ΤΕΔια ΔΙΑΒΑΣΕΜΕ

README για την εργασία στο μάθημα Τεχνολογίες Εφαρμογών Διαδικτύου

Γιατράκος Γεώργιος 1115201600036

Σάμιος Γρηγόρης 1115201500141

4.7.2020

Εισαγωγή

Ο σκοπός της εργασίας είναι η εξοικείωση με τις σύγχρονες τεχνολογίες διαδικτύου. Στα πλαίσια της εργασίας δουλέψαμε με Javascript framework(VueJS) και python micro-framework Flask. Η αρχιτεκτονική της εφαρμογής ακολούθησε το πρότυπο REST API. Η εφαρμογή υλοποιεί ιστοσελίδα ενοικίασης δωματίων από χρήστες. Η ακόλουθη αναφορά χωρίζεται σε τέσσερα κεφάλαια. Αρχικά, παρουσιάζουμε τις σχεδιαστικές επιλογές για το frontend. Στο δεύτερο κεφάλαιο παρουσιάζουμε τις σχεδιαστικές επιλογές για το backend. Στο τρίτο κεφάλαιο περιγράφουμε την διαδικασία που ακολουθούμε για την υλοποίηση του μπόνους, δημιουργία recommendation system βασισμένο σε Matrix Factorization. Το τελικό κεφάλαιο αναλύει τις παραδοχές που κάναμε στην υλοποίηση μας και τυχόν διαφοροποιήσεις απο την εκφώνηση.

Frontend

Για την υλοποίηση του user interface χωρίσαμε την εφαρμογή σε 3 τμήματα : appbar,main,footer

Κυρίως έγινε χρήση έτοιμων components του veutify[1]

Το appbar παραμένει σταθερό καθόλη την πλοήγηση του χρήστη εκεί βρίσκονται λειτουργίες εισόδου του χρηστη και λινκ προς την αρχική σελίδα.

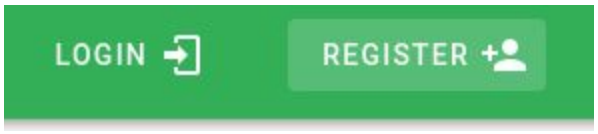
Το main τμήμα αλλάζει με βάση την πλοήγηση του χρήστη και αποτελεί το κεντρικό τμήμα της ιστοσελίδας.

Το footer παραμένει σταθερό καθόλη την πλοήγηση του χρήστη όπου εκεί βρίσκονται κάποια λινκ και γενικές πληροφορίες

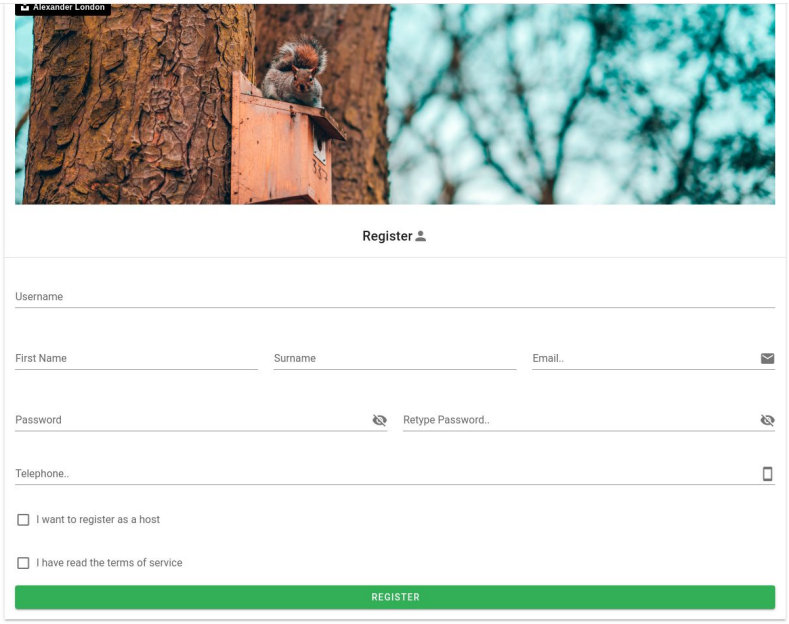
Κάθε ιστοσελίδα στο main αποτελείένα view το οποίο χωρίζεται σε επιμέρους components για καλύτερη διαχείριση, σχεδιασμό και οργάνωση του project. Στο αρχείο index.js στο φακελο store αποθηκεύονται global δεδομένα που μπορείνα αφορούν διαφορετικά components και όλα τα http requests προς το backend.

Παράδειγμα:

[1] https://vuetifyjs.com/en/



1. Ο χρήστης πατάει Register στο AppBar



2. Εισάγει στην φόρμα εγγραφής τα κενά πεδία. Πατώντας το Register καλείται το action register στο store/index.js

```
register({commit}, user) {
    return new Promise( executor: (resolve, reject) => {
        commit('auth_request')
        instance({url: 'user', data: user, method: 'POST'}) AxiosPromise<any>
            .then(resp => {
                console.log(resp.data)
                resolve(resp)
            })
            .catch(err => {
                commit('auth_error', err)
                localStorage.removeItem( key: 'token')
                reject(err)
            })
    })
},
```

3. Το action register εκτελεί μια POST http request στο endpoint '/user'.

Backend

Η υλοποίηση του backend έγινε στο framework flask[2]. Το συγκεκριμένο framework επιλέχθηκε λόγω της εξοικείωσης των συγγραφέων με την python και της γρήγορης και απλής δημιουργίας REST API.

Οι διάφορες λειτουργίες που ζητούνται αντιστοιχήθηκαν με endpoints και η συλλογή των δεδομένων από το frontend γίνεται με http requests στα κατάλληλα αυτά endpoints. Για την αποθήκευση των δεδομένων χρησιμοποιούμε βάση δεδομένων τύπου sqlite3[3]. Η βάση αυτή αποθηκεύεται σε ένα μόνο αρχείο καθιστώντας εύκολο τον συγχρονισμό των μελών της ομάδας αφού απαιτείται μόνο ένα commit και όχι database server. Η επικοινωνία του backend με την βάση γίνεται με

[2] https://flask.palletsprojects.com/en/1.1.x/
[3] https://www.sqlite.org/index.html

την χρήση της βιβλιοθήκης SQLAlchemy[4]. Η συγκεκριμένη βιβλιοθήκη επιτρέπει μια object-oriented προσέγγιση που αντιστοιχεί τους πίνακες της βάσης με object της γλώσσας python.

Για παράδειγμα, ας δούμε πώς γίνεται η εγγραφή ενός νέου χρήστη. Αρχικά το μοντέλο του χρήστη ορίζεται ως

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    public_id = db.Column(db.String(50), unique=True)
    isAdmin = db.Column(db.Boolean)
    isHost = db.Column(db.Boolean)
    isPending = db.Column(db.Boolean)
    uname = db.Column(db.String(32),unique=True)
    password = db.Column(db.String(80))
    fname = db.Column(db.String(32))
    surname = db.Column(db.String(32))
    email = db.Column(db.String(32))
    phone = db.Column(db.String(15))
    avatar = db.Column(db.String(50))
```

αντικείμενο της python στο αρχείο backend/models/User.py. Ο ορισμός είναι αυτός που φαίνεται στην φωτογραφία. Αποτι βλέπουμε η στήλες του πίνακα αντιστοιχούν με τις παραμέτρους της κλάσης User. Μπορούμε επίσης να ορίσουμε άλλες ιδιότητες,όπως το uniqueness. Η SQLAlchemy κάνει την μετατροπή σε SQL. Η δημιουργία νέας εγγραφής γίνεται εξίσου με εύκολα. Με την χρήση του constructor του αντικειμένου αρχικοποιούμε τις στήλες στις τιμές που θέλουμε. Στο παράδειγμα αυτό χρησιμοποιούμε και την δυνατότητα της python για default arguments.

[4] https://www.sqlalchemy.org/

```
def __init__(self,public_id,isHost,uname,password,fname,surname,email,phone,avatar,isPending=True,isAdmin=False):
    #self.id = id
    self.public_id = public_id
    self.isAdmin = isAdmin
    self.isHost = isHost
    self.isPending = isPending
    self.uname = uname
    self.password = password
    self.fname = fname
    self.surname = surname
    self.email = email
    self.phone = phone
    self.avatar = avatar
```

Στο αρχειο /routes/Users.py ορίζουμε όλα τα endpoints που ενεργούν πάνω στο μοντέλο μας (πχ. Login, register, acceptUser κτλ). Για την εγγραφή νέου χρήστη ορίζουμε το endpoint /user που δέχεται http requests τύπου POST. Η python παιρνει τα δεδομένα στο σώμα του http request με την βοήθεια του flask. Έπειτα, κάνουμε hash των κωδικόμε κρυπτογράφηση sha256. Στην

```
@users_blueprint.route('/user', methods=['POST'])
def addUserRequest():
    data = request.get_json()
    print(data,file=sys.stderr)
    hashed_password = generate_password_hash(data['password'], method='sha256')

    uName = User.query.filter_by(uname=data['uname']).all()
    if len(uName) > 0 :
        return jsonify({'message' : 'User already exists!'}),400

    uEmail = User.query.filter_by(email=data['email']).all()
    if len(uEmail) > 0 :
        return jsonify({'message' : 'Email already exists!'}),400

    new_user = User(
    public_id=str(uuid.uuid4()),
    uname=data['uname'],
    password=hashed_password,
    fname=data['fname'],
    surname=data['surname'],
    email=data['email'],
    phone=data['phone'],
    isHost=data['isHost'],
    isPending=data['isHost']
    )

    db.session.add(new_user)
    db.session.commit()
    return jsonify({'message' : 'New user created!'})
```

συνέχεια, ελέγχουμε την διαθεσιμότητα του ονόματος. Τέλος, δημιουργούμε ένα νέο αντικείμενο user το οποίο προσθέτουμε στην βάση δεδομένων μας. Η συνάρτηση επιστρέφει απάντηση στην αίτηση του χρήστη με μηνύματα για την έκβαση της αίτησης του. Η υλοποίηση των άλλων λειτουργιών γίνεται με παρόμοιο τρόπο.

Bonus

Το recommendation system έγινε με Matrix Factorization και Stochastic Gradient Descent. Η υλοποίηση γίνεται στο φακελο recommender του backend. Αρχικά, προσθέτουμε τα στοιχεία στην βάση με το αρχείο createData.py. Η διαδικασία είναι σχετικά αργή(5 ώρες) λόγω του όγκου των δεδομένων. Στο παραδοτέο τα δεδομένα είναι ήδη στην βάση οπότε δεν απαιτείται εκτέλεση του αρχείου. Το recommendation system καλείται απο το recommender.py. Τέλος, στο αρχείο ExplicitMF βρίσκεται η κλαση ExplicitMF που αντιστοιχείστο recommendation system μας. Για την υλοποίηση του recommendation system βασιστήκαμε σε πολλές πηγές που υπάρχουν στο διαδίκτυο με υλοποιήσεις στην python[5][6]. Ο αριθμός των components και των epochs είναι μεταβλητός και ορίζεται από τον χρήστη μέσω παραμέτρων κατα την αρχικοποίηση της κλάσης.

Το πρόβλημα του data sparsity δεν επιλύθηκε. Ο λόγος είναι ότι είχαμε τεχνικές δυσκολίες και περιορισμούς, λόγω της αρχιτεκτονικής της εφαρμογής που δεν επιτρέπει

[5] https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b21ea

[6] https://beckernick.github.io/matrix-factorization-recommender/

την αποθήκευση των αναζητήσεων. Επίσης, λόγω της απουσίας δοκιμαστικών δεδομένων θα ήταν δύσκολο να κάνουμε τις δοκιμές.

Παραδοχές

- Σε περίπτωση που ένας οικοδεσπότης επιθυμεί να θέσει ένα δωμάτιο διαθέσιμο μόνο για συγκεκριμένο διάστημα πχ [1 Ιουλίου,1 Σεπτεμβρίου] τότε δημιουργούνται κρατήσεις στα διαστηματα [today,30 Ιουνιου],[2 Σεπ,∞] όπου το ∞ αποθηκεύεται ως null στην βάση.

Επίλογος

Κατα την ανάπτυξη της εφαρμογής χωρίσαμε τα ζητούμενα της εκφώνησης. Έπειτα, υλοποιήσαμε κάθε ένα ξεχωριστά κάνοντας το frontend και backend κομματι.
Χρησιμοποιήσαμε git για version control. Κατα την ανάπτυξη της εργασίας παρουσιάστηκαν κάποια προβλήματα στο κομματι του SSL. Τα certificates είναι self signed και χρειάζεται ειδική ρύθμιση στον browser που επιτρέπει την σελίδα.