ΚΑΤΑΝΕΝΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

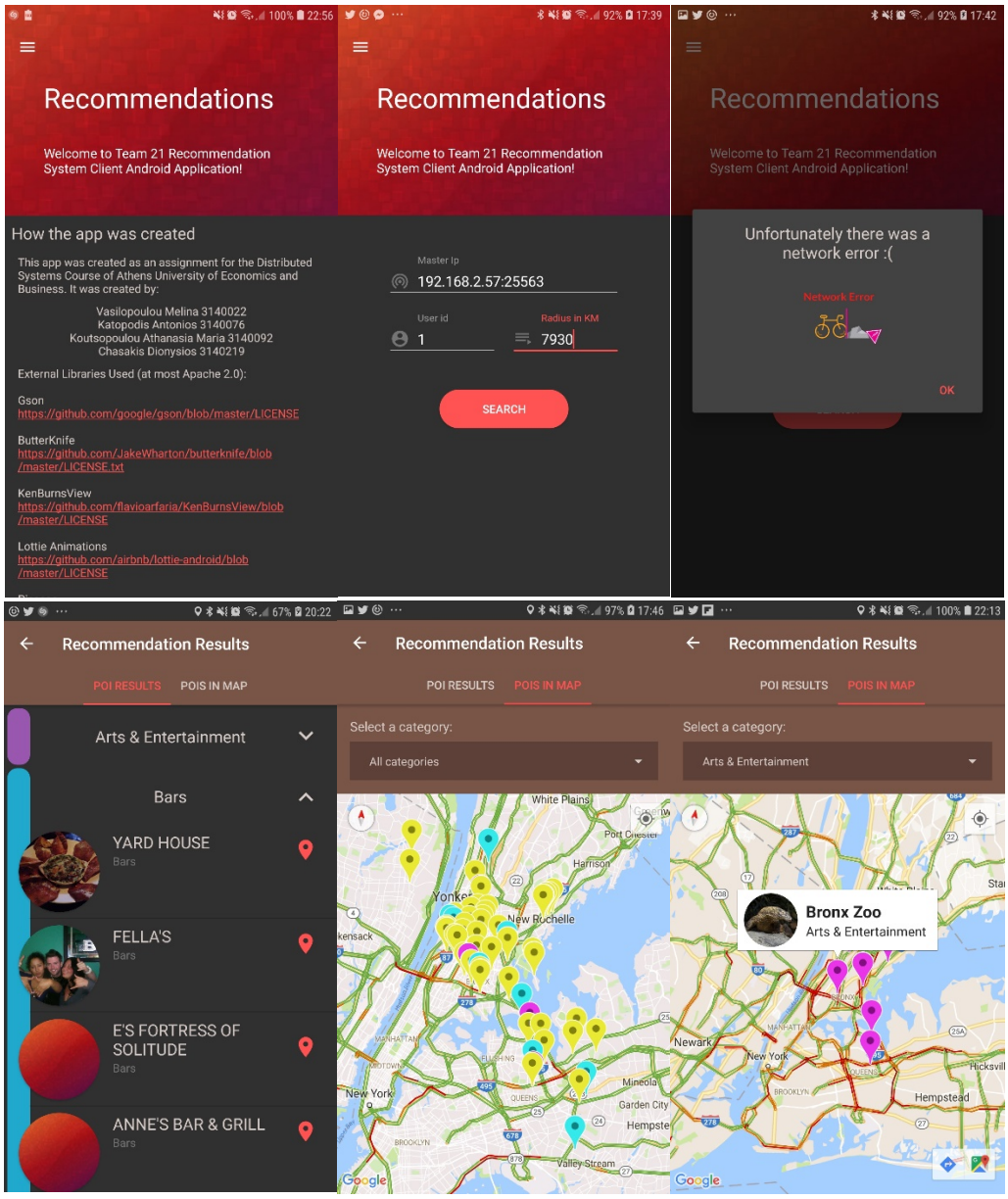
RECCOMENDATION SYSTEM

ΟΜΑΔΑ 21

Βασιλοπούλου Μελίνα 3140022
Κατωπόδης Αντώνιος 3140076
Κουτσοπούλου Αθανασία Μαρία 3140092
Χασακής Διονύσιος 3140219

ANDROID CLIENT

SCREENSHOTS



ACTIVITIES

Στο επίπεδο του Android έχουμε υλοποιήσει κάποιες βασικές κλάσεις όπως BaseActivity, BaseFragment, οι οποίες είναι abstract και απαιτούν τον ορισμό του Content XML. Τα Activities που έχουμε είναι:

- **MainActivity:** Το αρχικό Activity όπου κάνει spawn ένα NavigationDrawer όπου έχουμε 2 επιλογές. Ή τα Recommendations ή το About. Για τα fragments που χρησιμοποιούμε θα μιλήσουμε παρακάτω. Το NavigationView γίνεται wrap σε ένα DrawerLayout και εμφανίζεται σε ένα CoordinatorLayout, το οποίο περιέχει ένα FrameLayout το οποίο θα κάνουμε inflate με τα Fragments ώστε να αλλάζουμε το περιεχόμενό του, ανάλογα με το SelectedItem.
- **ResultsTabActivity:** Είναι το Activity στο οποίο κάνει navigate ο χρήστης όταν έχει εκτελεστεί επιτυχώς η αναζήτηση. Το Activity αυτό περιέχει ένα TabLayout ώστε να έχουμε 2 tabs. Ένα με ένα Grouping Recycler View, το οποίο κατηγοριοποιεί τα Poi με βάση την κατηγορία, και ένα με τον χάρτη όπου ο χρήστης επιλέγει ποια κατηγορία θέλει να δει. Το TabLayout χρησιμοποιεί ένα ViewPager το οποίο δέχεται ως είσοδο έναν Adapter. Ο Adapter είναι ο ResultsTabAdapter, που κάνει inherit από το FragmentPagerAdapter. Ο Adapter ανάλογα με το position γυρνά διαφορετικό fragment και αυτό θα είναι αυτό που θα εμφανιστεί στο ViewPager.

FRAGMENTS

Σαν fragments έχουμε υλοποιήσει τα Views που υπάρχουν στο DrawerLayout αλλά και όσα είναι σαν tabs στο ResultsTabActivity. Τα Fragments που έχουμε υλοποιήσει είναι:

- **SearchUserFragment:** Αυτό το fragment περιέχει 3 textViews όπου ο χρήστης θα εισάγει την IP και το Port του Server, το radius και το userId. Όταν ο χρήστης πατήσει το Search θα δημιουργηθεί ένας LoaderManager ο οποίος θα εκτελέσει έναν AsyncTaskLoader όπου θα γίνει η επικοινωνία με τον Server και θα γυρίσει τα αποτελέσματα. Ο λόγος που χρησιμοποιήσαμε έναν Loader είναι για τα Android Lifecycle Issues, όπου όταν ο χρήστης ενώ εκτελείται το Async Task κάνει rotate τη συσκευή, πάει σε sleep ή βγει από την εφαρμογή δεν δημιουργούνται Zombie Tasks όπου δεν επιστρέφουν δεδομένα σε κάποιο Activity και δεν εκτελούνται από την αρχή. Ένας Loader δεν θα δημιουργηθεί εκ νέου αλλά θα παρθεί από το Id του και θα γίνει restart. Το fragment αυτό έχει και ένα interface το οποίο θα υλοποιήσει ο πατέρας Activity ώστε να μεταβεί σε νέο Activity. Το interface αυτό είναι το OnResultsFetchedListener το οποίο θα γίνει Invoke όταν επιστρέψει ο Loader, ώστε από το MainActivity να ανοιχτεί το ResultsTabActivity.
  Επίσης έχουμε καλύψει και τη περίπτωση που όταν εκτελείται το request ο user κάνει rotate το κινητό, να αποθηκεύονται στο SaveInstance η IP του μάστερ, το userId και το Radius, ώστε στην onActivityCreated του fragment να γίνουν restore και να γίνει restart ο Loader. Αν δεν το είχαμε υλοποιήσει αυτό, όταν κάναμε rotate το request θα ακυρωνόταν (χωρίς όμως να είχαμε leaks (χρήση loader)). Η onSaveInstance καλείται σύμφωνα με το Lifecycle του Fragment όταν γίνει κάποια αλλαγή στο state όπως να πάει το κινητό σε sleep, γίνει rotate κλπ. Μετά την save, θα καλεστεί η

onActivityCreated η οποία είναι η onRestoreInstanceState, αλλά αντίστοιχη για τα Fragments.

- **InfoFragment:** Το fragment αυτό απλά δείχνει μία σελίδα με πληροφορίες σχετικές με το Project και την ομάδα μας.
- **ResultsListFragment:** Αυτό το fragment διαβάζει την τοπική βάση δεδομένων τα poi (σημείωση για την τοπική βάση πιο κάτω) και τα κατηγοριοποιεί με βάση την κατηγορία του poi. Με βάση τις κατηγορίες και τα poi της καθεμίας δημιουργείται ένα Grouping RecyclerView (αναλυτική σημείωση πιο κάτω). Για το κάθε item, γίνεται fetch η εικόνα του μέσω του Picasso και γίνονται inflate στο αντίστοιχο RoundedImageView.
- **PoisOnMapFragment:** Το fragment είναι υπεύθυνο για την εμφάνιση του χάρτη και την είσοδο των πινεζών. Στο fragment υπάρχει ένα Spinner το οποίο περιέχει όλες τις κατηγορίες των Poi. Επιλέγοντας μία, οι πινέζες αλλάζουν και δείχνουν μόνο τα poi της συγκεκριμένης αυτής κατηγορίας.

ΒΟΗΘΗΤΙΚΕΣ ΚΛΑΣΕΙΣ

Για χάριν επαναχρησιμοποίησης έχουν υλοποιηθεί μερικές Helper Classes (Package Utils) οι οποίες είναι:

- **DialogUtils:** Εμφανίζει μερικά custom AlertDialogs όπως την ένδειξη ότι το connection απέτυχε ή το dialog με το please wait…
- **LocationUtlis:** Η μέθοδος GetLastKnownLocation γυρνά την τελευταία τοποθεσία του χρήστη μέσω του LocationManager. Χρησιμοποιήθηκε η τελευταία τοποθεσία ώστε να αποφύγουμε τα listeners στο ILocationChanged interface και να αυξήσουμε τη πολυπλοκότητα, χωρίς τόσο σημασία.
- **NetworkUtils:** Η μέθοδος GetRecommendationPois παίρνει ως είσοδο το userId, το radius και το location του χρήστη. Φτιάχνει ένα CommunicationMessage με τα στοιχεία αυτά, ανοίγει ένα socket και κάνει connect με τη χρήση timeout. Κάνει Serialize σε JSON το object αυτό και το γράφει στο ObjectOutputStream. Έπειτα περιμένει για την απάντηση από τον Server, διαβάζει το αποτέλεσμα, το κάνει DeSerialize από JSON σε ένα CommunicationMessage και γυρνά τα Poi που έλαβε.
- **OpenPoiLocation:** Χρησιμοποιείται για όταν ο χρήστης κάνει κλικ στη πινέζα του RecyclerView να του ανοίξει το Maps με τοποθετημένη μία πινέζα στο Poi που επέλεξε.
- **PermissionUtils:** Ελέγχει αν τα permissions για το location(*ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION)* είναι ενεργοποιημένα και αν δεν είναι τα ζητά.

GROUPING RECYCLERVIEW

Για το Grouping RecyclerView χρησιμοποιήσαμε ένα third-party library και υλοποιήσαμε 2 ViewHolders. Τον PoiCategoryHolder που κάνει inherit από το GroupViewHolder

του library και τον PoiHolder που κάνει inherit από τον ChildViewHolder. Το RecyclerView θα περιέχει τα poi γκρουπαρισμένα ανά κατηγορία και όταν πατάει μία κατηγορία θα γίνεται expand και θα φαίνονται όλα τα poi της. Ο Adapter για αυτή τη διαδικασία είναι ο PoiCategoryGroupAdapter που περιέχει τους ViewHolders με το κάθε item. Όταν γίνεται κλικ στο item, γίνεται expand του item αυτού(της κατηγορίας) που θα δείξει τελικά τα Poi. Η διαφορά με την προηγούμενη έκδοση είναι ότι όλα πλέον βρίσκονται σε ένα RecyclerView και δεν υπάρχουν εμφολευμένα. Αυτό δημιουργεί μία εμπειρία χωρίς lags καθώς καμία βαριά διαδικασία δεν γίνεται στο UI thread (Το Picasso αξιοποιεί το ThreadPool).

ΧΡΗΣΗ SQLITE

Υλοποιήσαμε μία τοπική βάση δεδομένων με έναν πίνακα με σκοπό την αποθήκευση των Poi και την ανάκτησή τους από τα Fragments. Η ανάκτηση τους γίνεται πάλι μέσω Loader, για τους ίδιους λόγους με πριν. Ο λόγος που χρησιμοποιήθηκε η βάση είναι γιατί στην αρχή περνούσαμε στα Fragments το Arraylist από τα Poi μέσω των Extras του Intent. Όταν γυρνούσαν πολλά Pois (π.χ. όλα), το μέγιστο δυνατό όριο του Android για τα extras ήταν 1MB, οπότε γυρνούσε Exception γιατί η λίστα αυτή σε MB ξεπερνούσε το όριο αυτό. Ακολουθώντας τα design patterns του Android, για τη χρήση ContentProvider και τη δυνατότητα άλλοι developers να χρησιμοποιήσουν τη βάση μας, έχουμε:

Για την SQLite χρησιμοποιήσαμε 3 κλάσεις (Package Data):

- Ο PoiContentProvider είναι υπεύθυνος για τα transactions με τη βάση, με αποτέλεσμα το application να το αντιμετωπίζει σαν Black Box.
- Η SuggestedPoisContract περιέχει όλες τις στατικές παραμέτρους για ονόματα πινάκων, στηλών, directory της βάσης κλπ.
- Ο SuggestedPoisDbHelper φτιάχνει τον πίνακα στη βάση. Σε περίπτωση που θέλουμε να κάνουμε update το schema, απλά αλλάζουμε τη τιμή του static variable DATABASE_VERSION και θα καλεστεί η onCreate μόλις ξεκινήσει η εφαρμογή, με αποτέλεσμα να ανανεωθεί η βάση.

INFOWINDOWADAPTER

Για την εμφάνιση φωτογραφίας στο infoWindow του SupportMapFragment, χρησιμοποιήσαμε έναν Adapter ο οποίος θα δέχεται ένα HashMap με key τα marker ids και value το image url. Έτσι όταν ο χρήσης κάνει click σε ένα marker, κάνουμε inflate το xml που έχουμε φτιάξει με μία εικόνα και 2 textViews για την εμφάνιση του title και του snipper. Έτσι παίρνουμε από το marker το title και το snippet, τα ορίζουμε στα TextViews και έπειτα φορτώνουμε μέσω του Picasso την εικόνα που παίρνουμε από το HashMap στο ImageView. Για να φανεί η εικόνα, εφόσον το Picasso αξιοποιεί το ThreadPool, θα πρέπει να ορίσουμε ένα callback ώστε να επανεμφανίσουμε το InfoWindow ώστε να ανανεωθεί η εικόνα.

THIRD-PARTY LIBRARIES ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

Για την υλοποίηση της Android εφαρμογής χρησιμοποιήθηκαν τα libraries:

- ButterKnife (View Bindings): https://github.com/JakeWharton/butterknife
- GSON (Google JSON Serializing and Deserializing): https://github.com/google/gson
- Picasso Image Downloading and Caching: https://github.com/square/picasso
- KenBurnsView Image Animation: https://github.com/flavioarfaria/KenBurnsView
- Rounded Image View: https://github.com/vinc3m1/RoundedImageView
- Material Spinner(Dropdown): https://github.com/jaredrummler/MaterialSpinner
- Expandable RecyclerView: https://github.com/thoughtbot/expandable-recycler-view
- Lottie JSON Asset Animations: https://github.com/airbnb/lottie-android

ΑΛΛΑΓΕΣ ΣΤΟ BACKEND

Ο Master και το Android Client επικοινωνούν μέσω JSON. Τα CommunicationMessage γίνονται Serialize σε JSON με χρήση της βιβλιοθήκης GSON της Google και αποστέλλονται σαν objects. Έτσι σε περίπτωση που το object που διάβασε ο master είναι String, τότε πρόκειται για μία επικοινωνία μεταξύ Java Server και Android Client. Ο Master πλέον γνωρίζει ότι το result θα πρέπει να κάνει και αυτός Serialize το result και να το στείλει σαν String.

Ο υπολογισμός των Poi για κάθε χρήστη γίνεται ως εξής:

Ο Android Client στέλνει ένα CommunicationMessage όπου έχει ορίσει το id του χρήστη, το radius σε χιλιόμετρα και το LatLng του και κρατά το connection open έως ότου πάρει το αποτέλεσμα. Ο Master υπολογίζει για κάθε poi την απόσταση από τον χρήστη, φιλτράρει αυτά που είναι μέσα στο radius και τα τοποθετεί σε αύξουσα σειρά με βάση την απόσταση. Τέλος απαντά με ένα CommunicationMessage όπου έχει ορίσει το πεδίο poisToReturn που είναι μία λίστα με Poi. Η μέθοδος CalculateDistance, του Master δέχεται ως είσοδο ένα ζευγάρι από LatLng και υπολογίζει την απόστασής τους σε μέτρα.