MyWaste

Applicazioni e Servizi Web

Federico Pettinari - 988120 {federico.pettinari2@studio.unibo.it}
Hamado Dene - 973128 {hamado.dene@studio.unibo.it}

7 giugno 2021

0.1 Introduzione

Lo scopo del progetto è la realizzazione di un'applicazione web per la gestione dei dati dei rifiuti cittadini ed i relativi conferimenti, in modo che ogni cittadino possa sapere (anche in tempo reale) la quantità ed il costo dei rifiuti prodotti. A questo scopo si presume che ogni cassonetto per la raccolta differenziata sia dotato di lettore RFID per poterne identificare il padrone in modo automatico.

0.2 Requisiti

- Progettare e sviluppare una web app per la visualizzazione e gestione dei dati sui rifiuti cittadini
- I cassonetti sono dotati di sensore di peso e lettore rfid per identificare l'utente che sta conferendo i rifiuti
- La piattaforma deve calcolare il costo mensile in funzione del tipo di rifiuto e peso.
- Deve mostrare la notifica del conferimento all'utente
- Deve mostrare grafici statistici sul conferimento
- La lettura dei sensori dei cassonetti ed il conseguente inserimento dei dati sull'applicazione avvengono automaticamente al centro di smistamento, senza bisogno di intervento da parte dell'utente.

0.3 Design

0.3.1 Architettura

L'architettura è raffigurata in Figura 1. Ogni richiesta HTTP viene ricevuta e smistata dal reverse proxy. Le chiamate che iniziano con /api vengono passate al backend, mentre le altre al frontend. Questo aiuta a separare frontend e backend in modo trasparente e senza bisogno di utilizzare Cross-Origin Resource Sharing (CORS).

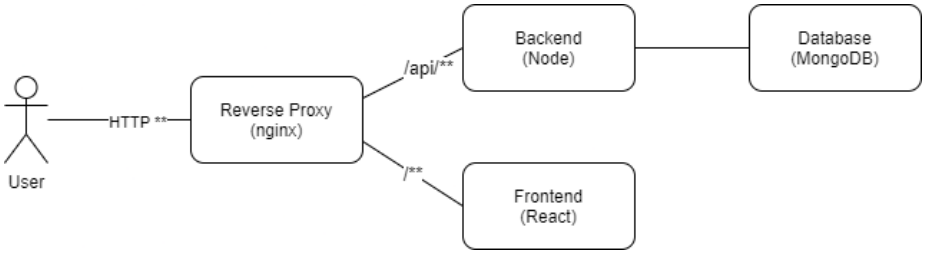

Figura 1: Architettura

0.3.2 Database

Il modello del database è semplice e raffigurato in Figura 2. L'applicazione può avere un numero variabile di account. Ad ogni account sono associate le proprie notifiche e conferimenti (Waste). Un account oltre ai dati di identificazione (email e password), ha un ruolo (role) per distinguere amministratori da utenti. Per ogni conferimento vengono memorizzati la data, il tipo di spazzatura (es. plastica o carta) e la quantità. Per ogni notifica vengono memorizzati la data di creazione, una flag per sapere se è stata letta, ed il tipo e quantità per i conferimenti.

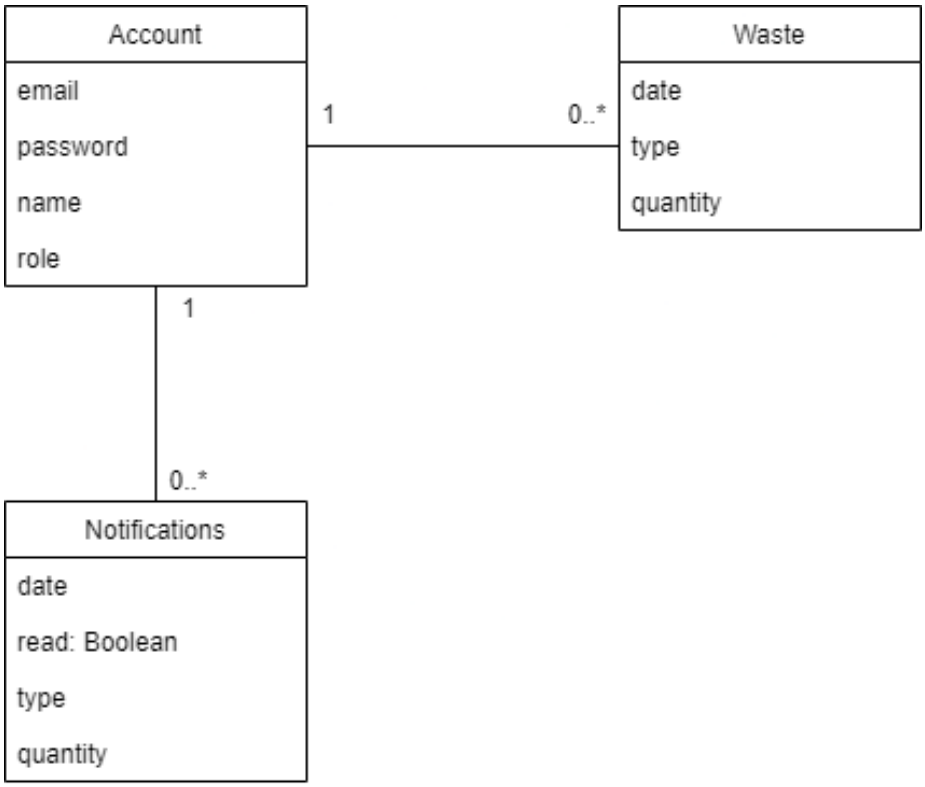

Figura 2: Modello del database

0.3.3 Interfaccia utente

Per il design dell'interfaccia utente è stato inizialmente sviluppato il mockup raffigurato in Figura 3, che si è man mano evoluto a seguito dei test di usabilità effettuati. E' stata prediletta la semplicità e l'immediatezza. Il progetto nasce mobile-first con interfaccia responsive per supportare anche schermi desktop.

Figura 3: Mockup interfaccia utente

0.4 Tecnologie

Il progetto utilizza le seguenti tecnologie:

- Github Actions - per continous integration
- Docker - per disaccoppiare il software dal sistema operativo e facilitarne il porting ed installazione.
- nginx - utilizzato come reverse proxy
- MongoDB - database non relazionale di documenti
- Node - come server web sia per frontend e backend
- Express - framework per gestire chiamate e routes HTTP

- Mongoose - come libreria di object modeling per mongodb
- Mocha e Supertest - per testing automatizzato nel backend
- mongodb-memory-server - per utilizzare un database mongodb in memoria, in modo da velocizzare l'esecuzione dei test
- bcrypt - per criptare le password degli utenti ai fini di sicurezza
- express-jwt - libreria per facilitare l'uso di tokens JWT utilizzati nel sistema di login utente
- Nodemon - tool che riavvia le applicazioni Node ad ogni modifica del sorgente per velocizzarne lo sviluppo e debugging
- React - framework per frontend
- SCSS - estensione di CSS che ne facilita il riuso
- Bootstrap - framework per CSS

0.5 Codice

0.5.1 Backend

Il backend è costituito da un servizio che espone API RESTful in formato JSON, ed è sviluppato con le tecnologie principali Node, Express e database MongoDB.

APIs

Le API implementate sono:

- `Post /login` Restituisce un json contente username,name,role e token JWT se i paramentri username e password passati nella fase di request risultano corretti.
- `Get /me` Restituisce le informazioni dell'utente loggato. La richiesta deve essere fatto usando un token valido.
- `Get /account` Restituisce la lista degli utenti registrati nella piattaforma. Questa api, può essere chiamata solo dagli utenti admin.
- `Post /account` Permette la creazione di nuovi account da parte di amministratori.
- `Patch /account/:account` Permette agli amminstratori di aggiornare le informazioni di un account esistente dato il suo id.
- `Delete /account/:account` Permette ad un amministratore di cancellare un account dalla piattaforma.

- `Post /waste` API per registrare i dati relativi ad un conferimento, come tipo e peso dei rifiuti.
- `Get /waste` Permette di effettuare query sui conferimenti, usando opportuni parametri quali:
  - `groupByType` Per richiedere i dati aggregati per tipo di rifiuto.
  - `includeDataPoints` Per includere la lista dei singoli dati in una query con raggruppamento.
  - `from` Data di inizio
  - `to` Data di fine
- `Get /notifications` Restituisce la lista delle notifiche di un determinato account.
- `Patch /notifications/:notification` Permette di aggiornare lo stato della notifica a Read.

Autenticazione

Il backend gestisce l'autenticazione ed i permessi degli utenti. Alla creazione di un account, la password viene criptata con algoritmo bcrypt e salvata su DB. Quando viene richiesta l'autenticazione e i parametri (username e password) corrispondono ai parametri salvati su DB, viene restituito un token JWT che permette l'accesso alla piattaforma. Esso contiene informazioni come id e permessi. Di seguito parte del codice volto alla gestione di permessi ed autenticazione:

```
1  const JWT_SECRET = '12345' // just for demonstration
2  const REQUEST_PROPERTY_DECODED_JWT = 'auth' // decoded JWT token
       gets inserted in request.auth
3
4  const jwt = require('jsonwebtoken')
5  // JWT decoding middleware
6  const auth = require('express-jwt')({ // JWT decoding middleware
7      secret: JWT_SECRET,
8      algorithms: ['HS256'],
9      credentialsRequired: true,
10     requestProperty: REQUEST_PROPERTY_DECODED_JWT
11 })
12 const guard = require('express-jwt-permissions')({
13     requestProperty: REQUEST_PROPERTY_DECODED_JWT,
14     permissionsProperty: 'role' // permissions are stored in the
           role field of the JWT token
15 })
16 //Generate token for user
17 module.exports = {
18     auth: auth,
19     guard: guard,
20     createToken(account) {
21         return jwt.sign({id: account._id, role:account.role},
               JWT_SECRET)
22     }
23 }
```

```
//Account model
const mongoose = require('mongoose')
const bcrypt = require('bcrypt')
const { Schema } = mongoose

const accountSchema = new Schema({
    email: {
        type: String,
        unique: true,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    role: {
        type: String,
        default: 'user'
    },
    name: {
        type: String,
        required: true
    }
})

accountSchema.pre('save', function(next) {
    if (!this.isModified('password')) {
        return next()
    }
    bcrypt.genSalt(2, (error, salt) => {
        if (error) {
            return next(error)
        }
        bcrypt.hash(this.password, salt, (error, hash) => {
            if (error) {
                return next(error)
            }
            this.password = hash
            next()
        })
    })
})

accountSchema.methods.toJSON = function() {
    var obj = this.toObject()
    delete obj.password
    obj.id = obj._id
    delete obj._id
    delete obj.__v
    return obj
}

module.exports = mongoose.model('Account', accountSchema)
```

Rifiuti

```
//Waste model
const mongoose = require('mongoose')
const { Schema } = mongoose

const schema = new Schema({
    account: { type: Schema.Types.ObjectId, required: true, ref: 'Account' },
    type: { type: String, required: true },
    quantity: { type: Number, required: true },
    date: { type: Date, default: Date.now, required: true, index: true }
})
schema.index({account: 1, date: -1}) // ascending accounts, descending dates

module.exports = mongoose.model('Waste', schema, 'waste')
```

```
//Waste controller
module.exports = {
    delivery(req, res, next) {
        Account.findById(req.body.account).exec()
            .then( acct => {
                let data = {
                    account: acct.id,
                    quantity: req.body.quantity,
                    type: req.body.type
                }
                if (req.body.date) {
                    data.date = new Date(req.body.date)
                }
                return new Waste(data).save()
            })
            .then(
                waste => {
                    new Notification({
                        account: waste.account,
                        date: waste.date,
                        message:`Delivered ${waste.quantity} Kg of ${waste.type}`
                    }).save()
                    res.status(201).json(waste)
                },
                err => next(err)
            )
    },
    query(req, res, next) {
        let q = null
        if (req.query.groupByType) {
            const pipeline = [
                {
                    $group: {
                        _id: "$type",
                        total: {$sum: "$quantity"}
                    }
                }, {
                    $project: {
                        _id: 0,
                        type: "$_id",
```

```
                        total: 1
                    }
                }
            ]
            if(req.query.includeDataPoints) {
                pipeline[0].$group.data = { $push:  { date: "$date"
                    , quantity: "$quantity" } }
                pipeline[1].$project.data = 1
            }
            let match = { }
            if (req.query.account) {
                match.account = ObjectId(req.query.account)
            }
            if (req.query.from) {
                match.date = { $gte: new Date(req.query.from) }
            }
            if (req.query.to) {
                match.date = match.date || {}
                match.date.$lte = new Date(req.query.to)
            }
            if(Object.keys(match).length > 0) {
                pipeline.unshift({ $match: match })
            }
            q = Waste.aggregate(pipeline)
        } else {
            q = Waste.find()
            if (req.query.account) {
                q.where('account', ObjectId(req.query.account))
            }
            if (req.query.from) {
                q.where('date').gte(new Date(req.query.from))
            }
            if (req.query.to) {
                q.where('date').lte(new Date(req.query.to))
            }
        }
        q.then(
            result => res.status(200).json(result),
            err => next(err)
        )
    }
}
```

Notifiche

Le notifiche vengono inserite automaticamente al momento del conferimento. Si presuppone che il frontend effettui polling.

```
//Notification model
const mongoose = require('mongoose')
const { Schema } = mongoose

const notificationSchema = new Schema({
    read: {
        type: Boolean,
        default: false
```

```
    },
    date: {
        type: Date,
        default: Date.now
    },
    message: {
        type: String,
        required: true
    },
    account: { type: Schema.Types.ObjectId, required: true, ref: '
        Account' },
})

module.exports = mongoose.model('Notification', notificationSchema)
```

```
//Notifications controller
const Notification = require('../models/notification')

module.exports = {
    query(req, res, next) {
        let q = Notification.find({account: req.auth.id})
        if (req.query.from) {
            q.where('date').gte(new Date(req.query.from))
        }
        if (req.query.to) {
            q.where('date').lte(new Date(req.query.to))
        }
        q.sort({date:'descending'})
        q.exec().then(notifications => res.status(200).json(
            notifications), next)
    },
    markAsRead(req, res, next) {
        Notification.findByIdAndUpdate(req.params.notification, req
            .body, {new:true})
            .then(n => res.json(n), next)
    }
}
```

0.5.2 Frontend

Il frontend è costituito da un servizio web sviluppato con Node e React.

Autenticazione

L'autenticazione è delegata al backend attraverso l'apposita API di login. Ricevuto il token JWT, esso viene salvato in localStorage per memorizzare la sessione.

```
import React, { useState } from 'react';
import PropTypes from 'prop-types';

async function loginUser(credentials) {
    return fetch('/api/login', {
        method: 'POST',
        headers: {
```

```
            'Content-Type':'application/json'
        },
        body: JSON.stringify(credentials)
    }).then(
        data => data.json()
    )
}

export default function Login({setToken}) {

    const [email, setEmail] = useState();
    const [password, setPassword] = useState();
    const [errorMessage, setErrorMessage] = useState();

    const handleSubmit = async e => {
        e.preventDefault();
        const token = await loginUser({
            email,
            password
        });
        if(token) {
            setErrorMessage('Authentication error. Incorrect
                username or password.')
        }
        setToken(token);
    }
    return(...)
}


export default function useToken() {
  const getToken = () => {
    const tokenString = localStorage.getItem('token');
    const userToken = JSON.parse(tokenString);
    return userToken?.token
  };

  const [token, setToken] = useState(getToken());

  const saveToken = userToken => {
    localStorage.setItem('token', JSON.stringify(userToken));
    setToken(userToken.token);
  };

  return {
    setToken: saveToken,
    token
  }
}
```

Gestione account

```
//Get users list
state = {
  users: []
}
```

```
5
6  getUserList() {
7    fetch("/api/account", {
8      method: 'GET',
9      withCredentials: true,
10     credentials: 'include',
11     headers: {
12       'Authorization': 'Bearer ' + JSON.parse(localStorage.getItem(
             'token')).token ,
13     }
14   })
15   .then(response => response.json())
16   .then(json => {
17     this.setState({users: json})
18   }).catch((error) => {
19     alert("some error: " + error);
20    });
21 }
```

```
1  //Add user
2  handleAddUser() {
3    if(this.state.user.length < 4) {
4      alert("All fields are required!")
5      return;
6    }
7    fetch("/api/account/", {
8      method: 'POST',
9      withCredentials: true,
10     credentials: 'include',
11     headers: {
12       'Authorization': 'Bearer ' + JSON.parse(localStorage.getItem(
             'token')).token,
13       'Content-Type': 'application/json'
14     },
15     body: JSON.stringify(this.state.user)
16   }).then((response) =>  {
17     if(response.ok) {
18       this.getUserList();
19       <Redirect to="/admin" />;
20     }
21   })
22 }
```

```
1  //Delete user
2  handleDelete() {
3    fetch("/api/account/" +this.state.modalInfo.id, {
4      method: 'DELETE',
5      withCredentials: true,
6      credentials: 'include',
7      headers: {
8        'Authorization': 'Bearer ' + JSON.parse(localStorage.getItem(
             'token')).token ,
9      }
10   }).then((response) =>  {
11     if(response.ok) {
12       this.getUserList();
13       <Redirect to="/admin" />;
```

```
    }
  });
}
```

```
//Update user data
handleUserSaveChange() {
  fetch("/api/account/" +this.state.modalInfo.id, {
    method: 'PATCH',
    withCredentials: true,
    credentials: 'include',
    headers: {
      'Authorization': 'Bearer ' + JSON.parse(localStorage.getItem(
          'token')).token ,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(this.state.user)
  }).then((response) =>  {
    if(response.ok) {
      this.getUserList();
      <Redirect to="/admin" />;
    }
  })
}
```

Grafici

Per creare i grafici statistici, bisogna fare una query alla tabella dei rifiuti. Con i dati che riceviamo, tramite highcharts li visualizziamo.

```
state = {
  chartData: []
}
componentDidMount(event,picker) {
   fetch("/api/waste?groupByType=true&includeDataPoints=true&from="
       +moment().subtract(60, 'days').format('YYYY-MM-DD')+"&to="+
       moment().add(1,'days').format('YYYY-MM-DD'))
   .then(response => response.json())
   .then(json => {
     this.setState({chartData: json})
   });
}

handleFilterChartsByDate(event, picker) {
   fetch("/api/waste?groupByType=true&includeDataPoints=true&from="
       +moment(picker.startDate).format('YYYY-MM-DD')+"&to="+moment
       (picker.endDate).format('YYYY-MM-DD'))
   .then(response => response.json())
   .then(json => {
     this.setState({chartData: json})
   });
}
```

Per la gestione del grafico lato utente standard, c'è stato la neccessità di far in modo che più componenti si passero le informazioni, in quanto viene aggiornato il campo costo e i grafici del mese selezionato.

```
class CostCalculator extends Component {

  constructor (props) {
    super(props)
    this.state = {
      startDate: new Date(),
      totalCost: ''
    };
    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(date) {
    this.setState({
      startDate: date
    })
  }

  componentDidMount() {
    this.calculateCost();
    this.handleChartUpdate();
  }

  componentDidUpdate(prevProps, prevState) {
    if(prevState.startDate !== this.state.startDate) {
        this.calculateCost();
        this.handleChartUpdate();
    }
  }

  getPickerDate() {
     return this.state.startDate
  }

    handleChartUpdate() {
       var accountId = JSON.parse(localStorage.getItem('token')).id
           ;
       var month = this.state.startDate.getMonth();
       var year = this.state.startDate.getFullYear();
       var from = moment(new Date(year, month, 1)).format('YYYY-MM-
           DD');
       var to = moment(new Date(year, month + 1, 0)).format('YYYY-
           MM-DD');
       fetch("/api/waste?account="+accountId+"&groupByType=true&
           includeDataPoints=true&from="+from+"&to="+to)
       .then(response => response.json())
       .then(json => {
           this.props.makeUpdateChartData(json)
       });
    }


   calculateCost() {
       var month = this.state.startDate.getMonth() + 1;
       var year = this.state.startDate.getFullYear();
       var accountId = JSON.parse(localStorage.getItem('token')).id
           ;

```

```
      fetch("/api/account/" +accountId + "/cost?month="+month+"&
          year="+year, {
           method: 'GET',
           withCredentials: true,
           credentials: 'include',
           headers: {
               'Authorization': 'Bearer ' + JSON.parse(localStorage
                   .getItem('token')).token,
               'Content-Type': 'application/json'
           }
       })
      .then(response => response.json())
      .then(json => {
          this.setState({totalCost: json.cost + " " + json.currency
              });
      });
   }
  render {
    return(...)
  }
}
```

```
class Statistics extends Component {
    constructor(props) {
        super(props);
        this.state = {
            chartData: []
        }
        this.updateChartData = this.updateChartData.bind(this);
    }

    updateChartData(data) {
        this.setState({chartData: data})
    }

    UNSAFE_componentWillReceiveProps(props) {
        if(this.state.chartData !== props.receiveChartData) {
            this.setState({chartData: props.receiveChartData})
        }
    }
  render {
    return(...)
  }
}
```

```
class Dashboard extends Component {
  constructor(props){
    super(props);
    this.state={
      chartData: []
    }
  }

  updateChartData(data) {
    this.setState({ chartData: data});
  }

```

```
13    render() {
14      return (
15        <div>
16          <CostCalculator makeUpdateChartData= {this.updateChartData.
                bind(this)}/>
17          <Statistics receiveChartData={this.state.chartData} />
18        </div>
19      )
20    }
21  }
```

0.6 Test

0.6.1 Codice

A livello di codice sono state adottate tecniche di continous integration con le Github Actions ed il framework e librerie di testing Mocha e Supertest. Ad ogni push di codice viene eseguita automaticamente la suite di testing opportunatamente creata. Questo sistema aiuta a prevenire regressioni e facilita la manutenzione.

0.6.2 Interfaccia

Il progetto è stato testato in modalità Cognitive Walkthrough con i seguenti task: . Gestione dei casi con input non valido (es. in caso di dati duplicati l'utente è in grado di capire cosa fare?).

- Visualizzazione e comprensione della propria dashboard
  - Login
  - Individuazione dashboard
  - Comprensione
    - Individuazione costo mensile
    - Individuazione quantità di spazzatura conferita in un determinato mese
    - Individuazione numero, data, e dettagli dei conferimenti avvenuti
  - Logout
- Ricevuta notifica a seguito di un conferimento
- Visualizzazione e comprensione dashboard per amministratori
  - Individuazione lista utenti
  - Individuazione quantità e ripartizione dei vari tipi di spazzatura in un determinato mese
- Creazione di un nuovo utente

Infine, una volta creato un prototipo funzionante e risolti i primi problemi individuati dal team di sviluppo, sono stati effettuati Usability Tests con 5 potenziali veri utenti. Ad ognuno di essi è stato chiesto di effettuare i task sopracitati senza però essere messi al corrente degli step intermedi. In questo modo è stato possibile osservare quanto fosse intuitiva l'interfaccia ed apportare migliorie.

0.7 Deployment

Il progetto può essere scaricato con `git clone https://github.com/fedpet/aws-project.git` ed avviato con `docker compose up` nella sua cartella root. A questo punto basterà aprire localhost con un browser web per vederne l'interfaccia. Di base per prova il progetto parte configurato con gli account admin e user1 e rispettivamente le password admin e user1.

0.8 Conclusioni

In conclusione pensiamo che questo progetto ci abbia aiutato a incrementare notevolmente la nostra conoscenza del mondo web e sopratutto aiutato a migliorare le nostre capacità di lavoro in team. Nonostante l'aspetto lavorativo ci abbia limitati a livello di tempo, siamo riusciti a strutturare il progetto in modo tale che ognuno potesse lavorare nel suo tempo libero senza la neccessità di dover coinvolgere continuamente gli altri membri. Per questo aspetto l'utilizzo corretto di github è stato fondamentale. Abbiamo inizialmente suddiviso il progetto in una lista di task da implementare, inseriti su Github sottoforma di "issue". Da tale lista, ognuno poteva prendere in carico un task, risolverlo e creare un pull request. Ogni pull request richiedeva la revisione da parte di almeno un altro membro del team, nonchè l'esecuzione con successo di tutti i test automatici.