פרויקט גמר
2019

Findel.site -

מנוע חיפוש שיתופי לתכנים לימודיים

המחלקה למדעי המחשב מכללת ספיר

מנחים: ד"ר שירה צוקר

מגישים

דין מורביה רן דוידאי

תוכן עניינים

תקציר

הבעיה אותה אנו רוצים לפתור:
ברשת האינטרנט קיימים אתרים ודפים רבים, המכילים מידע שגוי, שיווקי, מוטה (biased), ולעיתים לא רלוונטי כלל.
כיום קיימים מנועי חיפוש כמו גוגל, ואתרים לימודיים כגון ויקיפדיה, אך באחד חסרה התמקדות בתוכן החינוכי (דבר הגורם לתוצאות שאינם תמיד יובילו לתוכן לימודי), השני מספק מקור מידע יחיד שעשוי להיות מאוד מוגבל, ובשניהם אין משוב משתמשים עבור בקרת מידע ושאלת שאלות להרחבה על הנושא.

הפתרון שלנו:
אתר Findel נבנה על מנת למצוא אתרים חינוכיים, אמינים ורלוונטיים. לסנן אתרים פרסומיים, שיווקיים, ומוטים, המתחזים למקורות מידע אינפורמטיביים ואתרים אם תוכן שגוי.

האתר עושה זאת על ידי שימוש במנוע החיפוש של גוגל עבור תוצאות ראשוניות לחיפוש. לאחר מכן, בעזרת וויקיפדיה האתר מוצא מילות מפתח בנושא שבעזרתן מבצע סינון ראשוני עבור תכנים לימודיים, ונותן אופציות לחיפוש רחב יותר עבור נושאים קשורים.

לאחר קבלת התוצאות מוויקיפדיה וגוגל, האתר ישתמש בדירוגי משתמשים עבור שלושה מאפיינים (אהבו את התוכן, אמינות התוכן והאם התוכן חינוכי). דירוגי המשתמשים משפיעים באופן ישיר ומשמעותי על הדף אותו דירגו, ומשפיעים באופן עקיף על הדפים הנוספים, השייכים לאותו דומיין.

בנוסף לדירוג, יהיה ניתן להגיב על דפים או דומיינים, הקשורים לנושא, ולתגובות עצמן. באופן זה לא רק יהיה הדירוג שאומר אם הדף בעייתי, אלה מה בעייתי בו, וכן יהיה אפשר לשאול שאלות בנוגע לתוכן הדף.

מצב קיים

גוגל הינו אתר נהדר למציאת תכנים, אך אינו מקבל פידבק מהמשתמשים על אמינות מקורות המידע המופיעים בו, ובנוסף אינו אתר המתמקד בתכנים לימודיים ולכן קיימים תכנים להם קשה למצוא את האתרים הרלוונטיים.

וויקיפדיה יחסית אמין בשנים האחרונות, אך הוא מקור מידע יחיד וציבורי ועדיין יכולים להיות מקרים של אי דיוקים, או הטעיות על ידי אנשים מבפנים ומבחוץ.

Kidtopia הוא אתר המשתמש בgoogle custom engine לסינון רק עבור אתרים מסוימים. האתר מאוד נחמד, אך דורש לעדכן דומיינים באופן ידני, וגם אינו מספק פידבק ממשתמשים.

גלים הינו אתר בסגנון lemood אליו המורה מעלה תוכן לימודי עבור התלמיד. האתר אמין, אך אינו מספק מקורות ידע חיצוניים או שיתוף מידע באופן רחב.

קריטריונים	גוגל	ויקיפדיה	kidtopia.com	גלים	findel
אמינות המידע	✗	✓✗	✓	✓	✓
יכולת מציאת תכנים על נושא	✓	✓	✓	✓	✓
שיתופיות	✗	✓	✓	✗	✓
שמירת מקורות מידע	✗	✗	✗	✓	✓
חיפוש בטוח לתלמידים	✗	✓	✓	✗	✓
מאגר מידע עצמאי	✓	✓	✗	✗	✗
תמיכה בשפות	✓	✓	✗	✗	✗
מחיר	חינם	חינם	חינם	₪ 198	חינם

אופן מימוש הפרויקט

צד שרת: Node.js

Node.js הינה שפת צד שרת הרצה על בסיס מנוע JavaScript של Chrome V8.
בעזרתה בנינו REST API, המקבלת בקשות מצד הלקוח, מטפלת בהכנסה ושליפת מידע ממסד הנתונים, ומחזירה Response לצד הלקוח.

צד לקוח: React

React הינה ספריית JavaScript עבור בניית ממשק משתמש. השתמשנו בReact עבור הצגת המידע למשתמש, שליחת בקשות וקבלת תשובות ל REST API בשרת, ועבור out sourcing של עיבוד מידע, בצד המשתמש, ובכך לחסוך עלויות שרת.

מסד נתונים: MongoDB, עם שילוב Mongoose עבור מידול אובייקטים בצד שרת.

MongoDB הינו מסד נתונים NoSQL. בMongoDB נשמרים מסמכים באופן הדומה לJSON.

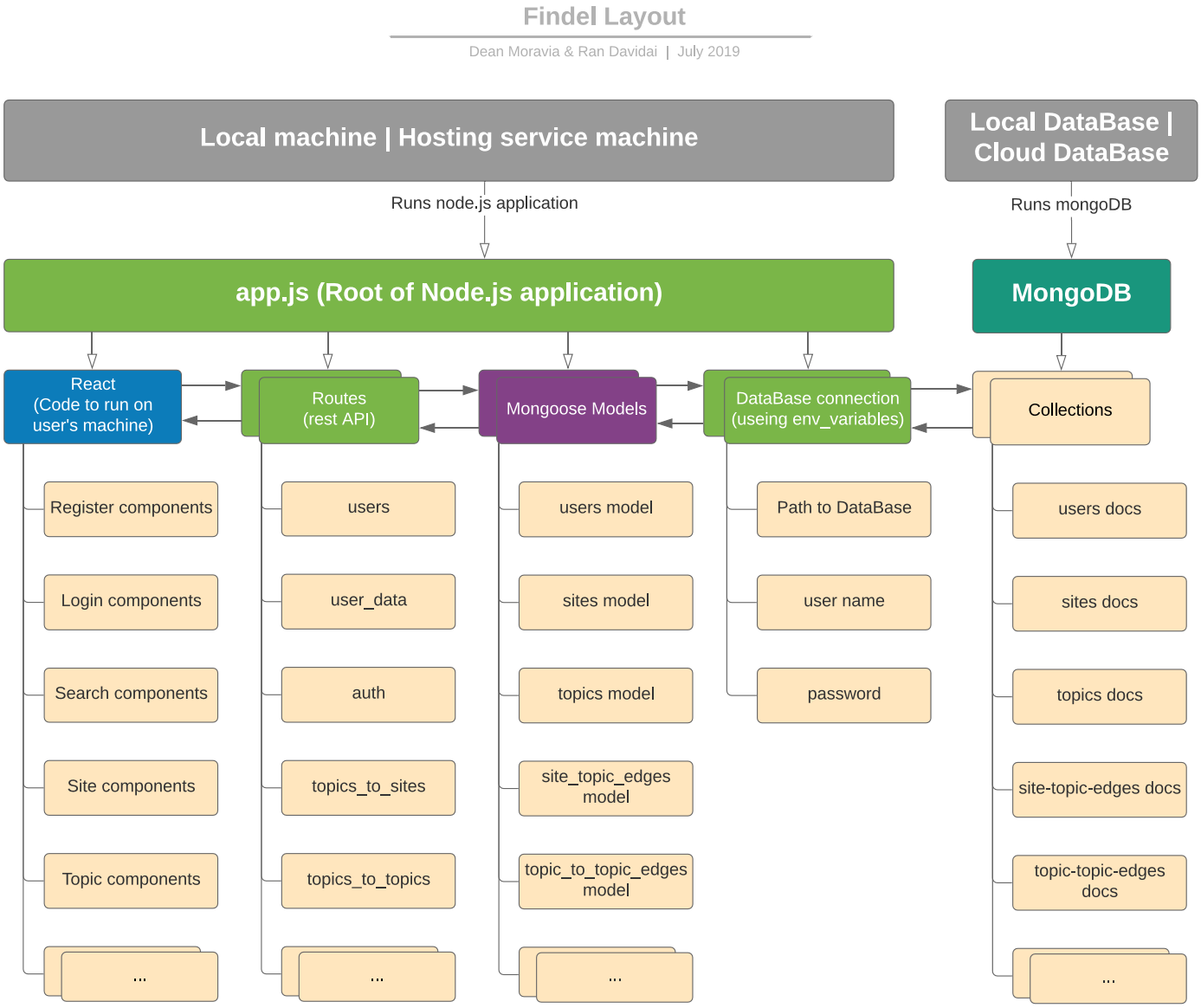
Mongoose הינו ספרייה עבור Node.js המאפשר ליצור תרשימים עבור אוספים של מסמכים, וככה לוודא את תקינות המידע הנכנס למסד הנתונים.

הפעלת הפרויקט:

הפרויקט יכול לעבוד בפלטפורמות שונות. אנו בחרנו להפעיל אותו באופן הבא:

- קוד הפרויקט שמור בGitHub של dean2400t@gmail.com תחת השם Findel.
- פתחנו משתמש בHeroku תחת dean2400t@gmail.com ובו פתחנו את האתר finde-site.
- קישרנו את הפרויקט בGitHub לfindel-site בHeroku וכל פעם שיש עדכון לMaster, האתר בHeroku מתעדכן אוטומטית.
- המסד נתונים נפתח בmLab תחת המשתמש dean2400t@gmail.com והכתובת, שם משתמש והסיסמא נשמרו במכונה עליה יושבת התוכנה של node.js תחת שם משתנה הסביבה "findel_db_connection".
- בנוסף, השתמשנו Custom Search Engine, של Google Developers Console, תחת המשתמש dean2400t@gmail.com על מנת לקבל תוצאות חיפוש מהAPI של Google

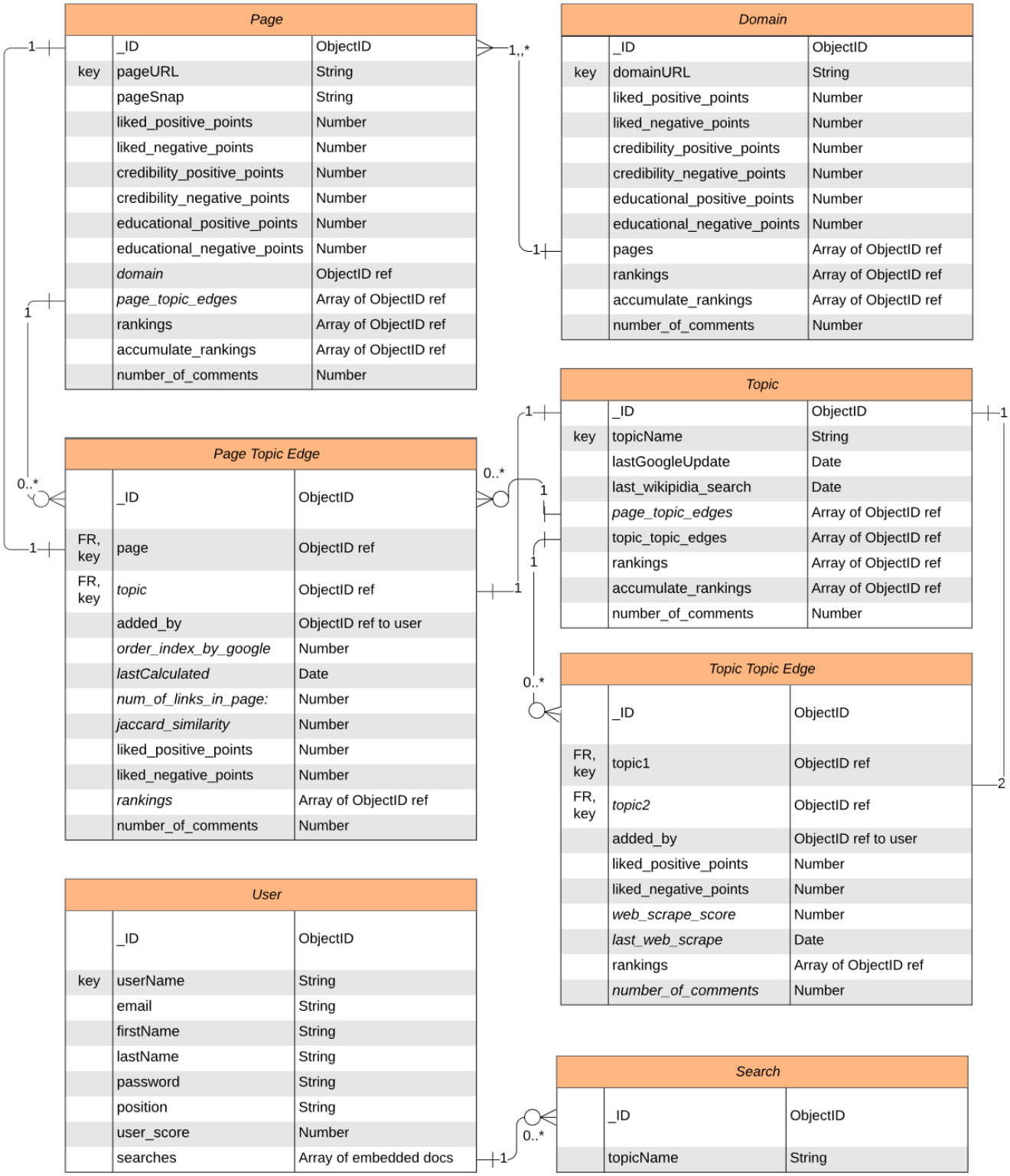
מבנה הפרויקט:



סכמת מסד הנתונים

Findel - DataBase Diagram

Dean Moravia & Ran Davidai | August 2019



Findel - DataBase Diagram

Dean Moravia & Ran Davidai | August 2019

Rankings		
	_ID	ObjectID
FR, key	object_id	ObjectID ref
key	object_collection_name	String
key	user	ObjectID ref
key	rank_type	String
	rank_code	Number
	score_added	Number

Accumulate Rankings		
	_ID	ObjectID
FR, key	object_id	ObjectID ref
key	object_collection_name	String
key	user	ObjectID ref
key	rank_type	String
	positive_rankings	Number
	negative_rankings	Number
	rankings	Array of ObjectID ref
	user_score	Number

1..*

1

Comments		
	_ID	ObjectID
FR, key	object_id	ObjectID ref
key	object_collection_name	String
key	user	ObjectID ref
key	text	String
	liked_positive_points	Number
	liked_negative_points	Number
	rankings	Array of ObjectID ref
	root_comment	ObjectID ref
	number_of_comments	Number

Rankings, Accumulate Rankings, Comments נרשמו לעבוד עם כל אוסף אפשרי. השילוב של שתי השדות הבאים שומר את המידע הדרוש על מנת להגיע לאובייקט הרלוונטי:

- **object_id-** הפנייה למזהה של מסמך מבין האוספים השונים, ההפנייה תהיה בעזרת object_collection_name.
- **object_collection_name-** שם האוסף אותו אנו מחפשים. משתמשים בפונקציה get_collection_from_collection_name הנמצאת בmodel על מנת לקבל את הmodel הרלוונטי.

הפעלת הפרויקט: Node.js -> app.js

React

קוד הNode.js של התוכנה רץ על שרת, ומפעיל עליו את הReact על ידי ניתוב לקוד הרלוונטי שצריך לרוץ אצל המשתמש:

בapp.js קוד ההפעלה:

```
const root = require('path').join(__dirname,
'client', 'build')
app.use(express.static(root));
app.get("*", (req, res) => {
    res.sendFile('index.html', { root }); });
```

Routes (rest API)

בNode.js יש גם ניתוב לRoutes של הAPI:

בapp.js:

```
require("./startup/routes")(app);
```

לאחר מכן בstartup/routes

```
const page=require('../routes/page');
```

הכוונה לקוד, בתיקיית routes, המטפל בבקשה

```
module.exports = function (app){
  app.use(cors());
  app.use(express.json());
  app.use('/api/page, page);
  …
```

בגרשיים מצוין הנתיב של הURL ובשמאל ההכוונה לקוד המטפל בבקשה.

לדוגמא, הקוד page , הנמצא בתיקיית routes, להזדהות המשתמש:

```
const retrieve_page_data = require('./retrieve_page_data');

router.get('/retrieve_page_data', async function(req, res) {
   var pageFormatedURL = req.query.pageURL;
   if (!pageFormatedURL)
      return res.status(400).send("No pageFormatedURL was sent")
   var token=req.headers['findel-auth-token'];
   var userID= checkAuthAndReturnUserID(token);
   return retrieve_page_data(pageFormatedURL, userID, res);
}
module.exports = router;
```

הנתיב retrieve_page_data הוא אחת הפעולות שניתן לבקש מהשרת. בנוסף גם קיימות פעולות נוספות כגון rank_page. אנו נבדוק שהבקשה קיבלה את

המידע הדרוש ולאחר מכן נעביר לפונקציה הנמצאת בקובץ רלוונטי, המטפלת בפרטי הבקשה אל מול המסד נתונים.

הקובץ retrieve_page_data המטפל בבקשה למידע על הדף

```
module.exports= async function retrieve_page_data(pageFormatedURL,
userID, res)
{
      var page = await Page.findOne({pageFormatedURL: pageFormatedURL})
      .select(page_selection(
         {
            include_edges: `page_topic_edges`,
            userID: userID
         }
      ))
      .populate(domain_populate())
      .populate(page_topic_edges_populate(
         {
            userID: userID,
            populate: topic_populate()
         }))
      .populate(page_usersRanking_populate({userID: userID}))
      .lean();

   if (!page)
      return res.status(400).send("Page " + pageURL +" not found in
database");

   return res.status(200).send(page);
}
```

Mongoose models:

המודלים יושבים תחת התיקייה models בשרת.

דוגמא למודל של page:

```
const Joi = require('joi');
Joi.objectId = require('joi-objectid')(Joi);
const mongoose = require('mongoose');

const pageSchema = new mongoose.Schema({
  pageURL: {
    type: String,
    required: true,
    unique: true, (חלק מהותי, אומר ששדה זה הוא ייחודי, ולא ניתן להכניס שניים עם שדה זהה)
    minlength: 4,
    maxlength: 1024
  },
```

```
pageSnap:{
  type: String,
  maxlength: 1024
},

liked_positive_points:{
  type: Number,
  required: true,
  default: 0
},
liked_negative_points:{
  type: Number,
  required: true,
  default: 0
},

credibility_positive_points:{
  type: Number,
  required: true,
  default: 0
},
credibility_negative_points:{
  type: Number,
  required: true,
  default: 0
},

educational_positive_points:{
  type: Number,
  required: true,
  default: 0
},
educational_negative_points:{
  type: Number,
  required: true,
  default: 0
},

domain:{ (הפנייה לדומיין, שדה המכיל מזהה ייחודי המפנה לדומיין)
  type: mongoose.Schema.ObjectId,
  ref: 'domains'
},

page_topic_edges:[{ (מערך של הפניות לקשתות בין האתר לנושאים הקשורים אליו)
  type: mongoose.Schema.ObjectId,
  ref: 'page-topic-edges'
}],
```

```
  edges_usersRanking:
    [{
      type: mongoose.Schema.ObjectId,
      ref: 'page-topic-edges-ranking'
    }],

  page_usersRanking:
    [{
      type: mongoose.Schema.ObjectId,
      ref: 'pages-ranking'
    }],

  number_of_comments:{
    type: Number,
    required: true,
    default: 0
  }
});

  const Page = mongoose.model('pages', pageSchema);

  function validateSite(page) { (פונקציה המוודא את תקינות המידע לפני הפנייה למסד נתונים)
    const schema = {
      pageURL: Joi.string().min(4).max(1024).required(),
      domain: Joi.objectId().required(),
      pageSnap: Joi.string().max(1024)
    };
    return Joi.validate(page, schema);
  }

  exports.Page = Page;
  exports.validate_page = validate_page;
```

בנוסף, עבור אבטחה ושליפה אחידה, יצרנו את התיקייה common_fields_selection בתוך models שמטרתה לטפל בשליפת השדות הרלוונטיים והpopulate בצורה אחידה, ובשמירה על שליחת מידע לא רגיש.

לצורך העזרה בselection מהמסד נתונים, אנו משתמשים בשתי פעולות מאוד חשובות אותם Mongoose מספק, בהם השתמשנו הרבה מאוד במהלך הפרויקט:

- **Populate**: פעולה המקבילה לLeft join ואוטומטית מחזירה את המסמך הרלוונטי מהאוסף המסופק בשדה "ref".
- **Lean**: Mongoose, כברירת מחדל ללא הפקודה Lean, ממיר את המסמך המבוקש מהמסד נתונים לאובייקט מסוג מודל. האובייקט מסוג model, מכיל את המידע על המסמך המבוקש ביחד עם פונקציות של

ספריית Mongoose, ופרטים נוספים. לרוב אנו לא נשתמש בפעולות אלה, וכן ההמרה לוקחת לנו זמן יקר.
עקב זאת, נתשמש בפקודת lean() שאומרת לMongoose לדלג על ההמרה ולהחזיר את המידע על המסמך כJSON.

דוגמא עבור selection: page_selections

```
const {page_usersRanking_populate} = require('./ranking_selections');

function page_selection(opts={})



{
   (שדות שאינם רגישים, ואינם מכילים מערך של אובייקטים נוספים העשויים להכביד על העברת הנתונים.
      בנוסף, במידה ונרצה להוסיף שדה חדש, לא נצטרך לעבור בכל מקום בו שלפנו מידע. כל שנצטרך הוא
                                                                                    להוסיף את השדה כאן)
    var selection = `
        pageURL
        pageSnap
        domain
        liked_positive_points
        liked_negative_points
        credibility_positive_points
        credibility_negative_points
        educational_positive_points
        educational_negative_points
        `
       (שדות המכילים מערך של אובייקטים, כגון קשתות, ייבחרו באופן פרטני כאשר נצטרך אותן)
    if (opts['include_edges'])
        selection+=opts['include_edges'];

       (כאשר יש לנו זהות של משתמש, אז נשלוף גם את הדירוג עבורו באופן אוטומטי)
    if (opts['userID'] != null)
        selection += `page_usersRanking`
    return selection;
}

 (populate הינה פעולה המקבילה לleft_join. אנו צריכים אותו כאשר מודל אחר מבקש לקבל את מידע עבור
                                                                                              דף)
function page_populate(opts={})
{
    var populate = {
        path: 'page',
        select: page_selection(opts), (שימוש בפונקציה מעלה)
        }

       (אפשרות לעשות populate בתוך הpopulate של הדף. למשל עבור דומיין או מערך קשתות)
    if (opts['populate'])
        populate.populate = opts['populate'];
```

```
    if (opts['userID']) (אם מסופק מזהה משתמש, אזי נחזיר את הדירוגים הרלוונטיים)
    {
        if (populate.populate == null)
            populate.populate = [];
        populate.populate.push(page_usersRanking_populate({userID:
opts['userID']}));
    }
    return populate;
}
```

באופן זה, השאילתות לשליפה מהמסד נתונים מתקצרות מאוד, ברורות הרבה יותר, ועלות השינויים במסד הנתונים קטנה באופן משמעותי.

DataBase connection:

הכתובת, שם המשתמש והסיסמא למסד הנתונים יושבים במכונה המארחת את תוכנת node.js שעובדת עליה. הם שמורים בEnvironment variables של המכונה.

בתיקיית config של הפרויקט בקובץ custom-environment-variables.json, רשומים שמות משתני הסביבה הנחוצים לפרויקט.

```
{
    "jwtPrivateKey":"findel_jwtPrivateKey",
    "db_connection": "findel_db_connection"
}
```

'findel_db_connection'. בהפעלת הפרויקט, הם נשלפים משם ונשמרים כמשתנה בספריית הMongoose של הפרויקט.

```
var db_connection=config.get('db_connection');
    if (!db_connection)
        var db_connection="mongodb://localhost/findel";
    mongoose.connect(db_connection)
    .then (()=> console.log("connected to mongodb"))
```

הפעלה -React

קוד הReact מתחיל בindex.js בו הוא מפעיל את App.js ,React, ReactDom
(קוד התוכנה אותו כתבנו) Bootstrap שירוץ בפרויקט.

בתוך הApp.js אנו מריצים רכיב מספריית הReact router dom בשם Route על מנת לעבור בין הדפים השונים באתר.
בכדי לעשות זאת, אנו נעשה Import לקבצים המכילים את הדפים, לדוגמא:

```
import SearchPage from './components/SearchPageComponents/SearchPage';
```

לאחר מכן, בתוך הreturn של רכיב הApp.js תחת רכיב Router אנו שמים את הנתיבים שלנו באתר. כל Route מכוון לקוד הדף עבור הכתובת הרלוונטית

```
<Router>
      <Route exact path="/" component={SearchPage} />
      <Route path="/Login_page" component={LoginPage} />
</Router>
```

כל קוד לדף נמצא בתיקיית components תחת תיקייה ייעודית עבורו.
בהמשך לדוגמא של הsearchPage, הנתיב יהיה
components/SearchPageComponents ובתוכו כל הרכיבים הדרושים עבור הצגת הדף. רכיבים ref_to_pages המטפלים באופן תצוגת קישור בין הנושא לדף מסוים, וגם כל הפונקציות לחישוב וטיפול בחיפוש תחת התיקייה search_functions.
App.js גם יכיל ויציג קישורים לדפים, אותם ייראו באופן תמידי בכל עמוד.

```
<text className="link_text" onClick={() =>
this.push_to_history_and_go('/')}>חפש</text>
```

React- State and Props

State- המצב בדף, משתנים ומערכים המשתנים עקב בקשות ופעולות משתמש נשמרים לנו בState של הדף עצמו, למשל

```
this.state = {
        search_text_box: search_text_box,
        pages_in_search:[],
        expandedContents: [],
        pages_ref: [],
        ambigousData: [],
}
```

בשלב זה, כאשר רק נכנסנו לדף, אין לנו state. אנו מאתחלים את הstate עם מערכים, המוצגים למשתמש, כמערכים ריקים, ולכן לא יוצג למשתמש כלום.
הsearch_text_box לא בהכרח יהיה כי עשוי לקבל ערך מקישור לחיפוש עם ערך מסוים או שמירה להיסטוריה.

המערכים יוצגו למשתמש בעתיד לאחר שנקבל נתונים מהשרת. שינוי הstate יביא לשינוי באופן אוטומטי לשדות this.state.<field> הרלוונטים ויטענו בדף:

```
<div hidden={!this.state.is_topic_loaded} style={{textAlign: 'right'}}>
    <Comments_loader data_for_comments={this.state.data_for_comments}/>
</div>
<div id="pages_ref">
    <Pages_ref pages_ref={this.state.pages_ref}/>
    <button onClick= {() => this.more_pages_clicked()}
     hidden={this.state.is_more_pages_button_hidden}> אתרים
     נוספים...</button>
</div>
<div id="pages">
    <Pages_in_search
     pages_in_search={this.state.pages_in_search}/>
</div>
<div id="ambigous">
     <Ambigous ambigousData={this.state.ambigousData}/>
</div>
```

כל הרכיבים Pages_ref, Pages_in_seach נמצאים בComponents משלהם, ומקבלים את המידע מהState לאחר שאנו מקבלים מידע מהשרת ומבצעים setState הממלא את המערך הרלוונטי.

Props- Props הוא מידע, הנתון לשינויים, שרכיב מקבל מרכיב אב. למשל הרכיב Pages_ref המקבל את this.state.pages_ref לתוך המשתנה pages_ref.
לכן, כאשר המערך בתוך this.state.pages_ref ישתנה, אז יתעדכן באופן אוטומטי המערך בתוך Pages_ref מכיוון שיהיה תחת הprops.

משתמשים ואבטחה

יצירת משתמש

המשתמש נוצר תחת api/users/createxxxx

המפתח לuser הוא הuserName שהינו ייחודי.

מייל גם כן ייחודי, אך אינו שדה חובה.

```
req.body.position="Student";
  user = new User(_.pick(req.body, ['email', 'userName', 'firstName',
'lastName', 'password', 'position']));

  user.user_score=1;

  const salt = await bcrypt.genSalt(10);
  user.password = await bcrypt.hash(user.password, salt);

  await user.save();

  const token = user.generateAuthToken();
  return res.status(200).send({token: token, userName: user.userName});
```

אנו משתמשים בbcrypt לביצוע hash על סיסמאות המשתמשים לפני שמירתם במסד הנתונים. השימוש בsalt נועד עבור חוסן לתקיפת Rainbow table בה משתמשים בטבלה המחושבת מראש לטובת פיצוח הצפנת פונקציית hash.

לאחר שמירת המשתמש וגם בתהליך ההתחברות, נשתמש בפונקציה generateAuthToken הנמצאת בתוך הmodel של users.

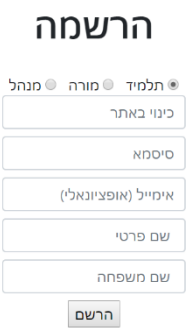
```
userSchema.methods.generateAuthToken = function() {
    var jwtPrivateKey=config.get('jwtPrivateKey');
    const token = jwt.sign({ _id: this._id, email: this.email,
password: this.password, position: this.position}, jwtPrivateKey);
    return token;
  }
```

ההתחברות מחדש קורת בapi/auth לשם המשתמש שולח שם משתמש וסיסמא, ואם הכול תקין, אזי מקבל token

```
const validPassword = await bcrypt.compare(req.body.password,
user.password);
  if (!validPassword) return res.status(400).send('שם משתמש או סיסמא
שגויים');

  const token = user.generateAuthToken();
  var dataToSend= {
                      token:token,
                      userID:user.id
                    };
  res.status(200);
  res.send(dataToSend);
```

דוגמא לאבטחה- דף הרשמה:



בדף זה ניתן לבחור איזה סוג משתמש לרשום.
באופן רגיל ניתן להירשם כתלמיד ללא רישום קודם לאתר, אך עבור הרשמת מורה יש להיות מחוברים כבר כמורה או כמנהל (מורה או מנהל רושם מורה חדש) ועבור הרשמת מנהל נדרש להיות מנהל.

האבטחה עובדת באופן הבא:

```
router.post('/createTeacherAccount', [auth, isAdminOrTeacher],async
(req, res) => {

  const { error } = validate(req.body);
```

בכניסה לנתיב הרישום תחת api/users/createTeacherAccount מופעלות שתי פונקציות אחת אחרי השנייה.

הראשונה auth: בודקת האם הtoken בו משתמש המשתמש תקין בהתאם להצפנה הפנימית של השרת. זאת תחת config.get('jwtPrivateKey') המכוון אותנו לenvironment variable היושב במכונה עליה רץ הnode.js.

```
module.exports = function (req, res, next) {
  var token;
  if (req.headers['findel-auth-token']!=null)
    token = req.headers['findel-auth-token'];
  if (!token)
  {
    return res.status(401).send('יש להתחבר על מנת לבצע פעולה זאת');
  }

  try {
    const decoded = jwt.verify(token, config.get('jwtPrivateKey'));
    req.user = decoded;
    next();
  }
  catch (ex) {
    //res.redirect(`/authPage`);
    return res.status(400).send('Invalid token.');
  }
}
```

אם הtoken תקין, אזי עוברים לשלב הבא, הפונקציה isAdminOrTeacher

```
module.exports = function (req, res, next) {
    // 401 Unauthorized
    // 403 Forbidden

    if (req.user.position!="Admin" && req.user.position!="Teacher")
      return res.status(403).send('Access denied.');
    next();
  }
```

אם המשתמש אינו admin או אינו teacher אנו מחזירים לו Access denied.

אבטחה זאת, ניתן לבצע בכניסה לכל נתיב בתוכנה.
במקרים בהם אנו רוצים לוודא שהtoken תקין, אך רוצים להמשיך גם אם המשתמש אינו מחובר, אזי נשתמש בפונקציה checkAuthAndReturnUserID היושבת בmiddelware

```
module.exports=function checkAuthAndReturnUserID(token)
{
  try {
    const decoded = jwt.verify(token, config.get('jwtPrivateKey'));
    return decoded._id;
  }
  catch (ex) {
    return null;
  }
}
```

היא עושה בדיקה, ומחזירה null במקרה שהtoken אינו תקין. ככה במקרים בהם משתמש עם token מזויף יכול לראות את המידע הציבורי, אך לא את המידע של המשתמש עליו ניסה להשיג מידע.
מקום בו יש שימוש הוא בקבלת תוצאות חיפוש והדירוגים של המשתמש. ככה יוחזרו רק תוצאות החיפוש ללא הדירוגים.

תהליך החיפוש

הרעיון מאחוריי:

הרעיון הוא להשתמש בוויקיפדיה על מנת לשלוף נושאים הקשורים לנושא, ולהכניס אותם לקשתות topic topic edges.
לאחר מכן לבצע בגוגל חיפוש של אתרים הקשורים לנושא ולהכניס אותם לpage topic edges.
לשמור את תאריך החיפושים תחת last_wikipedia_search עבור נושא אל מול וויקיפדיה, וlastGoogleUpdate עבור נושא אל מול דפים. זאת על מנת לחסוך זמן ריצה, ושימוש במשאבים של גוגל באופן יעיל.

מטרה אחת היא שסדר האתרים שיוצגו למשתמש יהיה בהתאם לנתונים שאנו מהחסנים בקשתות, ובמידע על הדף והדומיין

בקשתות:
order_index_by_google: הסדר על פי גוגל.
num_of_links_in_page: כמות המופעים בדף של נושאים הקשורים לנושא.
Jaccard_similarity: הדמיון בין הדף לבין העמוד ויקיפדיה.
rankings: דירוגים ישירים לקשת.

בדף ובדומיין: דירוגים לאמינות וחינוכיות.
בנוסף, בזכות דירוג זה, ניתן לערוך את מנוע החיפוש של גוגל להימנע לחפש באתרים מסוימים, ובכך להוריד מראש אתרים לא חינוכיים או אמינים להם הצטבר דירוג שלילי (זאפ, yad2, ynet, ksp וכדומה).

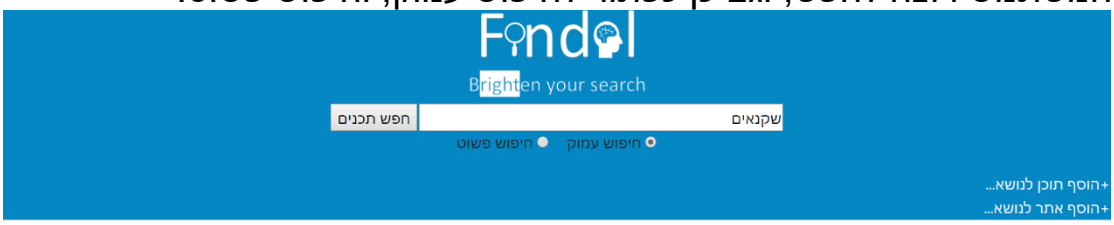
בשלב זה, מניסוי וטעיה, אנו שמים את הדגש הרב ביותר על דירוג ישיר של המשתמשים לקשת, אחריו הסדר שגוגל סיפק, אז לדירוג המצטבר של האתר והדומיין, ואז לחיפוש טקסט.
הסיבה שהחיפוש טקסט אינו אמין, היא שאתרי פרסומות עשויים להכיל הרבה מאוד טקסט הקשור לעמוד ויקיפדיה, ובכך לתת תוצאות לא רלוונטיות (למשל עבור החיפוש של "מחשב" או "טלוויזיה").
אם זאת, חיפוש הטקסט משמש למטרה השנייה שלנו, חיפוש תכנים קשורים:

המטרה השנייה שלנו היא לחפש נושאים הקשורים לנושא אותו חיפשנו. לשם כך נשתמש במספר ההופעות, של הנושא הקשור, בתוך ה6 אתרים אותם אנו מחשיבים כקשורים ביותר לנושא.
התוצאות עם חיפוש זה הן לרוב טובות.

תהליך החיפוש, צד לקוח:

תהליך החיפוש מתחיל בReact בSearchPageComponents.
הקובץ SearchPage.jsx מטפל בנראות, בסדר הדף, וכן בקריאה לפונקציה הראשית main_search_function בתיקייה search_functions המטפלת בחיפוש.

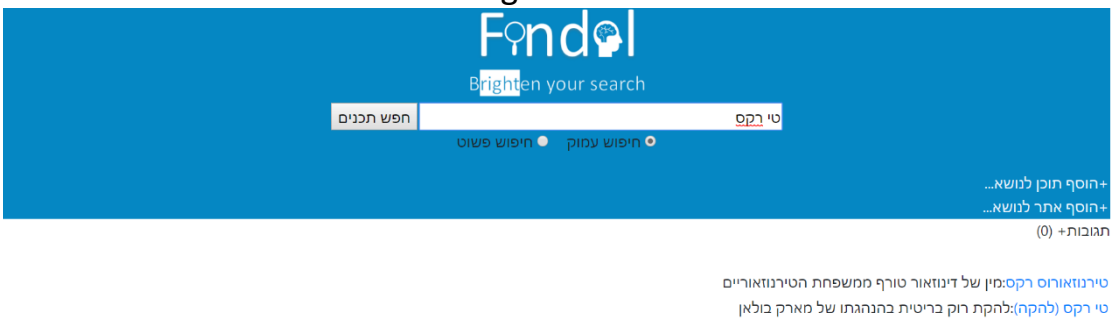
בתוך הSearchPage קיים השדה "search_box_text" המכיל את הנושא אותו המשתמש רוצה לחפש, וגם כן כפתור לחיפוש עמוק, וחיפוש פשוט.



חיפוש עמוק- ייתן ערך "אמת" לחיפוש RabinKarp, וJaccard similarity במידה ונמצא עמוד ויקיפדיה מתאים.
חיפוש פשוט- ייתן ערך "שקר" ויציג את המידע מיד לאחר קבלת האתרים ונושאים קשורים מהשרת. הם יקבלו תוצאות חיפוש מחיפושים קודמים (אם בוצעו).
לאחר לחיצה על כפתור החיפוש, הדף יפנה אותנו לmain_search_function.

main_search_function:

- עבור תחילת התהליך, נפעיל:
  `request_topic_and_connected_topics_from_server`
  הפונקציה תבקש מהשרת לחפש האם קיים עמוד ויקיפדיה, לעדכן תכנים קשורים בהתאם (topic topic edges) ולהחזיר את המידע שנמצא.
  *אם בוויקיפדיה נמצא שהנושא יכול להתכוון לנושאים רבים, אזי נציג זאת למשתמש תחת AmbigousContents ונסיים את החיפוש.



- לאחר שהוחזר המידע על הנושא והנושאים הקשורים אליו מהשרת, נפעיל את הפונקציה:

```
request_pages_from_server
```

הפונקציה תקבל את הנושא אותו אנו מחפשים, ותבקש מהשרת לחפש אתרים רלוונטיים בגוגל וליצור ולהחזיר קשתות page topic edges בין האתרים לנושא. קשתות אלה יקבלו את משקלם מהדירוגים ותוצאות החיפוש העמוק בין הנושא ועמוד הוויקיפדיה שלו לדפים.
הפונקציה תשתמש בפונקציה sort_page_topic_edges, המשתמשת במידע שיש לנו עד כה, על מנת לסדר את הדפים בהתאם למידע נמצא בקשתות ובמידע על הדף והדומיין ממסד הנתונים. המידע כולל דירוגים, תוצאות מחיפושים קודמים: כמה נושאים הקשורים לנושא מופיעים בדף והדמיון בין דף הוויקיפדיה לדף שגוגל סיפק "jaccard similarity", וכמו כן סדר ההופעה על פי חיפוש גוגל.

- אם אין נושאים קשורים, אזי נציג מיידית את המידע על האתרים שנמצאו. אחרת, נעשה חיפוש עמוק:

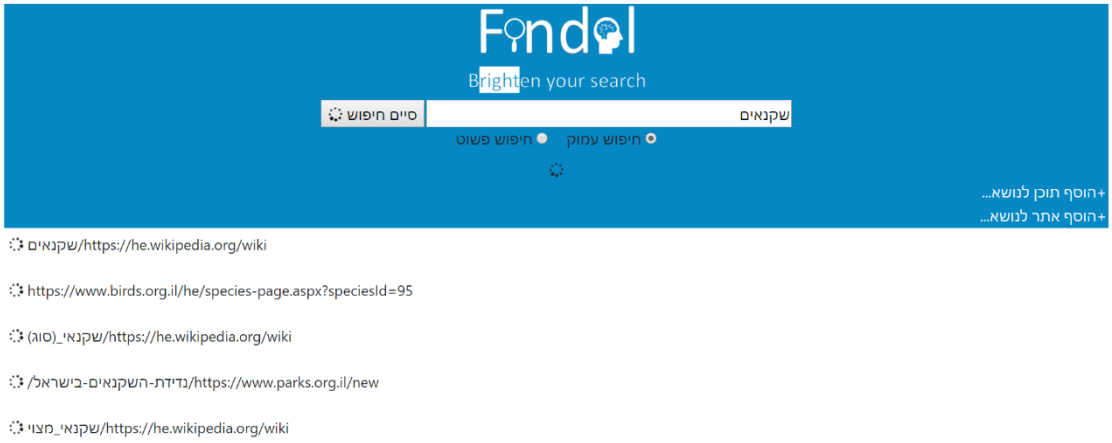
חיפוש עמוק:

<u>תהליך החיפוש:</u>

- נפעיל את הפונקציה:

```
display_pages_scrape_process
```

כל שהפונקציה עושה היא לאתחל את pages_in_search שיציג את כתובת הדפים עם icon מסתובב המראה שמחפשים בהם תכנים.



- לאחר מכן נאתחל את הRabinKarp והJaccard similarity ונבנה בהם את הטבלאות לעמוד ויקיפדיה (הסבר מפורט על שניהם לאחר הסבר התהליך).

- נפעיל את הפונקציה refreshSearchStatus שעובדת באינטרוולים של 0.75 שניות והמעדכנת את מצב החיפוש למשתמש. לאחר שכולם סיימו חיפוש, היא תפעיל פונקציות לסידור מחדש של הדפים, שליחת התוצאות לשרת, והצגת המידע למשתמש.

- נפעיל את web_scrape_pages בו אנו מבקשים מהשרת לבדוק האם התוצאות חיפוש לקשת עדכניות, אם כן להחזירן, ואם לא, אזי להחזיר את הדף אינטרנט תחת הכתובת pageURL.
  אם קיבלנו את תוכן הדף, אנו נבצע חיפוש טקסט בעזרת שימוש במחלקות RabinKarp וJaccard similarity המכילות את המידע מוויקיפדיה.

- לאחר שסיימנו עם חיפוש הדף, נחליף את סימן החיפוש בpages_in_search לv, אם הצלחנו, ולx במידה והייתה בעיה.

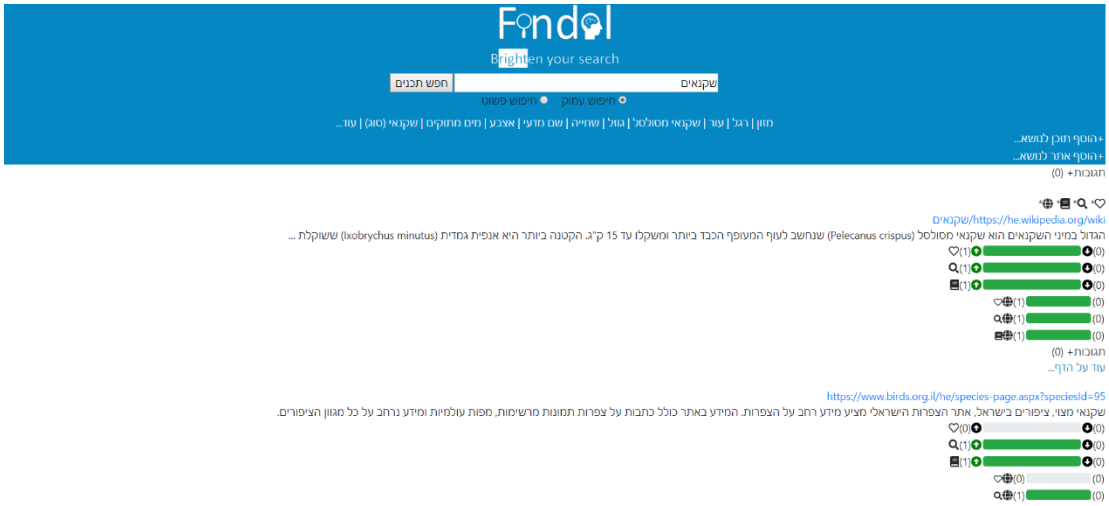
- לאחר שקיבלנו את כל התוצאות, הפונקציה refreshSearchStatus נכנסת לשלב השני שלה:

<u>תוצאות החיפוש:</u>

- לאחר שקיבלנו את התוצאות של הקשתות page topic edges אנו נפעיל את rank_pages על מנת לסדר מחדש את הדפים, בהתאם לתוצאות, ולהציגם למשתמש.

- הפונקציה search_for_expanded_content תבדוק האם יש קשתות הצריכות עדכון, ואם כן, אזי תפעיל את הפונקציה best_pages_results שתלך ל6 דפים עם הציון הכי גבוהה, תשקלל את תוצאות החיפוש של הRabin-Karp search לנושאים הקשורים, ותשלח לשרת את התוצאות עבור כל נושא קשור.
  בנוסף, הפונקציה תבדוק אם אחד או יותר מששת הדפים המדורגים כטובים ביותר לא עבר חיפוש מכיוון שקיבל תוצאות לקשר בינו לנושא בזמן האחרון. במידה וזה המצב, הפונקציה תיזום web_scrape עבור הדף/דפים.

- בסופו של דבר נפעיל את הפונקציה display_expanded_content שתציג למשתמש את התכנים הנוספים הרלוונטיים לאחר החיפוש



ניתן לראות כי מצאנו נושאים קשורים, אך גם נמצאו נושאים המופיעים הרבה באתרים הנוספים, אך אינם מתאימים. בשלב זה נכנס הדירוג:
על ידי לחיצה על "עוד..." בסוף הנושאים הקשורים, אנו נכנסים לדף המציג ונותן לדרג נושאים

מים מתוקים
דירוג משתמשים: (0)(0)
ציון מחיפוש באתרים: 4
חיפוש אחרון: 2019-09-01T17:33:29.454Z

זנב
דירוג משתמשים: (0)(0)
ציון מחיפוש באתרים: 2
חיפוש אחרון: 2019-09-01T17:33:29.454Z

טגואניים
דירוג משתמשים: (0)(1)
ציון מחיפוש באתרים: 2
חיפוש אחרון: 2019-09-01T17:33:29.454Z

כנף
דירוג משתמשים: (0)(0)
ציון מחיפוש באתרים: 2
חיפוש אחרון: 2019-09-01T17:33:29.454Z

נחשוניים
דירוג משתמשים: (0)(1)
ציון מחיפוש באתרים: 2
חיפוש אחרון: 2019-09-01T17:33:29.454Z

לאחר הדירוג:

באותו אופן ניתן לקדם ולהוריד את הקשת המחברת בין אתר לנושא.

Rabin-Karp search

- על מנת לאתחל את החיפוש, צריך את המשתנים הבאים:
  - Q- מספר ראשוני גדול עבור גודל המערך אליו מבצעים hash.
  - R- כמות האותיות בw3schools רשום שיש שיכולים להיות עד 10176 סמלים בcharacter code, זה יותר מידי, אך מבטיח שזאת לא תהיה הסיבה לתקלה, ולכן זהו המספר שבחרנו ככמות האותיות הפוטנציאליות בא"ב.
  - המספר המקסימלי של תווים במחרוזת אותה אנו מחפשים. אנו בחרנו 20 תווים במחרוזת, כלומר לא נחפש לינקים של וויקיפדיה לנושאים המכילים יותר מ20 אותיות.
  - מספר דפים לחיפוש- כל דף מקבל מערך משלו המכיל 20 מערכים עבור כל טבלה למספר אותיות.
- בניית טבלה מהטקסט בוויקיפדיה hashWikiLinks:
  - אנו נאתחל 20 מערכים, אחד עבור כל כמות אותיות.
  - עבור כל מחרוזת של לינק מוויקיפדיה, נבדוק האם היא קטנה מ20, ואם כן נאתחל אותה עם:
    - השם
    - האינדקס במערך המקורי של הconnected_topics_edges שישמש כסמן אליו נשלח את התוצאות כשיגיע הזמן.
    - hash המאותחל ל1- כי עדיין לא חישבנו אותו
    - מערך בגודל מספר הדפים, השומר כמות הופעות בכל דף.
  - עבור כל link נחשב את הhash עבור השדה hash. לאחר מכן נכניס את הlink לאחת מ20 הטבלאות בהתאם לאורך המחרוזת לתא עם הערך hash שלו. הוא ישמר עם האינדקס שלו במערך הlinks כמצביע אליו, ועם הstring של המחרוזת כזיהוי במקרה שיש התנגשות hash.
- createHashTables:
  - עבור כל אות מ1 עד 20 הכנס:

```
var hashTableLength=text.length-curNumOfChars+1;
            var hashedTable=[];
            var txtHash = this.hash(text, curNumOfChars);
            hashedTable[0]=txtHash;
            for (var index = 1; index < hashTableLength; index++) {
                txtHash = (txtHash + this.Q -
                this.RMarray[curNumOfChars-1]*text.charCodeAt(index-1)%
                this.Q) % this.Q;
```

```
txtHash = (txtHash*this.R + text.charCodeAt(index-
1+curNumOfChars)) % this.Q;
hashedTable[index]=txtHash;
}
this.pages_hashes[pageIndex][curNumOfChars-1]=hashedTable;
```

עבדנו על בסיס הנוסחה שנלמדה בכיתה, ושינינו אותה בכך שאנו מכניסים את מערך התוצאות למערך המספר אותיות הרלוונטי

- add_hits_from_pages מופעל עבור כל דף:
  - עבור כל מערך של אורך מחרוזת:
    - עבור כל תא במערך המכיל hash של מחרוזת, בהתאם לכמות האותיות:
      - אם הערך מצביע לתא במערך (מאותה אורך מחרוזת) של לינקים מוויקיפדיה, אזי תרכיב את המחרוזת הרלוונטית מטקסט האתר, ואם היא תואמת לאחד התאים, אזי תוסיף 1+ למופע זה תחת האתר.

- בסוף פעולה זאת, יהיה בwikiLinks_hashes_to_words_by_length יהיו 20 מערכים, ובכל אחד מהם מערך של המחרוזות עם כמות המופעים שלהן עבור כל דף.

Jaccard similarity

חיפוש כלל הטקסט מוויקיפדיה עם כלל הטקסט של כל דף, ובסופו של דבר נותן ציון על פי החיתוך חלקי האיחוד של הסט של מספר אותיות או מספר מילים הנקבע מראש (K). אנו החלטנו להשתמש ב10 אותיות כK שלנו.

החיפוש הוא בעזרת האלגוריתם של Rabin Karp, אך במקרה זה אנו עושים זאת בין שני טקסטים שלמים, והQ שלנו הוא עכשיו 20399 בגלל שיש יותר סיכוי להתנגשויות עקב יותר מילים.

עבור כל סט יש לנו ערך בינארי 1/0 להאם הסט נמצא בוויקיפדיה או לא, וערך דומה עבור האם הערך נמצא בדף. לאחר שמילאנו את כולם, אנו עוברים על הסטים ועבור 1, 1 נוסיף 1+ גם לחיתוך וגם לאיחוד, ועבור 1, 0 או 0, 1 נוסיף 1+ רק לחיתוך. בסוף אנו מקבלים את התוצאה הסופית מהחיתוך חלקי האיחוד שסכמנו.

תהליך החיפוש, צד שרת

בצד השרת קיימים שתי נתיבים לצורך החיפוש:

```
/api/topic_topic_edges/search_for_connected_topics_in_db_and_wikipedia
/api/page_topic_edges/update_and_retrieve_topic_to_pages_edges_using_google
```

search_for_connected_topics_in_db_and_wikipedia

- מתחילים עם חיפוש הנושא במסד נתונים.
- אם לא קיים, אזי יוצרים אחד חדש.
- אם המשתמש מחובר, אזי שומרים את החיפוש למסד הנתונים.
- משתמשים בwtf_wikipedia ספרייה המטפלת בשליפת מידע מהapi של וויקיפדיה ומחזירה את התוצאות כjson.
- עם התשובה מוויקיפדיה יכולים לקרות שלושה דברים:
  - יש דף וויקיפדיה.
  - ערך לא ברור ורשימה של ערכים אפשריים.
  - לא קיים דבר בוויקיפדיה על החיפוש.
- במקרה הראשון מתחילים לחפש ולשמור לינקים.
- במקרה השני מחזירים למשתמש רשימה של נושאים אפשריים אותם וויקיפדיה הציעה.
- בשלישי, שולפים את כל מה ששמור לנו על הנושא, ושולחים למשתמש.

- שליפת נושאים קשורים מויקיפדיה:
  מעמוד וויקיפדיה יש לינקים בתוך הטקסט המקשרים בין הנושא העכשווי לנושאים קשורים. אנו משתמשים בהם על מנת לקבל נושאים פוטנציאלים הקשורים לנושא המדובר.
  - שמים את הלינקים במערך.
  - ממיינים אותם
  - מורידים לינקים כפולים מהמערך הממוין.
  - מתחילים ליצור קשתות בין הנושאים.

- יצירת קשתות בין הנושאים:
  עבור יצירת הקשתות, אנו משתמשים בפונקציה update_connected_topics_using_wikipidias_links המקבלת נושא ומערך לינקים.
  - נבדוק האם ביצענו חיפוש בזמן האחרון. אם כן, אזי נדלג על החיפוש במסד הנתונים. אם לא, נמשיך:
  - נשלוף ממסד הנתונים את כל הנושאים שהנושא הנוכחי מקושר אליהם.
  - נעשה חיפוש בינארי על מנת למצוא את הנושאים שלא מחוברים לנושא הנוכחי.

- אם הנושא מחובר, אזי נעבור לנושא הבא, אחרת ננסה לשמור נושא חדש למסד הנתונים, ואם יוחזר כי כבר קיים, אזי נשלוף את הנושא הקיים.
- נחבר בין הנושאים בtopic_topic_edge חדשה, כאשר הtopicName בעל ערך המחרוזת הקטנה יותר יהיה בtopic1, והשני בtopic2.

לאחר שסיימנו את הפעולה, נשתמש בפונקציה
`get_topic_and_connected_topics_edges_for_search`
הפונקציה שולפת ממסד הנתונים את הנושא, הקשתות אל מול הנושאים האחרים, הנושאים האחרים, והדירוג של המשתמש (אם מחובר) לכל קשת. לאחר מכן משתמשת בזמן שנקבע מראש על מנת להחליט האם יש צורך לעדכן את תוצאות החיפוש עבור כל קשת, ושולחת את המידע למשתמש.

<u>update_and_retrieve_topic_to_pages_edges_using_google</u>

- בודקים האם הנושא קיים במסד הנתונים, ויוצרים אותו במקרה הצורך.
- בודקים האם בוצע חיפוש גוגל, בזמן האחרון, עבור הנושא.
- אם בוצע חיפוש, אזי מחזירים את הpage topic edges והמידע שלהן. אם לא, אזי מתחילים את תהליך החיפוש בגוגל בעזרת הפונקציה
  `search_Google_and_orgenize`

<u>search_Google_and_organize:</u>

- שימוש בפונקציה googleSearch המקבלת ערך לחפש ומספר דפים למצוא, על מנת לקבל את התוצאות חיפוש. מהתוצאות אנו לוקחים את הpageURL, siteSnap, order_index_by_google.

- לאחר מכן אנו נשתמש בפונקציה
  `add_new_pages_and_return_pages_from_database`
  הפונקציה מבקשת את כל הדפים מהמסד נתונים עם pageURL שגוגל סיפק. ולאחר חיפוש אילו דפים לא נמצאים, מוסיפה דפים חדשים למסד נתונים (כולל דומיינים חדשים), ומחזירה את הדפים שאוסיפה ביחד עם הדפים שכבר היו.

- עכשיו כאשר יש לנו את הדפים ממסד הנתונים, נשלוף את הקשתות page topic edges מהנושא ונבצע חיפוש, עבור כל דף, האם קיימת קשת בינו לבין הנושא. אם קיימת, אזי נעדכן את index_google עבורה, ובמידה ואין קשת כזאת, ניצור אחת חדשה ונכניס לתוכה את המידע.

לאחר שסיימנו את החיפוש, או שכבר היה חיפוש עדכני, נשתמש בפונקציה
`retrieve_topic_to_pages_edges_from_topic`
הפונקציה מחזירה את הקשתות page topic edges, הדף תחת כל קשת page והדומיין של הדף domain. כמו כן, אם המשתמש מחובר, אזי נמלא בrankings את הדירוגים שלו עבור כל אחד מהאוספים

דירוג ישיר ודירוג מצטבר

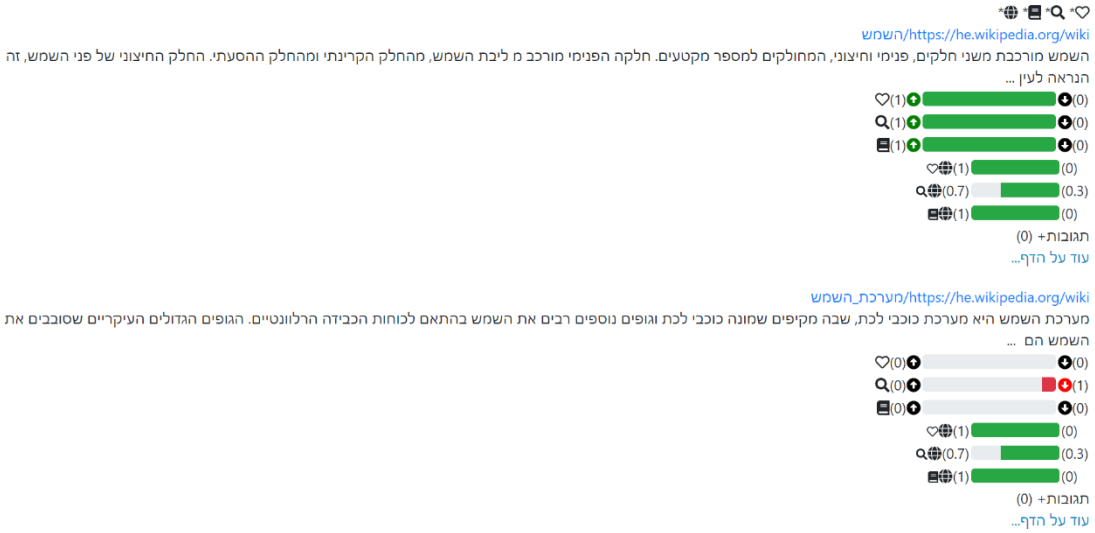
הרעיון מאחוריי

קיימים שלושה סוגי דירוגים: "credibility"ו "educational" בהם משתמשים לדירוג ישיר של דף ודירוג מצטבר לדומיין, ו"liked" איתו ניתן לדרג
קיימים דירוגים ישירים תחת האוסף rankings ודירוגים מצטברים תחת האוסף accumulate_rankings.

דירוגים ישירים הינם דירוגים המקבלים את כלל הניקוד של המשתמש באופן ישיר, הדירוגים הישירים הם:

- אהבה של קשת בין דף לנושא (דף תחת נושא מסוים).
- אהבה לקשת בין נושא לנושא.
- אמינות הדף (דירוג ישיר לדף ללא קשר לנושא).
- חינוכיות הדף.
- אהבה של תגובה.

דירוגים מצטברים הינם דירוגים הנובעים מדירוגים ישירים.
הסיבה שצריך דירוג מצטבר היא שאנו רוצים לשקלל באופן לוגי את ציון הדפים, ומהם להפיק את ציון הדומיין.
על מנת לעשות זאת, המשתמש לא יכול להביא לדומיין או לדף ניקוד גבוה יותר ממה שהוקצה לו.
כלומר, אם משתמש שיש לו דירוג בעל ערך מספרי 1 ידרג 2 דפים באתר מסוים כאמינים ו1 כלא אמין, אזי הניקוד בפועל שהדומיין צריך לקבל מהמשתמש הוא 0.666 דירוג חיובי ו0.333 דירוג שלילי. על מנת לבצע מעקב אחר הדירוגים וחישוב הניקוד המשוקלל, בזמן יעיל, אנו צריכים להיות במעקב אחר הדירוגים הישירים המצטברים.
בדוגמא מטה מוצג סך דירוג של משתמש יחיד לאחר שדירג דף נוסף מחוץ לתמונה כאמין:



דוגמא להסברת אופן המימוש: עבור כל קשת בין דף לנושא אותה המשתמש דירג שאהב, הדף עצמו מקבל את הדירוג הזה לאוסף הrankings בתוך הדירוג המצטבר, ועל ידי הוספה והורדה לשדות של ערך 1 כל פעם מpositive_rankings וnegative_rankings, העלות לעדכון ניקוד הדירוג שווה לשליפה אחת מהמסד נתונים + פעולה מתמטית קלה + עדכון הדירוג המצטבר + עדכון שינוי באובייקט המדורג.
הדירוגים המצטברים הם:

- אהבה של דף מהקשתות שלו
- אהבה של דומיין מהקשתות של הדפים שלו.
- אמינות הדומיין מהדפים שלו.
- חינוכיות הדומיין מהדפים שלו.

<u>המימוש בפועל (rankings):</u>
כל אובייקט ranking מכיל:
_id ייחודי לו.
object_id: מזהה ייחודי לאובייקט אותו אנו מדרגים.
object_collection_name: האוסף בו נמצא האובייקט.
user: מזהה ייחודי של המשתמש המדרג.
rank_type: "educational"/"credibility"/"liked".
ארבעת השדות מעלה יוצרים מזהה ייחודי במסד נתונים.

rank_code: ערך 1 עבור דירוג חיובי ו2 עבור דירוג שלילי
score_added: כמה הדירוג אוסיף לניקוד האובייקט המדורג. השדה תלוי בניקוד אותו המשתמש יכול להוסיף.

<u>צד לקוח:</u>
בצד הלקוח, קיים חץ למעלה וחץ למטה עבור כל דירוג שניתן לדרג באופן ישיר. החץ הזה שולח בקשה לפונקציה rank_function שבתוך הcomponent של הדף, המטפלת בשליחה לנתיב הנכון בשרת:
לapi/object_route/rank_object בשרת. הobject ניתן להחליף באוספים שונים בpage, topic_topic_edges וכדומה. כל אוסף מתחיל את הדירוג בעצמו אל מול פונקציה כללית של דירוג (עוד על זה בהמשך).
בנוסף הפונקציה דואגת לטפל בלוגיקה מאחוריי הדירוגים בהתאם לתשובות המתקבלות מהשרת. הסיבה לכך היא שבקשות מסוימות עשויות להתקבל ולהגיע באיחור לשרת, ולכן נקבל תשובה לאחר ששלחנו בקשה כפולה, או שלחנו בקשה אחרת. זה מתבצע על ידי שמירת המזהה הייחודי של הדירוג, סוג הדירוג וזמן הדירוג. הפונקציה גם דואגת לעדכן את הstate של הדף אחר הדירוג החדש, צביעת החצים הרלוונטיים.

<u>צד שרת:</u>
בצד השרת לכל route של פונקציות ייחודיות לטיפול בהזנת הניקוד אליו, וקריאה לפונקציה הכללית, המכניסה דירוג למסד הנתונים וקוראת לפונקציות

עדכון, הסרה ושליחת נתונים ללקוח, אותן קיבל תחת המשתנה update_remove_response_handlers.
לדוגמא:
api/pages/rank_page

```
var page= await Page.findOne({_id: pageID})
  .select(page_selection())
  .populate(domain_populate())
  .lean();
  if (!page)
    return res.status(400).send("Page not found");

  const update_remove_response_handlers=
require('./rank_page_update_remove_response_handlers');

  return await rank(
    page,
    'pages',
    rank_type,
    rank_code,
    userID,
    update_remove_response_handlers,
    res)
}
```

אנו שולפים את הדף והדומיין שלו מהמסד נתונים, בונים את הפונקציות לעדכון והסרת הדירוג, ואז שולחים את כל המידע הדרוש לביצוע העדכון. הלוגיקה מתבצעת פונקציה rank היושבת בmodels.

הפונקציה rank עובדת באופן הבא:

- תשיג את הuser_score מהמשתמש במסד נתונים.
- תבדוק האם דירוג קיים:
  - אם הדירוג לא קיים:
    - אם rank_code הוא 0, אזי תחזיר שלא קיים דירוג (כי המשתמש מבקש לבטל דירוג שכבר הנראה בוטל)
    - אם rank_code הוא לא 0, אזי תנסה להכניס דירוג חדש. אם הוכנס תבצע עדכון, על פי העדכון הפרטני שסיפק האובייקט.
  - אם הדירוג קיים:
    - בדוק האם הrank_code שווה לrank_code החדש, ואם כן, אזי תחזיר את הדירוג הנשלף ללא שינוי, כי מדובר בבקשה ישנה.
    - אם הם אינם שווים, אזי תבקש למחוק את הדירוג הקיים. במידה והצליחה, תקרא להסרה הפרטנית אותה סיפק האובייקט.

- תנסה להכניס דירוג חדש, ואם הצליח, אזי תקרא לעדכון הפרטני אותו סיפק האובייקט.

בלוגיקה זאת, רק קריאה שהצליחה להכניס דירוג חדש למסד נתונים תעדכן את ניקוד האובייקט, ורק קריאה שמחקה דירוג תקרא להסרת דירוג מהאובייקט.

דוגמא לupdate_remove _handlers של pages

```
async function object_update_score_function(ranking, page, rank_type,
user)
{
  var score_field_name = get_score_field_name(rank_type,
ranking.rank_code);
  var field_and_score_in_json = {};
  field_and_score_in_json[score_field_name] = ranking.score_added;


  await Page.findOneAndUpdate({_id: page._id},
    {$inc: field_and_score_in_json, $push: {rankings: ranking._id}});
  page[score_field_name] += ranking.score_added;

  await rank_domain_add_accumulate_ranking(ranking, page.domain, user);
  return true;
}

async function object_remove_score_function(ranking, page, rank_type,
user)
{
  var delete_result = await Ranking.deleteOne({_id: ranking._id});
  if (delete_result != null)
    if (delete_result.n == 1)
    {
      var score_field_name = get_score_field_name(rank_type,
ranking.rank_code);
      var field_and_score_in_json = {};
      field_and_score_in_json[score_field_name] = -ranking.score_added;

      await Page.findOneAndUpdate({_id: page._id},
        {$inc: field_and_score_in_json, $pull: {rankings:
ranking._id}});
        page[score_field_name] -= ranking.score_added;

      await rank_domain_remove_accumulate_ranking(ranking, page.domain,
user)

      return true;
    }
  return false
}
```

הפעולות המתבצעות הן הוספת והורדת ניקוד במידה של הכנסת ומחיקת דירוג מהמסד נתונים, הוספת הקשת למערך rankings של הדף, והפנייה לעדכון accumulate rankings של הדומיין של הדף.
הלוגיקה הזאת נכונה עבור כל אובייקט, ובאותו זמן ייחודית עבור כל אובייקט.

המימוש בפועל (Accumulate rankings):

פשוט יחסית אך יעיל:
אנו שומרים מערך של דירוגים ישירים אותו האובייקט קיבל ומוסיפים 1+ לpositive_rankings וnegative_rankings עבור כל דירוג, ומחסירים אחד עבור כל הסרת דירוג.
הניקוד המשוקלל הוא
(positive_rankings/(positive_rankings+negative_rankings))*user_score
עבור positive_points אותם מוסיפים לאובייקט המדורג.
ו(negative_rankings/(positive_rankings+negative_rankings))*user_score
עבור negative_points אותם מוסיפים לאובייקט המדורג.

על מנת לדעת כמה להוסיף לאחר שינוי, אנו צריכים לחשב את הניקוד המשוקלל לפני ההוספת/הסרת דירוג ישיר, ואת הניקוד לאחר הפעולה.
ההפרש בין שני הניקודים הוא הניקוד אותו צריך להוריד מהאובייקט המדורג.

כמובן שיש מספר דברים הדורשים התייחסות מיוחדת כמו המקרים בהם נשלפו כל הדירוגים, ואז צריך להציב ידנית 0 על מנת שלא נחלק ב0.
כמו כן מכיוון שהפעולה המתמטית מתרחשת בשרת, השתמשנו ב version control פשוט שמתעדכן בכל עדכון לדירוג. באופן זה, אם התקבלו שני דירוגים בו זמנית, אזי לא נפספס או נבצע הוספה לא נכונה לניקוד.

דוגמא של add_to_accumulate_ranking לpage

```
async function rank_page_add_accumulate_ranking(ranking, page, user)
{
    [accumulate_ranking, diff_positive, diff_negative]
    = await add_rank_to_accumulate_ranking(ranking, page._id, 'pages',
user);

    var json_for_inc={};
    json_for_inc[ranking.rank_type + "_positive_points"]=
diff_positive;
    json_for_inc[ranking.rank_type + "_negative_points"]=
diff_negative;
    await Page.findByIdAndUpdate(page._id,
        {
            $addToSet: {accumulate_rankings: accumulate_ranking},
            $inc: json_for_inc
        });
```

```
    page[ranking.rank_type + "_positive_points"]+= diff_positive;
    page[ranking.rank_type + "_negative_points"]+= diff_negative;
    await rank_domain_add_accumulate_ranking(ranking, page.domain,
user)
}
```

תגובות

הרעיון מאחוריי:

תגובה יכולה להשתייך לכל סוג של אובייקט כולל תגובה אחרת.

המזהה הייחודי של תגובה הם:
object_id: מזהה ייחודי של האובייקט עליו מגיבים.
object_collection_name: שם האוסף של האובייקט
user: המשתמש שהוסיף את התגובה
text: הטקסט של התגובה (על מנת למנוע הוספה כפולה של תגובה)

בנוסף, קיים שדה חשוב הכרחי נוסף:
root_comment, במקרה שהתגובה היא תגובת בת של תגובה אחרת, אזי הroot_comment תכיל את הערך מזהה ייחודי של התגובת שורש. במידה שהתגובה היא תגובת שורש לאובייקט מאוסף אחר, אזי הroot_comment תכיל את המזהה הייחודי שלה עצמה (_id)

בעזרתו אנו יכולים לשלוף את כל התגובות עבור אובייקט מסוים בשתי שליפות, הראשונה

```
var root_comments  = await Comment.find({
        object_id: object_id,
        object_collection_name: object_collection_name
    })
    .select(
        '_id'
    ).lean();
```

פה אנו שולפים את כל תגובות השורש המגיבות לאובייקט.

לאחר מכן:

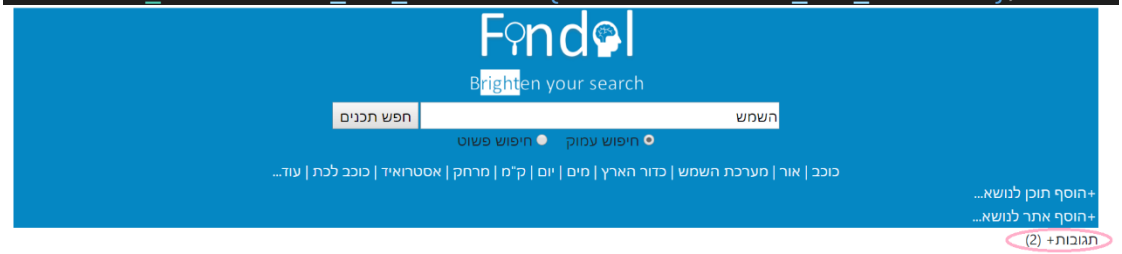
```
var comments = await Comment.find({root_comment: {$in:
root_comments_IDs_array}})
    .select(comment_selection({userID: userID}))
    .populate('user', 'userName position')
    .populate(rankings_populate(
        {
            userID: userID,
            object_collection_name: 'comments'
        }))
    .lean();
```

אנו שולפים את כל התגובות שהן תגובות לתגובות שורש עבור האובייקט.

המימוש בפועל:

צד לקוח:
על מנת לספק לרכיב תגובות, כל שצריך הוא לייבא את הרכיב Comments_loader מתוך Comments_components ולקרוא לו במקום בו רוצים להציג את התגובות:

```
<Comments_loader data_for_comments={this.state.data_for_comments}/>
```



בנוסף,
ביצירת הstate הראשוני להכניס בthis.state.data_for_comments ערכים ריקים שמוסתרים עד לאחר טעינת העמוד:

```
data_for_comments: {
                object_id: "",
                object_collection_name: '',
                number_of_comments: 0
            }
```

לאחר טעינת העמוד, לעדכן לערכים האמיתיים:

```
page.data_for_comments={
                    object_id: page._id,
                    object_collection_name: 'pages',
                    number_of_comments: page.number_of_comments
                }
```

הComments_loader מכיל מערך ריק של Comments.
לאחר לחיצה על "תגובות +", תשלח בקשה לשרת להביא את התגובות.

```
axios.get("/api/comments/retrieve_comments/?object_id="+opts.object_id+
"&object_collection_name="+opts.object_collection_name, {
            headers: {'findel-auth-token': this.token}
```

התשובה מהשרת היא מערך של תגובות לא ממוינות.
מיון התגובות מתבצע על ידי הפונקציה arrange_comments:

על מנת לסדר את התגובות, אנו יוצרים מספר דברים:

- מערך ראשי המכיל את תגובות השורש.
- מערך לכל תגובה המכיל את תגובות הבת שלה.
- JSON הנותן מצביע לכל תגובה על פי הid_ הייחודי שלה.

עכשיו נשתמש במצביעים של JSON ובobject_id אליו התגובה מצביע על מנת להכניס אותה כתגובת בת של התגובה בJSON[object_id].

```
comments_array.forEach(comment => {
        if (comment.object_collection_name=="comments")
            comments[comment.object_id].push(comment);
    });
```

בסופו של התהליך, כל התגובות ייכנסו למערך של התגובות אב שלהן, והתגובות הראשיות יהיו במערך הroots.
JSON מבצע את החיפוש על ידי פונקציית hash אז תאורטית זמן העבודה הוא O(n), כאשר n הינו מספר התגובות.

לאחר קבלת התגובות, הComments_loader ימלא את הרכיב Comments במידע הדרוש לו.

Comments_component:
מכיל Comments_array_mapper המקבל את מערך התגובות שורש, וממפה כל אחת מהן לרכיב Comment_component אשר מטפל בהצגת ודירוג תגובות, וגם כן מכיל Comments_array_mapper על מנת לטעון את התגובות תחתיו.

הרכיב Comments_component (תגובות) נותן את האפשרות להוסיף תגובת שורש על ידי שימוש ברכיב add_comment_component.

הרכיב Comment_component (תגובה) נותן את האפשרות להוסיף תגובה לעצמו גם כן, על ידי שימוש בadd_comment_component

על מנת להוסיף תגובה, צריך:

```
var opts={
            text: this.state.comment_text,
            object_id: this.props.parrent_object_data.object_id,
            object_collection_name:
             this.props.parrent_object_data.object_collection_name,
            root_comment_id:
             this.props.parrent_object_data.root_comment_id
          };
        axios.post('/api/comments/add_comment', opts, {
        headers: {'findel-auth-token': this.token}}
```

לאחר מכן, אם התגובה נכנסה בהצלחה, אזי נציג אותה מיידית למשתמש.

<u>צד שרת</u>:
עברנו בהתחלה על אופן שליפת התגובות לאובייקט.

הכנסת תגובה:

```
var opts={
            text: this.state.comment_text,
            object_id: this.props.parrent_object_data.object_id,
            object_collection_name:
             this.props.parrent_object_data.object_collection_name,
            root_comment_id:
             this.props.parrent_object_data.root_comment_id
          };
        axios.post('/api/comments/add_comment', opts, {
        headers: {'findel-auth-token': this.token}}
```

root_comment_id:
אנו צריכים אותו במקרה ואנו בתגובה עמוקה בשורש, וצריכים לספק לתגובת בת את המזהה הייחודי של תגובת השורש.

מעבר לזה, אם אנו שולחים root_comment_id=null, אזי אנו מודיעים לשרת שמדובר בתגובת שורש חדשה לאובייקט והוא יטפל בה בהתאם.

סיכום

בפרויקט השגנו דברים רבים:

- האתר משלב בין גוגל לוויקיפדיה על מנת למצוא אתרים ונושאים הקשורים לתוכן לימודי.

- האתר מספק שמירת היסטוריה, ושמירת אתרים מועדפים בהתאם לנושא שחיפשנו, ובהתאם לזמן בו דירגנו אותם. באופן זה התלמיד יכול לשמור את מקורות המידע שלו באופן מסודר וברור.

- האתר נותן אפשרות למשתמשים לדרג את התכנים עם התוצאה הישירה שהחיפוש העתידי יהיה יעיל יותר, ומספק את התוצאה העקיפה, בה התלמיד עוצר לחשוב האם הדף אליו הגיע מכיל תוכן אמין.

- האתר נותן למשתמשים את האפשרות להוסיף קשתות משלהם בין אתרים לנושאים ובין נושאים לנושאים.

- האתר נותן לכולם להגיב כמעט על כל תוכן בו, ויכולת לדרג את התגובות על מנת לספק את התגובות המועילות ביותר. בנוסף התגובה מציגה את תפקידי האנשים באופן ברור (תלמיד/ מורה/ מנהל) כך שניתן לדעת את אמינות התגובה מעבר לדירוג הבסיס.

כיוונים להרחבה

קיימים דברים רבים נוספים שניתן לקדם:

- חיבור בין תלמיד למורה על מנת לספק הכוונה אישית ובקרה על מקורות המידע בהם משתמש התלמיד.

- שיפור יכולת מציאת התכנים. בשלב זה מציאת התכנים הנוספים עדיין איננה מושלמת, ואם זמן נוסף והשקעה ניתן להגיע לתוצאות טובות יותר.

- דירוג משתנה למשתמשים: משתמשים שמגיבים ומקבלים דירוג טוב לתגובותיהם, ולתכנים הנוספים שהם מעלים, יקבלו עם הזמן משקל גדול יותר לדירוגים שלהם, ומאידך משתמשים שמגיבים באופן לא קשור או לא רלוונטי יהיו עם דירוג מופחת.

- הרחבת יכולות admin:
  - ביטול אובייקטים וקשתות לא רצויים (מחיקה מאבדת את המידע, ביטול שומר שלא יישמר המידע שוב).
  - צפייה ישירה בכל דירוגי המשתמשים עבור אובייקט, כך שניתן לראות תבניות בדירוג המשתמשים.

ביבליוגרפיה

הקורס איתו למדנו node.js, React, וmongoDB	Code with mosh: https://codewithmosh.com/
דוקומנטציה של node.js	https://nodejs.org/en/docs/
דוקומנטציה של mongoDB	https://docs.mongodb.com/
דוקומנטציה של mongoose	https://mongoosejs.com/docs/guide.html
דוקומנטציה של React	https://reactjs.org/docs/forwarding-refs.html