



| **Νο** | **Ονοματεπώνυμο** | | **Α.Μ.** |
| --- | --- | --- | --- |
| 1 | Γιαρμαδούρου Αλεξάνδρα | | 154871 |
| 2 | Θεολόγου Σπυρίδων | | 156451 |
| 3 | Μαρκόπουλος Χριστόφορος | | 156513 |
| **ΣΧΟΛΙΑ ΠΡΟΣ ΤΟΝ ΚΑΘΗΓΗΤΗ** | |  | |

***Υπεύθυνη Δήλωση Φοιτητή****: Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία αυτής της εργασίας, είναι πλήρως αναγνωρισμένη και αναφέρεται, είτε στο σημείο «Σχόλια προς καθηγητή», είτε μέσα στην εργασία. Επίσης, έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς, είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τη συγκεκριμένη Θεματική Ενότητα.*

 Συμφωνώ και αποδέχομαι την ανωτέρω δήλωση

 Δε συμφωνώ και δεν αποδέχομαι την ανωτέρω δήλωση (στην περίπτωση αυτή, ο Κ-Σ έχει δικαίωμα να μην αξιολογήσει την εργασία του φοιτητή)

| Ημερομηνία ανακοίνωσης εργασίας:, 22/01/2024  Ημερομηνία παράδοσης εργασίας: 06/03/2024  ΔΕΝ ΘΑ ΔOΘΟΥΝ ΠΑΡΑΤΑΣΕΙΣ |
| --- |

ΣΤΟΙΧΕΙΑ ΠΟΥ ΣΥΜΠΛΗΡΩΝΕΙ Ο ΚΑΘΗΓΗΤΗΣ

| **ΗΜΕΡΟΜΗΝΙΑ ΑΞΙΟΛΟΓΗΣΗΣ** |  |
| --- | --- |
| **ΒΑΘΜΟΣ** |  |

**ΣΧΟΛΙΑ ΠΡΟΣ ΦΟΙΤΗΤΗ / ΦΟΙΤΗΤΡΙΑ**

##### Αναλυτική Αξιολόγηση

| **Άσκηση** | **Περιγραφή** | **Ποσοστό** |
| --- | --- | --- |
| ΔΡΑΣΤΗΡΙΟΤΗΤΑ 1 | Διαχείριση του Έργου με έμφαση στη χρήση του εργαλείου συνεργασίας | 20 |
| ΔΡΑΣΤΗΡΙΟΤΗΤΑ 2.Α | Διάγραμμα ER, Διάγραμμα Κλάσεων και Υλοποίηση Κλάσεων σε Java (περιλαμβάνει χρήση API και επεξεργασία json) | 20 |
| ΔΡΑΣΤΗΡΙΟΤΗΤΑ 2.Β | Δημιουργία GUI Εφαρμογής | 15 |
| ΔΡΑΣΤΗΡΙΟΤΗΤΑ 2.Γ | Παρουσίαση Στατιστικών Στοιχείων | 15 |
| ΔΡΑΣΤΗΡΙΟΤΗΤΑ 2.Δ | Συνολικός Έλεγχος και Εκτέλεση της Εφαρμογής | 15 |
| ΔΡΑΣΤΗΡΙΟΤΗΤΑ 3 | Κριτικός Απολογισμός του Έργου | 5 |
| ΔΡΑΣΤΗΡΙΟΤΗΤΑ 4 | Δημιουργία video | 10 |
|  | **Σύνολο** | 100 |

ΠΕΡΙΕΧΟΜΕΝΑ

[**1**](#_heading=h.gjdgxs) **[ΕΙΣΑΓΩΓΗ 4](#_heading=h.gjdgxs)**

[1.1 Σύντομη περιγραφή του συστήματος 4](#_heading=h.44sinio)

[1.2 Βασικοί Στόχοι Του Έργου 5](#_heading=h.2jxsxqh)

[1.3 Χαρακτηριστικά και Λειτουργίες Της WeatherApp- Ικανοποίηση Απαιτήσεων. 5](#_heading=h.z337ya)

[1.4 Παραδοχές 7](#_heading=h.3j2qqm3)

[**2**](#_heading=h.30j0zll) **ΔΙΑΧΕΙΡΙΣΗ ΕΡΓΟΥ 8**

[2.1](#_heading=h.1fob9te) Υπολογισμός της απαιτούμενης προσπάθειας ανά απαίτηση 8

[2.2](#_heading=h.3znysh7) Υπολογισμός των προτεραιοτήτων 10

[2.3](#_heading=h.2et92p0) To product backlog 11

[2.4](#_heading=h.tyjcwt) Οργάνωση ομάδας και αναθέσεις αρμοδιοτήτων 12

[2.5](#_heading=h.3dy6vkm) Παρακολούθηση της προσπάθειας κατά τη διάρκεια του έργου 13

[2.6](#_heading=h.1t3h5sf) Χρήση εργαλείου trello 16

[**3**](#_heading=h.2s8eyo1) **ΥΛΟΠΟΙΗΣΗ ΕΡΓΟΥ 17**

[3.1](#_heading=h.2s8eyo1) Ερώτημα Α– Διάγραμμα Κλάσεων και Υλοποίηση Κλάσεων σε Java 17

[3.2](#_heading=h.17dp8vu) Ερώτημα Β – Δημιουργία GUI Εφαρμογής 30

[3.3](#_heading=h.3rdcrjn) Ερώτημα Γ – Παρουσίαση στατιστικών στοιχείων καιρού 37

[3.4](#_heading=h.26in1rg) Ερώτημα Δ – Συνολικός Έλεγχος και Εκτέλεση της Εφαρμογής 40

[**4**](#_heading=h.lnxbz9) **ΚΡΙΤΙΚΟΣ ΑΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ΕΡΓΟΥ 45**

[**5**](#_heading=h.35nkun2) **VIDEO 46**

[**6**](#_heading=h.1ksv4uv) **ΑΝΑΦΟΡΕΣ 47**

# ΕΙΣΑΓΩΓΗ

Η παρούσα ομαδική εργασία της ΘΕ ΠΛΗ-24 περιλαμβάνει το σχεδιασμό, την ανάπτυξη και την υλοποίησης της εφαρμογής WeatherApp, η οποία παρουσιάζει, επεξεργάζεται και αποθηκεύει δεδομένα καιρικών συνθηκών που λαμβάνει από την υπηρεσία ιστού (web service) της ιστοσελίδας wttr.in: <https://wttr.in/>.

## Σύντομη περιγραφή του συστήματος

Η WeatherApp δίνει τη δυνατότητα, μέσω ενός γραφικού συστήματος διεπαφής (GUI), να προβληθούν τα δεδομένα καιρού πόλης που λαμβάνονται από την wttr.in: <https://wttr.in/> . Η wttr.in παρέχει τις υπηρεσίες αυτές μέσω web services τύπου REST (Representational State Transfer). Η εφαρμογή WeatherApp λαμβάνει τα δεδομένα μέσω της κλήσης του API ως πελάτης(client), σε μορφή JSON (JavaScript Object Notation). Για τη χρήση των web services https://wttr.in δεν απαιτείται η δημιουργία λογαριασμού ή/και API KEY.

Στη συνέχεια η εφαρμογή WebApp παρέχει μια σειρά από δυνατότητες:

1. Αποθήκευση των παρακάτω καιρικών δεδομένων για κάθε πόλη σε ΒΔ:

* Θερμοκρασία (temp\_C)
* Υγρασία (humidity)
* Ταχύτητα ανέμου (windspeedKmph)
* Δείκτης UV (uvIndex)
* Λεκτική πρόγνωση καιρού (weatherDesc)

1. Διαγραφή των σχετικών δεδομένων από τη ΒΔ, βάσει ονόματος πόλης.
2. Τροποποίηση των δεδομένων για την τελευταία εγγραφή μιας πόλης.
3. Αναζήτηση ιστορικού για λίστα πόλεων για τις οποίες έχει γίνει αποθήκευση δεδομένων καιρού.
4. Αναζήτηση ιστορικού για ημερομηνίες αποθήκευσης δεδομένων ανά πόλη.
5. Προβολή στατιστικών στοιχείων με τις πόλεις με τις περισσότερες αναζητήσεις και εκτύπωση των στοιχείων αυτών σε PDF.

## Βασικοί στόχοι του έργου

* Η ανάπτυξη μιας πλήρως λειτουργικής εφαρμογής αποτελεί τον κύριο στόχο του έργου. Αυτό επιτυγχάνεται με τη δημιουργία ενός REST API για τη λήψη των δεδομένων καιρού, το σχεδιασμό ενός φιλικού προς το χρήστη GUI, και την υλοποίηση ενός συστήματος βάσης δεδομένων.
* Ο δεύτερος στόχος επικεντρώνεται στην απόκτηση και εφαρμογή γνώσεων για διάφορα εργαλεία και τεχνολογίες, με χρήση της γλώσσας Java. Αυτό περιλαμβάνει την απόκτηση εμπειρίας στη χρήση εργαλείων για την ανάπτυξη του GUI, του REST API και της DB. Μέσω αυτής της διαδικασίας, η ομάδα στοχεύει στην ομαλή εκτέλεση των εργασιών του έργου και στην παράδοση υψηλής ποιότητας λογισμικού.
* Ένας άλλος σημαντικός στόχος είναι η κατανόηση και εφαρμογή των μεθόδων Agile και Scrum κατά τη διάρκεια του κύκλου ζωής του έργου. Αυτό περιλαμβάνει τον διαχωρισμό της διαδικασίας ανάπτυξης σε διαχειρίσιμες περιόδους (sprints), την ιεράρχηση των εργασιών και την επανεξέταση των απαιτήσεων.
* Ο τέταρτος στόχος αφορά στη συνεργασία μεταξύ των μελών της ομάδας. Μέσω αποτελεσματικής επικοινωνίας, συντονισμού και αμοιβαίας υποστήριξης, η ομάδα στοχεύει στην εκμετάλλευση των ατομικών δυνατοτήτων και εμπειριών για την επίτευξη κοινού στόχου. Σε αυτό το συνεργατικό περιβάλλον, η ομάδα πετυχαίνει μέγιστη απόδοση, δημιουργικότητα, αλλά και ευκαιρίες μάθησης, βελτιώνοντας τελικά τα συνολικά αποτελέσματα του έργου.

## Χαρακτηριστικά και λειτουργίες της WeatherApp - Ικανοποίηση απαιτήσεων

Τα χαρακτηριστικά και οι λειτουργίες της εφαρμογής προσδιορίζονται στην εκφώνηση της εργασίας και εμπεριέχονται και στο αρχείο Excel Product Backlog. Eίναι οι εξής:

| Νο | Περιγραφή |
| --- | --- |
| R1 | Το σύστημα περιλαμβάνει τις παρακάτω επιλογές:   1. Προβολή καιρικών δεδομένων πόλης 2. Προβολή λίστας πόλεων για τις οποίες έχει γίνει αναζήτηση καιρού 3. Προβολή λίστας ημερομηνιών για πόλη για την οποία έχει γίνει αναζήτηση καιρού 4. Προβολή στατιστικών δεδομένων πόλεων και εκτύπωση σε αρχείο pdf 5. Έξοδος |
| R2 | Με την προβολή των δεδομένων πόλης εμφανίζεται οθόνη στην οποία ο χρήστης επιλέγει τα δεδομένα που επιθυμεί να αντληθούν από το API. Συγκεκριμένα, πληκτρολογεί το όνομα μιας πόλης, και εφόσον υπάρχει στο API, επιστρέφονται τα κατάλληλα δεδομένα.  Οι λεπτομέρειες που μας ενδιαφέρουν να αντληθούν για κάθε πόλη είναι οι ακόλουθες 5:   * Θερμοκρασία (temp\_C) * Υγρασία (humidity) * Ταχύτητα ανέμου (windspeedKmph) * Δείκτης UV (uvIndex) * Λεκτική πρόγνωση καιρού (weatherDesc)   Εν συνεχεία, ο χρήστης έχει τη δυνατότητα να αποθηκεύσει τα δεδομένα που ανακτήθηκαν από το API σε ΒΔ. Σε αυτή την περίπτωση προηγείται έλεγχος ύπαρξης στη βάση δεδομένων των δεδομένων αυτών (από προηγούμενη αναζήτηση) και αν δεν υπάρχουν, αποθηκεύονται στη βάση δεδομένων.  Στην οθόνη προβολής των δεδομένων εμφανίζονται επιπλέον τα πλήκτρα «**Διαγραφή δεδομένων πόλης**» και «**Επεξεργασία δεδομένων πόλης**»  Με την επιλογή της διαγραφής δεδομένων πόλης διαγράφονται από τη βάση δεδομένων όλα τα σχετικά δεδομένα βάσει του ονόματος της πόλης. Αντίστοιχα, με την επιλογή της τροποποίησης των δεδομένων καιρού μιας πόλης, ο χρήστης έχει τη δυνατότητα να τροποποιήσει μία ή/και περισσότερες από τις αποθηκευμένες 5 καιρικές ενδείξεις, για την τελευταία εγγραφή καιρικών δεδομένων για την εν λόγω πόλη .  Η ολοκλήρωση οποιασδήποτε διαγραφής ή επεξεργασίας δεδομένων γίνεται αφού προηγηθεί ερώτημα επαλήθευσης με επιβεβαίωση από το χρήστη. |
| R3 | Κατά την προβολή λίστας πόλεων για τις οποίες έχει γίνει αναζήτηση καιρού ο χρήστης έχει τη δυνατότητα να ανατρέξει μέσα στη ΒΔ για έγγραφές οι οποίες αφορούν αναζητήσεις δεδομένων καιρού για πόλεις, οι οποίες έχουν γίνει στο παρελθόν .  Αντίστοιχα, κατά την προβολή λίστας ημερομηνιών για πόλη για την οποία έχει γίνει αναζήτηση καιρού, ο χρήστης μπορεί να δει τις εγγραφές των ημερομηνιών στις οποίες έχει γίνει αναζήτηση και αποθήκευση δεδομένων καιρού της συγκεκριμένης πόλης στο παρελθόν. |
| R4 | Με την επιλογή προβολής στατιστικών καιρικών δεδομένων και εκτύπωση σε αρχείο pdf, αναζητάμε μέσα στην ΒΔ που έχει δημιουργηθεί, ποιες είναι οι πιο δημοφιλείς αναζητήσεις δεδομένων καιρού πόλεων, δηλαδή τις πόλεις που έχει αναζητήσει ο χρήστης της εφαρμογής περισσότερες φορές. Τα στατιστικά αυτά πρέπει να μπορούν να προβληθούν υπό τη μορφή πίνακα (δημοφιλέστερες πόλεις σε φθίνουσα μορφή), ενώ στη συνέχεια με την ύπαρξη αντίστοιχης επιλογής, δίνεται η δυνατότητα εκτύπωσης του παραγόμενου πίνακα σε αρχείο PDF. |

Οι απαιτήσεις αυτές ικανοποιούνται από την εφαρμογή και αναλύονται με λεπτομέρεια στο εγχειρίδιο χρήσης, στην παράγραφο 3.4.

## Παραδοχές

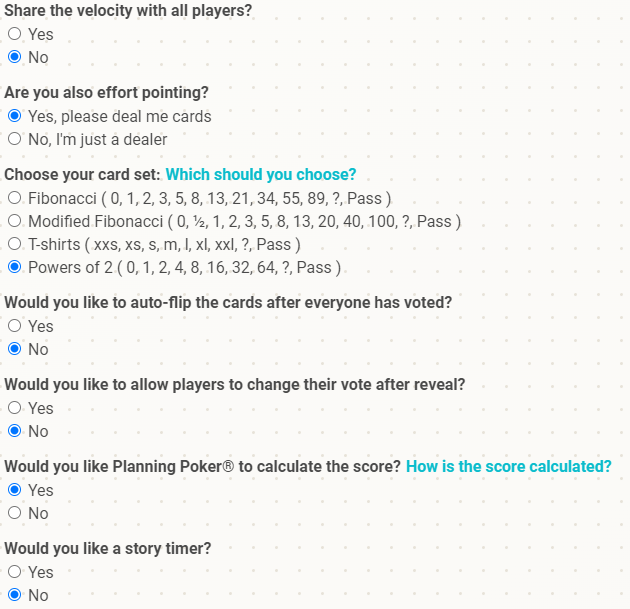
1. Στα στατιστικά δεδομένα, ο αριθμός των αναζητήσεων αφορά στο πόσες φορές έχει αναζητηθεί ο καιρός μιας πόλης και όχι πόσες φορές έχουν αποθηκευτεί τα δεδομένα καιρού της.
2. Στον έλεγχο που πραγματοποιείται πριν αποθηκευτούν τα καιρικά δεδομένα, ελέγχεται αν υπάρχει άλλη καταχώριση για τη συγκεκριμένη πόλη, στη συγκεκριμένη ημερομηνία. Στην περίπτωση αυτή, τα δεδομένα δεν αποθηκεύονται και προβάλλεται σχετικό μήνυμα.
3. Η διαγραφή γίνεται από τη λίστα πόλεων για τις οποίες έχει γίνει αποθήκευση. Στη διαγραφή όλων των στοιχείων μιας πόλης από τη ΒΔ, διαγράφεται και ο αριθμός των αναζητήσεων της. Λόγω της παραδοχής 1, πόλεις που μόνο αναζητήθηκαν αλλά δεν αποθηκεύτηκαν δεδομένα καιρού για αυτές, δεν διαγράφονται. Αυτό όμως δεν αποτελεί πρόβλημα, καθώς η απαίτηση για τη διαγραφή όλων των καιρικών δεδομένων από τη βάση για τη συγκεκριμένη πόλη ικανοποιείται.

# ΔΙΑΧΕΙΡΙΣΗ ΕΡΓΟΥ

Το έργο ολοκληρώθηκε σε 3 επαναλήψεις (Sprints) διάρκειας δύο εβδομάδων η κάθε μια. Η ομάδα αποφάσισε ποιες από τις απαιτήσεις του Product Backlog θα συμπεριληφθούν σε κάθε επανάληψη. Το αρχείο Product Backlog.xlsx βρίσκεται στα παραδοτέα της εργασίας.

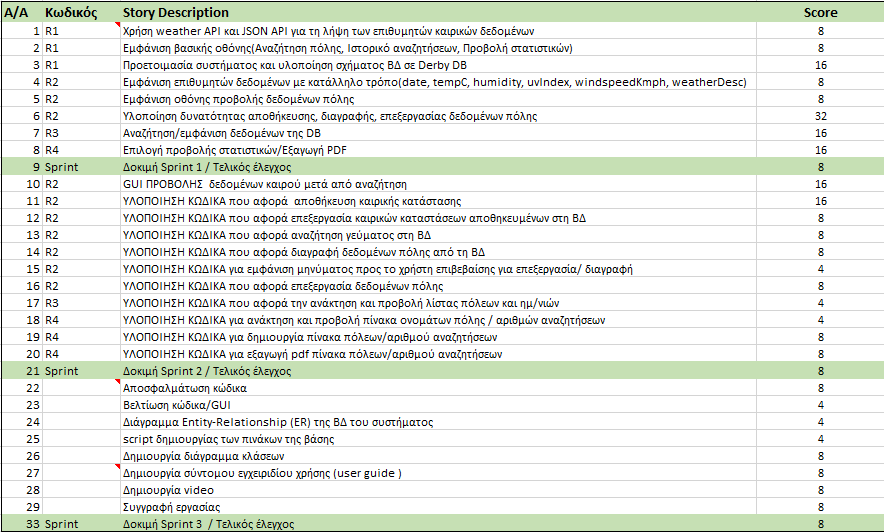
## Υπολογισμός της απαιτούμενης προσπάθειας ανά απαίτηση

Σε κάθε επανάληψη υπολογίστηκε η απαιτούμενη προσπάθεια (effort) ανά απαίτηση με τη χρήση της εφαρμογής <http://www.planningpoker.com> .



Η ομάδα δημιούργησε λογαριασμό στην εφαρμογή και ξεκίνησε ένα game ψηφοφορίας. Επιλέχθηκαν κάρτες με δυνάμεις του 2, οι οποίες θα αντιπροσώπευαν ώρες εργασίας. Δεν υπήρχαν σημαντικές αποκλίσεις στην εκτίμηση του κάθε μέλους για την απαιτούμενη προσπάθεια ανά απαίτηση.

Η εκτιμώμενη προσπάθεια ανά απαίτηση φαίνεται στον παρακάτω πίνακα:

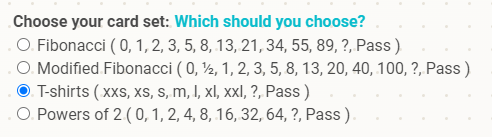


Ο συνολικός χρόνος που απαιτείται είναι 272 ώρες ή 34 εργατοημέρες. Όπως θα δούμε στη συνέχεια(2.5) οι εκτιμήσεις αποκλίνουν μερικώς του πραγματικού χρόνου καθώς υπήρξαν απρόοπτες καθυστερήσεις.

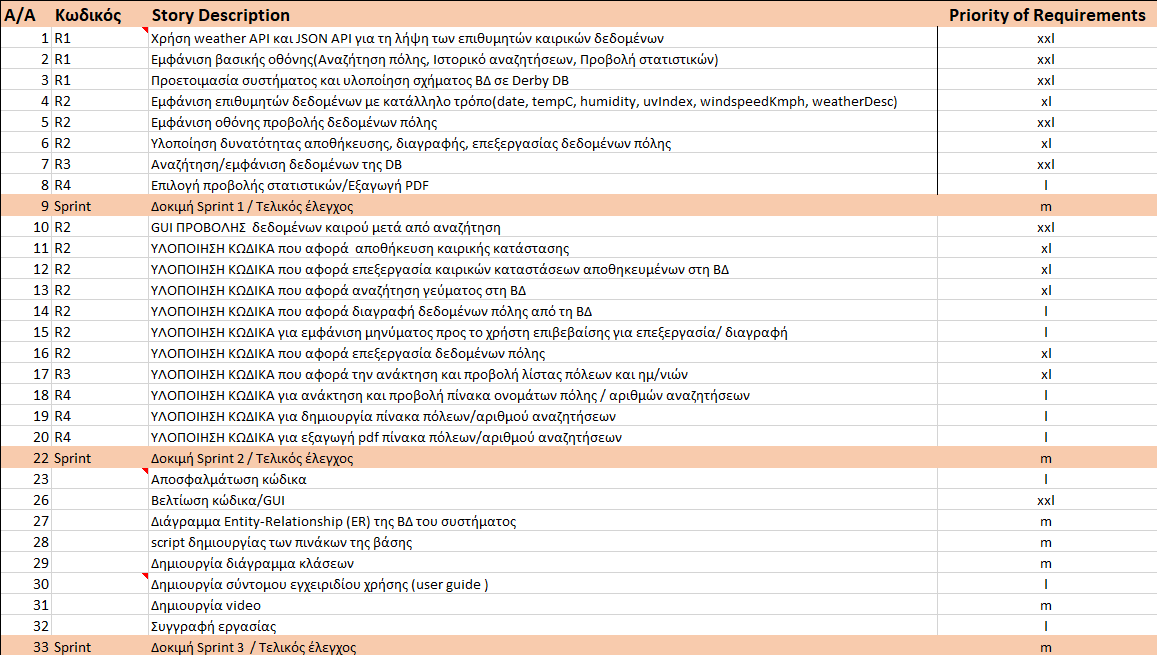
## Υπολογισμός των προτεραιοτήτων

Με τη χρήση της εφαρμογής <http://www.planningpoker.com> υπολογίστηκε και η προτεραιότητα (priority) ανά απαίτηση.

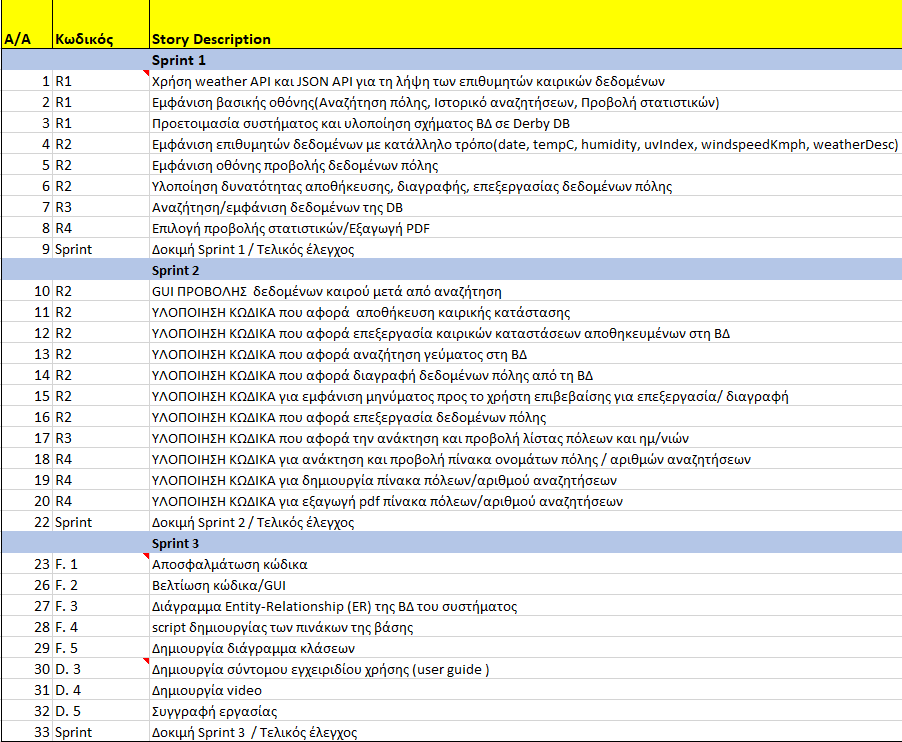
Επιλέχθηκε game ψηφοφορίας με μεγέθη:



Η προτεραιότητα ανά απαίτηση φαίνεται στον παρακάτω πίνακα:



## To product backlog

**

## Οργάνωση ομάδας και αναθέσεις αρμοδιοτήτων

**Product Owner (Ιδιοκτήτης Προϊόντος): Παναγιώτης Φιτσιλής, ΣΕΠ του τμήματος**

Σε συνεργασία με την ομάδα ανάπτυξης, ο product owner επιλέγει ποιες απαιτήσεις θα υλοποιηθούν σε κάθε επανάληψη. Επίσης, αναλαμβάνει ενεργό ρόλο στην εκτίμηση της προσπάθειας που απαιτείται για την υλοποίηση των απαιτήσεων και συμβάλει στην αναζήτηση λύσεων όταν παρουσιάζονται προβλήματα.

**Scrum Master: Χριστόφορος Μαρκόπουλος**

Ο Scrum Master είναι υπεύθυνος για τη σωστή εφαρμογή της μεθοδολογίας Scrum. Λειτουργεί ως μεσολαβητής μεταξύ του Product Owner και της ομάδας εργασίας. Συντονίζει την ομάδα υλοποίησης του έργου και διασφαλίζει ότι αυτή είναι πλήρως λειτουργική και αποτελεσματική. Επίσης, βοηθά στην αντιμετώπιση δυσκολιών ώστε η ομάδα να πετύχει τους στόχους της και προωθεί την συνεχή βελτίωση της ομάδας και της μεθόδου εργασίας.

**Scrum Team: Γιαρμαδούρου Αλεξάνδρα, Θεολόγου Σπυρίδων, Μαρκόπουλος Χριστόφορος**

Η Scrum Team είναι υπεύθυνη για την ανάπτυξη και την παράδοση συγκεκριμένων εργασιών σε κάθε Sprint.

Στην αρχή του έργου, η ομάδα συναποφάσισε για την κατανομή των εργασιών.

O Μαρκόπουλος Χριστόφορος ανέλαβε το σχεδιασμό και την υλοποίηση του GUI της εφαρμογής και τη δημιουργία του video.

Ο Θεολόγου Σπυρίδων ανέλαβε την υλοποίηση REST API, την ανάπτυξη κώδικα για τη δημιουργία PDF, τη σύνταξη του product backlog, τη δημιουργία των burnt down charts, καθώς και τις ψηφοφορίες Planning poker και Priority poker.

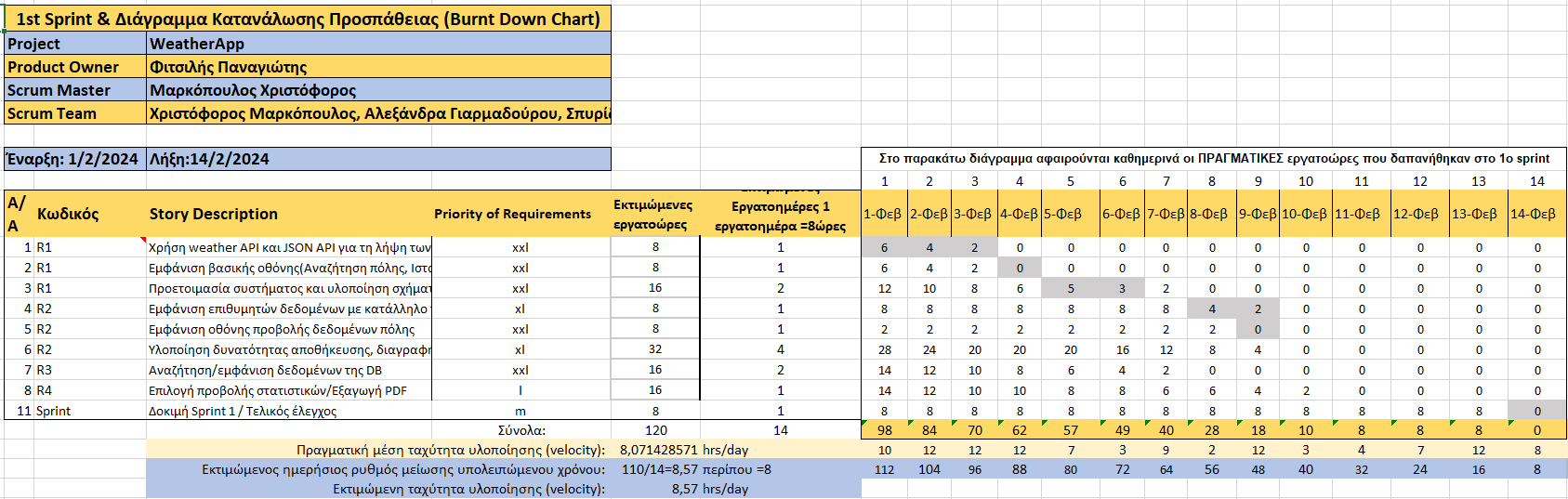
Η Γιαρμαδούρου Αλεξάνδρα ανέλαβε το σχεδιασμό και την υλοποίηση της ΒΔ, την ανάπτυξη κώδικα για τις CRUD operations της ΒΔ, και τη συγγραφή του συνοδευτικού εγγράφου της εργασίας.

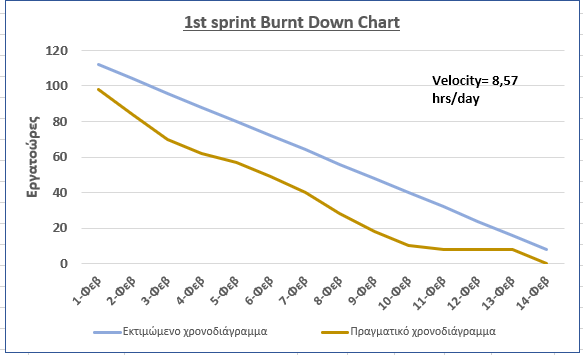
Η ομάδα συντονίζεται με συχνές ομαδικές τηλεδιασκέψεις και ανταλλαγή μηνυμάτων, μέσω της εφαρμογής Discord. Σκοπός αυτών των συναντήσεων είναι τα μέλη της ομάδας να μπορούν να αναζητήσουν βοήθεια, να επισημάνουν προβλήματα που συνάντησαν στην ανάπτυξη του έργου και να αναφέρουν τι έχουν καταφέρει. Η ανταλλαγή αρχείων μεταξύ των μελών της ομάδας γίνεται με χρήση του Google Drive.

## Παρακολούθηση της προσπάθειας κατά τη διάρκεια του έργου

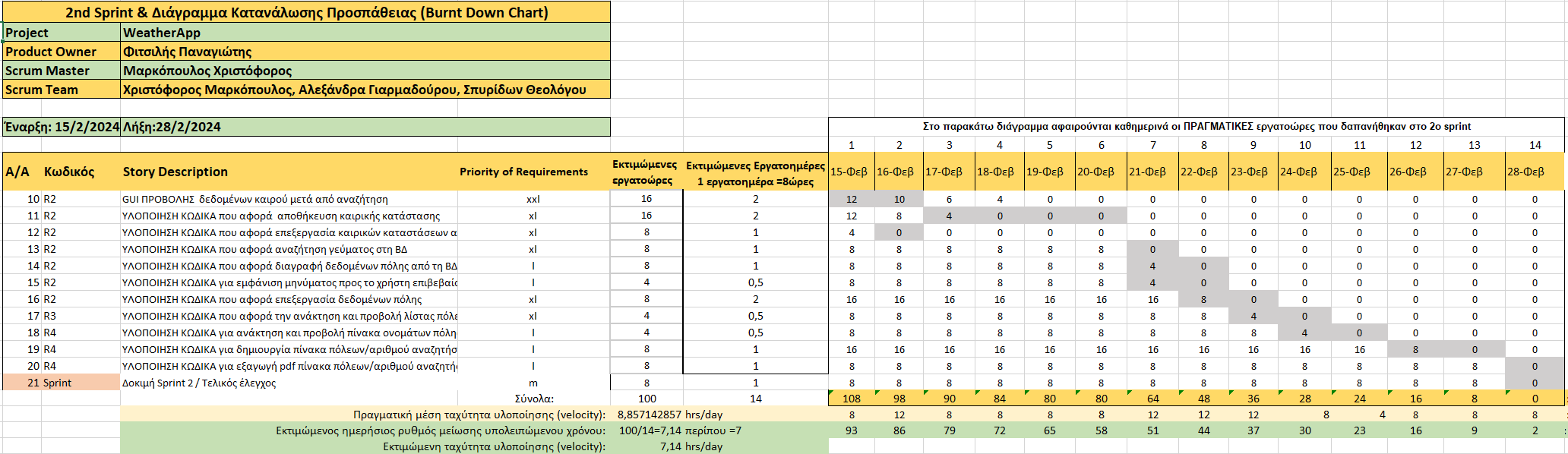
Παρατίθεται ο πραγματικός χρόνος υλοποίησης ανά απαίτηση για κάθε Sprint, σε σύγκριση με τον εκτιμώμενο χρόνο. Οι αποκλίσεις διαφαίνονται πιο καθαρά στα burn – down charts, τα οποία περιέχονται και στα 3 φύλλα(Sprint 1, Sprint 2, Sprint 3) του Product Backlog.xlsx που περιέχεται στα συνοδευτικά έντυπα της εφαρμογής.

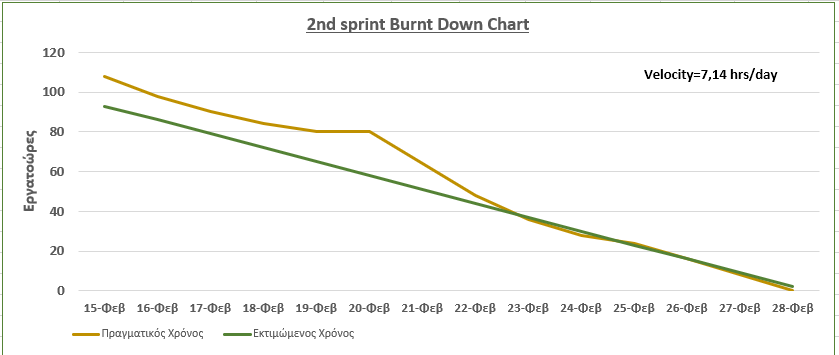
**Sprint 1:**

****

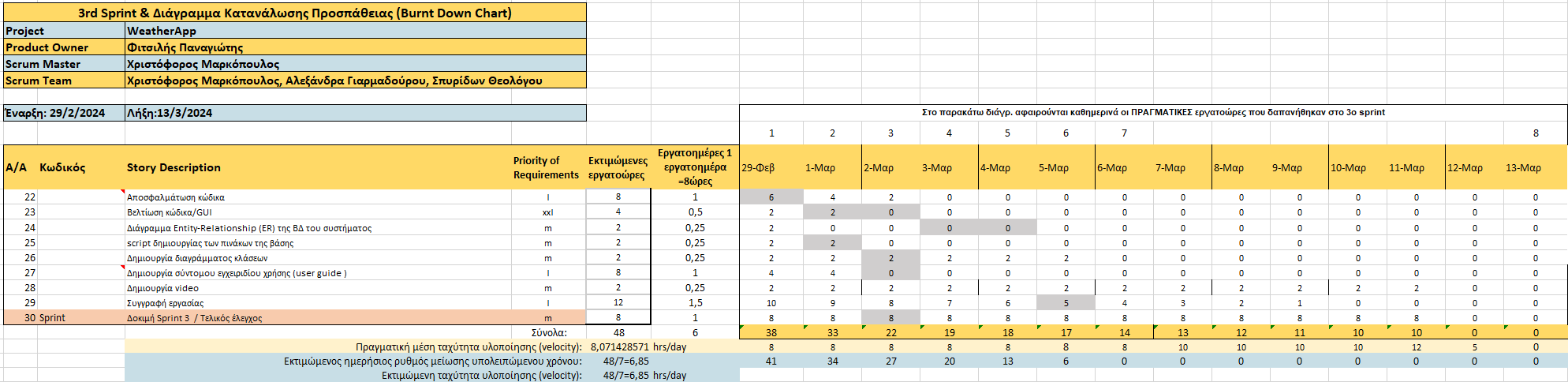
****

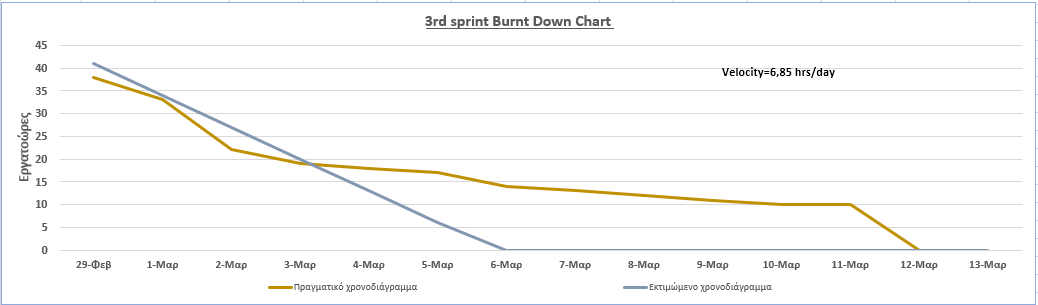
**Sprint 2:**

**

**

**Sprint 3:**

**

**

Στο 1ο Sprint παρατηρούμε πως η διεκπεραίωση των εργασιών ολοκληρώνεται σε χρόνο μικρότερο από τον εκτιμώμενο, καθώς είχε γίνει μια υπερεκτίμηση του χρόνου για την αναζήτηση πληροφοριών σχετικά με τα νέα εργαλεία που καλούμασταν να χρησιμοποιήσουμε.

Στο 2ο Sprint παρατηρούμε πως η διεκπεραίωση των εργασιών ολοκληρώνεται σε χρόνο μεγαλύτερο από τον εκτιμώμενο, καθώς υπήρξαν αναπάντεχες δυσκολίες που μας ώθησαν στο να αφιερώσουμε μεγάλο κομμάτι του χρόνου μας σε επίλυση σφαλμάτων και προβλήματα ασυμβατότητας.

Στο 3ο Sprint υπάρχει μεγάλη διαφορά καθώς δόθηκε παράταση στην προθεσμία υποβολής. Ο επιπλέον χρόνος της παράτασης χρησιμοποιήθηκε για την «εκλέπτυνση» των λειτουργιών και του GUI.

## Χρήση εργαλείου Trello

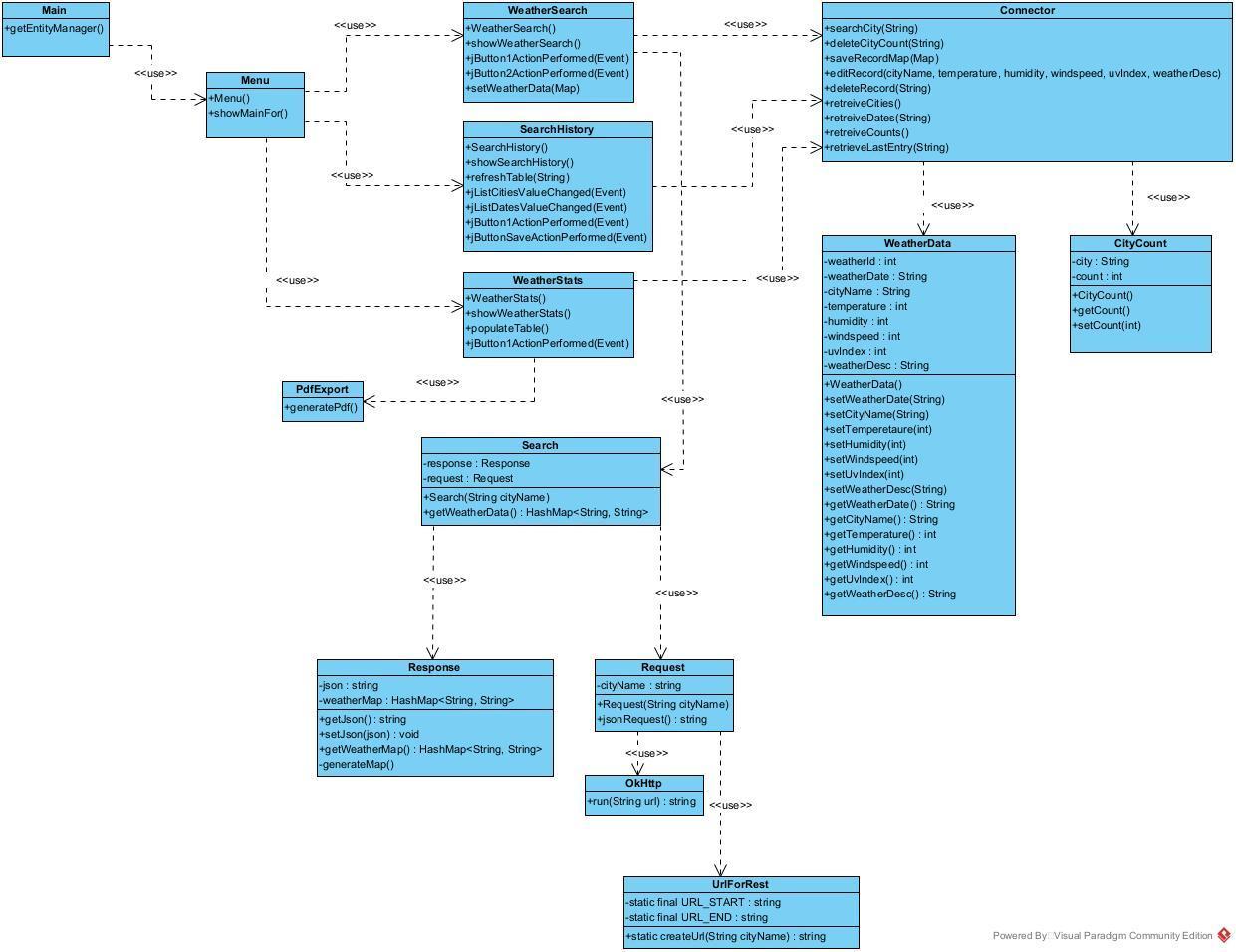
Το Trello ([www.trello.com](http://www.trello.com)) είναι ένα εργαλείο διαχείρισης έργων που βασίζεται σε κάρτες και πίνακες (boards). Ο κάθε πίνακας αντιπροσωπεύει ένα έργο ή έναν τομέα εργασίας και περιλαμβάνει κάρτες που αντιστοιχούν σε διαφορετικές εργασίες ή εργαλεία. Οι χρήστες μπορούν να μετακινούν τις κάρτες από το ένα σημείο του πίνακα στο άλλο, σύμφωνα με την πρόοδο του έργου ή την αλλαγή της κατάστασής τους. Οι κάρτες μπορεί να περιέχουν πληροφορίες όπως λεπτομέρειες εργασίας, προτεραιότητες, σχόλια και συνημμένα αρχεία.

Η ομάδα δημιούργησε ένα board για το project και τοποθέτησε τις κάρτες με τις εργασίες του αρχικού sprint. Ωστόσο, αν και έγινε προσπάθεια, η ομάδα δεν κατάφερε να το ενσωματώσει επιτυχώς στις διαδικασίες. Η ομάδα αντιμετώπισε προβλήματα με την οργάνωση και την ενημέρωση του πίνακα, καθώς επικεντρώθηκε περισσότερο σε καθημερινές συναντήσεις και συζητήσεις για την πρόοδο του έργου, παρά στη χρήση του εργαλείου. Ως αποτέλεσμα, η ομάδα επέλεξε να απομακρυνθεί από το Trello και να χρησιμοποιήσει άλλες μεθόδους οργάνωσης των εργασιών της.

# ΥΛΟΠΟΙΗΣΗ ΕΡΓΟΥ

## Ερώτημα Α– Διάγραμμα Κλάσεων και Υλοποίηση Κλάσεων σε Java

Α) Το διάγραμμα κλάσεων βρίσκεται στο συνοδευτικό αρχείο Class\_diagram.vpp:



Το script που χρησιμοποιήθηκε για τη δημιουργία των πινάκων της βάσης είναι το παρακάτω:

**CREATE** **TABLE** WEATHER\_DATA

**(**

WEATHER\_ID INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY** **(START** **WITH** 1**,** **INCREMENT** **BY** 1**),**

WEATHER\_DATE VARCHAR**(**10**)** **NOT** **NULL,**

CITY\_NAME VARCHAR**(**60**)** **NOT** **NULL,**

TEMPERATURE INTEGER **NOT** **NULL,**

HUMIDITY INTEGER **NOT** **NULL,**

WINDSPEED INTEGER **NOT** **NULL,**

UV\_INDEX INTEGER **NOT** **NULL,**

WEATHER\_DESC VARCHAR**(**100**)** **NOT** **NULL**

**)**

**CREATE** **TABLE** CITY\_COUNT

**(**

CITY VARCHAR**(**60**)** **NOT** **NULL** **PRIMARY** **KEY,**

COUNT INTEGER **NOT** **NULL**

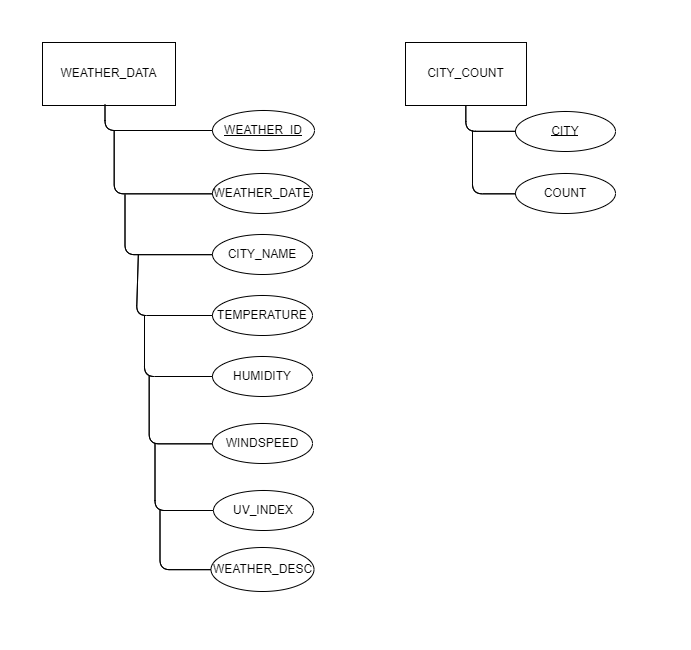
**)**

Ο πίνακας WEATHER\_DATA είναι αυτός στον οποίο αποθηκεύονται όλες οι εγγραφές των καιρικών δεδομένων. Το primary key του είναι το WEATHER\_ID, ένας ακέραιος που γίνεται auto-generated κάθε φορά που πραγματοποιείται μια εγγραφή στον πίνακα. Με αυτόν τον τρόπο εξασφαλίζεται η ύπαρξη διαφορετικών εγγραφών για την ίδια πόλη, με αναγνωριστικό κάθε εγγραφής το WEATHER\_ID. Οι υπόλοιπες στήλες του πίνακα είναι: WEATHER\_DATE (ημερομηνία εγγραφής), CITY\_NAME (όνομα πόλης), TEMPERATURE (θερμοκρασία), HUMIDITY(υγρασία), WINDSPEED(ταχύτητα ανέμου), UV\_INDEX(δείκτης UV), WEATHER\_DESC(περιγραφή καιρού).

Ο πίνακας CITY\_COUNT είναι αυτός στον οποίο αποθηκεύονται όλες οι αναζητήσεις με βάση το όνομα της πόλης. Το primary key του είναι το CITY, το όνομα της πόλης για την οποία πραγματοποιήθηκε αναζήτηση. Η άλλη στήλη του πίνακα είναι η COUNT, η οποία ανανεώνεται με κάθε αναζήτηση για την κάθε πόλη.

Δεν υπάρχει κάποια συσχέτιση μεταξύ των δυο πινάκων.

Το ER διάγραμμα της βάσης φαίνεται παρακάτω:



Β)

**Κώδικας της κλάσης Main:**

1 package pli24.weatherapp;

2

3 import javax.persistence.EntityManager;

4 import javax.persistence.EntityManagerFactory;

5 import javax.persistence.Persistence;

6 import pli24.weatherapp.windows.Menu;

7

8 /\*\*

9 \*

10 \* **@author** alexa

11 \*/

12 public class **Main** {

13

14 //--------------------------------------------------------------------------

15 // obtaining an EntityManager instance

16 private static EntityManagerFactory *entityManagerFactory*;

17 public static EntityManager ***getEntityManager***() {

18 if (*entityManagerFactory* == null) {

19 *entityManagerFactory* =

20 Persistence.*createEntityManagerFactory*("weatherAppPU");

21 }

22 return *entityManagerFactory*.createEntityManager();

23 }

24 //--------------------------------------------------------------------------

25

26

27

28

29 public static void ***main***(String[] args) {

30

31 Menu.*showMenu*();

32

33

34

35 }

36

37 }

38

**Κώδικας των κλάσεων για την υλοποίηση του REST API:**

OkHttp

1 package pli24.weatherapp.REST;

2

3 /\*\*

4 \*

5 \* **@author** stheo

6 \*/

7

8 import java.io.IOException;

9 import okhttp3.OkHttpClient;

10 import okhttp3.Request;

11 import okhttp3.Response;

12

13 public class **OkHttp** {

14

15 // create new OkHttpClient

16 private OkHttpClient client = new OkHttpClient.Builder().build();

17

18 // returns a response string of the json object requested

19 public String **run**(String url) {

20 Request request = new Request.Builder().url(url).build();

21

22 try (Response response = client.newCall(request).execute()) {

23 if (response.isSuccessful() && response.body() != null) {

24 return response.body().string();

25 }

26 }

27 catch (IOException e){

28 System.*err*.println(e);

29

30 }

31 return null;

32 }

33 }

34

Request

1 package pli24.weatherapp.REST;

2

3 /\*\*

4 \*

5 \* **@author** stheo

6 \*/

7 public class **Request** {

8 private String cityName;

9

10 // request builder

11 public **Request**(String cityName){

12 this.cityName = cityName;

13 System.*out*.println("Creating request for '" + this.cityName + "'");

14 }

15

16 // returns a json with the given city's weather data

17 public String **jsonRequest**(){

18 if (this.cityName.equals(""))

19 return null;

20 OkHttp callHttp = new OkHttp();

21 String url = UrlForRest.*createUrl*(this.cityName);

22 String json = callHttp.run(url);

23 return json;

24 }

25

26 }

27

Response

1 package pli24.weatherapp.REST;

2

3 /\*\*

4 \*

5 \* **@author** stheo

6 \*/

7

8 import com.google.gson.\*;

9 import java.util.Map;

10 import java.lang.reflect.Type;

11 import com.google.gson.reflect.TypeToken;

12 import java.util.HashMap;

13 import java.util.List;

14

15

16

17 public class **Response** {

18

19 private String json;

20 private HashMap<String, String> weatherMap = new HashMap<String, String>();

21

22 // builder

23 public **Response**(Request request){

24 this.json = request.jsonRequest();

25 try{

26 this.generateMap();

27 }

28 catch(NullPointerException e){

29 System.*err*.println("Failed to generate weather map!");

30 }

31

32 }

33

34

35 public void **setJson**(String json){ this.json = json;}

36

37 public String **getJson**() { return this.json;}

38

39

40 // generates a weatherMap for the Response object {date, weatherDesc, cityName, windspeedKmph,humidity, uvIndex, tempC }

41 private void **generateMap**(){

42 Gson gson = new Gson();

43 Type mapType = new TypeToken<Map<String, List<Map<String, Object>>>>() {}.getType();

44 Map<String, List<Map<String, Object>>> map = gson.fromJson(this.json, mapType);

45

46 //cityName

47 this.weatherMap.put("cityName", map.get("nearest\_area").get(0).get("areaName").toString().replace("[{value=", "").replace("}]",""));

48

49 //date(YY-MM-DD)

50 this.weatherMap.put("date",

51 map.get("current\_condition").get(0).get("localObsDateTime").toString().substring(0, 10));

52

53 //tempC

54 this.weatherMap.put("tempC",

55 map.get("current\_condition").get(0).get("temp\_C").toString());

56

57 //humidity

58 this.weatherMap.put("humidity",

59 map.get("current\_condition").get(0).get("humidity").toString());

60

61 //uvIndex

62 this.weatherMap.put("uvIndex",

63 map.get("current\_condition").get(0).get("uvIndex").toString());

64

65 //windspeedKmph

66 this.weatherMap.put("windspeedKmph",

67 map.get("current\_condition").get(0).get("windspeedKmph").toString());

68

69 //weatherDesc

70 this.weatherMap.put("weatherDesc",

71 map.get("current\_condition").get(0).get("weatherDesc").toString().replace("[{value=", "").replace("}]",""));

72

73

74 System.*out*.println("Map Generated!");

75 }

76

77

78 public HashMap<String, String> **getWeatherMap**(){ return this.weatherMap; }

79 }

80

Search

1 package pli24.weatherapp.REST;

2

3 /\*\*

4 \*

5 \* **@author** stheo

6 \*/

7 import java.util.HashMap;

8

9 public class **Search** {

10

11 private Response response;

12 private Request request;

13

14 public **Search**(String cityName){

15 this.request = new Request(cityName);

16 this.response = new Response(request);

17 }

18

19 /\* returns a map with the keys :

20 "weatherDesc",

21 "cityName",

22 "uvIndex",

23 "date",

24 "tempC",

25 "windspeedKmph" \*/

26 public HashMap<String, String> **getWeatherData**(){

27 return response.getWeatherMap();

28 }

29 }

30

UrlForRest

1 package pli24.weatherapp.REST;

2

3 /\*\*

4 \*

5 \* **@author** stheo

6 \*/

7 public class **UrlForRest** {

8

9 // start and end of url requests

10 private static final String *URL\_START* = "https://wttr.in/";

11 private static final String *URL\_END* = "?format=j1";

12

13 // given a city name, returns a formatted url i.e. "https://wttr.in/Patras/format=j1";

14 public static String ***createUrl***(String cityName){

15 return *URL\_START* + cityName + *URL\_END*;

16 }

17 }

18

**Γ) Κώδικας της κλάσης Connector**

Με τη χρήση του JPA, από τη βάση δεδομένων δημιουργήθηκαν δύο POJO κλάσεις, η CityCount και η WeatherData. Δημιουργήσαμε μια τρίτη, την Connector, στην οποία περιλαμβάνονται όλες οι μέθοδοι για τα CRUD operations που χρησιμοποιήσαμε:

1 package pli24.weatherapp.db;

2

3 import java.util.List;

4 import java.util.Map;

5 import java.util.Objects;

6 import javax.persistence.EntityManager;

7 import javax.persistence.Query;

8 import pli24.weatherapp.Main;

9

10 /\*\*

11 \*

12 \* **@author** alexa

13 \*/

14 public class **Connector** {

15

16 //-------------------------------------------------------------------------

17 //New search: ADD city in CITY\_COUNT, sets count to 1

18 //If city already in table, count +1

19 public static void ***searchCity***(String city){

20 EntityManager em = Main.*getEntityManager*();

21 em.getTransaction().begin();

22 Query query = em.createNamedQuery("CityCount.findByCity");

23 query.setParameter("city", city);

24 List resultList = query.getResultList();

25

26 if (resultList.isEmpty()){

27 CityCount c = new CityCount(city, 1);

28 em.persist(c);

29 }

30 else {

31 CityCount c=(CityCount) resultList.get(0) ;

32 int currentCount=c.getCount();

33 c.setCount(currentCount + 1);

34 em.merge(c);

35 }

36 em.getTransaction().commit();

37 em.close();

38 }

39 //--------------------------------------------------------------------------

40

41 //-------------------------------------------------------------------------

42 //DELETE City from CityCount

43 public static void ***deleteCityCount***(String city){

44 EntityManager em = Main.*getEntityManager*();

45 em.getTransaction().begin();

46 CityCount c = em.createNamedQuery("CityCount.findByCity",CityCount.class).

47 setParameter("city", city).getSingleResult();

48 em.remove(c);

49 em.getTransaction().commit();

50 em.close();

51 }

52 //--------------------------------------------------------------------------

53

54 //--------------------------------------------------------------------------

55 //SAVE weather record in WEATHER\_DATA if it doesn't already exist (map)

56 public static void ***saveRecordMap***(Map<String, String> map){

57 EntityManager em = Main.*getEntityManager*();

58 em.getTransaction().begin();

59

60 List<WeatherData> wList=em.createQuery(

61 "SELECT wd FROM WeatherData wd WHERE wd.cityName LIKE :cityName AND wd.weatherDate LIKE :weatherDate",

62 WeatherData.class).setParameter("cityName",map.get("cityName"))

63 .setParameter("weatherDate",map.get("date")).getResultList();

64

65

66 if (wList.isEmpty()){

67 WeatherData w = new WeatherData();

68 w.setCityName(map.get("cityName"));

69 w.setWeatherDate(map.get("date"));

70 w.setTemperature(Integer.*parseInt*(map.get("tempC")));

71 w.setHumidity(Integer.*parseInt*(map.get("humidity")));

72 w.setWindspeed(Integer.*parseInt*(map.get("windspeedKmph")));

73 w.setUvIndex(Integer.*parseInt*(map.get("uvIndex")));

74 w.setWeatherDesc(map.get("weatherDesc"));

75 em.persist(w);

76 }

77 else {

78 System.*out*.println("The record for "+map.get("cityName")+" "+map.get("date")+" already exists.");

79 }

80 em.getTransaction().commit();

81 em.close();

82 }

83 //--------------------------------------------------------------------------

84

85 //-------------------------------------------------------------------------

86 //EDIT weather latest record in WEATHER\_DATA (null for args that won't be changed)

87 public static void ***editRecord***(String cityName, Integer temperature, Integer humidity,

88 Integer windspeed, Integer uvIndex, String weatherDesc){

89

90 Objects.*requireNonNull*(cityName, "City cannot be null");

91

92 EntityManager em = Main.*getEntityManager*();

93 em.getTransaction().begin();

94

95 WeatherData w = em.createQuery(

96 "SELECT wd FROM WeatherData wd WHERE wd.cityName LIKE :cityName ORDER BY wd.weatherDate DESC",

97 WeatherData.class)

98 .setParameter("cityName", cityName)

99 .setMaxResults(1)

100 .getSingleResult();

101

102 if (temperature!=null){

103 w.setTemperature(temperature);

104 }

105 if (humidity!=null) {

106 w.setHumidity(humidity);

107 }

108 if (windspeed!=null) {

109 w.setWindspeed(windspeed);

110 }

111 if (uvIndex!=null) {

112 w.setUvIndex(uvIndex);

113 }

114 if (weatherDesc!=null) {

115 w.setWeatherDesc(weatherDesc);

116 }

117 em.merge(w);

118 em.getTransaction().commit();

119 em.close();

120 }

121 //--------------------------------------------------------------------------

122

123 //-------------------------------------------------------------------------

124 //DELETE all weather records in WEATHER\_DATA for a city

125 public static void ***deleteRecord***(String cityName){

126 EntityManager em = Main.*getEntityManager*();

127 em.getTransaction().begin();

128

129 List <WeatherData> wList = em.createQuery(

130 "SELECT wd FROM WeatherData wd WHERE wd.cityName LIKE :cityName",

131 WeatherData.class)

132 .setParameter("cityName", cityName)

133 .getResultList();

134

135 for (WeatherData w: wList){

136 em.remove(w);

137 }

138 em.getTransaction().commit();

139 em.close();

140 }

141 //--------------------------------------------------------------------------

142

143 //--------------------------------------------------------------------------

144 //RETREIVE all cities in WEATHER\_DATA

145 public static List<String> ***retreiveCities***() {

146 EntityManager em = Main.*getEntityManager*();

147 em.getTransaction().begin();

148

149 List <String> cList=em.createQuery(

150 "SELECT DISTINCT wd.cityName FROM WeatherData wd",String.class)

151 .getResultList();

152 for (String c:cList){

153 System.*out*.println(c);

154 }

155 em.close();

156 return cList;

157

158 }

159 //--------------------------------------------------------------------------

160

161 //--------------------------------------------------------------------------

162 //RETREIVE all dates for a specific city in WEATHER\_DATA

163 public static List<String> ***retreiveDates***(String cityName) {

164 EntityManager em = Main.*getEntityManager*();

165 em.getTransaction().begin();

166

167 List <String> cList=em.createQuery(

168 "SELECT wd.weatherDate FROM WeatherData wd WHERE wd.cityName LIKE :cityName",

169 String.class)

170 .setParameter("cityName", cityName)

171 .getResultList();

172 for (String c:cList){

173 System.*out*.println(c);

174 }

175 em.close();

176 return cList;

177 }

178 //--------------------------------------------------------------------------

179

180 //--------------------------------------------------------------------------

181 //RETREIVE all cities in CITY\_COUNT in descending order(counts)

182 public static List<Object[]> ***retreiveCounts***() {

183 EntityManager em = Main.*getEntityManager*();

184 em.getTransaction().begin();

185

186 List <Object[]> cList=em.createQuery(

187 "SELECT c.city,c.count FROM CityCount c ORDER BY c.count DESC",Object[].class)

188 .getResultList();

189

190 //print count for each city

191 for (Object[] row : cList) {

192 String city = (String) row[0];

193 int count = (int) row[1];

194 System.*out*.printf("%-20s | %d%n", city, count);

195 }

196 em.close();

197 return cList; //Return list of records as objects

198 }

199 //--------------------------------------------------------------------------

200

201 //--------------------------------------------------------------------------

202 //RETRIEVE the last entry for a specific city

203 public static WeatherData ***retrieveLastEntry***(String cityName){

204

205 Objects.*requireNonNull*(cityName, "City cannot be null");

206

207 EntityManager em = Main.*getEntityManager*();

208 em.getTransaction().begin();

209

210 WeatherData w = em.createQuery(

211 "SELECT wd FROM WeatherData wd WHERE wd.cityName LIKE :cityName ORDER BY wd.weatherDate DESC",

212 WeatherData.class)

213 .setParameter("cityName", cityName)

214 .setMaxResults(1)

215 .getSingleResult();

216

217 em.close();

218 return w;

219 }

220 //--------------------------------------------------------------------------

221

222 }

223

## Ερώτημα Β – Δημιουργία GUI Εφαρμογής

**Κώδικας των κλάσεων για την υλοποίηση του GUI(έχουν αφαιρεθεί τα auto Generated κομμάτια κώδικα):**

**Κώδικας κλάσης αρχικής οθόνης /Menu:**

1 package pli24.weatherapp.Windows;

2

3 /\*\*

4 \*

5 \* **@author** Xristoforos

6 \*/

7 public class **Menu** extends javax.swing.JFrame {

8

9 /\*\*

10 \* Creates new form Menu

11 \*/

12 public **Menu**() {

13 *initComponents*();

14 setLocationRelativeTo(null);

15 }

16

17

18 private void **jButtonSearchLocationActionPerformed**(java.awt.event.ActionEvent evt) {

19 WeatherSearch.*showWeatherSearch*();

20 }

21

22 private void **jButton2showSearchHistoryActionPerformed**(java.awt.event.ActionEvent evt) {

23 SearchHistory.*showSearchHistory*();

24 }

25

26 private void **jButton3shoWeatherStatsActionPerformed**(java.awt.event.ActionEvent evt){

27 WeatherStats.*showWeatherStats*();

28 }

29

30 /\*\*

31 \* **@param** args the command line arguments

32 \*/

33 public static void ***showMenu***() {

34

35 /\* Create and display the form \*/

36 java.awt.EventQueue.*invokeLater*(new Runnable() {

37 public void **run**() {

38 new Menu().setVisible(true);

39 }

40 });

41 }

42 }

**Κώδικας οθόνης αναζήτησης τοποθεσίας / WeatherSearch:**

1 package pli24.weatherapp.Windows;

2

3 import java.util.HashMap;

4 import java.util.Map;

5 import javax.swing.JOptionPane;

6 import REST.Search;

7 import DB.Connector;

8

9 /\*\*

10 \*

11 \* **@author** Alexandra

12 \*/

13 public class **WeatherSearch** extends javax.swing.JFrame {

14 private Map<String, String> weatherMap;

15

16 /\*\*

17 \* Creates new form WeatherSearch

18 \*/

19 public **WeatherSearch**() {

20 *initComponents*();

21 setLocationRelativeTo(null);

22 }

23

24 private void **jTextField1ActionPerformed**(java.awt.event.ActionEvent evt) {

25

26 //

27 }

28

29

30

31

32 private void **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {

33 // Call the saveRecordMap method from Connector

34 Connector.*saveRecordMap*(weatherMap);

35 // Display a message

36 JOptionPane.*showMessageDialog*(this, "Η αποθήκευση ήταν επιτυχής.");

37 }

38

39 private void **jButton2ActionPerformed**(java.awt.event.ActionEvent evt) {

40 // Retrieve weather data

41 HashMap<String, String> weatherMap = new Search(*jTextField1*.*getText*()).getWeatherData();

42

43 // Check if weather data exists

44 if (!weatherMap.isEmpty()) {

45

46 // Set the weather data in WeatherForm

47 setWeatherData(weatherMap);

48

49 //set CityCount in DB

50 Connector.*searchCity*(weatherMap.get("cityName"));

51 } else {

52 // Show an error message if weather data does not exist

53 JOptionPane.*showMessageDialog*(this,"Η πόλη δεν υπάρχει");

54 }

55 }

56

57 public void **setWeatherData**(HashMap<String, String> weatherMap) {

58 // Set the city name in the JTextField

59 *jTextField1*.*setText*(weatherMap.get("cityName"));

60

61 // Format the weather data

62 StringBuilder weatherDataString = new StringBuilder();

63

64 weatherDataString.append("Ημερομηνία: ").append(weatherMap.get("date")).append("**\n**");

65 weatherDataString.append("Θερμοκρασία (C): ").append(weatherMap.get("tempC")).append("**\n**");

66 weatherDataString.append("Υγρασία (%): ").append(weatherMap.get("humidity")).append("**\n**");

67 weatherDataString.append("Ταχύτητα Ανέμου (Kmph): ").append(weatherMap.get("windspeedKmph")).append("**\n**");

68 weatherDataString.append("Δείκτης UV: ").append(weatherMap.get("uvIndex")).append("**\n**");

69 weatherDataString.append("Περιγραφή Καιρού: ").append(weatherMap.get("weatherDesc")).append("**\n**");

70

71 // Update the JTextArea with the weather data

72 *jTextArea2*.*setText*(weatherDataString.toString());

73

74 // Assign weatherMap to the class-level variable

75 this.weatherMap = weatherMap;

76 }

77

78

79 public static void ***showWeatherSearch***() {

80 java.awt.EventQueue.*invokeLater*(() -> {

81 new WeatherSearch().setVisible(true);

82 });

83 }

84 }

85

**Κώδικας οθόνης εμφάνισης ιστορικού / SearchHistory:**

1 package pli24.weatherapp.Windows;

2

3

4 import java.util.List;

5 import javax.swing.DefaultListModel;

6 import javax.swing.JOptionPane;

7 import DB.Connector;

8 import DB.WeatherData;

9 import javax.swing.table.DefaultTableModel;

10 import java.lang.NullPointerException;

11

12

13 public class **SearchHistory** extends javax.swing.JDialog {

14

15 /\*\*

16 \* Creates new form MealCategoriesForm

17 \*/

18 public **SearchHistory**() {

19 *initComponents*();

20 setLocationRelativeTo(null);

21

22 *jListCities*.*setEnabled*(false);

23 *jListDates*.*setEnabled*(false);

24

25 List<String> cities = Connector.*retreiveCities*();

26 DefaultListModel<String> citiesList = new DefaultListModel<>();

27

28 for (String city : cities) {

29 citiesList.addElement(city);

30 }

31 *jListCities*.*setModel*(citiesList);

32 *jListCities*.*setEnabled*(true);

33

34 pack();

35 setLocationRelativeTo(null);

36 setModal(true);

37

38 }

39

40 // creates/refreshes the last entry table

41 public void **refreshTable**(String cityName){

42 try{

43 WeatherData w = Connector.*retrieveLastEntry*(cityName);

44 //column Names

45 String cols[] = {"date",

46 "tempC",

47 "humidity",

48 "weatherDesc",

49 "windspeedKmph",

50 "uvIndex"};

51

52

53 //getting the values from weatherdata

54 String data[][] = {{

55 w.getWeatherDate(),

56 String.*valueOf*(w.getTemperature()),

57 String.*valueOf*(w.getHumidity()),

58 w.getWeatherDesc(),

59 String.*valueOf*(w.getWindspeed()),

60 String.*valueOf*(w.getUvIndex())

61 }};

62

63 // recreating the table

64 DefaultTableModel model = new DefaultTableModel(data, cols){

65 @Override

66 public boolean **isCellEditable**(int row, int column) {

67 // Make the cell in the first column (Date) uneditable

68 return column != 0;

69 }

70 };

71

72 *jTableLastEntry*.*setModel*(model);

73 }

74 catch(java.lang.NullPointerException e){

75 System.*out*.println(e);

76 }

77 }

78

79

80 private void **jListCitiesValueChanged**(javax.swing.event.ListSelectionEvent evt) {

81 if (!evt.getValueIsAdjusting()) {

82 *jListDates*.*setEnabled*(false);

83 List<String> datesPerCity = Connector.*retreiveDates*(*jListCities*.*getSelectedValue*());

84 DefaultListModel<String> datesPerCityList = new DefaultListModel<>();

85

86 for (String datePerCity : datesPerCity) {

87 datesPerCityList.addElement(datePerCity);

88 }

89 *jListDates*.*setModel*(datesPerCityList);

90 *jListDates*.*setEnabled*(true);

91

92 // change the table values

93 refreshTable(*jListCities*.*getSelectedValue*());

94 }

95 }

96

97 private void **jListDatesValueChanged**(javax.swing.event.ListSelectionEvent evt) {

98 if (!evt.getValueIsAdjusting() && *jListDates*.*isEnabled*()) {

99

100 }

101 }

102

103 private void **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {

104

105 // Get the selected city from the jListCities component.

106 String selectedCity = *jListCities*.*getSelectedValue*();

107

108 // Check if a city is selected.

109 if (selectedCity != null) {

110 //confirmation dialog

111 int option = JOptionPane.*showConfirmDialog*(this,

112 "Διαγραφή "+selectedCity+" ?", "Επιβεβαίωση Διαγραφής",

113 JOptionPane.*YES\_NO\_OPTION*);

114 if (option == JOptionPane.*YES\_OPTION*) {

115 // Remove the selected city from the list.

116 DefaultListModel<String> model = (DefaultListModel<String>) *jListCities*.*getModel*();

117 model.removeElement(selectedCity);

118

119 //DELETE the city from DB

120 Connector.*deleteCityCount*(selectedCity);

121 Connector.*deleteRecord*(selectedCity);

122 }

123 } else {

124 // Show a message if no city is selected.

125 JOptionPane.*showMessageDialog*(this,

126 "Επιλέξτε πόλη για διαγραφή",

127 "Δεν έχει επιλεγεί πόλη", JOptionPane.*INFORMATION\_MESSAGE*);

128 }

129 }

130

131

132 // button for last entry update confirmation

133 private void **jButtonSaveActionPerformed**(java.awt.event.ActionEvent evt) {

134 // Get the selected city from the jListCities component.

135 String selectedCity = *jListCities*.*getSelectedValue*();

136

137 // Check if a city is selected.

138 if (selectedCity != null) {

139 //confirmation dialog

140 int option = JOptionPane.*showConfirmDialog*(this,

141 "Ενημέρωση τελευταίας καταχώρησης για πόλη: "+selectedCity+" ?", "Επιβεβαίωση Ενημέρωσης",

142 JOptionPane.*YES\_NO\_OPTION*);

143

144

145 if (option == JOptionPane.*YES\_OPTION*) {

146 try{

147 // update record

148 Connector.*editRecord*(selectedCity,

149 Integer.*valueOf*((String) *jTableLastEntry*.*getModel*().*getValueAt*(0,1)),

150 Integer.*valueOf*((String) *jTableLastEntry*.*getModel*().*getValueAt*(0,2)),

151 Integer.*valueOf*((String) *jTableLastEntry*.*getModel*().*getValueAt*(0,4)),

152 Integer.*valueOf*((String) *jTableLastEntry*.*getModel*().*getValueAt*(0,5)),

153 (String) *jTableLastEntry*.*getModel*().*getValueAt*(0,3)

154 );

155 JOptionPane.*showMessageDialog*(this,

156 "Ενημέρωση Επιτυχής!",

157 "Επιτυχία", JOptionPane.*INFORMATION\_MESSAGE*);

158 }

159 catch(Exception e){

160 JOptionPane.*showMessageDialog*(this,

161 "Αποτυχία Ενημέρωσης!",

162 "Σφάλμα", JOptionPane.*INFORMATION\_MESSAGE*);

163

164 }

165 }

166 }

167 // Show a message if no city is selected.

168 else {

169 JOptionPane.*showMessageDialog*(this,

170 "Επιλέξτε πόλη!",

171 "Δεν έχει επιλεγεί πόλη", JOptionPane.*INFORMATION\_MESSAGE*);

172 }

173 }

174

175

176

177

178 /\*\*

179 \* **@param** args the command line arguments

180 \*/

181 public static void ***showSearchHistory***() {

182 /\* Create and display the form \*/

183 java.awt.EventQueue.*invokeLater*(new Runnable() {

184 public void **run**() {

185 new SearchHistory().setVisible(true);

186 }

187 });

188 }

189 }

190

## Ερώτημα Γ – Παρουσίαση στατιστικών στοιχείων καιρού

**Κώδικας οθόνης εμφάνισης ιστορικού(GUI) / SearchHistory:**

1 package pli24.weatherapp.Windows;

2

3 import pli24.weatherapp.PDF.PdfExport;

4 import java.awt.Color;

5 import java.io.FileNotFoundException;

6 import java.awt.Dimension;

7 import java.io.IOException;

8 import java.util.List;

9 import javax.swing.JFileChooser;

10 import javax.swing.JOptionPane;

11 import javax.swing.event.TableModelListener;

12 import javax.swing.filechooser.FileNameExtensionFilter;

13 import javax.swing.table.DefaultTableModel;

14 import DB.Connector;

15

16 /\*\*

17 \*

18 \* **@author** stheo

19 \*/

20 public class **WeatherStats** extends javax.swing.JFrame {

21

22 /\*\*

23 \* Creates new form WeatherStats

24 \*/

25

26

27 public **WeatherStats**() {

28 *initComponents*();

29 setLocationRelativeTo(null);

30 populateTable();

31 }

32

33

34 // populates the table with City, Search Count values

35 public void **populateTable**(){

36 String cols[] = {"City", "Search Count #"};

37 List<Object[]> cityCount = Connector.*retreiveCounts*();

38 System.*out*.println(cityCount.size());

39 String data[][] = new String[cityCount.size()][2];

40

41 // actual population of table

42 int i = 0;

43 for (Object[] row: cityCount){

44 data[i][0] = row[0].toString();

45 data[i][1] = row[1].toString();

46 i++;

47 }

48

49 // disabling cell editing option

50 DefaultTableModel model = new DefaultTableModel(data, cols){

51 @Override

52 public boolean **isCellEditable**(int row, int column) {

53 //all cells false

54 return false;

55 }

56 };

57

58 *jTable1*.*setModel*(model);

59 }

60

61 // creates a PDF of the table

62 private void **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {

63 try{

64 // delete error label text

65 *errorInExport*.*setText*("Pdf Created!");

66 *errorInExport*.*setForeground*(Color.*BLUE*);

67 PdfExport.*generatePdf*();

68 }

69 catch(FileNotFoundException e){

70 System.*out*.println("Something went wrong..");

71 // show error message

72 *errorInExport*.*setForeground*(Color.*red*);

73 *errorInExport*.*setText*("Something went wrong..");

74 }

75 }

76

77

78 public static void ***showWeatherStats***() {

79 java.awt.EventQueue.*invokeLater*(() -> {

80 new WeatherStats().setVisible(true);

81 });

82 }

83 }

**Κώδικας της κλάσης για τη δημιουργία PDF / PdfExport:**

1 package pli24.weatherapp.PDF;

2

3 /\*\*

4 \*

5 \* **@author** stheo

6 \*/

7 import com.itextpdf.kernel.pdf.PdfDocument;

8 import com.itextpdf.kernel.pdf.PdfWriter;

9 import pli24.weatherapp.db.Connector;

10 import com.itextpdf.layout.Document;

11 import com.itextpdf.layout.element.Cell;

12 import com.itextpdf.layout.element.Table;

13 import com.itextpdf.layout.element.Paragraph;

14 import com.itextpdf.kernel.colors.ColorConstants;

15 import java.io.FileNotFoundException;

16 import java.util.List;

17

18

19 public class **PdfExport** {

20

21 // creates CITY\_COUNT table in PDF form

22 public static void ***generatePdf***() throws FileNotFoundException{

23

24

25 List<Object[]> cityCounts = Connector.*retreiveCounts*();

26

27 // Creating a PdfDocument object

28 final String dest = "CityCounts.pdf";

29 PdfWriter writer = new PdfWriter(dest);

30

31 // Creating a PdfDocument object

32 PdfDocument pdf = new PdfDocument(writer);

33

34 // Creating a Document object

35 Document doc = new Document(pdf);

36

37 // Creating a table

38 float [] pointColumnWidths = {275F, 275F};

39 Table table = new Table(pointColumnWidths);

40

41

42 // Adding header cells to the table

43 table.addCell(new Cell().add(new Paragraph("City").setBold()).setBackgroundColor(ColorConstants.*GRAY*));

44 table.addCell(new Cell().add(new Paragraph("Search Count#").setBold()).setBackgroundColor(ColorConstants.*GRAY*));

45

46

47 // Adding the appropriate values to the table

48 for (Object[] row : cityCounts) {

49 String city = (String) row[0];

50 int count = (int) row[1];

51 table.addCell(new Cell().add(new Paragraph(city)));

52 Paragraph countParagraph = new Paragraph().add(String.*valueOf*(count));

53 table.addCell(new Cell().add(countParagraph));

54 }

55

56 // Adding Table to document

57 doc.add(table);

58

59 // Closing the document

60 doc.close();

61 System.*out*.println("Table created successfully..");

62

63 }

64 }

65

## Ερώτημα Δ – Συνολικός Έλεγχος και Εκτέλεση της Εφαρμογής

| **Απαίτηση** | **Εικόνες** |
| --- | --- |
| **R1** | Εμφάνιση του αρχικού μενού με τις διαθέσιμες επιλογές. |
| **R2** | Πατώντας το κουμπί «Αναζήτηση Τοποθεσίας» του αρχικού μενού, εμφανίζεται το παράθυρο «Αναζήτηση». Στο πεδίο «Όνομα Πόλης» εισάγεται το όνομα της πόλης για να πραγματοποιηθεί αναζήτηση.      Εμφανίζονται τα στοιχεία καιρού για την πόλη που αναζητήθηκε. Η επιλογή «Αποθήκευση δεδομένων καιρού» δίνει τη δυνατότητα να αποθηκευτούν τα δεδομένα.    Μετά την αποθήκευση προβάλλεται σχετικό μήνυμα. |
| **R3** | Πατώντας το κουμπί «Προβολή Ιστορικού Αναζήτησης» του αρχικού μενού, εμφανίζεται το παράθυρο «Λίστα Πόλεων»    Στη λίστα «Πόλεις» εμφανίζονται οι πόλεις για τις οποίες έχουν αποθηκευτεί δεδομένα καιρού. Με την επιλογή μιας πόλης από τη λίστα, εμφανίζεται λίστα ημερομηνιών στις οποίες πραγματοποιήθηκε αποθήκευση στοιχείων. Επιπλέον, εμφανίζονται τα καιρικά στοιχεία της τελευταίας καταχώρισης για την πόλη αυτή. Τα στοιχεία αυτά μπορούν να τροποποιηθούν κατευθείαν μέσα στα αντίστοιχα πεδία μετά από διπλό κλικ, πληκτρολογώντας τις νέες τιμές και ENTER. Για να αποθηκευτούν, πρέπει να πατηθεί το κουμπί «Αποθήκευση Αλλαγών».    Προβάλλεται σχετικό μήνυμα για επιβεβαίωση, και τα στοιχεία αποθηκεύονται.  Το κουμπί «Διαγραφή Πόλης» δίνει τη δυνατότητα να διαγραφούν όλες οι εγγραφές για τη συγκεκριμένη πόλη που έχει επιλεγεί από τη λίστα. Εμφανίζεται μήνυμα για επιβεβαίωση.    Μετά την επιτυχή διαγραφή, εμφανίζεται η ανανεωμένη λίστα πόλεων. |
| **R4** | Πατώντας το κουμπί «Στατιστικά Καιρού» του αρχικού μενού, εμφανίζεται το παράθυρο «Στατιστικά Καιρού». Στο παράθυρο εμφανίζονται οι πόλεις για τις οποίες έχει πραγματοποιηθεί αναζήτηση, και το πόσες φορές έχουν αναζητηθεί.    Με την επιλογή «Εξαγωγή σε PDF», δημιουργείται στο φάκελο του project το αντίστοιχο έγγραφο PDF που περιέχει τα στατιστικά, και εμφανίζεται σχετικό μήνυμα. |

# ΚΡΙΤΙΚΟΣ ΑΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ΕΡΓΟΥ

Μια από τις πιο σημαντικές δυσκολίες που αντιμετωπίσαμε ήταν η αρχική εγκατάσταση της Derby DB. Τα βήματα που περιγράφονταν στις παρεχόμενες οδηγίες, δούλευαν για κάποια μέλη της ομάδας ενώ για άλλα όχι, και επειδή ήταν προσανατολισμένες σε Ant projects, ήταν δυσκολότερο να αναζητήσουμε πληροφορίες από τρίτες πηγές, γιατί η πλέον διαδεδομένη τεκμηρίωση αφορά κυρίως σε Maven projects. Τελικά καταφέραμε να εξασφαλίσουμε επιτυχή εγκατάσταση και στα τρία μέλη της ομάδας αλλά αυτό περιόρισε σημαντικά το διαθέσιμο χρόνο υλοποίησης.

Μια από τις αστοχίες μας ήταν η μη ενσωμάτωση του εργαλείου Trello στην διαδικασία. Η άμεση καθημερινή επικοινωνία μας φάνηκε πιο ευέλικτη και δεν καταφέραμε να το χρησιμοποιήσουμε ως σημείο αναφοράς και διαχείρισης.

Σε ένα επόμενο αντίστοιχο έργο θα εστιάζαμε στην καλύτερη διαχείριση του χρόνου, ώστε να είχαμε τη δυνατότητα να εστιάσουμε σε λεπτομέρειες που θα έκαναν την εφαρμογή ακόμη πιο λειτουργική και καλαίσθητη.

Ωστόσο, ανάμεσα στις προκλήσεις αυτές, προέκυψαν αρκετά θετικά στοιχεία. Μέσω του έργου δημιουργήθηκε μια σχέση συνεργασίας και φιλίας μεταξύ μας. Η από κοινού προσπάθεια και η ανοιχτή επικοινωνία μας επέτρεψε να αντιμετωπίσουμε τα εμπόδια. Παρά τις τεχνικές δυσκολίες και τις προκλήσεις, ο κοινός ενθουσιασμός και το πάθος για το έργο παρέμειναν ακλόνητα, κινητοποιώντας μας να ξεπεράσουμε τις δυσκολίες και να παράγουμε αποτελέσματα.

# VIDEO

Το video, όπως και ολόκληρο το project μαζί με όλα τα σχετικά αρχεία, βρίσκεται στο Google Drive:

[**https://drive.google.com/drive/folders/1bu1UPhlgeaAesG6zQ43LqoAGHgvX3XvE?usp=drive\_link**](https://drive.google.com/drive/folders/1bu1UPhlgeaAesG6zQ43LqoAGHgvX3XvE?usp=drive_link)

# ΑΝΑΦΟΡΕΣ

1. Deitel, Paul, και Harvey Deitel. 2015. Java Προγραμματισμός. Δέκατη Έκδοση. Αθήνα: Εκδόσεις: Μ. Γκιούρδας.
2. Φιτσιλής, Παναγιώτης. “Panos Fitsilis- YouTube.” [www.youtube.com/@panosfitsilis6349](http://www.youtube.com/@panosfitsilis6349).
3. W3schools. “Java Tutorial.” [www.w3schools.com/java/default.asp](http://www.w3schools.com/java/default.asp)
4. Stack Overflow. [www.stackoverflow.com](http://www.stackoverflow.com)
5. Apache NetBeans. “Working with the Java DB (Derby) Database” [https://netbeans.apache.org/tutorial/main/kb/docs/ide/java-db](https://netbeans.apache.org/tutorial/main/kb/docs/ide/java-db/)
6. Geeks for Geeks. <https://www.geeksforgeeks.org>
7. ChatGPT. <https://chat.openai.com>