τόσο του ArrayList όσο και του JComboBox με μοναδικές τοποθεσίες.

Τέλος, η κλάση περιέχει μια μέθοδο που ονομάζεται "getSelectedLocation()" η οποία επιστρέφει την τρέχουσα επιλεγμένη θέση από το JComboBox.

| public class LocationFilterWidget extends JPanel {  public JComboBox<String> locationDropdown;  private ArrayList<String> locations;    public LocationFilterWidget(Firestore db, final String defaultLocation) {  setLayout(new FlowLayout());  JLabel locationLabel = new JLabel("Location:");  add(locationLabel);  locations = new ArrayList<String>();  locationDropdown = new JComboBox<String>();    add(locationDropdown);  CollectionReference collection = db.collection("images");  ApiFuture<QuerySnapshot> future = collection.get();  locations.add("All");  locationDropdown.addItem("All");  try {  QuerySnapshot documents = future.get();  for (DocumentSnapshot document : documents) {  String location = document.getString("location");  if (location != null && !locations.contains(location)) {  locations.add(location);  locationDropdown.addItem(location);  }  }  } catch (Exception e) {  System.err.println("Error getting documents: " + e.getMessage());  }  }  public String getSelectedLocation() {  return (String) locationDropdown.getSelectedItem();  }  } |
| --- |

### 3.7.4 Εφαρμογή Φίλτρων

Τα πεδία αυτά προστίθενται στο γραφικό περιβάλλον στη μέθοδο loadImagesFromFirebaseStorage, όπως φαίνεται παρακάτω:

| YearFilterWidget yearFilter = new YearFilterWidget(db);  filtersPanel.add(new JLabel("Year: "));  filtersPanel.add(yearFilter);        PeopleFilterWidget peopleFilter = new PeopleFilterWidget(db);  filtersPanel.add(new JLabel("Person: "));  filtersPanel.add(peopleFilter);      final JComboBox<String> peopleDropdown = peopleFilter;      final JComboBox<String> yearDropdown = yearFilter;    LocationFilterWidget locationfilterWidget = new LocationFilterWidget(db,location);  final JComboBox<String> locationDropdown = locationfilterWidget.locationDropdown;  JButton applyButton = new JButton("Apply");  filtersPanel.add(locationfilterWidget);  filtersPanel.add(applyButton); |
| --- |

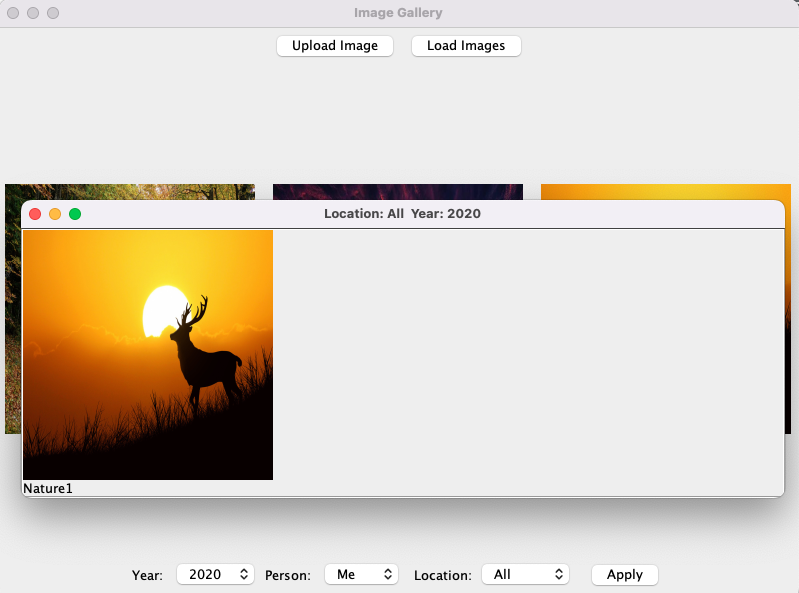
Μαζί με αυτά τα widgets εισάγουμε και ένα κουμπί Αpply, το οποίο θα εφαρμόσει τα επιλεγμένα φίλτρα. Τα αποτελέσματα των φίλτρων θα εμφανιστούν σε ξεχωριστό παράθυρο μόλις πατηθεί το κουμπί Αpply.

Ακολουθεί ο listener του apply button:

| applyButton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  String selectedLocation = locationDropdown.getSelectedItem().toString();  String selectedYear = yearDropdown.getSelectedItem().toString();  String selectedPerson = peopleDropdown.getSelectedItem().toString();    // Filter images by location  Firestore db;    FileInputStream serviceAccount = null;  try {  serviceAccount = new FileInputStream(SERVICE\_ACCOUNT\_KEY\_PATH);  } catch (FileNotFoundException e2) {  // TODO Auto-generated catch block  e2.printStackTrace();  }  FirebaseOptions options = null;  try {  options = new FirebaseOptions.Builder()  .setCredentials(GoogleCredentials.fromStream(serviceAccount))  .build();  } catch (IOException e2) {  // TODO Auto-generated catch block  e2.printStackTrace();  }  try {  FirebaseApp.initializeApp(options);  } catch (Exception aa) {  System.out.println("Already Exists");  }  db = FirestoreClient.getFirestore();  CollectionReference imagesCollection = db.collection("images");  Query query = null;  if (!selectedLocation.equals("All")) {  query = imagesCollection.whereEqualTo("location", selectedLocation);  }      // If a year other than "All" is selected, add a whereEqualTo filter to the query  if (!selectedYear.equals("All")) {  if (query != null) {  query = query.whereEqualTo("year", selectedYear);  }  else {  query = imagesCollection.whereEqualTo("year", selectedYear);  }  }    if (query == null) {  query = imagesCollection.whereNotEqualTo("year", -9999);  }    ApiFuture<QuerySnapshot> querySnapshot = query.get();  List<QueryDocumentSnapshot> documents = null;  try {  documents = querySnapshot.get().getDocuments();  //System.out.println("==========>>");  //System.out.println(documents);  } catch (InterruptedException e1) {  // TODO Auto-generated catch block  e1.printStackTrace();  } catch (ExecutionException e1) {  // TODO Auto-generated catch block  e1.printStackTrace();  }    // Create a new window to display filtered images  JFrame frame = new JFrame("Location: " + selectedLocation + ' ' + " Year: "+ selectedYear);  frame.setMinimumSize(new Dimension(400, 200)); // set minimum size to 400x200 pixels  JPanel panel = new JPanel(new GridLayout(0, 3, 5, 5));  JScrollPane scrollPane = new JScrollPane(panel); // Add a scroll pane in case there are many images      for (DocumentSnapshot document : documents) {  String imageUrl = document.getString("originalImageName");  String caption = document.getString("image\_name");      //name filtering  String[] nameArray = document.getString("people").split(",");  if (!selectedPerson.equals("All")) {  if (!Arrays.asList(nameArray).contains(selectedPerson)) {  continue;  }  }      imageUrl = "https://firebasestorage.googleapis.com/v0/b/javaimagegallery-68cd9.appspot.com/o/images%2F"+imageUrl+"?alt=media&token=4e5aafa0-f77a-448e-b19c-193dc6e60de1";    // Create a new panel for each image  JPanel imagePanel = new JPanel(new BorderLayout());  JLabel imageLabel = null;  imageLabel = new JLabel(createScaledImageIcon(imageUrl, 250, 250));  JLabel captionLabel = new JLabel(caption);  imagePanel.add(imageLabel, BorderLayout.CENTER);  imagePanel.add(captionLabel, BorderLayout.SOUTH);    // Add image panel to the grid layout  panel.add(imagePanel);  }    // Add scroll pane to the frame and show the window  frame.add(scrollPane);  frame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);  frame.pack();  frame.setVisible(true);  }  }); |
| --- |

O listener αυτός λειτουργεί ως εξής:

* Ανακτά την επιλεγμένη τοποθεσία, έτος και άτομο από τα αντίστοιχα αναπτυσσόμενα μενού.
* Αρχικοποιεί έναν Firebase client χρησιμοποιώντας τη διαδρομή αρχείου κλειδιού λογαριασμού υπηρεσίας και λαμβάνει μια αναφορά Firestore.
* Δημιουργεί ένα ερώτημα με βάση τα επιλεγμένα φίλτρα, ξεκινώντας με φιλτράρισμα ανά τοποθεσία και στη συνέχεια προσθέτοντας επιπλέον φίλτρα για το έτος και τα άτομα, εάν υπάρχουν.
* Εάν δεν έχουν επιλεγεί φίλτρα, ορίζει ένα προεπιλεγμένο φίλτρο για να εξαιρούνται έγγραφα με τιμή "έτος" -9999.
* Εκτελεί το ερώτημα και ανακτά μια λίστα εγγράφων.
* Δημιουργεί ένα νέο παράθυρο για την εμφάνιση των φιλτραρισμένων εικόνων και ορίζει τον τίτλο του παραθύρου με βάση τα επιλεγμένα φίλτρα.
* Δημιουργεί ένα πλαίσιο με διάταξη πλέγματος και προσθέτει ένα παράθυρο κύλισης σε αυτό.
* Για κάθε έγγραφο στη λίστα φιλτραρισμένων εγγράφων, ανακτά τη διεύθυνση URL και τη λεζάντα της εικόνας και δημιουργεί ένα νέο πλαίσιο για κάθε εικόνα.
* Προσθέτει κάθε πίνακα εικόνας στον πίνακα διάταξης πλέγματος.
* Προσθέτει το παράθυρο κύλισης στο παράθυρο, ρυθμίζει το παράθυρο να απορρίπτεται στο κλείσιμο και εμφανίζει το παράθυρο.



Εικόνα 3.13: Παράδειγμα αποτελέσματος εφαρμογής φίλτρου

Στο project

# **Συμπεράσματα και μελλοντική εργασία**

Σε αυτό την εργασία αναπτύχθηκε μια εφαρμογή συλλογής εικόνων Java που επιτρέπει στους χρήστες να προβάλλουν και να φιλτράρουν εικόνες με βάση την τοποθεσία, το έτος και τους ανθρώπους. Έχουμε χρησιμοποιήσει τη βάση δεδομένων Firebase Cloud Firestore και τον χώρο αποθήκευσης για την αποθήκευση και ανάκτηση δεδομένων εικόνας και έχουμε ενσωματώσει διάφορες βιβλιοθήκες Java και στοιχεία όπως το Swing, το AWT και το Google Cloud SDK για τη δημιουργία της διεπαφής χρήστη και την εκτέλεση εργασιών επεξεργασίας εικόνας.

H εφαρμογή πληροί τις λειτουργικές απαιτήσεις που περιγράφονται στην αρχική πρόταση έργου και έχει αποδείξει τη σκοπιμότητα δημιουργίας μιας εφαρμογής συλλογής εικόνων που βασίζεται σε Java. Ωστόσο, υπάρχουν ακόμη αρκετοί τομείς για βελτίωση και μελλοντικές εργασίες που θα μπορούσαν να βελτιώσουν τη λειτουργικότητα και τη χρηστικότητα της εφαρμογής.

Ένας πιθανός τομέας για βελτίωση είναι ο σχεδιασμός της διεπαφής χρήστη. Αν και η τρέχουσα σχεδίαση είναι λειτουργική, μπορεί να μην είναι η πιο διαισθητική ή οπτικά ελκυστική για όλους τους χρήστες. Η ενσωμάτωση των σχολίων των χρηστών και η διεξαγωγή δοκιμών χρηστικότητας θα μπορούσαν να βοηθήσουν στον εντοπισμό περιοχών προς βελτίωση και στην ενημέρωση του σχεδιασμού των μελλοντικών επαναλήψεων της εφαρμογής.

Ένας άλλος τομέας για μελλοντική εργασία είναι η επέκταση των δυνατοτήτων φιλτραρίσματος της εφαρμογής. Προς το παρόν, οι χρήστες μπορούν να φιλτράρουν εικόνες μόνο βάσει τοποθεσίας, έτους και ατόμων. Η προσθήκη πρόσθετων φίλτρων, όπως λέξεις-κλειδιά ή ετικέτες εικόνας, θα μπορούσε να διευκολύνει τους χρήστες να βρίσκουν συγκεκριμένες εικόνες.

Επιπλέον, η εφαρμογή θα μπορούσε να επωφεληθεί από βελτιωμένες δυνατότητες επεξεργασίας εικόνας. Για παράδειγμα, η ενσωμάτωση αλγορίθμων αναγνώρισης εικόνας ή όρασης υπολογιστή θα μπορούσε αυτόματα να προσθέσει ετικέτες ή να κατηγοριοποιήσει τις εικόνες με βάση το περιεχόμενό τους.

Τέλος, η ενσωμάτωση των δυνατοτήτων κοινής χρήσης μέσων κοινωνικής δικτύωσης θα μπορούσε να διευκολύνει τους χρήστες να μοιράζονται εικόνες με άλλους και να αυξήσουν την εμβέλεια και την ορατότητα της εφαρμογής.

Ενώ η τρέχουσα έκδοση της εφαρμογής γκαλερί εικόνων Java είναι λειτουργική και πληροί τις αρχικές απαιτήσεις του έργου, υπάρχει ακόμα χώρος για βελτίωση και μελλοντική εργασία. Με συνεχή ανάπτυξη και επανάληψη, η εφαρμογή θα μπορούσε να γίνει ένα ισχυρό και πολύτιμο εργαλείο για τη διαχείριση και την κοινή χρήση εικόνων.

# **Αναφορές**

[1] Qian, L., Luo, Z., Du, Y. and Guo, L., 2009. Cloud computing: An overview. In Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1 (pp. 626-631). Springer Berlin Heidelberg.

[2] Moroney, L. and Moroney, L., 2017. Cloud storage for firebase. The definitive guide to firebase: build android apps on google's mobile platform, pp.73-92.

[3] Gosling, J., 1995. Java™: An Overview. Recuperado d e http://www. cs. dartmouth. edu/~ mckeeman/cs118/references/OriginalJavaWhitep aper. pdf.

[4] Loy, M., Eckstein, R., Wood, D., Elliott, J. and Cole, B., 2002. Java swing. " O'Reilly Media, Inc.".

[5] https://firebase.google.com/docs/storage

[6] https://firebase.google.com/docs/firestore

[7] https://www.java.com/en/

[8] Smart, J.F., 2005. An introduction to Maven 2. JavaWorld Magazine. Available at: http://www. javaworld. com/javaworld/jw-12-2005/jw-1205-maven. html, p.27.