**Αναφορά Έργου**

**Εφαρμογή Προβολής Καιρικών Δεδομένων**

**ΕΑΠ – ΠΛΗ 24**

**3η ΓΡΑΠΤΗ ΕΡΓΑΣΙΑ 2023-2024**

|  |  |  |  |
| --- | --- | --- | --- |
| **Νο** | **Ονοματεπώνυμο** | | **Α.Μ.** |
| 1 | ΣΟΦΙΑΝΟΠΟΥΛΟΣ ΠΑΝΑΓΙΩΤΗΣ | | 156606 |
| 2 |  | |  |
| 3 |  | |  |
| **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.*

Χ Συμφωνώ και αποδέχομαι την ανωτέρω δήλωση

χ

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: 13/03/2024  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 ΕΙΣΑΓΩΓΗ 4](#_Toc156823891)

[2 ΔΙΑΧΕΙΡΙΣΗ ΕΡΓΟΥ 5](#_Toc156823892)

[2.1 Υπολογισμός της απαιτούμενης προσπάθειας ανά απαίτηση 5](#_Toc156823893)

[2.2 Υπολογισμός των προτεραιοτήτων 5](#_Toc156823894)

[2.3 To product backlog 5](#_Toc156823895)

[2.4 Οργάνωση ομάδος και αναθέσεις αρμοδιοτήτων 5](#_Toc156823896)

[2.5 Παρακολούθηση της προσπάθειας κατά τη διάρκεια του έργου 5](#_Toc156823897)

[2.6 Χρήση εργαλείου trello 6](#_Toc156823898)

[3 ΔΡΑΣΤΗΡΙΟΤΗΤΑ 2 - ΥΛΟΠΟΙΗΣΗ ΕΡΓΟΥ 7](#_Toc156823899)

[*3.1* Ερώτημα Α– Διάγραμμα Κλάσεων και Υλοποίηση Κλάσεων σε Java 7](#_Toc156823900)

[3.2 Ερώτημα Β – Δημιουργία GUI Εφαρμογής 7](#_Toc156823901)

[3.3 Ερώτημα Γ – Παρουσίαση στατιστικών στοιχείων καιρού 7](#_Toc156823902)

[3.4 Ερώτημα Δ – Συνολικός Έλεγχος και Εκτέλεση της Εφαρμογής 7](#_Toc156823903)

[4 ΚΡΙΤΙΚΟΣ ΑΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ΕΡΓΟΥ 8](#_Toc156823904)

[5 VIDEO 8](#_Toc156823905)

[6 ΑΝΑΦΟΡΕΣ 9](#_Toc156823906)

# ΕΙΣΑΓΩΓΗ

*[Δώστε μια σύντομη περιγραφή του συστήματος. Παρουσιάστε τους βασικούς στόχους του έργου, καθώς και τις παραδοχές σας. Αναφέρετε τα χαρακτηριστικά και τις λειτουργίες που πρέπει να επιτελεί το έργο ώστε να ανταποκρίνεται στις απαιτήσεις και τις ανάγκες του πελάτη. Αναφέρετε με ποιόν/ποιους τρόπο/τρόπους το έργο ικανοποιεί αυτές τις απαιτήσεις.]*

Με την παρούσα ομαδική εργασία 3η κατα σειρά στο ακαδημαικό έτος 2023-2024 για την Θ.Ε. της ΠΛΗ 24 συνέρχεται μία ομάδα ατόμων με σκοπό την δημιουργία της εφαρμογής WeatherApp που επεξεργάζεται καιρικά δεδομένα που λαμβάνει από την υπηρεσία ιστού <https://wttr.in/>

* 1. **Περιγραφή της εφαρμογής WeatherApp**

Η εφαργμοή αντλεί δεδομένα απ’ την υπηρεσία ιστου wttr.in η οποία παρέχει τις υπηρεσίες μέσω web services τύπου REST ώστε να χρησιμοποιούνται ευρέως. Τα δεδομένα επιστρέφονται σε μορφή JSON. Η εφαρμογή WeatherApp επεξεργάζεται αυτά τα δεδομένα για την παροχή ανάλογων υπηρεσιών προς τους χρήστες της όπως:

1. Προβολή καιρικών δεδομένων ανα πόλη (θερμοκρασία, υγρασία, ταχύτητα ανέμου, δείκτης UV, λεκτική πρόγνωση καιρού).
   1. Επιλογή συγκεκριμένων δεδομένων προς άντληση
   2. Επεξεργασία, αποθήκευση δεδομένων στην βάση
   3. Διαγραφή δεδομένων από την βάση
2. Προβολή λίστας πόλεων για τις οποίες έχει γίνει αναζήτηση καιρικών δεδομένων.
3. Προβολή λίστας ημερομηνιών για πόλη για την οποία έχει γίνει αναζήτηση.
4. Προβολή στατιστικών δεδομένων πόλεων και εκτύπωση σε αρχείο pdf
5. Έξοδος
   1. **Στόχος του Έργου**

Ο στόχος του έργου είναι η δημιουργία μίας εύχρηστης εφαρμογής που μέσω της γραφικής της διεπαφής θα επιτρέπει στον χρήστη να αναζητεί τα καιρικά δεδομένα για την πόλη που ενδιαφέρεται, να τα επεξεργάζεται να τα αποθηκεύει αλλά και να παρακολουθεί στατιστικά των δεδομένων αυτών βάσει των αναζητήσεων του.

* 1. **Χαρακτηριστικά και λειτουργίες της εφαρμογής WeatherApp**

**Προδιαγραφή R1: Βασικό menu**Κατά την έναρξη της εφαρμογής εμφανίζεται το μενού επιλογών κεντραρισμένο στο κύριο παράθυρο. Οι επιλογές του είναι σε μορφή κουμπιών.  
  
1. Προβολή καιρικών δεδομένων ανά πόλη  
2. Προβολή λίστας πόλεων και ημερομηνιών  
3. Προβολή στατιστικών ανά πόλη  
4. Exit

**Προδιαγραφή R2: Προβολή δεδομένων πόλης**

Με την επιλογή αυτή, πρώτη κατά σειρά στο βασικό μενού εμφανίζεται ένα καινούργιο παράθυρο με τίτλο «Προβολή καιρικών δεδομένων ανά πόλη» και τα αντίστοιχα πεδία πόλη, θερμοκρασία, υγρασία, ταχύτητα ανέμου, UV index, weather description, date και προβολές(πόσες φορές έχει προβληθεί η πόλη που αναζητούμε). Δίπλα στο πεδίο της πόλης υπάρχει το κουμπί “search” για να αναζητήσουμε την συγκεκριμένη πόλη στο wttr.in αφού την εισάγουμε στο πεδίο. Τέλος υπάρχουν σε μορφή κουμπιών οι επιλογές «αποθήκευση δεδομένων» για να αποθηκεύσουμε τα δεδομένα που αντλησαμε από την αναζήτηση, «διαγραφή δεδομένων» για να διαγράψουμε τα δεδομένα και «πίσω στο μενού» για την επιστροφή στο Βασικό μενού.

**Προδιαγραφή R3: Προβολή λίστας πόλεων και ημερομηνιών**Στην επιλογή αυτή, ανοίγουμε παράθυρο μέσα στο οποίο εμφανίζεται λίστα με τις πόλεις που έχουμε αναζητήσει καθώς και οι ημερομηνίες που έγιναν οι αναζητήσεις αυτές. Στην οθόνη αυτή ο χρήστης επιλέγει μία πόλη από την λίστα για να εμφανίσει το ιστορικό αναζητήσεων και τροποποίησεις δεδομένων για την συγκεκριμένη πόλη.

**Προδιαγραφή R4: Προβολή στατιστικών ανά πόλη**Στην επιλογή αυτή διατρέχουμε την Βάση Δεδομένων για να βρούμε τις αναζητήσεις που έχει κάνει ο χρήστης ως προς τις πόλεις και τις εμφανίζουμε σε πίνακα με φθίνουσα κατάταξη από την πιο δημοφιλή πόλη των αναζητήσεων έως την λιγότερο δημοφιλή. Στην συνέχεια ο χρήστης έχει την επιλογή να εκτυπώσει τον παραγόμενο πίνακα σε αρχεία PDF.

## Πληροφορίες για την υλοποίηση της εφαρμογής WeatherApp

Η ομάδα που εκπόνησε την εργασία αποτελείται από δύο άτομα:  
  
1. Σοφιανόπουλος Παναγιώτης (ΑΜ: )  
2. Τζανής Ιωάννης (ΑΜ: )

Με επιβλέπων καθηγητή τον κ. Φιτσιλή Παναγιώτη.

Για την λειτουργία της εφαρμογής απαιτείται

* Πρόσβαση στο internet
* χρήση των απαιτούμενων βιβλιοθηκών που δόθηκαν από το ίδρυμα στο Apache Netbeans
* η εφαρμογή Apache Derby για την υλοποίηση και χρήση της ΒΔ

Η εφαρμογή υλοποιήθηκε σε γλώσσα Java με τα παρακάτω εργαλεία:¨

* Java jdk 17
* Περιβάλλον ανάπτυξης Java apache Netbeans
* Visual paradigm για τον σχεδιασμό των κλάσσεων και την παραγωγή SQL
* Planning poker για τον υπολογισμό του φόρτου εργασίας καθώς και τον επιμερισμό των εργασιών
* Μέθοδος Scrum για την βέλτιση οργάνωση και υλοποίηση του project
* Github
* Microsoft office/excel για την συγγραφή της εργασίας
* Discord σαν μέσο επικοινωνίας μεταξύ των μελών της ομάδας

Για την κατασκευή του βίντεο χρησιμοποιήθηκε το OBS.

# ΔΙΑΧΕΙΡΙΣΗ ΕΡΓΟΥ

## Υπολογισμός της απαιτούμενης προσπάθειας ανά απαίτηση

Χρησιμοποιήσαμε την μέθοδο Planning poker μέσω του planningpoker.com για τον υπολογισμό της εκτιμώμενης προσπάθειας κάθε απαίτησης. Συγκεκριμένα οργανώθηκε παιχνίδι που περιλαμβάνει 20 stories τα οποία αντιπροσωπεύουν το συνολικό έργο που κληθήκαμε να υλοποιήσουμε. Για την βαθμολόγηση των ιστοριών επιλέξαμε το σετ «powers of 2» όπου κάθε αριθμός αντανακλά την απαιτούμενη προσπάθεια σε ώρες. Καθ’όλη την διάρκεια του παιχνιδιού υπήρχε επικοινωνία με σκοπό την καλύτερη αντιστοίχιση των ιστοριών σε ώρες.

A screenshot of a computer

Description automatically generated

Εικόνα 1. Οργάνωση του planningpoker



Εικόνα 2. Στιγμιότυπο από την διαδικασία του παιχνιδιού

A screenshot of a computer

Description automatically generated

Εικόνα 3. Τελικό αποτέλεσμα του παιχνιδιού

Βαθμολογήσαμε το έργο με 155 πόντους συνολικά. Θεωρήσαμε ότι κάθε ανθρωποημέρα περιλαμβάνει περίπου 8 ώρες εργασίας άρα για την υλοποίηση του έργου απαιτούνται 155/8 = 19,375 ανθρωποημέρες συνολικά, περίπου 9,5 μέρες για κάθε μέλος δηλαδή.

## Υπολογισμός των προτεραιοτήτων

Με την χρήση του planningpoker εκτός από την απαιτούμενη προσπάθεια κάναμε και την προτεραιοποίηση των απαιτήσεων του έργου. Οργανώσαμε παιχνίδι με τις ίδιες ιστορίες και αυτή την φορά επιλέξαμε το σετ «T-shirts» με μονάδες μέτρησης small, medium, large κλπ. Για μεγαλύτερη προτεραιότητα των απαιτήσεων επιλέγαμε μεγαλύτερα μεγέθη και αντίστοιχα για μικρότερες απαιτήσεις, μικρότερα.

A screenshot of a computer

Description automatically generated

Εικόνα 4. Προτεραιότητες μετά το τέλος του παιχνιδιού.

Με βάση το αποτέλεσμα της μεθόδου Planningpoker για την απαιτούμενη προσπάθεια και τις προτεραιότητες που πρέπει να θέσουμε καταλήξαμε στο παρακάτω χρονοδιάγραμμα για τα Sprints.

|  |
| --- |
| Sprint 1. 06-10/03  Ανάλυση απαιτήσεων, ανάθεση αρμοδιοτήτων/προγραμματισμός sprint, Σχεδίαση ΒΔ/Pojo κλάσεων σχεδιασμός GUI και υλοποίηση απαιτήσεων R1,R2,R3,R4 |
| Sprint 2. 11-12/03  Ελέγχος των απαιτήσεων, δημιουργία διαγράμματος κλάσεων, βελτίωση κώδικα, δημιουργία product backlog |
| Sprint 3. 13/03  Τελικός έλεγχος εφαρμογής, δημιουργία εγχειριδίου, συγγραφή της εργασίας, δημιουργία βίντεο |

## To product backlog

Το product backlog είναι εργαλείο της μεθοδολογίας Agile στα πλαίσια του Scrum. Αναφέρεται στην λίστα των απαιτήσεων και της λειτουργικότητας που παρέχει το προιόν στον πελάτη. Αυτές οι απαιτήσεις ονομάζονται PBIs – product back log Items.

Το product backlog συνήθως δημιουργεί ο product owner και είναι μία συλλογή των ιστοριών, των προδιαγραφών και όποιων άλλων πληροφοριών αφορούν το προιόν που διατίθεται. Το product backlog τροποποιείται κατάλληλα κατά την υλοποίηση του project και η λίστα των προτεραιότητων ταξινομείτε με τρόπο τέτοιο ώστε να υλοποιούνται πρώτα τα σημαντικότερα χαρακτηριστικά.  
  
A screenshot of a computer screen

Description automatically generated

Εικόνα 5. sprint 1

A screenshot of a computer

Description automatically generated

Εικόνα 6.sprint 2

A screenshot of a computer screen

Description automatically generated

Εικόνα 7. sprint 3

## Οργάνωση ομάδος και αναθέσεις αρμοδιοτήτων

*[Αναφέρετε την οργάνωση της ομάδας του έργου, καθώς και τους ρόλους των μελών της ομάδας. Πιο συγκεκριμένα:*

* *Ποιος ήταν ο ρόλος του κάθε μέλους της ομάδας;*
* *Ποιος ήταν ο υπεύθυνος για την υλοποίηση της κάθε απαίτησης;*
* *Αναφέρετε τον τρόπο επικοινωνίας των μελών της ομάδας, τη διαχείριση διαφορών μεταξύ των μελών, κ.λπ. και ποιες από τις πρακτικές του SCRUM χρησιμοποιήσατε;*

*Εάν δεν έχετε δώσει απάντηση γράψτε με κεφαλαία γράμματα, ΔΕΝ ΑΠΑΝΤΗΘΗΚΕ. Εάν εν γνώση σας δίνετε ελλιπή απάντηση γράψτε με κεφαλαία γράμματα, ΕΛΛΙΠΗΣ ΑΠΑΝΤΗΣΗ]*

**Product Owner**

Είναι ο υπέυθυνος του έργου. Διαχειρίζεται και ελέγχει το product backlog δηλαδή τις απαιτήσεις του project. Φροντίζει για την έγκαιρη υλοποίηση της εφαρμογής παρέχοντας την κατάλληλη τεχνοτροπία, τα εργαλεία που πρέπει να χρησιμοποιηθούν, την χρονική εκτίμηση των απαιτήσεων, δίνοντας όλα τα παραπάνω με τρόπο ουσιαστικό και κατανοητό στην ομάδα ανάπτυκης. Τον ρόλο του product owner στο Project έχει ο καθηγητής κ. Π. Φιτσιλής.

**Scrum Master**

Φροντίζει ώστε η μέθοδος scrum να χρησιμοποιηθεί σωστά κατα την διάρκεια υλοποίησης του έργου. Έχει ρόλο επιτηρητή της πορείας του project και στόχος του είναι η επιτυχής ολοκλήρωση του. Παρέχει βοηθητική εργασία σε ότι αφορά την κατανόηση των απαιτήσεων του project προς την ομάδα ανάπτυξης, επικοινωνεί με τον Product owner προς επίλυση των διαφόρων προβλημάτων και ακόμα αναλαμβάνει να διατηρήσει/μεγιστοποιήσει την παραγωγικότητα της ομάδας ανάπτυξης για την οποία είναι υπέυθυνος.

Η ομάδα μας είναι διμελής οπότε τον ρόλο αυτό αναλάβαμε από κοινού.

**Scrum Team**

Η scrum team δεν είναι παρα η ομάδα ανάπτυξης του έργου. Οφείλει να πράξει ό,τι χρειάζεται ώστε να υλοποιηθεί το project με τον καλύτερο δυνατό τρόπο παίρνοντας τις κατάλληλες αποφάσεις και εκτελώντας τις ανάλογες διεργασίες ώστε στο τέλος κάθε Sprint να επιτυγχάνονται οι ορισμένοι στόχοι. Η scrum team αποτελείται από τους Σοφιανόπουλο Παναγιώτη και Τζανή Ιωάννη.

Οι εργασίες της ομάδας πραγματοποιούνται με την συμμετοχή και παρακολούθηση του συνόλου της κατά την διάρκεια των sprints. Η επικοινωνία της ομάδας έγινε μέσω της εφαρμογής discord καθώς και της εφαρμογής github για την ανταλλαγή υλικού και την ενσωμάτωση των υλοποιήσεων κάθε μεριάς.

Στον λίγο χρόνο που διαθέσαμε - λόγω παραλείψεων από μεριάς μας και έλλειψης επικοινωνίας με τους συναδέλφους μας - για την ολοκλήρωση της εργασίας κάθε ένας απο εμάς έδωσε τα μέγιστα, ωστόσο ο συγγραφέας της παραγράφου αυτής αναγνωρίζει τον σημαίνων ρόλο του Σοφιανόπουλου Παναγιώτη που με την εμπειρία του υλοποίησε ουσιαστικά μερικά από τα δυσκολότερα σημεία.

## Παρακολούθηση της προσπάθειας κατά τη διάρκεια του έργου

*[Παρουσιάστε το χρόνο υλοποίησης ανά απαίτηση/ανά επανάληψη (sprint) σε σύγκριση με αυτά που είχαν προϋπολογιστεί, το διάγραμμα κατανάλωσης προσπάθειας (burnt down chart) ανά sprint, καθώς και την ταχύτητα υλοποίησης (velocity). Εάν δεν έχετε δώσει απάντηση γράψτε με κεφαλαία γράμματα, ΔΕΝ ΑΠΑΝΤΗΘΗΚΕ. Εάν εν γνώση σας δίνετε ελλιπή απάντηση γράψτε με κεφαλαία γράμματα, ΕΛΛΙΠΗΣ ΑΠΑΝΤΗΣΗ]*

**Sprint 1**

A graph on a sheet of paper

Description automatically generated

**Sprint 2**

A screenshot of a spreadsheet

Description automatically generated

**Sprint 3**

A screenshot of a spreadsheet

Description automatically generated

## Χρήση εργαλείου github

Λόγο του ελλειπής χρόνου, καταλήξαμε στο να χρησιμοποιήσουμε αποκλειστικά το github για την συνεργασία μας στον κώδικα και το discord για την εποικηνωνία μεταξύ μας.

Στο παρακάτω link θα βρείτε το project μας καθώς και τα commits που κάναμε κατά την διάρκεια συγγραφής της εργασίας (Παρακάτω εικόνα)

<https://github.com/Jukehere/eap-weather>

A screenshot of a computer

Description automatically generated

# ΔΡΑΣΤΗΡΙΟΤΗΤΑ 2 - ΥΛΟΠΟΙΗΣΗ ΕΡΓΟΥ

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

1. ER

A computer screen shot of a computer

Description automatically generated

Το αρχείο ονομάζεται weatherer.vpp

SQL:

CREATE TABLE City (areaname varchar(255) NOT NULL, views integer NOT NULL, PRIMARY KEY (areaname));

CREATE TABLE Citydata (id integer GENERATED BY DEFAULT AS IDENTITY, weather\_date varchar(255) NOT NULL, C\_temp varchar(255) NOT NULL, humidity varchar(255) NOT NULL, wspeedinkmph varchar(255) NOT NULL, uvindex varchar(255) NOT NULL, weatherdesc varchar(255) NOT NULL, cityname varchar(255) NOT NULL, PRIMARY KEY (id));

ALTER TABLE Citydata ADD CONSTRAINT FKCitydata428547 FOREIGN KEY (cityname) REFERENCES City (areaname);

DB Name: weatherdata

User: weatherdata

Pass: weatherdata

1. EapWeather.java

/\*\*

 \*

 \* @author softa

 \*/

import java.awt.EventQueue;

public class EapWeather {

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

            @Override

            public void run() {

                guir1 frame = new guir1();

                frame.show();

            }

        });

    }

}

1. City.java

package eapweather;

import java.io.Serializable;

import java.util.List;

import javax.persistence.Basic;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.OneToMany;

import javax.persistence.Table;

/\*\*

 \*

 \* @author softa

 \*/

@Entity

@Table(name = "CITY")

@NamedQueries({

    @NamedQuery(name = "City.findAll", query = "SELECT c FROM City c"),

    @NamedQuery(name = "City.findByAreaname", query = "SELECT c FROM City c WHERE c.areaname = :areaname"),

    @NamedQuery(name = "City.findByViews", query = "SELECT c FROM City c WHERE c.views = :views")})

public class City implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id

    @Basic(optional = false)

    @Column(name = "AREANAME")

    private String areaname;

    @Basic(optional = false)

    @Column(name = "VIEWS")

    private int views;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "cityname")

    private List<Citydata> citydataList;

    public City() {

    }

    public City(String areaname) {

        this.areaname = areaname;

    }

    public City(String areaname, int views) {

        this.areaname = areaname;

        this.views = views;

    }

    public String getAreaname() {

        return areaname;

    }

    public void setAreaname(String areaname) {

        this.areaname = areaname;

    }

    public int getViews() {

        return views;

    }

    public void setViews(int views) {

        this.views = views;

    }

    public List<Citydata> getCitydataList() {

        return citydataList;

    }

    public void setCitydataList(List<Citydata> citydataList) {

        this.citydataList = citydataList;

    }

    @Override

    public int hashCode() {

        int hash = 0;

        hash += (areaname != null ? areaname.hashCode() : 0);

        return hash;

    }

    @Override

    public boolean equals(Object object) {

        // TODO: Warning - this method won't work in the case the id fields are not set

        if (!(object instanceof City)) {

            return false;

        }

        City other = (City) object;

        if ((this.areaname == null && other.areaname != null) || (this.areaname != null && !this.areaname.equals(other.areaname))) {

            return false;

        }

        return true;

    }

    @Override

    public String toString() {

        return "eapweather.City[ areaname=" + areaname + " ]";

    }

}

Citydata.java

package eapweather;

import java.io.Serializable;

import java.util.Date;

import javax.persistence.Basic;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

import javax.persistence.Temporal;

import javax.persistence.TemporalType;

/\*\*

 \*

 \* @author softa

 \*/

@Entity

@Table(name = "CITYDATA")

@NamedQueries({

    @NamedQuery(name = "Citydata.findAll", query = "SELECT c FROM Citydata c"),

    @NamedQuery(name = "Citydata.findById", query = "SELECT c FROM Citydata c WHERE c.id = :id"),

    @NamedQuery(name = "Citydata.findByDate", query = "SELECT c FROM Citydata c WHERE c.date = :date"),

    @NamedQuery(name = "Citydata.findByCTemp", query = "SELECT c FROM Citydata c WHERE c.cTemp = :cTemp"),

    @NamedQuery(name = "Citydata.findByHumidity", query = "SELECT c FROM Citydata c WHERE c.humidity = :humidity"),

    @NamedQuery(name = "Citydata.findByWspeedinkmph", query = "SELECT c FROM Citydata c WHERE c.wspeedinkmph = :wspeedinkmph"),

    @NamedQuery(name = "Citydata.findByUvindex", query = "SELECT c FROM Citydata c WHERE c.uvindex = :uvindex"),

    @NamedQuery(name = "Citydata.findByWeatherdesc", query = "SELECT c FROM Citydata c WHERE c.weatherdesc = :weatherdesc")})

public class Citydata implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Basic(optional = false)

    @Column(name = "ID")

    private Integer id;

    @Basic(optional = false)

    @Column(name = "date")

    @Temporal(TemporalType.DATE)

    private Date date;

    @Basic(optional = false)

    @Column(name = "C\_TEMP")

    private int cTemp;

    @Basic(optional = false)

    @Column(name = "HUMIDITY")

    private int humidity;

    @Basic(optional = false)

    @Column(name = "WSPEEDINKMPH")

    private int wspeedinkmph;

    @Basic(optional = false)

    @Column(name = "UVINDEX")

    private int uvindex;

    @Basic(optional = false)

    @Column(name = "WEATHERDESC")

    private String weatherdesc;

    @JoinColumn(name = "CITYNAME", referencedColumnName = "AREANAME")

    @ManyToOne(optional = false)

    private City cityname;

    public Citydata() {

    }

    public Citydata(Integer id) {

        this.id = id;

    }

    public Citydata(Integer id, Date date, int cTemp, int humidity, int wspeedinkmph, int uvindex, String weatherdesc) {

        this.id = id;

        this.date = date;

        this.cTemp = cTemp;

        this.humidity = humidity;

        this.wspeedinkmph = wspeedinkmph;

        this.uvindex = uvindex;

        this.weatherdesc = weatherdesc;

    }

    public Integer getId() {

        return id;

    }

    public void setId(Integer id) {

        this.id = id;

    }

    public Date getDate() {

        return date;

    }

    public void setDate(Date date) {

        this.date = date;

    }

    public int getCTemp() {

        return cTemp;

    }

    public void setCTemp(int cTemp) {

        this.cTemp = cTemp;

    }

    public int getHumidity() {

        return humidity;

    }

    public void setHumidity(int humidity) {

        this.humidity = humidity;

    }

    public int getWspeedinkmph() {

        return wspeedinkmph;

    }

    public void setWspeedinkmph(int wspeedinkmph) {

        this.wspeedinkmph = wspeedinkmph;

    }

    public int getUvindex() {

        return uvindex;

    }

    public void setUvindex(int uvindex) {

        this.uvindex = uvindex;

    }

    public String getWeatherdesc() {

        return weatherdesc;

    }

    public void setWeatherdesc(String weatherdesc) {

        this.weatherdesc = weatherdesc;

    }

    public City getCityname() {

        return cityname;

    }

    public void setCityname(City cityname) {

        this.cityname = cityname;

    }

    @Override

    public int hashCode() {

        int hash = 0;

        hash += (id != null ? id.hashCode() : 0);

        return hash;

    }

    @Override

    public boolean equals(Object object) {

        // TODO: Warning - this method won't work in the case the id fields are not set

        if (!(object instanceof Citydata)) {

            return false;

        }

        Citydata other = (Citydata) object;

        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {

            return false;

        }

        return true;

    }

    @Override

    public String toString() {

        return "eapweather.Citydata[ id=" + id + " ]";

    }

}

CityJpaController.java

package eapweather;

import eapweather.exceptions.IllegalOrphanException;

import eapweather.exceptions.NonexistentEntityException;

import eapweather.exceptions.PreexistingEntityException;

import java.io.Serializable;

import javax.persistence.Query;

import javax.persistence.EntityNotFoundException;

import javax.persistence.criteria.CriteriaQuery;

import javax.persistence.criteria.Root;

import java.util.ArrayList;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

/\*\*

 \*

 \* @author softa

 \*/

public class CityJpaController implements Serializable {

    public CityJpaController(EntityManagerFactory emf) {

        this.emf = emf;

    }

    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {

        return emf.createEntityManager();

    }

    public void create(City city) throws PreexistingEntityException, Exception {

        if (city.getCitydataList() == null) {

            city.setCitydataList(new ArrayList<Citydata>());

        }

        EntityManager em = null;

        try {

            em = getEntityManager();

            em.getTransaction().begin();

            List<Citydata> attachedCitydataList = new ArrayList<Citydata>();

            for (Citydata citydataListCitydataToAttach : city.getCitydataList()) {

                citydataListCitydataToAttach = em.getReference(citydataListCitydataToAttach.getClass(), citydataListCitydataToAttach.getId());

                attachedCitydataList.add(citydataListCitydataToAttach);

            }

            city.setCitydataList(attachedCitydataList);

            em.persist(city);

            for (Citydata citydataListCitydata : city.getCitydataList()) {

                City oldCitynameOfCitydataListCitydata = citydataListCitydata.getCityname();

                citydataListCitydata.setCityname(city);

                citydataListCitydata = em.merge(citydataListCitydata);

                if (oldCitynameOfCitydataListCitydata != null) {

                    oldCitynameOfCitydataListCitydata.getCitydataList().remove(citydataListCitydata);

                    oldCitynameOfCitydataListCitydata = em.merge(oldCitynameOfCitydataListCitydata);

                }

            }

            em.getTransaction().commit();

        } catch (Exception ex) {

            if (findCity(city.getAreaname()) != null) {

                throw new PreexistingEntityException("City " + city + " already exists.", ex);

            }

            throw ex;

        } finally {

            if (em != null) {

                em.close();

            }

        }

    }

    public void edit(City city) throws IllegalOrphanException, NonexistentEntityException, Exception {

        EntityManager em = null;

        try {

            em = getEntityManager();

            em.getTransaction().begin();

            City persistentCity = em.find(City.class, city.getAreaname());

            List<Citydata> citydataListOld = persistentCity.getCitydataList();

            List<Citydata> citydataListNew = city.getCitydataList();

            List<String> illegalOrphanMessages = null;

            for (Citydata citydataListOldCitydata : citydataListOld) {

                if (!citydataListNew.contains(citydataListOldCitydata)) {

                    if (illegalOrphanMessages == null) {

                        illegalOrphanMessages = new ArrayList<String>();

                    }

                    illegalOrphanMessages.add("You must retain Citydata " + citydataListOldCitydata + " since its cityname field is not nullable.");

                }

            }

            if (illegalOrphanMessages != null) {

                throw new IllegalOrphanException(illegalOrphanMessages);

            }

            List<Citydata> attachedCitydataListNew = new ArrayList<Citydata>();

            for (Citydata citydataListNewCitydataToAttach : citydataListNew) {

                citydataListNewCitydataToAttach = em.getReference(citydataListNewCitydataToAttach.getClass(), citydataListNewCitydataToAttach.getId());

                attachedCitydataListNew.add(citydataListNewCitydataToAttach);

            }

            citydataListNew = attachedCitydataListNew;

            city.setCitydataList(citydataListNew);

            city = em.merge(city);

            for (Citydata citydataListNewCitydata : citydataListNew) {

                if (!citydataListOld.contains(citydataListNewCitydata)) {

                    City oldCitynameOfCitydataListNewCitydata = citydataListNewCitydata.getCityname();

                    citydataListNewCitydata.setCityname(city);

                    citydataListNewCitydata = em.merge(citydataListNewCitydata);

                    if (oldCitynameOfCitydataListNewCitydata != null && !oldCitynameOfCitydataListNewCitydata.equals(city)) {

                        oldCitynameOfCitydataListNewCitydata.getCitydataList().remove(citydataListNewCitydata);

                        oldCitynameOfCitydataListNewCitydata = em.merge(oldCitynameOfCitydataListNewCitydata);

                    }

                }

            }

            em.getTransaction().commit();

        } catch (Exception ex) {

            String msg = ex.getLocalizedMessage();

            if (msg == null || msg.length() == 0) {

                String id = city.getAreaname();

                if (findCity(id) == null) {

                    throw new NonexistentEntityException("The city with id " + id + " no longer exists.");

                }

            }

            throw ex;

        } finally {

            if (em != null) {

                em.close();

            }

        }

    }

    public void destroy(String id) throws IllegalOrphanException, NonexistentEntityException {

        EntityManager em = null;

        try {

            em = getEntityManager();

            em.getTransaction().begin();

            City city;

            try {

                city = em.getReference(City.class, id);

                city.getAreaname();

            } catch (EntityNotFoundException enfe) {

                throw new NonexistentEntityException("The city with id " + id + " no longer exists.", enfe);

            }

            List<String> illegalOrphanMessages = null;

            List<Citydata> citydataListOrphanCheck = city.getCitydataList();

            for (Citydata citydataListOrphanCheckCitydata : citydataListOrphanCheck) {

                if (illegalOrphanMessages == null) {

                    illegalOrphanMessages = new ArrayList<String>();

                }

                illegalOrphanMessages.add("This City (" + city + ") cannot be destroyed since the Citydata " + citydataListOrphanCheckCitydata + " in its citydataList field has a non-nullable cityname field.");

            }

            if (illegalOrphanMessages != null) {

                throw new IllegalOrphanException(illegalOrphanMessages);

            }

            em.remove(city);

            em.getTransaction().commit();

        } finally {

            if (em != null) {

                em.close();

            }

        }

    }

    public List<City> findCityEntities() {

        return findCityEntities(true, -1, -1);

    }

    public List<City> findCityEntities(int maxResults, int firstResult) {

        return findCityEntities(false, maxResults, firstResult);

    }

    private List<City> findCityEntities(boolean all, int maxResults, int firstResult) {

        EntityManager em = getEntityManager();

        try {

            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();

            cq.select(cq.from(City.class));

            Query q = em.createQuery(cq);

            if (!all) {

                q.setMaxResults(maxResults);

                q.setFirstResult(firstResult);

            }

            return q.getResultList();

        } finally {

            em.close();

        }

    }

    public City findCity(String id) {

        EntityManager em = getEntityManager();

        try {

            return em.find(City.class, id);

        } finally {

            em.close();

        }

    }

    public int getCityCount() {

        EntityManager em = getEntityManager();

        try {

            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();

            Root<City> rt = cq.from(City.class);

            cq.select(em.getCriteriaBuilder().count(rt));

            Query q = em.createQuery(cq);

            return ((Long) q.getSingleResult()).intValue();

        } finally {

            em.close();

        }

    }

}

CitydataJpaController.java

package eapweather;

import eapweather.exceptions.NonexistentEntityException;

import java.io.Serializable;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Query;

import javax.persistence.EntityNotFoundException;

import javax.persistence.criteria.CriteriaQuery;

import javax.persistence.criteria.Root;

/\*\*

 \*

 \* @author softa

 \*/

public class CitydataJpaController implements Serializable {

    public CitydataJpaController(EntityManagerFactory emf) {

        this.emf = emf;

    }

    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {

        return emf.createEntityManager();

    }

    public void create(Citydata citydata) {

        EntityManager em = null;

        try {

            em = getEntityManager();

            em.getTransaction().begin();

            City cityname = citydata.getCityname();

            if (cityname != null) {

                cityname = em.getReference(cityname.getClass(), cityname.getAreaname());

                citydata.setCityname(cityname);

            }

            em.persist(citydata);

            if (cityname != null) {

                cityname.getCitydataList().add(citydata);

                cityname = em.merge(cityname);

            }

            em.getTransaction().commit();

        } finally {

            if (em != null) {

                em.close();

            }

        }

    }

    public void edit(Citydata citydata) throws NonexistentEntityException, Exception {

        EntityManager em = null;

        try {

            em = getEntityManager();

            em.getTransaction().begin();

            Citydata persistentCitydata = em.find(Citydata.class, citydata.getId());

            City citynameOld = persistentCitydata.getCityname();

            City citynameNew = citydata.getCityname();

            if (citynameNew != null) {

                citynameNew = em.getReference(citynameNew.getClass(), citynameNew.getAreaname());

                citydata.setCityname(citynameNew);

            }

            citydata = em.merge(citydata);

            if (citynameOld != null && !citynameOld.equals(citynameNew)) {

                citynameOld.getCitydataList().remove(citydata);

                citynameOld = em.merge(citynameOld);

            }

            if (citynameNew != null && !citynameNew.equals(citynameOld)) {

                citynameNew.getCitydataList().add(citydata);

                citynameNew = em.merge(citynameNew);

            }

            em.getTransaction().commit();

        } catch (Exception ex) {

            String msg = ex.getLocalizedMessage();

            if (msg == null || msg.length() == 0) {

                Integer id = citydata.getId();

                if (findCitydata(id) == null) {

                    throw new NonexistentEntityException("The citydata with id " + id + " no longer exists.");

                }

            }

            throw ex;

        } finally {

            if (em != null) {

                em.close();

            }

        }

    }

    public void destroy(Integer id) throws NonexistentEntityException {

        EntityManager em = null;

        try {

            em = getEntityManager();

            em.getTransaction().begin();

            Citydata citydata;

            try {

                citydata = em.getReference(Citydata.class, id);

                citydata.getId();

            } catch (EntityNotFoundException enfe) {

                throw new NonexistentEntityException("The citydata with id " + id + " no longer exists.", enfe);

            }

            City cityname = citydata.getCityname();

            if (cityname != null) {

                cityname.getCitydataList().remove(citydata);

                cityname = em.merge(cityname);

            }

            em.remove(citydata);

            em.getTransaction().commit();

        } finally {

            if (em != null) {

                em.close();

            }

        }

    }

    public List<Citydata> findCitydataEntities() {

        return findCitydataEntities(true, -1, -1);

    }

    public List<Citydata> findCitydataEntities(int maxResults, int firstResult) {

        return findCitydataEntities(false, maxResults, firstResult);

    }

    private List<Citydata> findCitydataEntities(boolean all, int maxResults, int firstResult) {

        EntityManager em = getEntityManager();

        try {

            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();

            cq.select(cq.from(Citydata.class));

            Query q = em.createQuery(cq);

            if (!all) {

                q.setMaxResults(maxResults);

                q.setFirstResult(firstResult);

            }

            return q.getResultList();

        } finally {

            em.close();

        }

    }

    public Citydata findCitydata(Integer id) {

        EntityManager em = getEntityManager();

        try {

            return em.find(Citydata.class, id);

        } finally {

            em.close();

        }

    }

    public int getCitydataCount() {

        EntityManager em = getEntityManager();

        try {

            CriteriaQuery cq = em.getCriteriaBuilder().createQuery();

            Root<Citydata> rt = cq.from(Citydata.class);

            cq.select(em.getCriteriaBuilder().count(rt));

            Query q = em.createQuery(cq);

            return ((Long) q.getSingleResult()).intValue();

        } finally {

            em.close();

        }

    }

}

## Ερώτημα Β – Δημιουργία GUI Εφαρμογής

*[Εισάγετε τον κώδικα java για τις σχετικές κλάσεις. Θα πρέπει να εισάγεται ΜΟΝΟ τον κώδικα που έχετε γράψει εσείς και όχι αυτόν που παράγεται αυτόματα προσδιορίζοντας ευκρινώς το όνομα της κλάσης και της μεθόδου Φροντίστε ο κώδικας να είναι μορφοποιημένος κατάλληλα και να είναι ευανάγνωστος. Θα πρέπει να υπάρχει τεκμηρίωση με μορφή σχολίων. Εάν δεν έχετε δώσει απάντηση γράψτε με κεφαλαία γράμματα, ΔΕΝ ΑΠΑΝΤΗΘΗΚΕ. Εάν εν γνώση σας δίνετε ελλιπή απάντηση γράψτε με κεφαλαία γράμματα, ΕΛΛΙΠΗΣ ΑΠΑΝΤΗΣΗ.]*

## Ερώτημα Γ – Παρουσίαση στατιστικών στοιχείων καιρού

*[Εισάγετε τον κώδικα java που αναφέρεται στην παρουσίαση των στατιστικών στοιχείων. Θα πρέπει να προσδιορίζεται ευκρινώς το όνομα της κλάσης και της μεθόδου. Φροντίστε ο κώδικας να είναι μορφοποιημένος κατάλληλα και να είναι ευανάγνωστος. Θα πρέπει να υπάρχει τεκμηρίωση με μορφή σχολίων. Εάν δεν έχετε δώσει απάντηση γράψτε με κεφαλαία γράμματα, ΔΕΝ ΑΠΑΝΤΗΘΗΚΕ. Εάν εν γνώση σας δίνετε ελλιπή απάντηση γράψτε με κεφαλαία γράμματα, ΕΛΛΙΠΗΣ ΑΠΑΝΤΗΣΗ.]*

## Ερώτημα Δ – Συνολικός Έλεγχος και Εκτέλεση της Εφαρμογής

*[Να εκτελέσετε την εφαρμογή και να εισαγάγετε σε μορφή πίνακα εικόνες (screendumps), όπου θα φαίνεται το αποτέλεσμα της εκτέλεσης. Εάν δεν έχετε δώσει απάντηση γράψτε με κεφαλαία γράμματα, ΔΕΝ ΑΠΑΝΤΗΘΗΚΕ. Εάν εν γνώση σας δίνετε ελλιπή απάντηση γράψτε με κεφαλαία γράμματα, ΕΛΛΙΠΗΣ ΑΠΑΝΤΗΣΗ.*

*Εάν δεν έχετε υλοποιήσει μια απαίτηση γράψτε στην αντίστοιχη γραμμή ΔΕΝ ΥΛΟΠΟΙΗΘΗΚΕ.*

*Εάν δεν έχετε υλοποιήσει ΠΛΗΡΩΣ μια απαίτηση γράψτε στην αντίστοιχη γραμμή ΔΕΝ ΥΛΟΠΟΙΗΘΗΚΕ ΠΛΗΡΩΣ, προσδιορίζοντας στη συνέχεια το βαθμό υλοποίησής της.]*

|  |  |
| --- | --- |
| **Απαίτηση** | **Εικόνα** |
| R1 | *Εισαγωγή σχετικής εικόνας. Σύντομα σχόλια που να επεξηγούν τη λειτουργία.* |
| R2 |  |
| R3 |  |
| R4 |  |
| R5 |  |

# ΚΡΙΤΙΚΟΣ ΑΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ΕΡΓΟΥ

*[Κάντε έναν αναλυτικό και κριτικό απολογισμό του έργου αναφέροντας*

#### τα προβλήματα που αντιμετωπίσατε, τι θα αλλάζατε στο έργο αν είχατε τη δυνατότητα,

#### τις αποκλίσεις που είχατε στις επαναλήψεις σε σχέση με αυτά που ορίζονται στην εργασία,

#### το χρόνο που απαιτήθηκε για την κάθε δραστηριότητα με βάση τα στοιχεία που κρατήσατε και σε σχέση με το χρόνο που προϋπολογίσατε,

#### πως αξιολογείτε το εργαλείο συνεργασίας trello και με ποιο τρόπο σας βοήθησε στην καθημερινή σας εργασία

#### τους κινδύνους που πραγματώθηκαν, καθώς και πώς αποκριθήκατε σε αυτούς,

#### τι θα αλλάζατε σε ένα επόμενο αντίστοιχο έργο, καθώς και

#### ποια ήταν τα θετικά σημεία που αποκομίσατε από αυτή την εργασία].

# VIDEO

*[Παραθέστε το link του video που δημιουργήσατε]*

# ΑΝΑΦΟΡΕΣ

*[Παραθέστε τις βιβλιογραφικές αναφορές (βιβλία, άρθρα, σύνδεσμοι στο διαδίκτυο) που χρησιμοποιήσατε για την υλοποίηση της εργασίας.]*