Технически Университет София
Факултет по компютърни системи и технологии

Проект по Web технологии

Тема:

Hirundo

https://github.com/kakato10/hirundo2

Светимир Игнатов 661317018

Ръководител:

(гл. ас. А. Ташева)

Съдържание

1. Идея на проекта

В днешно време едни от най-известните сайтове, са социалните мрежи (Facebook, Twitter, Instagram, Google+ и др.) . Всички тези сайтове след време започват да се разширяват в много и различни насоки и по този начин първоначалната идея залегнала в тях остава на заден план.

Проектът Hirundo има за цел точно това, да предостави на потребителите социална мрежа, в която комуникацията между тях самите ще бъде в центъра на вниманието. Социална мрежа, която да заеме освободилата се ниша, и да даде алтернатива на по-горе изброените гиганти.

2. Типове потребители

- Анонимни потребители

Това са потребителите, които нямат регистрация в сайта. Те имат право да се регистрират в сайта и да разгледат различните статистики, които са генерирани;

- Регистрирани потребители

Това са потребителите, които вече имат създадена регистрация в Hirundo. Преди да се впишат сайта, им се предоставя възможност за вписване и да разгледат статстиките, които представят потока на информация в сайта;

- Вписани потребители

Това са същинските потребители на Hirundo. Те имат достъп до пълната функционалност на системата.

- Системата предоставя възможност на всеки вписан потребител да следват или да престане да следва други потребители;
- На специална страница наречена Feed вписаните потребители могат да видят съобщенията на всички последвани, от тях, потребители;
- Отново на страницата Feed всеки вписан потребитен може да съставя собствени съобщения, които могат да съдържат в себе си hashtag-ове;
- На страницата Hashtag Feed може да се проверяват всички съобщения съдържащи даден hashtag;
- Към всяко съобщение могат да се създават коментари, да се харесват и да се премахва добавено харесване;
- В страницата My Posts всеки вписан потребител може да види списък със собствените съобщения;
- В страницата Settings потребителите могат да сменят темата (цветовата гама) на системата;

- На специална страница Statistics могат да се разгледат статистики представящи различни аспекти на данните в системата;

3. Основни потребителски случаи (use case-и)

Потребителски случай	Кратко описание	Участващи потребители
Регистриране	Анонимен потребител, може да се регистрира в системата предоставяйки потребителско име, валиден email адрес и парола.	Анонимен потребител
Вписване	Регистриран потребител, може да се впише в системата предоставяйки своя email и парола.	Регистриран потребител
Следване на потребител	Вписан потребител, може да последва друг потребител, което му дава възможност да вижда съобщенията публикувани от втория.	Вписан потребител
Прекратяване следването на потребител	Вписан потребител, може да спре да следва друг потребител, като съответно спира да вижда публикуваните от него съобщения.	Вписан потребител
Публикуване на съобщене	Вписан потребител може да публикува съобщение. В съобщението си може да отбележи и hashtag-ове свързани с него.	Вписан потребител
Преглед на съобщения	Вписан потребител може да вижда публикуваните съобщения от потребители, които той следва.	Вписан потребител
Преглед на собствените съобщения	Всеки потребител може да разглежда своите публикувани съобщения.	Вписан потребител
Харесване на съобщение	Вписан потребител може да харесва съобщения.	Вписан потребител
Отказ от харесване на съобщение	След като вписан потребител е харесал дадено съобщение, той има възможност да се откаже от харесването.	Вписан потребител
Преглед на съобщения по hashtag.	Вписан потребител може да филтрира всички съобщения съдържащи конкретен hashtag (независимо дали следва авторът на даденото съобщение).	Вписан потребител
Добавяне на коментар към съобщение	Всеки вписан потребител има възможност за добавяне на коментар към дадено съобщение, на някой от последваните от него потребители.	Вписан потребител
Смяна на темата	Всеки вписан потребител може да смени цветовата гама (темата) на системата в страницата за настройки.	Вписан потребител
Разглеждане на	Всеки потребител може да разгледа	Анонимен

статистики	статистики представящи различни аспекти на системата.	потребител / Регистриран потребител / Вписан потребител

4. Основни изгледи (main views)

Име на изгледа	Кратко описание	URI
Регистриране (Register)	Предоставя форма за регистрация, давайки възможност на даден потребител да се регистрира.	/register
Вписване (Login)	Предоставя форма за вписване в системата.	/login
Главен списък със съобщения (Feed)	Визуализира всички съобщения на потребители следвания от текущо вписания потребител в системата. Също така предоставя въозможност за публикуване на ново съобщение	/feed
Списък със съобщения филтрирани по hashtag (Hashtag Feed)	Визуализира всички съобщения съдържащи конкретен hashtag.	/hashtag_feed
Списък със собствени съобщения (My Posts)	Визуализира всички съобщения публикувани от текущо вписания потребител	/my_posts
Настройки (Settings)	Визуализира настройките, които системата предоставя на потребителя.	/settings
Статистики (Statistics)	Визуализира различни статистики генерирани от системата, с цел представяне на различни нейни аспекти	/statistics

5. Server Resource API

В таблицата отдолу са описани само използваните endpoint-и. Сървърът поддържа още.

Име на ресурса	Кратко описание	URI
Authentication (login)	**POST**: Дава възможност потребител да се впише в системата предоставяйки email адрес и парола	/auth/login
Users	**GET:** Връща данни за всички регистрирани потребители в системата **POST**: Позволява създаването на нов потребител в системата	/users
User	**GET:** Връща данните за потребителя с конкретно userId	/users/{userId}
	POST: Позволява харесване на съобщение с дадено postId	/users/like
	POST: Позволява на потребител да спре да харесва съобщение с дадено postId	/users/dislike
	POST: Позволява на вписания потребител да последва потребител с дадено userId	/users/follow

	POST: Позволява на вписания потребител да спре да следва потребител с дадено userId	/users/unfollow
Posts	**GET:** Връща данните за всички съобщения, създадени от потребители следвани от текущо вписания потребител. **GET** (с query hashtag): Връща данните за всички съобщения съдържащи конкретния hashtag. **POST:** Създава ново съобщение.	/posts
	GET: Връща данните за съобщенията на текущо вписания потребител.	/posts/own
Comments	**GET** (с query postId): Връща данните за всички коментари към съобщение с посоченото postId. **POST:** Позволява създаването на ново съобщение.	/comments
Comment	**GET:** Връща данните за коментар с конкретно commentId	/comments/{commentId}
Settings	**GET**: Връща настройките за текущия потребител; **POST:** Обновява настройките за текущия потребител;	/comments/{:userId}
Statistics	**GET:** Връща генерираните от системата статистикит	/stats

6. Използвани технологии

6.1. Клиентска част

- React – изграждане на структурата на UI-а на системата;
- React-router – навигиране между различните изгледи на системата;
- Redux – управляване на данните по време на изпълнението на клиентската част на системата;
- Material UI – изграждане на красив и модерен UI на системата;
- Bluebird – библиотека за promise-и;
- Lodash – библиотека предоставяща множество функции за работа с обекти, списъци и стрингове;
- Superagent – библиотека за изпращане на заявки към сървъра;
- React-c3js – библиотека за генериране на различни графики за представяне на данни;

6.2. Сървърна част

- Express – библиотека улесняваща изграждането на NodeJS приложения;
- Passport – библиотека даваща възможност за лесно изграждане на аутентукация в NodeJS приложения;
- jsData – библиотека за управляване на данните в сървърната част на системата.

Лесно изграждане на моделен слой;

- jsData-schema – библиотека за валидация чрез схеми на данни пристигащи към сървъра;
- js-data-mongodb – адаптер за комуникация на jsData с нерелационата база данни MongoDB;
- Lodash – библиотека предоставяща множество функции за работа с обекти, списъци и стрингове;
- MongoDB – нерелационна база от данни;

7. Структура (архитектура) и работен процес на системата

Кода на системата може да бъде намерен на следния URL: https://github.com/kakato10/hirundo2

7.1. Клиентска част

Архитектурата на клиентската част на системата е базирана на изискваните в работния процес React-Redux елементи.

- Store – централизирано хранилище на информация заредена в клиентската част на приложението. Представя състоянието на клиентската част;
- Actions – разпределители на събития отговарящи за модифициране състоянието на store-а;
- Reducers – функции модифициращи store-а на базата на събития генерирани от action-ите;
- Services – класове тип Singleton отговарящи за комуникация със сървъра, отнасяща се за конкретен модел. Класът Connection отговаря за изпращането на заявки към сървъра;
- Containers – класове wrapper-и, които имат директен достъп до състоянието на клиентската част. Тяхната цел е да работят като посредник между същинското представяне на информацията за потребител и бизнес логиката на приложението;
- Components – класове отговарящи за визуалузурането на информацията към потребителите на системата;

Работен процес:

User action -> Component processing -> Action call -> Service call -> Server request -> Action triggered -> Store update -> Components rerendering

7.2. Сървърна част

Сървърът на приложението е изграден като Express.js приложение за Node.js. Данните генерирани от приложението се съхраняват в нерелационна база от данни MongoDb. За

програмно представяне на различните типове данни (моделите/ресурсите) в системата се използва библиотеката jsData, с нейни допълнения за комуникация с MongoDB и валидиране на данните изграждащи моделите.

- Models – Папка съдържаща ресурси на системата. При дефинирането на даден модел се посочва и неговата схема, спрямо, която ще бъдат валидирани данните свързани с него.
- Routes – Папка съдържаща дефинициите на различните пътища изграждащи публичния интерфейс на сървъра. За ресурси, за които е необходимо да се реализират стандартните CRUD възможности има функция, която автоматично създава необходимите крайни точки. Тя позволява и модифицане на поведението, в случай на необходимост от страна на бизнес логиката. Също така възможна конфигурация е и забраняването на някой от методите за CRUD, в случай, че не е необходимо да има такава публична крайна точка.
- Services – различни услуги допълващи работата на приложението;

8. Представяне данните в базата от данни

Модел	Атрибут	Тип
User	_id	id
	username	String
	email	String
	password	String
	followedUsers	Array (of user ids)
Post	_id	Id
	authorUsername	String
	authordId	String
	content	String
	likes	Array (of user ids)
	hashtags	Array (of strings)
Comment	_id	Id
	authorUsername	String
	authorId	String
	content	String
	postId	String
Seetings	_id	Id
	userId	String
	theme	string

9. Нефункционални характеристики

9.1. Сигурност

- Валидация на данните

При създаването на всеки ресурс се създава схема, спрямо която биват валидирани всички пристигащи данни от страна на клиентската част на системата;

Примерна схема на модел (server/models/comments.js):

```
module.exports = (DS, schemator) => {
    schemator.defineSchema('Comment', {
        _id: 'id',
        authorUsername: 'string',
        authorId: 'string',
        content: 'string',
        postId: 'string'
    });
    return DS.defineResource('comment');
};
```

Дефиниция на тип данни за схема на модел (server/services/schemator).js:

```
schemator.defineDataType('id', x => {
// allow own id not to be defined
// should end up here only on
// entity create
    const type = typeof x;
    if (type === 'undefined') {
        return null;
    }
    if (type === 'string') {
        return null;
    }
    return {
        rule: 'type',
        actual: type,
        expected: 'string'
    };
});
```

Обработка на POST заявка спрямо схема на модел (server/routes/basicListeners.js):

```
router.post(CLREndpoint, (req, res) => {
    if (!req.body) {
        return res.status(400).send('No data provided!');
    }
    let data = req.body;
    if (filters && filters.postCLR) {
        data = filters.postCLR(data, req);
    }
    const errors = req.app.locals.schemator.validateSync(resourceName, data);
    if (errors) {
        res.status(403).send(errors);
        return;
    }
```

```
    req.app.locals[resourceName].create(data).then((entity)  =>  {
        res.status(201).send({
            location:   `${CLREndpoint}/${entity._id}`
        });
    },   (e)  =>  {
        res.status(500).send(e);
    });
});
```

- Наличие на проекции (projections)

  За всеки ресурс могат да се дефинират, множество проекции. При прилагането на дадена проекция, данните, които се изпращат към клиента се филтрират спрямо отбелязаната проекция.

  Налагане на проекция над инстанция от даден модел (/server/services/helper.js):

```
static applyProjectionOnEntity(entity, projection) {
    const keysToGet = _.keys(projection);

    return _.pick(entity, keysToGet);
}
```

  Дефиниране на проекция за даден модел (server/services/projections.js):

```
module.exports = {
    user: {
        basic: {
            _id: true,
            username: true,
            email: true,
            followedUsers: true
        }
    },
    ...
};
```

- Проверка за вписан потребител при навигиране към страници изискващ такъв:
  Системата налага ограничения до страници изискващи да бъдат достъпвани само от потребители, вписани в системата. При опит за достъпване от невписан потребител, системата го пренасочва към страница за вписване.
  Проверка наличието на вписан потребител и навигиране към страницата за вписване при нужда (client/src/comonents/App.js):

```
function requiresAuth(store, nextState, replace) {
    const currentUser = store.getState().auth.user;
    if (!currentUser) {
        replace({
            pathname: '/login',
            state: {
                nextPathname: nextState.location.pathname
            }
        });
    }
}
```

Изискване за вписан потребител за достъп до дадена страница:

```
<Route path = "feed"
       onEnter = {
         requireUser
       }
       component = {
         Feed
       }
/>
```

Разрешаване автоматично създаване на бисквитки (от страна на passport.js) за вписаните потребители (server/server.js):

```
app.use(cookieSession({
    keys: ['secret1', 'secret2'],
    maxAge: 24 * 60 * 60 * 1000 // 1 day
}));
app.use(passport.initialize());
app.use(passport.session());
```

9.2. Скалируемост

- Лесно конструиране и изграждане на заявки към сървъра

Наличието на service-а Connection, осигурява лесен начин за изпращане на заявки към сървъра. API-то му предоставя функция send, чрез която се извършва самото изпращане. Функцията приема като аргументи endpoint-а, данните които да се изпратят и тип на заявката. Ако заявката не е от тип POST или PUT данните се добавят като query-параметри на заявката.

- Лесно добавяне на нови модели (ресурси) на сървъра

Основните endpoint-и като Collection Level Resource и Individual Level Resource са отделени в самостоятелна функция, която може да закачи съответните функции, които да следят за заявки от страна на клиентската част на системата. При нужда функцията предоставя и възможност за конфигуриране (модифициране) поведението на съответните listener-и.

Закачане на функция за GET заявки за ресурс (server/routes/basicListeners.js):

```
module.exports = (router, {
    CLREndpoint, ILREndpoint, resourceName, permissions = {
        getCLR: true,
        put: true,
        post: true,
        delete: true
    }
}, projection, filters) => {
    if (permissions.getCLR) {  // check permission
        router.get(CLREndpoint, (req, res) => {
            let query;
            if (_.keys(req.query).length && filters && filters.getCLRWithQuery) {
                query = filters.getCLRWithQuery(req);  // client-side query
```

```
            } else if (filters && filters.getCLR) {  // default query
                query = filters.getCLR(req);
            }
            req.app.locals[resourceName].findAll(query).then((entities) => {
                if (projection) {
                    entities = Helpers.applyProjectionOnCollection(entities, projection);
                }
                res.send(entities);
            });
        });
    }
    ...
};
```

- Лесно превключване от един persistence layer към друг

Благодарение на библиотеката JSData използвана при реализацията на сървъра на системата, в случай на нужда сървърът лесно може да бъде превключен да работи с друг persistence слой.

```
const jsData = require('js-data');
const { MongoDBAdapter } = require('js-data-mongodb');
const mongoAdapter = new MongoDBAdapter({
    uri: 'mongodb://127.0.0.1:27017/hirundo'
});
const DS = new jsData.DS({
    cacheResponse: false,
    idAttribute: '_id',
    beforeUpdate: (resource, data, cb) => {
        const {
            _id, ...restProps
        } = data;
        cb(null, restProps);
    }
});
DS.registerAdapter('mongoAdapter', mongoAdapter, {
    default: true
});
```

9.3. Отказоустойчивост

- Валидация на данни (подробно описание в точка 9.1).
  Валидирането на данните предотвратява постъпването на некоректна информация към системата, като по-този начин намалява шанса за съхранението на повредени или грешни данни в базата от данни на системата.
- Редовна проверка за грешки.
  В кода на системата в случаите, където се използват асинхронен поток на данните, се прави прави проверка за възникнала грешка по-време на осъществената операция;

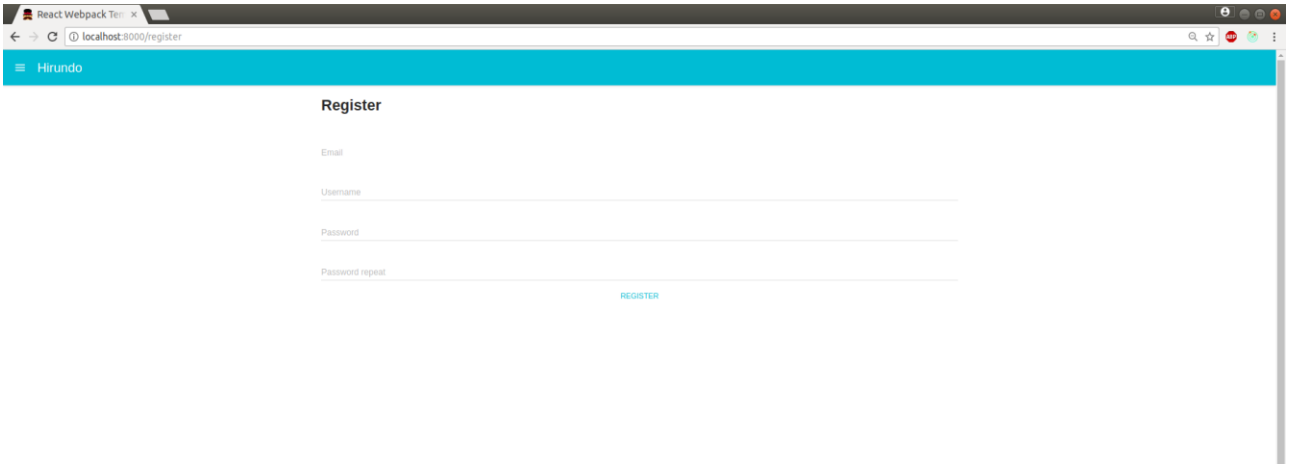
10. Дизайн

Дизайнът на сайта е реализиран спрямо най-съвременните стандарти в уеб приложенията. Използван е стандарта Material Design наложен от Google през последните няколко години.

Този тип дизайн се характеризира с изчистен външен вид, в който се цели да се открояват главно акцентите в съдържанието на дадения сайт. Същевременно съдържанието се наслагва в координатната ос “z”, така че да се реализира многослойност на съдържанието. Основна част в този аспект на дизайна е наслагването на пластове хартия един върху друг.

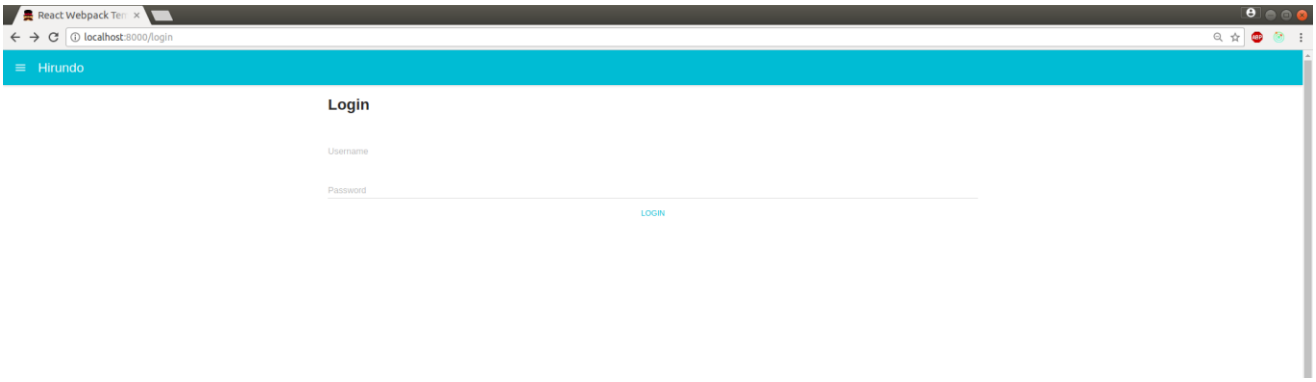
10.1. Дизайн на регистрацията

Дизайнът за изгледа за регистрация използва стандартни TextField компоненти от Material-UI



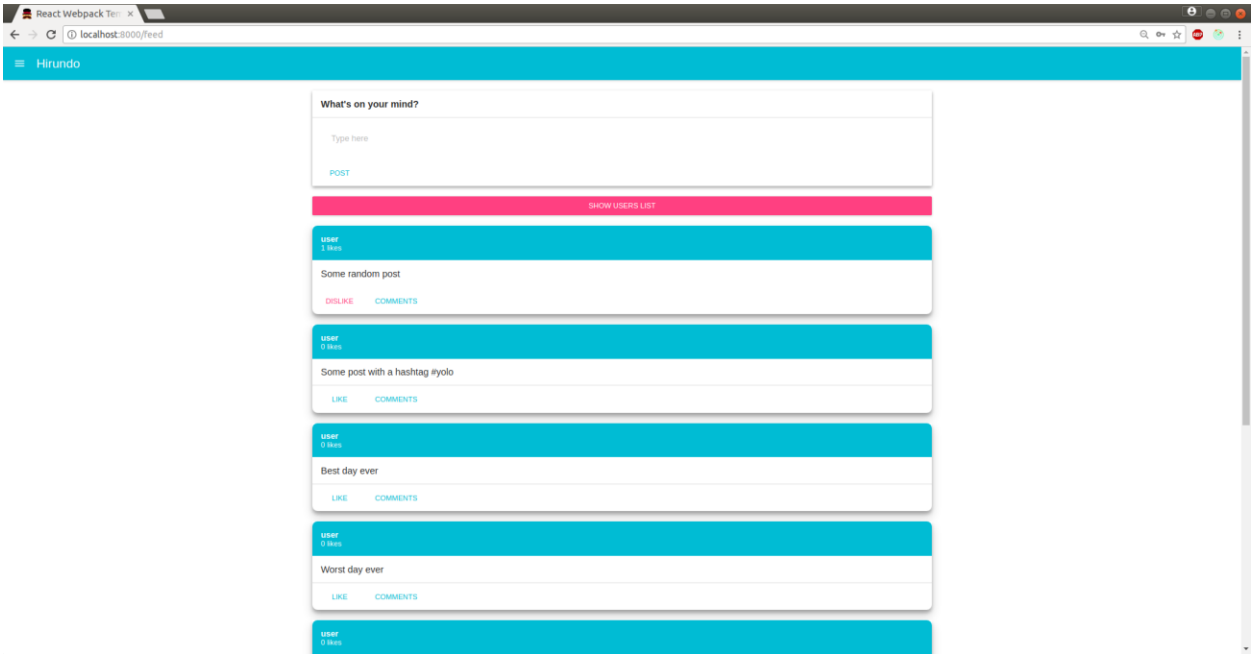
10.2. Дизайн на login

Този изглед използва същите ресурси, като по-горния, но са нужни по-малко полета:

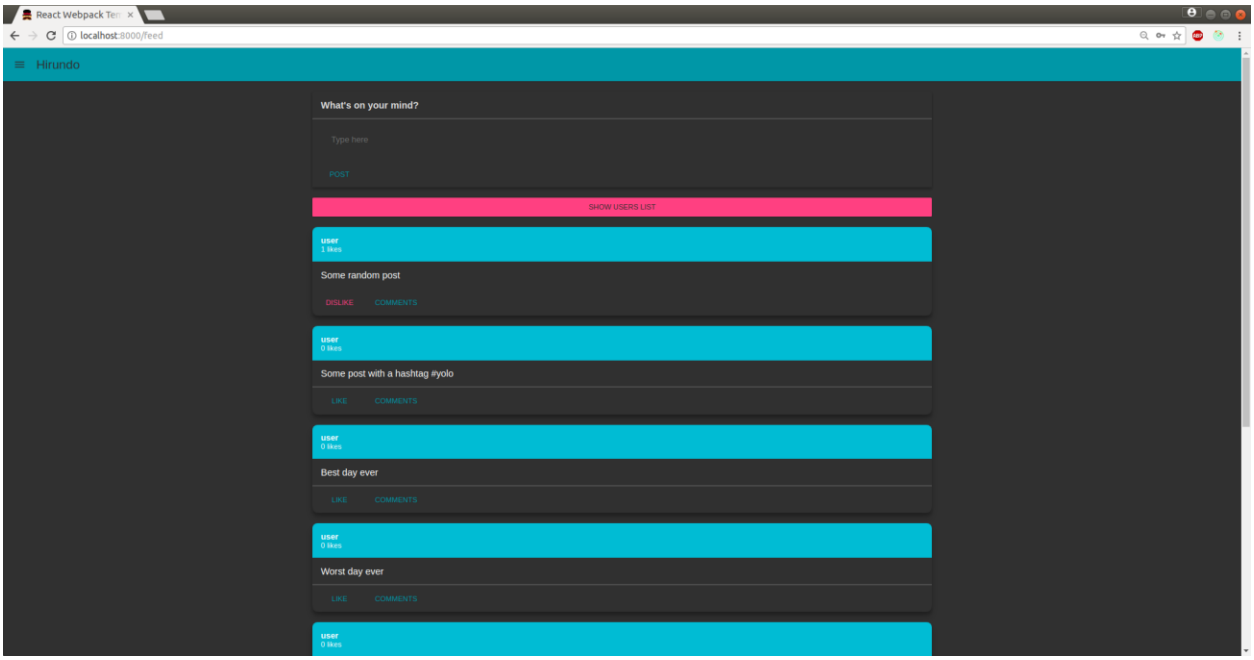


10.3. Дизайн на страницата за показване на постовете

Светла тема:

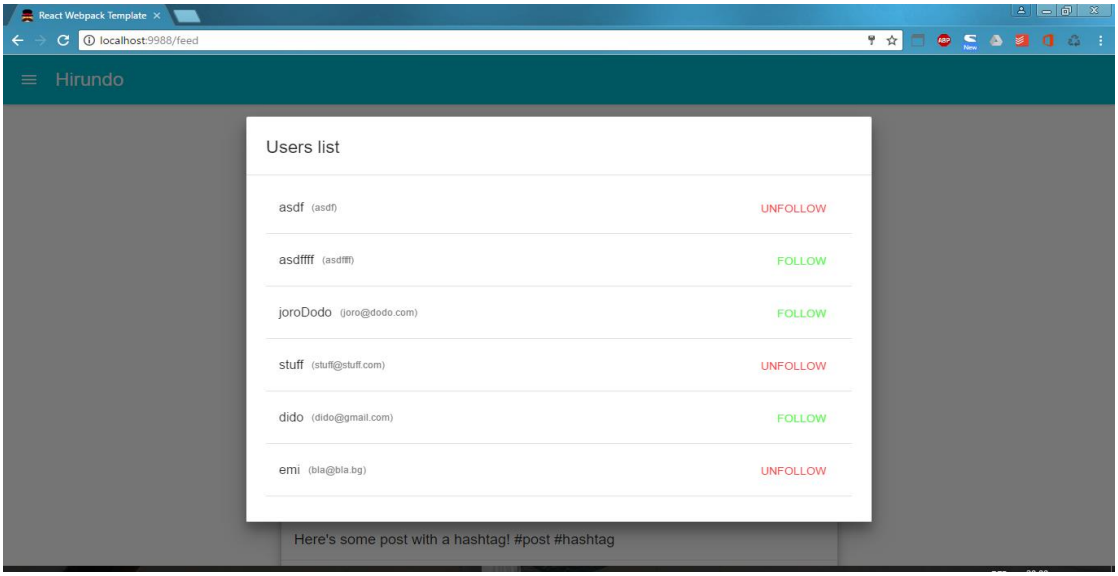


Тъмна тема:



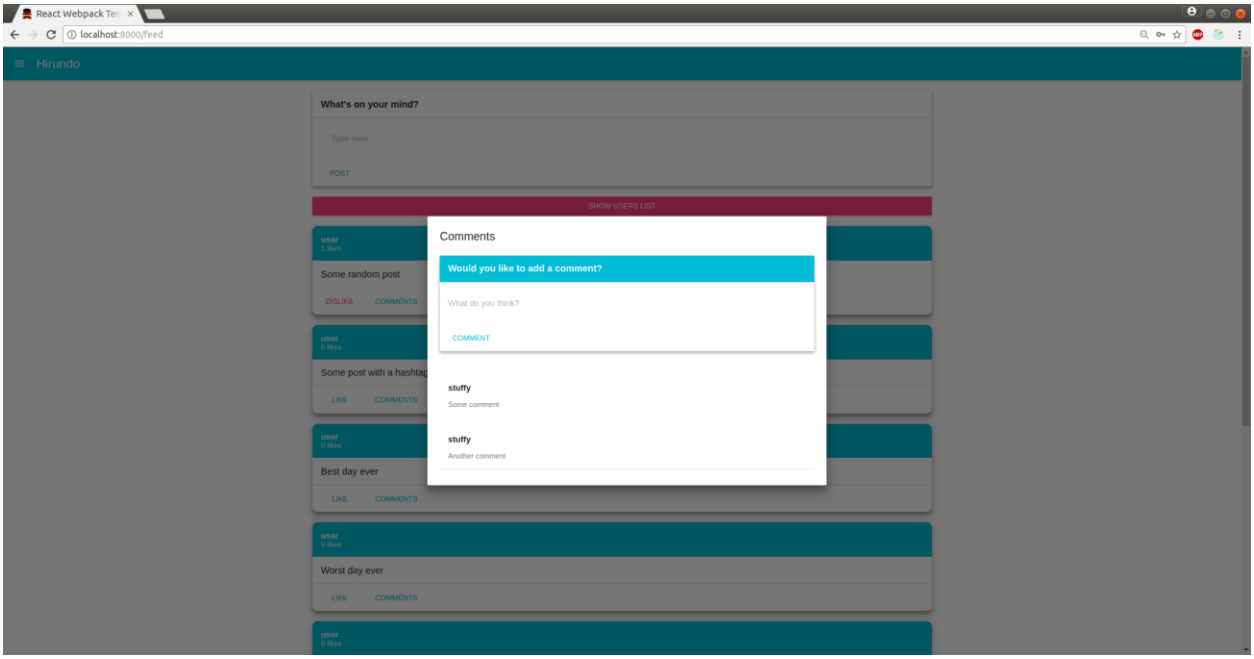
10.4. Дизайн на страницата за показване на потребителите

Създава се диалог след натискане на “SHOW USERS LIST”, в който може да се последват потребители. Това тяхно последване се отразява веднага на показваните в момента постове.

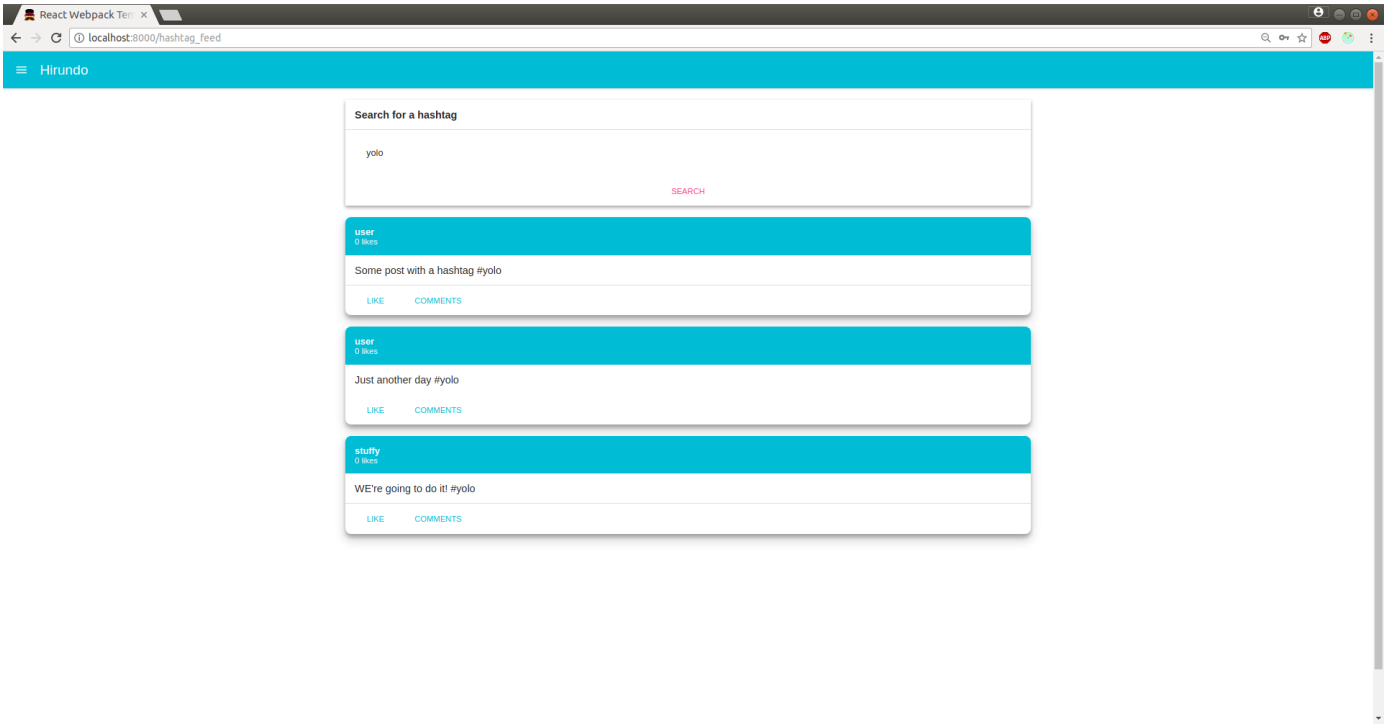


10.5. Дизайн на страницата за показване на коментарите

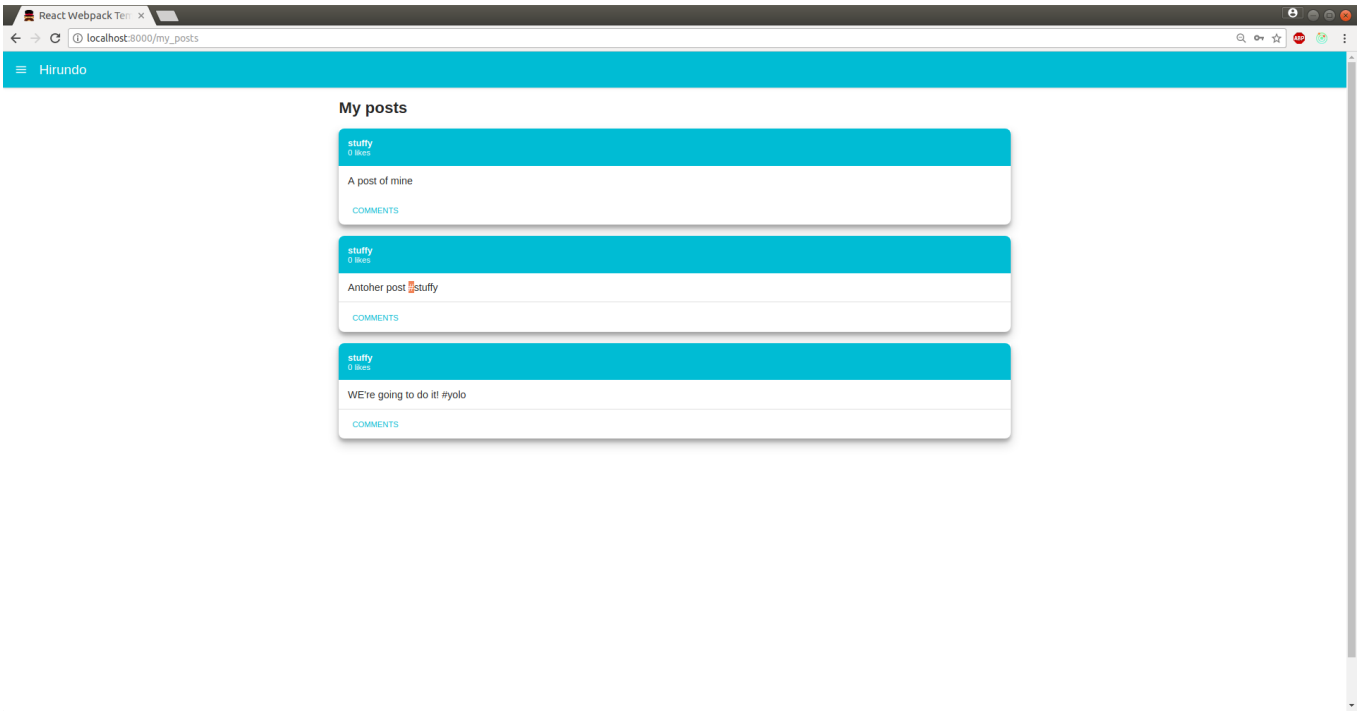
Също се създава диалог, в който се показва компонент за създаване на коментари към пост, а под нея и вече съществуващите коментари.



10.6. Дизайн на страницата за показване на постовете по търсен hashtag



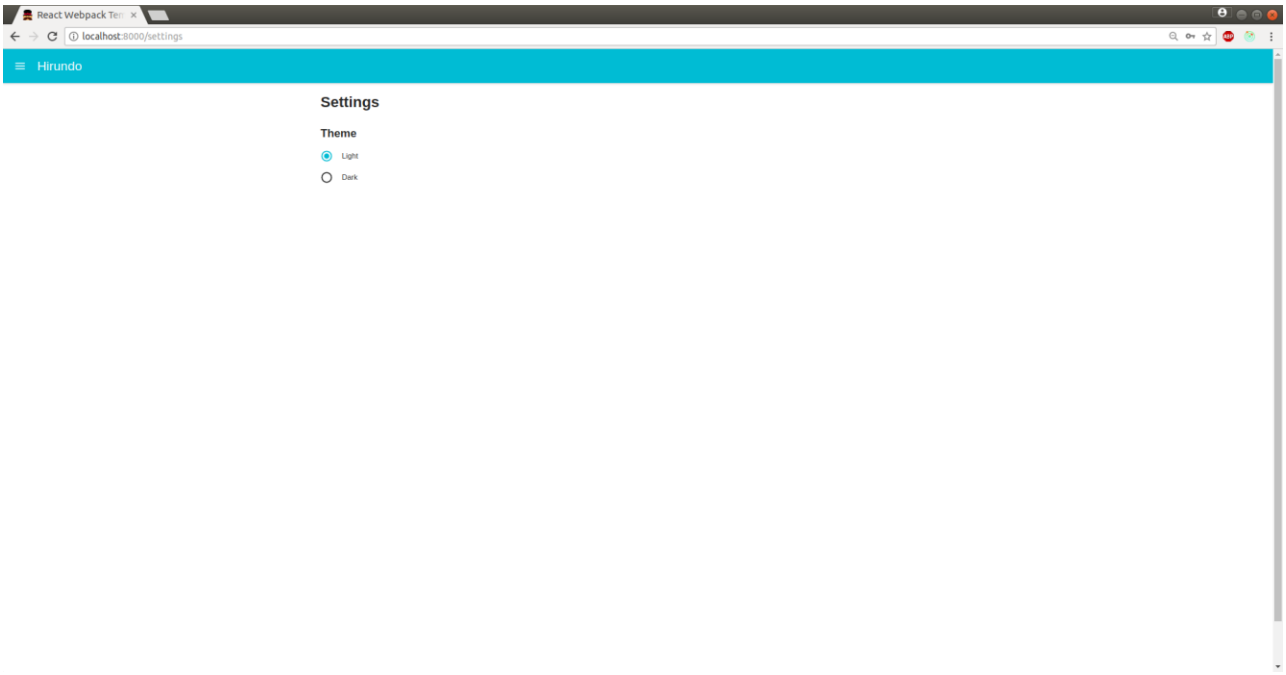
10.7. Дизайн на страницата за разглеждане на собствените постове



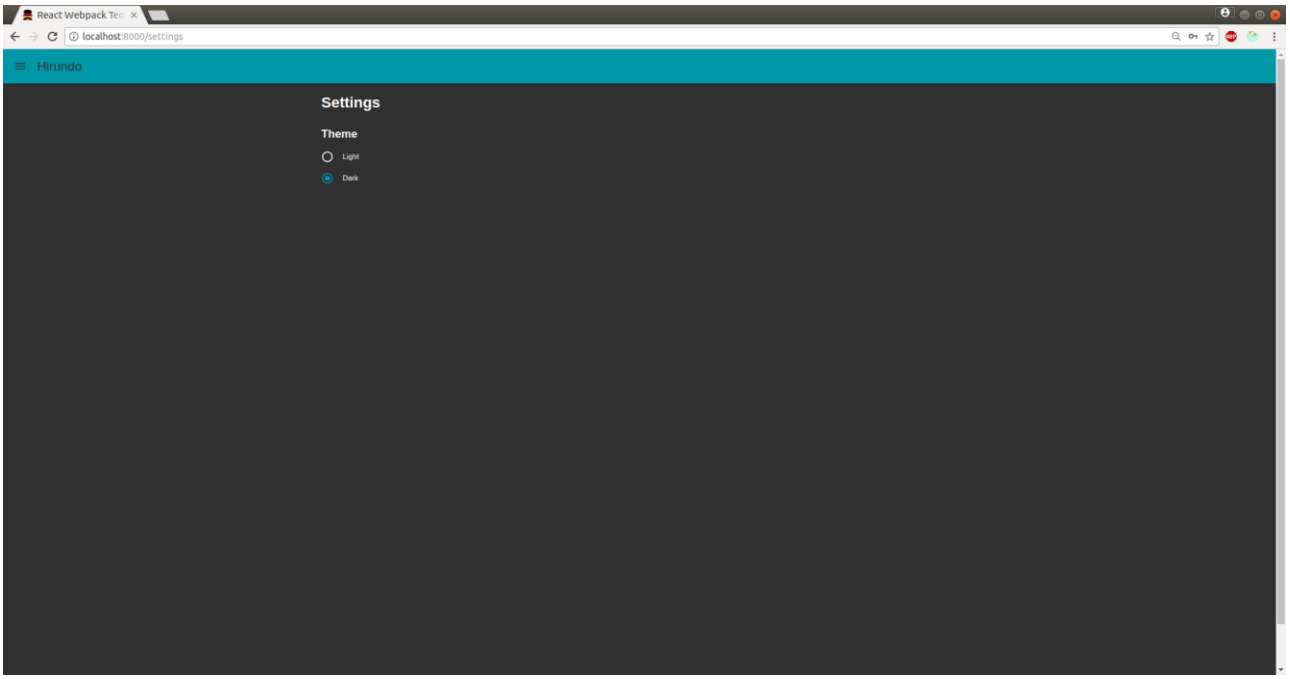
10.8. Дизайн на страницата за настройки

Това е страницата, на която потребителите могат да избират темата на системата

Светла тема:



Тъмна тема:



10.9. Дизайн на страницата за разглеждане на стастики за системата