Оглавление

[1. Описание компонента пользовательский интерфейс 2](#_Toc8909398)

[2. Детали реализации компонента 3](#_Toc8909399)

[2.1 Используемые технологии. Описание 3](#_Toc8909400)

[2.1.1 Py2neo 3](#_Toc8909401)

[2.1.2 Django REST Framework 3](#_Toc8909402)

[2.1.3 React 4](#_Toc8909403)

[2.1.4 React-D3-Graph 4](#_Toc8909404)

[3.2 Архитектура компонента 5](#_Toc8909405)

[3.2.1 Структура БД 5](#_Toc8909406)

[3.2.2 Веб-сервер 6](#_Toc8909407)

[3.2.3. Клиентская часть 8](#_Toc8909408)

[3. Запросы, реализованные в компоненте 14](#_Toc8909409)

[4.1 Поиск публикаций, авторов, областей знаний 14](#_Toc8909410)

[4.2 Выборка вероятных авторитетов в некоторых областях знаний 18](#_Toc8909411)

[4.3 Исследование динамики публикаций в областях знаний 21](#_Toc8909412)

[4.4 Произвольный запрос на Cypher 23](#_Toc8909413)

[4.5 Показ состояния баз данных СКА 27](#_Toc8909414)

# Описание компонента пользовательский интерфейс

Пользовательский интерфейс – это компонент Системы, задачей которого являетсяобеспечить возможность большому числу пользователей производить желаемые запросы к Графу знаний, не вникая в особенности работы графовой БД, языка запросов просматривать результаты запросов в удобной форме, а также не дать пользователю при работе навредить БД преднамеренно или случайно.

Он должен предоставлять конечным пользователям системы удобный интерфейс, позволяющий легко находить требуемую информацию, выдавать результаты не только в текстовом виде, но и, где возможно, иллюстрировать их, рисуя графы, диаграммы.

Клиентская часть предоставляет следующие возможности:

- поиск научных публикаций, авторов, тем по части названия или с использованием различных фильтров.

- составление и показ выборки авторов, отсортированных по их значимости в некоторых областях науки (темы вводятся пользователем). Выборка основана на числе и индексе цитирования публикаций, значениях Page Ranking для статей. Исследуя связи цитирования, можно выявлять «экспертов» в области, на труды которых ссылаются многие другие авторы, и в первую очередь изучить их. Это может помочь существенно сэкономить время поиска информации для собственных исследований.

- составление и показ списка областей знаний с указанием в виде диаграмм популярности их у авторов по периодам времени, по общему числу публикаций. Так можно предположить, какие темы сейчас в тренде, к каким интерес нарастает, какие были популярны у исследователей, на пример в период 1950-1970 гг.- возможность выполнить любой запрос к Графу знаний на языке Cypher; ответ предоставляется в виде таблицы или графа.

- для отдельной публикации, темы, автора есть страницы с подробной информацией о них, представленной в текстовом виде и в виде изменяемого графа.

- возможность выполнить любой запрос к Графу знаний на языке Cypher; ответ предоставляется в виде таблицы или графа.

- для отдельной публикации, темы, автора есть страницы с подробной информацией о них в текстовом виде и в виде изменяемого графа. Так можно пошагово исследовать взаимосвязи сущностей.

# Детали реализации компонента

Компонент представляет собой веб-приложение.

Серверная часть имеет связь с компонентом Граф знаний и может выполнять все необходимые запросы к БД Neo4j. Для инкапсуляции процесса построения и выполнения запросов к БД сервер предоставляет API. В методы помещены запросы, которые могли бы быть потенциально востребованными при работе с такой системой как СКА. Например, поиск и просмотр информации о цитировании той или иной публикации. API используется клиентской частью компонента.

Клиентская часть создана для того, чтобы предоставить контролируемый доступ широкому кругу простых пользователей к накопленным в БД Системы данным и отображать аналитические расчеты и выборки в читаемом и наглядном виде. Для того, чтобы получить желаемую информацию пользователю не требуется составлять запрос на языке запросов, вникать в особенности сущностей и аналитических коэффициентов. От него требуется только, используя традиционные для веб-страниц способы ввода показать системе, что он хочет найти или узнать.

## Используемые технологии. Описание

### Py2neo

Py2neo - это клиентская библиотека и инструментарий для работы с Neo4j из приложений Python и из командной строки. Библиотека обернула официальный драйвер, добавив поддержку HTTP, API более высокого уровня, OGM, инструменты администратора, интерактивную консоль, лексер Cypher для Pygments и многие другие навороты. В отличие от предыдущих выпусков, Py2neo v4 больше не требует HTTP и может работать полностью через Bolt.

В системе эта библиотека – основной способ работы с БД Neo4j. Все запросы, которые необходимы компоненту Пользовательский интерфейс для получения данных выполняются на сервере при помощи нее.

### Django REST Framework

Django REST Framework – мощный и гибкий инструмент для разработки Web API. Он позволяет настроить различные виды аутентификации, выбрать удобный способ сериализации данных для передачи в каждом отдельном случае.

Это надстройка над фреймворком для разработки веб-приложений Django, ориентированная именно на разработку API.

Он является основой веб-сервера компонента.

### React

React - JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. Он позволяет сравнительно быстро создавать интерактивный пользовательский интерфейс веб-приложения. Одна из существенных плюсов использования такой библиотеки является то, что при работе приложения она следит за потоком данных в клиентском приложении и старается снизить нагрузку на устройство клиента, оптимизируя перерисовку страниц: перерисовке будут, преимущественно, подвергаться только те части, которые действительно должны отреагировать на изменения данных, а не вся страница целиком.

С помощью этой библиотеки разработана клиентская часть веб-приложения компонента Пользовательский интерфейс СКА.

### React-D3-Graph

React-D3-Graph – это библиотека, оборачивающая мощь D3.js в компоненты React. D3.js представляет собой JS библиотеку для создания сложных визуализаций данных и, так же, как и, например, React, оптимизирующая работу с DOM. Однако, работать с ней в приложении, где все остальные элементы созданы с помощью React было бы неудобно. React-D3-Graph – это развивающаяся библиотека, как раз и нацеленная на облегчение создания сложных, красивых и информативных вставок в React приложение, опираясь на богатый функционал D3.js.

Эта библиотека позволила разнообразить страницы компонента разноцветными графами. Так как много уже было написано на React к моменту, когда было решено нарисовать что-то более интересное, чем текст и диаграммы, эта библиотека оказалась очень кстати. Библиотек для создания диаграмм и графиков существует гораздо больше. В том числе и основанных на D3.js. Но с функционалом создания графов нашлась только эта. В ней можно создавать «живые графы», присутствуют события пользовательского взаимодействия с элементами графа, широкие возможности настройки внешнего вида графа. Очень упрощает работу хорошее автопозиционирование графа. Очень точно реализована реакция графа на движение, перетягивание отдельного узла - граф плавно перестраивается, сохраняя форму симметричного облака.

И, несмотря на то, что иногда обновления библиотеки вместе с решением старых ошибок приносят новые, ее достоинства все равно не меркнут.

## 3.2 Архитектура компонента

### 3.2.1 Структура БД

На данный момент в графовой БД существуют следующие сущности и связи:

Author – тот, кто опубликовал статью. Содержит данные о своем имени и о статьях, которые написал (отношение WROTE)

Publication – публикация написанная автором. Содержит в себе имя, год публикации, идентификатор (sha256), ISBN, количество страниц, язык, расширение файла, теги. C автором, написавшем статью, соединена связью WROTE. Ссылки на темы, к которым с определенной вероятностью относится (вероятностная модель, отношение THEME\_RELATION, отношение содержит в себе вероятность). Имеет ссылки на статьи, на которые ссылается (LINKS\_TO, необходимо для page ranking). Имеет ссылки на ключевые фразы, которые входят в ее текст (FREQUENCY, содержит количество вхождений, эдакая интерпретация мешка слов).

Theme – тема, к которой может относиться публикация. Имеет в себе свое уникальное название и также имеет возможность просмотреть все публикации, относящиеся к ней через THEME\_RELATION.

Token – сущность, которая представляет себя уникальным именем. Имеется возможность просмотреть все публикации, которые имеют в своем тексте вхождение ключевой фразы и также можно просмотреть количество вхождений (все это хранится в связи FREQUENCY). Графически все поля и типы в виде множества и графовая модель представлены на рис. 3.

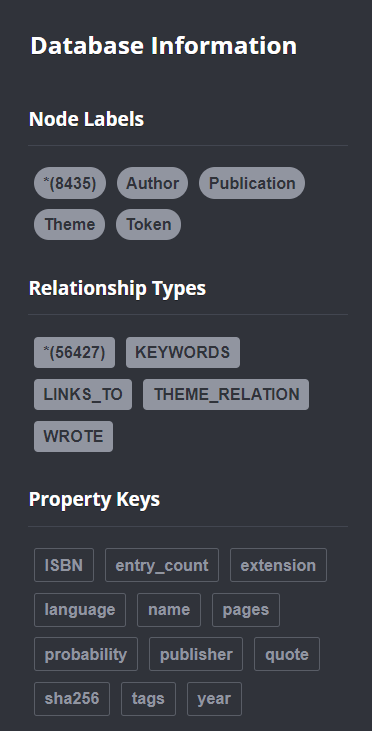


Рисунок 3. Структура БД, показанная СУБД Neo4j

### 3.2.2 Веб-сервер

Веб-сервер размещен на одной из виртуальных машин, выделенных для проекта, имеющей доступ по локальной сети к виртуальной машине, которая является хостом компонента Граф знаний и его БД Neo4j.

Веб-сервер создается на языке Python на фреймворке Django REST Framework.

С помощью драйвера Py2Neo сервер устанавливает соединение с БД Neo4j и выполняет к ней запросы. На клиентскую часть посылаются ответы в виде объектов JSON.

Работу с Py2Neo инкапсулирует класс NeoQuerier. При создании объекта этого класса в \_\_init\_\_ создается объект py2neo.Graph, представляющий программный интерфейс для работы с отдельным графом в БД Neo4j. Подключение его выполняется с помощью константных данных, таких как имя пользователя БД, пароль, IP адрес сервера БД. Они вынесены в файл констант. Объект py2neo.Graph помещается в поле объекта класса NeoQuerier. Для повышения безопасности (предотвращения действий сторонних пользователей по изменению данных в БД) для предоставления компоненту Пользовательский интерфейс ограниченного доступа к данным на уровне СУБД Neo4j был зарегистрирован пользователь с правами «только для чтения».

Для выполнения любого из предусмотренных запросов нужно вызвать соответствующий метод у объекта класса NeoQuerier.

В дальнейшем этот класс будет продолжать эволюционировать. В основном, будут добавляться новые запросы и происходить переход на использование Py2Neo OGM, если это будет целесообразно.

В файле views.py находятся классы, соответствующие запросам, оговоренным в требованиях к компоненту Пользовательский интерфейс. Все они являются наследниками rest\_framework.views.APIView. Класс позволяет указать в качестве свойств уровень ограничения доступа (permission\_classes) пользователей к данному запросу (IsAdminUser, IsAuthenticated, AllowAll) и класс для преобразования данных перед отправкой клиенту (renderer\_classes). В каждом классе переопределены методы, соответствующие нужным HTTP глаголам (в основном GET). Из параметров запроса выделяются нужные, затем производится создание объекта NeoQuerier и вызов метода. Результат запроса для отправки клиенту преобразовывается в JSON. В данный момент из БД возвращаются объекты типа py2neo.Node и py2neo.Relationship. Они содержат достаточно много полезных свойств. Однако при сериализации их в JSON с использованием встроенного в Python класса JSONEncoder некоторые важные иногда свойства, такие как identity (автоматически создаваемый уникальный для узлов одной группы идентификатор), labels(список имен групп узлов, к которым принадлежит данный). В некоторых выдачах, таких как, например, просто список публикаций, или список имен для работы подсказок на клиенте они не требуются. И можно использовать сериализатор по умолчанию. Но отсутствие этих свойств – большая проблема при построении графов, которые требовалось рисовать на некоторых страницах клиентской части. Иногда можно было передавать ID отдельным полем, но это не всегда удобно. Потому был сделана несложная модификация класса JSONEncoder, позволявшая сохранить объектам их свойства, а так же делать некоторые небольшие преобразования, например, приводить индентификаторы типа «число» к типу «строка». (Далее, в разделе о фронт-части приложения будет говориться о сложностях с одинаковыми идентификаторами у узлов разных типов).

Файл urls.py, как во всех приложениях на Django, определяет соответствие URL и представлений (Листинг 2).



Листинг 2 – urls.py

Кроме функционала обеспечения данными из БД веб-сервер выполняет также аутентификацию пользователей, и предоставляет доступ к панели администратора Django (входит в состав Django Framework, предоставляется по умолчанию). В ней можно управлять правами пользователей системы, удалять и добавлять их. В соответствии с текущими требованиями регистрация пользователей возможна только непосредственно администратором вручную. Пользователям с правами администратора также предоставляется доступ к функционалу выполнения CYPHER-запроса к БД Neo4j через веб-сервер, и получение информации о количестве узлов в БД.

### 3.2.3. Клиентская часть

Клиентская часть разработана с помощью библиотеки React и некоторых вспомогательных библиотек. В качестве менеджера пакетов и development-сервера используется Yarn.

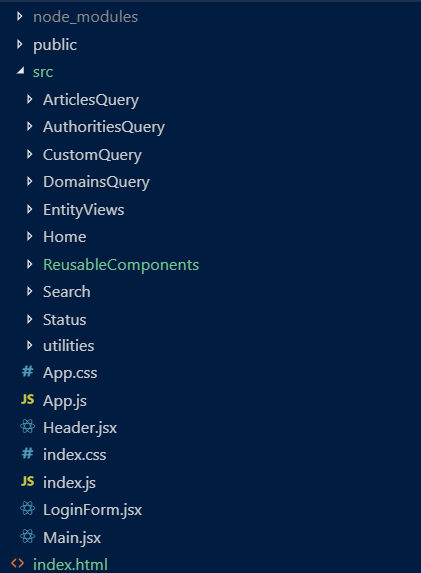


Рисунок 4 – структура проекта клиентской части

Основные структурные составляющие – это традиционные React компоненты и контейнеры для них. Было принято решение пока не использовать инструменты управления как состоянием данных, такие как Redux, так как у приложения нет какого-то глобального состояния, глобальных данных, которые затрагивают больше одного раздела, а максимальная вложенность компонентов, где нужна передача данных не только вниз, но и вверх (к родительскому компоненту) не превышает трех. И реализация на традиционных callback-ах не выглядит слишком ужасной.

Для организации навигации используются возможности библиотеки React Router. Маршрутизатор позволяет сопоставлять маршруты и компоненты, используя несложный JSX-синтаксис. Структура url приложения – на листинге 3.

<Switch>

<Route exact path='/' component={Home}/>

<Route path={SEARCH} component={SearchWithResults} />

<Route path={AUTHOR} component={AuthorGraphView}/>

<Route path={DOMAIN} component={Domain}/>

<Route path={PUBLICATION} component={PublicationGraphView}/>

<Route path={AUTHORITIES\_QUERY} component={AuthoritiesQuery}/>

<Route path={DOMAINS\_POPULARITY\_QUERY} component={DomainsQuery}/>

<Route path={TEST} component={Test} />

<Route path={CUSTOM\_QUERY} component={CustomQuery}/>

<Route path={STATUS} component={Status} />

</Switch>

Листинг 3 – маршруты

Общий жизненный цикл компонентов, представляющих отдельные страницы Пользовательского интерфейса:

1. Параметры, такие как идентификатор, параметры фильтрования выделяются из строки запроса (для получения их используется вспомогательная библиотека queryString).
2. Выполняется запрос на сервер, и, при успешном получении ответа, подготавливаются и выводятся данные
3. При выполнении поиска установленные пользователем параметры, такие как текст, по которому следует подбирать результаты, значения фильтров не только отправляются на сервер, но и дописываются в адресную строку клиента. Это позволяет приложению корректно и ожидаемо работать при нажатии кнопки «Назад» в браузере, а так же организовывать ссылки на конкретные поисковые запросы с других страниц. Например, со страницы выборки авторов-экспертов в указанных пользователем областях науки существуют для каждого автора из списка ссылки на поиск публикаций, и при переходе по ссылке фильтры авторов и тем публикаций поиска сразу будут заполнены. Это позволяет не создавать отдельные страницы для разнообразных простых выборок.

Листинг 4 – пример текста в адресной строке при выполнении поиска публикаций на темы Biology и Programming

http://localhost:3000/search?type=publication&theme=Biology&theme=Programming

Для организации загрузки данных с веб-сервера в файле ./src/utitilies/ loaders.js созданы методы, которые скрывают от компонентов, в которых они применяются, особенности формирования пути, привязки токена аутентификации к запросу. Строковые константы путей вынесены в отдельные файлы.

// loaders.js

const getPublicationGraph = (id, token) => {

const query = `${PYTHON\_BACKEND\_API\_PUBLICATION\_GRAPH}?id=${id}`;

return getLoaderPromise(query, authOptions(token));

}

const getAuthorGraph = (id, token) => {

const query = `${PYTHON\_BACKEND\_API\_AUTHOR\_GRAPH}?id=${id}`;

return getLoaderPromise(query, authOptions(token));

}

const getAuthoritiesInDomainsList = (domains\_list, token) => {

let query = buildQueryParametersList('domain', domains\_list);

return getLoaderPromise(PYTHON\_BACKEND\_API\_AUTHORITIES\_QUERY + `?${query}`, authOptions(token));

};

Листинг 5 – пример методов из loaders.js

Рассмотрим особенности реализации некоторых компонентов. Наиболее интересно выглядят страницы для показа информации об отдельном авторе и отдельной публикации, так как в них совмещены текст и граф. Пример – на рис. 5 и рис. 6.

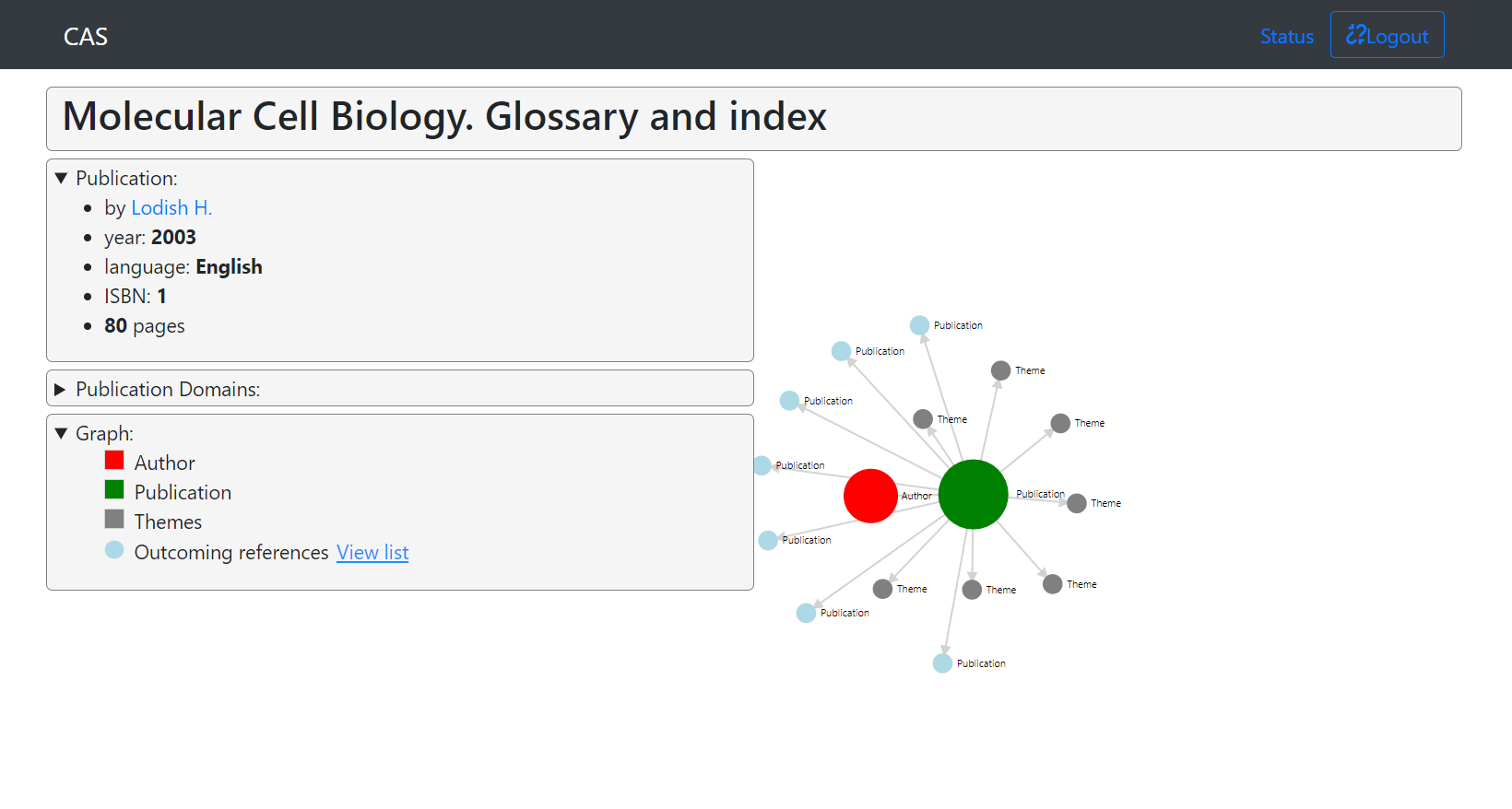


Рисунок 5 – страница отдельной публикации

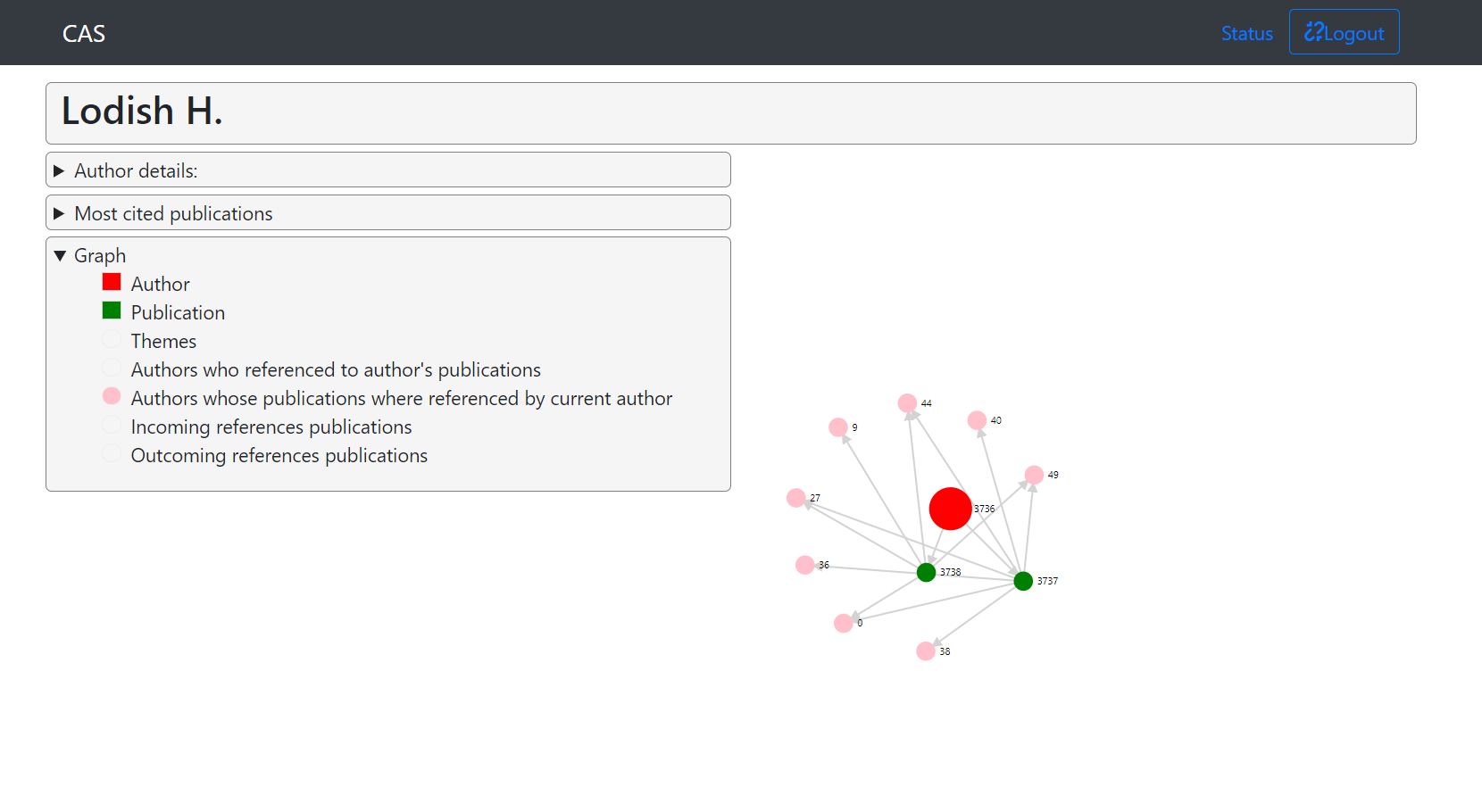


Рисунок 6 – страница отдельного автора

Внешний вид страниц идентичен, они состоят из одних и тех же компонентов, отличия лишь в отображаемых данных.

Краткий список особенностей:

* текстовая информация находится в сворачиваемых списках <details>
* На графе, который занимает всю площадь страницы (в т.ч. и под списками), отображаются узлы и связи, которые имеют отношение к предмету страницы.
* Есть группы узлов, которые можно сменять или отключать, чтобы они не мешали просмотру.
* При наведении курсора мыши на узел появляется всплывающая подсказка с краткой информацией о нем
* При клике на узел происходит переход по ссылке на соответствующую страницу подробной информации о предмете этого узла

Для создания графа использован компонент <Graph> библиотеки react-d3-graph. С сервера приходит уже подготовленная информация для построения графа – ребра и узлы, полученные запросом из самой БД Neo4j. Они сгруппированы по своему значению: узел автора/публикации, о котором идет речь на странице, узлы и связи наиболее значимых публикаций автора, узлы и связи цитирования и т.д. Код клиентской части занимается формированием графа, соответствующего текущим требованиям пользователя (состоянию селекторов групп узлов на странице, подробнее о них будет рассказано ниже). К объектам узлов добавляется свойство цвета, размера, ссылка на страницу подробной информации, функция формирования подсказки из данных узла. Узлы собираются в объекты-словари (ключ – их свойство id), чтобы избежать коллизий и иметь быстрый доступ к объекту конкретного узла по id, ребра – в списки, и все это передается в компонент Graph для отрисовки.

Компонент Graph предоставляет возможность подписаться на события пользовательского клика на узел, наведения мыши. В функцию-обработчик передается значение свойства id того узла, у которого произошло событие.

В библиотеке не было возможности показа всплывающих подсказок, поэтому эта возможность реализована вручную в компоненте-обертке <EntityGraph>. Компонент отслеживает положение мыши, и в любой момент предоставляет информацию о текущих ее координатах. По событию Graph.onMouseOverNode показывается подсказка немного правее курсора.

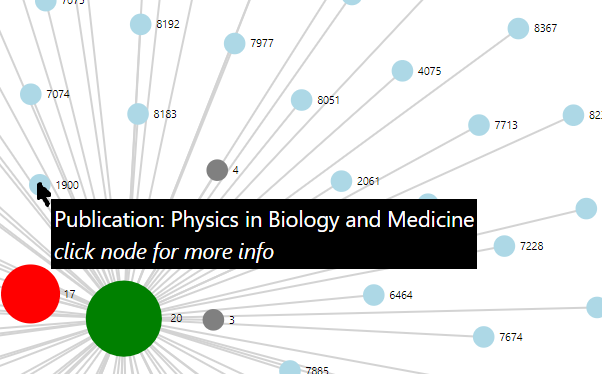


Рисунок 7 – подсказка с информацией (курсор нарисован)

По нажатию на конкретный узел происходит переход на страницу с подробностями о предмете, который отражает узел.

Самой значимой трудностью, с которой пришлось столкнуться при разработке графов с помощью react-d3-graph были коллизии идентификаторов.

Первоначально планировалось, что можно будет включать одновременно несколько опциональных групп узлов, однако из-за того, что стали частыми коллизии идентификаторов узлов, относящихся к разным группам, и особенно ребер, которые библиотекой рисования различаются не по свойству id, а по связке «начальный-конечный узлы», было решено сделать группы замещающими друг друга.

Кроме того, перед добавлением или исключением узлов было вставлено временное решение – компоненту графа предварительно передается пустой список узлов и ребер, а затем уже – новый список. Для того, чтобы не перегружать клиентские машины повторяющимися одинаковыми пересортировками и переформированиями ранее сформированные наборы узлов и ребер сохраняются («мемоизируются») в специальных функциях.

# Запросы, реализованные в компоненте

## 4.1 Поиск публикаций, авторов, областей знаний

Пользователю предоставляется возможность найти среди всех имеющихся в Графе знаний узлов нужные ему, указывая в качестве параметров поиска тип узла, часть имени, и параметры, определяющие, с какими другими узлами должны быть связи.

Поле для ввода части имени пользователь видит прямо на главной странице приложения (рис. 8).

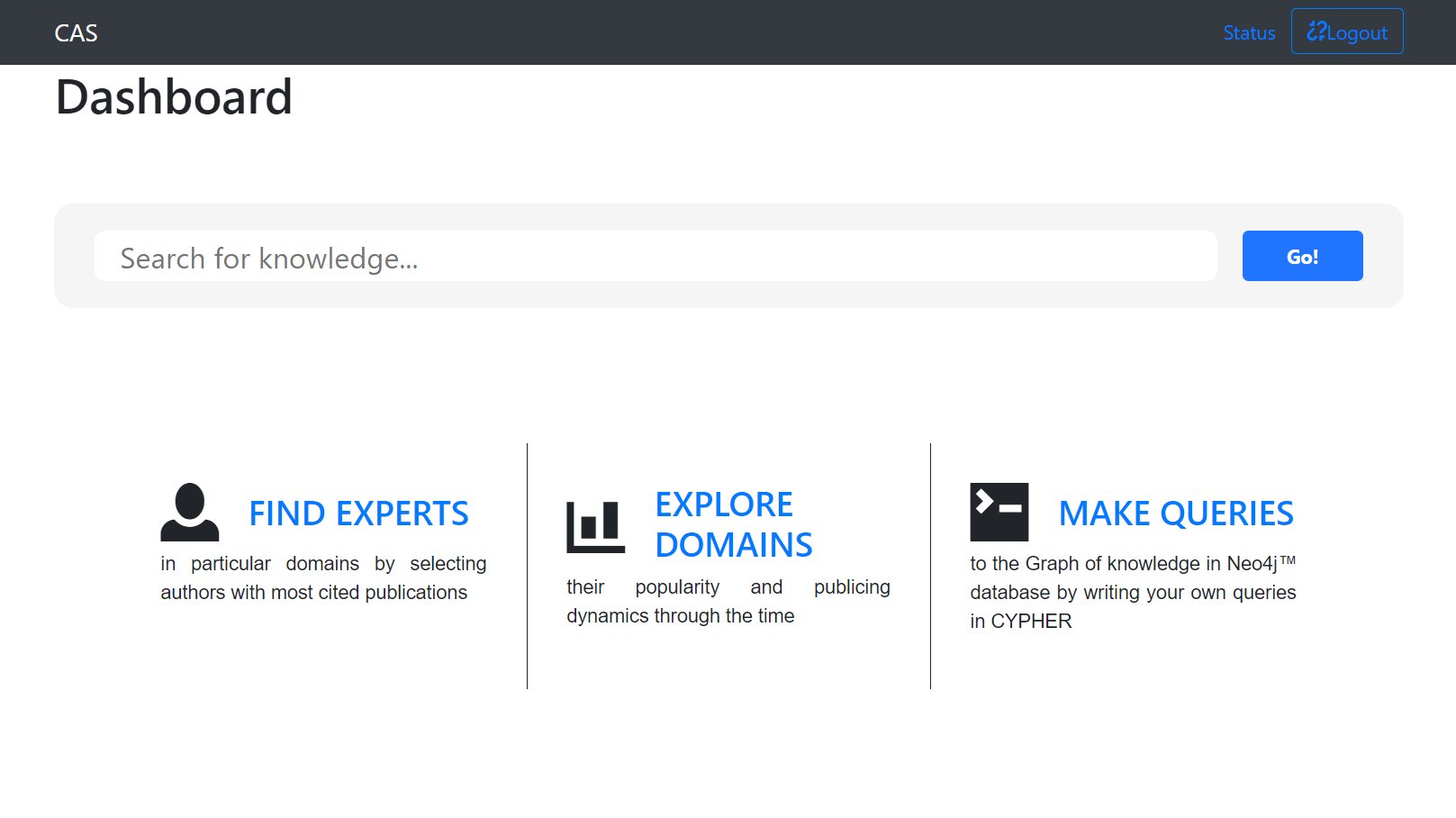


Рисунок 8 – главная страница

На странице поиска (пример – на рис. 9) можно выбирать тип (публикации, авторы, темы или все вместе), и для поиска авторов и публикаций есть дополнительные фильтры. Если мы ищем публикации, то в фильтры позволяют указать, работы каких именно авторов мы хотели бы видеть, и/или указать темы, к которым должны относиться искомые публикации. В случае поиска авторов пока что есть фильтр только по темам, в которых автор должен иметь хотя бы одну публикацию.

Результаты поиска сортируются по значимым для каждого типа числам: авторы и темы – по количеству публикаций, публикации – по количеству ссылок на них.

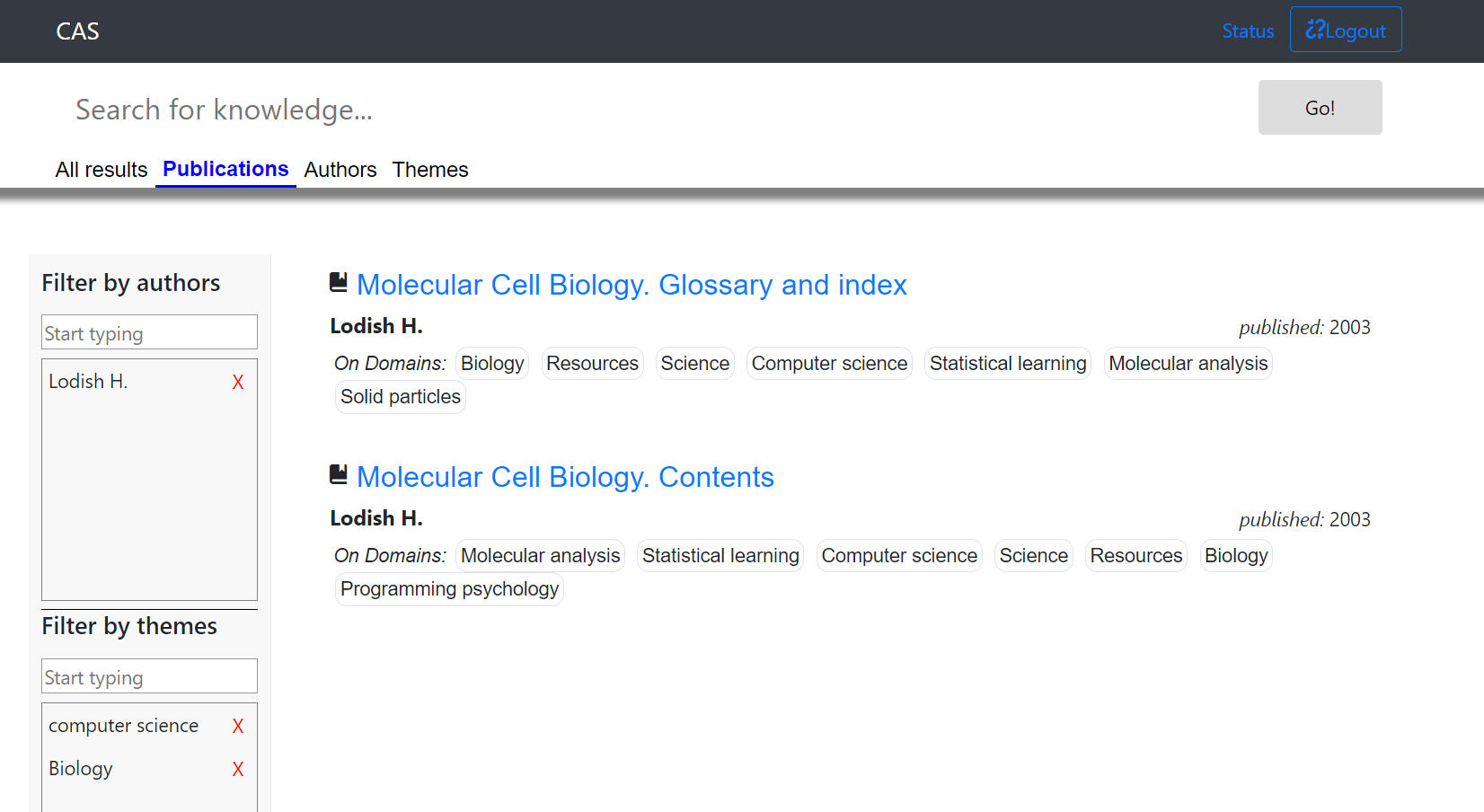
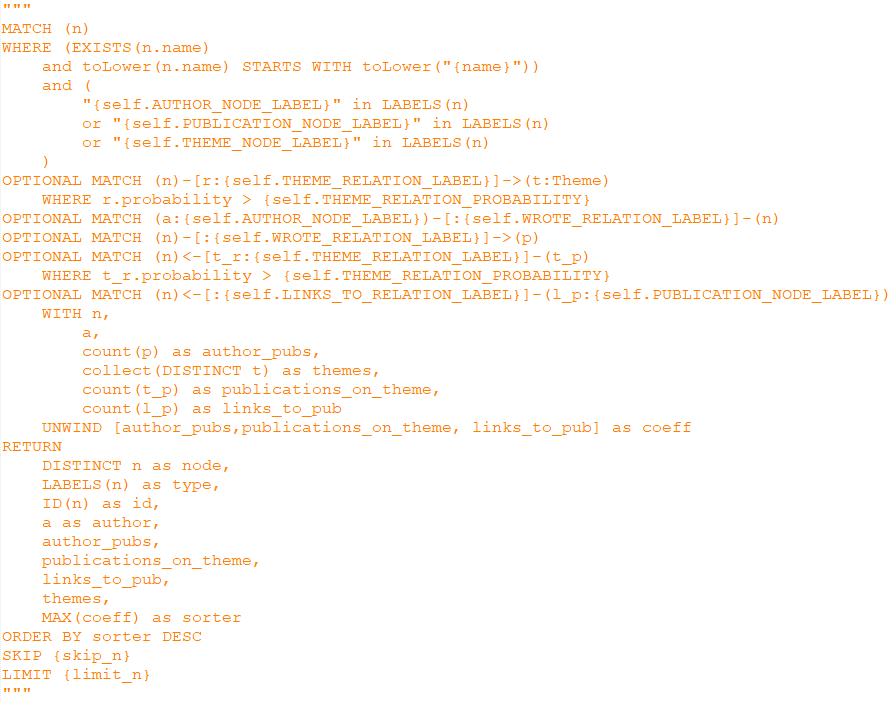


Рисунок 9 – поиск публикаций по авторам и темам

На листинге 4 представлены запросы на языке CYPHER, которые выдают требуемую информацию.

Листинг 4 – запрос на поиск элементов по части названия, расставляющий результаты по убыванию значимого коэффициента

Для просмотра подробностей о интересующем авторе, публикации или области знаний созданы отдельные страницы.

На них, как уже описывалось в разделе 2, совмещены текстовая информация и граф, являющийся фрагментом Графа знаний. На нем показаны узлы и ребра, имеющие отношение к предмету страницы. Некоторые группы узлов (опциональные) можно скрывать или показывать, чтобы просмотр был более удобным.

На странице детальной информации о публикации представлены:

- название, автор, дополнительные свойсва (ISBN, язык и т.д.)

- список тем, к которым она относится

- граф, на нем следующие узлы и ребра:

- автор, публикация, темы

- публикации, ссылающиеся на предмет страницы (опционально)

- публикации, на которые ссылается публикация-предмет страницы (опционально).

На странице детальной информации об авторе представлены:

- имя, организация

- список из наиболее цитируемых его публикаций (до 10 штук)

- граф, на нем следующие узлы:

- автор, наиболее цитируемые его публикации (до 10)

- темы публикаций (опционально)

- авторы, которые ссылались в своих публикациях на публикации этого автора (опционально)

- авторы, на публикации которых ссылался текущий автор при написании своих наиболее популярных публикаций (опционально)

- публикации, ссылающиеся на публикации автора (опционально)

- публикации, на которые ссылался автор (опционально)

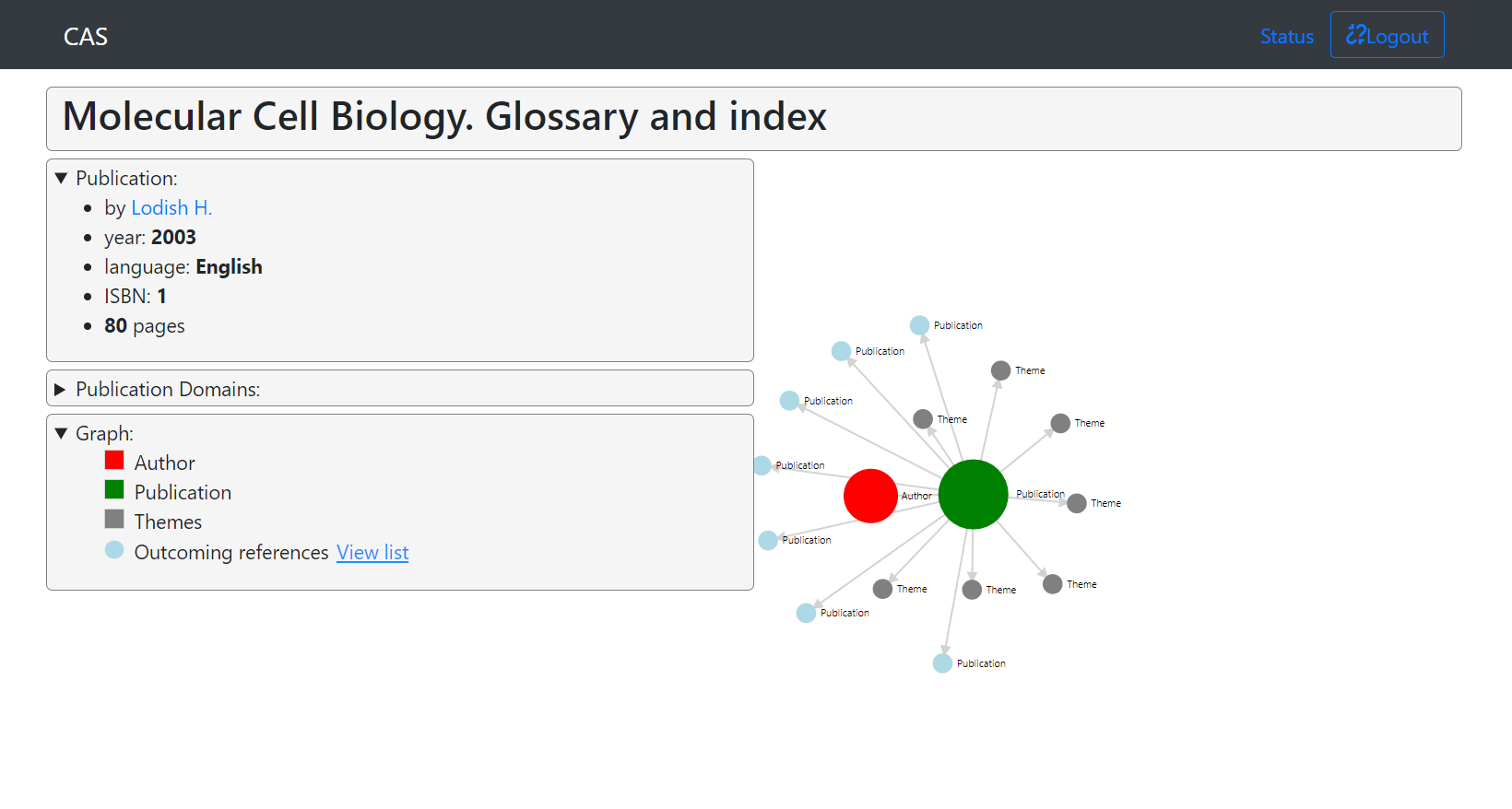


Рисунок 10 – страница отдельной публикации

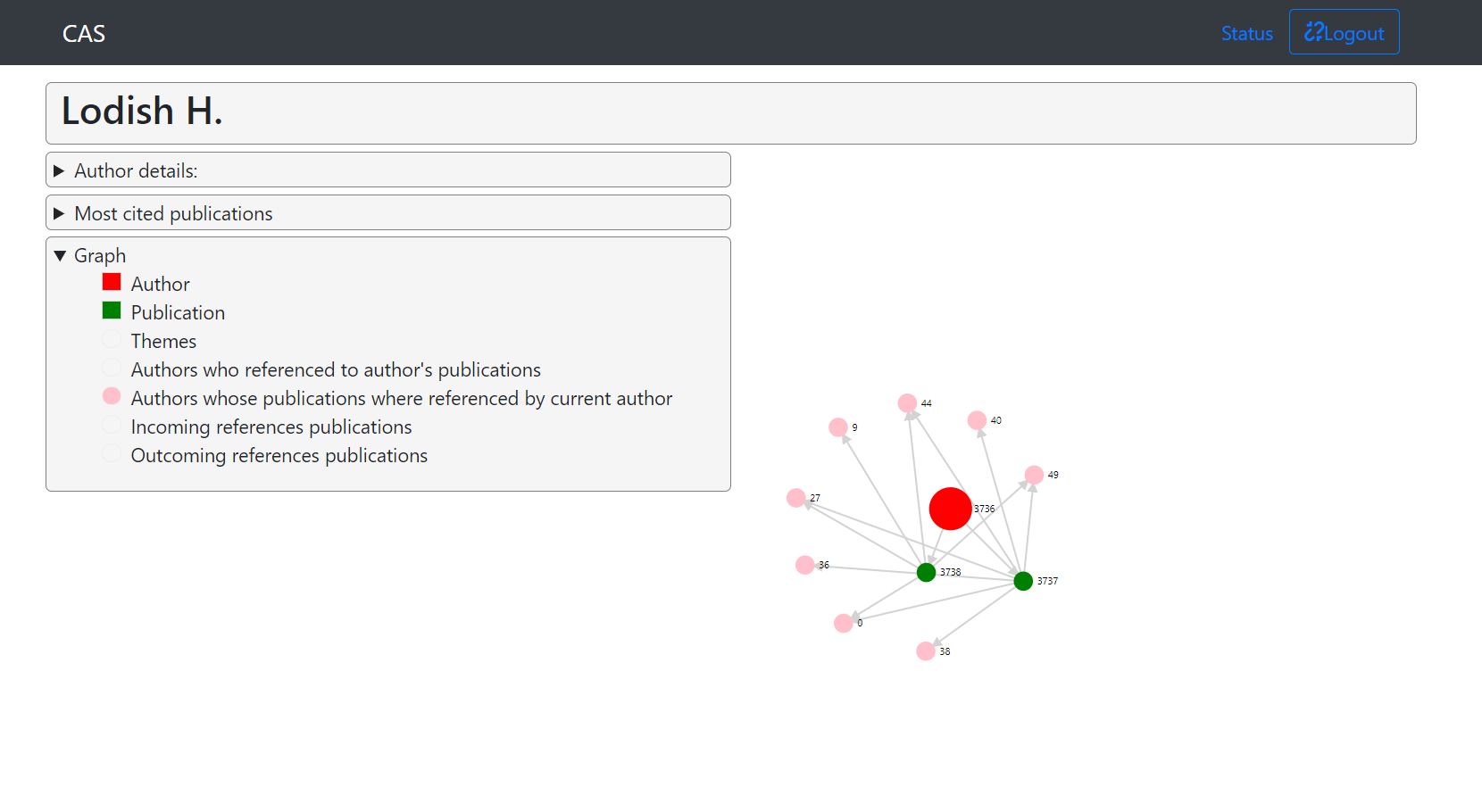


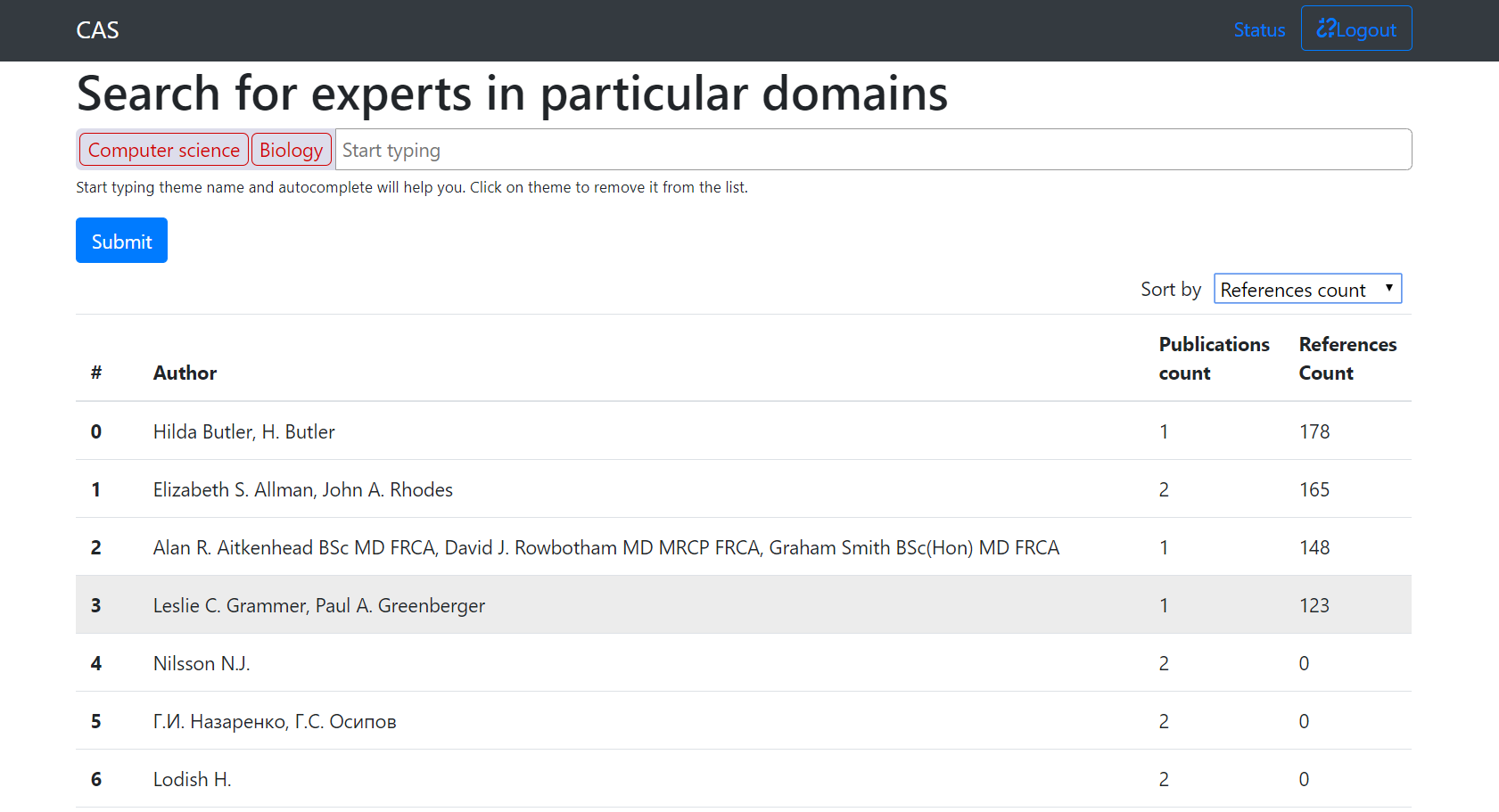
Рисунок 11 – страница отдельного автора

## 4.2 Выборка вероятных авторитетов в некоторых областях знаний

Одним из аргументов в пользу использования графовой БД в проекте СКА было то, что сама суть графов позволяет строить более наглядные зависимости и производить аналитические вычисления. Выборка авторов, которых можно было бы назвать экспертами в каких-то областях знаний основана на использовании рассчитанного для каждой публикации коэффициента pageRank, количества публикаций авторов и количество ссылок на публикации отдельного автора.

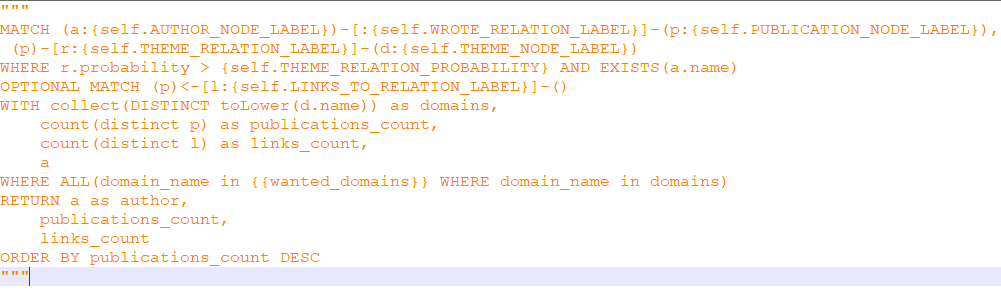
Для справки. Page rank – это алгоритм, который позволяет выбирать наиболее важные статьи в графе. Данный алгоритм также применяется такими корпорациями как Google, Twitter для выдачи результатов запроса и сортировки поиска информации по интересам.

Страница для просмотра описанной выборки приведена на рис. 10. В поле ввода пользователь вводит области знаний, в которых должны иметь публикации интересующие его авторы. Когда список сформирован, авторов можно упорядочить по числу публикаций и по индексу цитирования.

Рисунок 12 – выборка «экспертов»

По клику на любой из результатов в списке пользователя направляют на страницу поиска публикаций с заполненными фильтрами.

Темы, к которым относится публикация, в Системе определяются автоматически – это одна из задач компонента Фильтрации и Аналитики. В каждом экземпляре отношения «публикация-тема» (в БД Neo4j – THEME\_RELATION) устанавливается свойство probability – вероятность того, что публикация действительно относится к этой теме. Выборки публикаций осуществляются с учетом этого коэффициента. Настраивать его можно в конфигурации сервера. Запрос на CYPHER приведен ниже в листинге 5.



Листинг 5 – запрос выборки авторитетов

## 4.3 Исследование динамики публикаций в областях знаний

Одной из функций компонента составление и показ списка областей знаний с указанием в виде диаграмм популярности их у авторов по периодам времени, по общему числу публикаций. Так можно предположить, какие темы сейчас в тренде, к каким интерес нарастает, какие были популярны у исследователей, на пример в период 1950-1970 гг.

Пример списка приведен на рис. 11, 12. Диаграммы строятся с помощью вспомогательной библиотеки react-vis. Она ориентирована на создание красивых диаграмм и графиков. Числовые данные для построения диаграммы каждой отдельной темы формируются на сервере. Все диаграммы построены в одинаковом масштабе, чтобы можно было визуально оценивать различия динамики.

С этой страницы пользователь может или перейти на страницу подробной информации для отдельной темы, или с помощью кнопки «Add to query» собрать список из тем и затем просмотреть выборку потенциальных авторитетов в этих областях (была описана ранее в пункте 4.1). (Пример – на рис. 13).

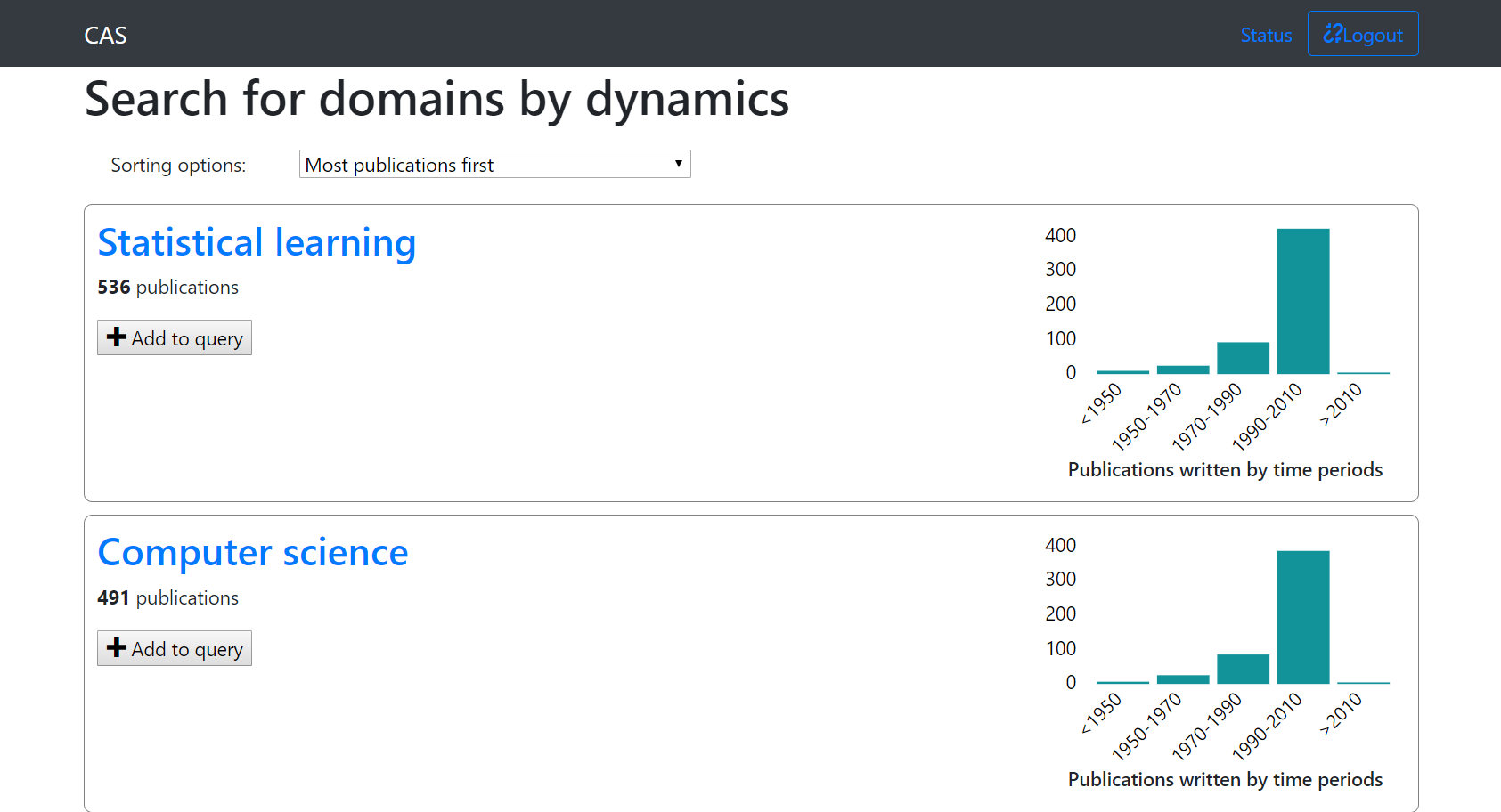


Рисунок 13 – страница областей знаний

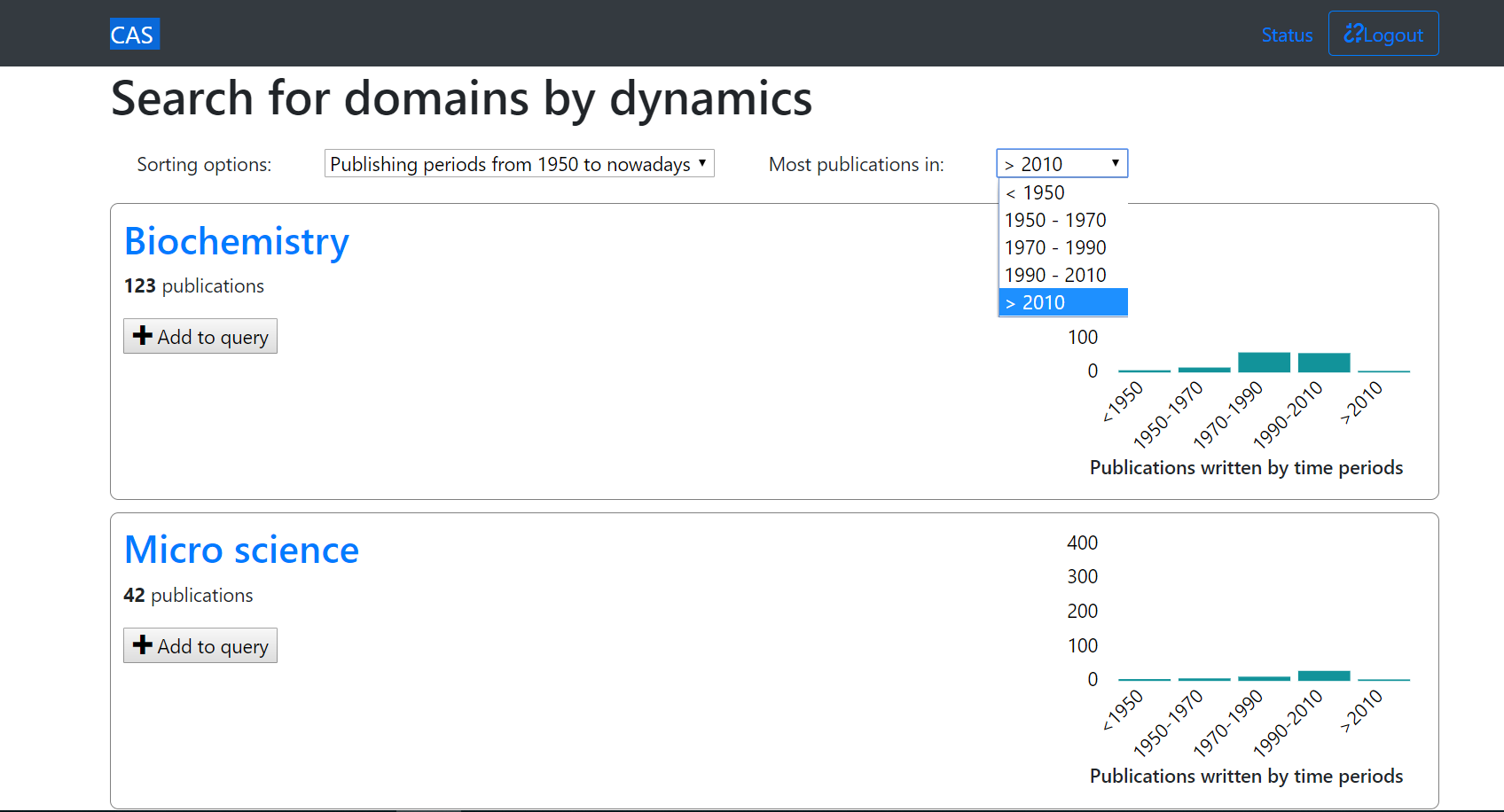


Рисунок 14 – вариант сортировки областей знаний

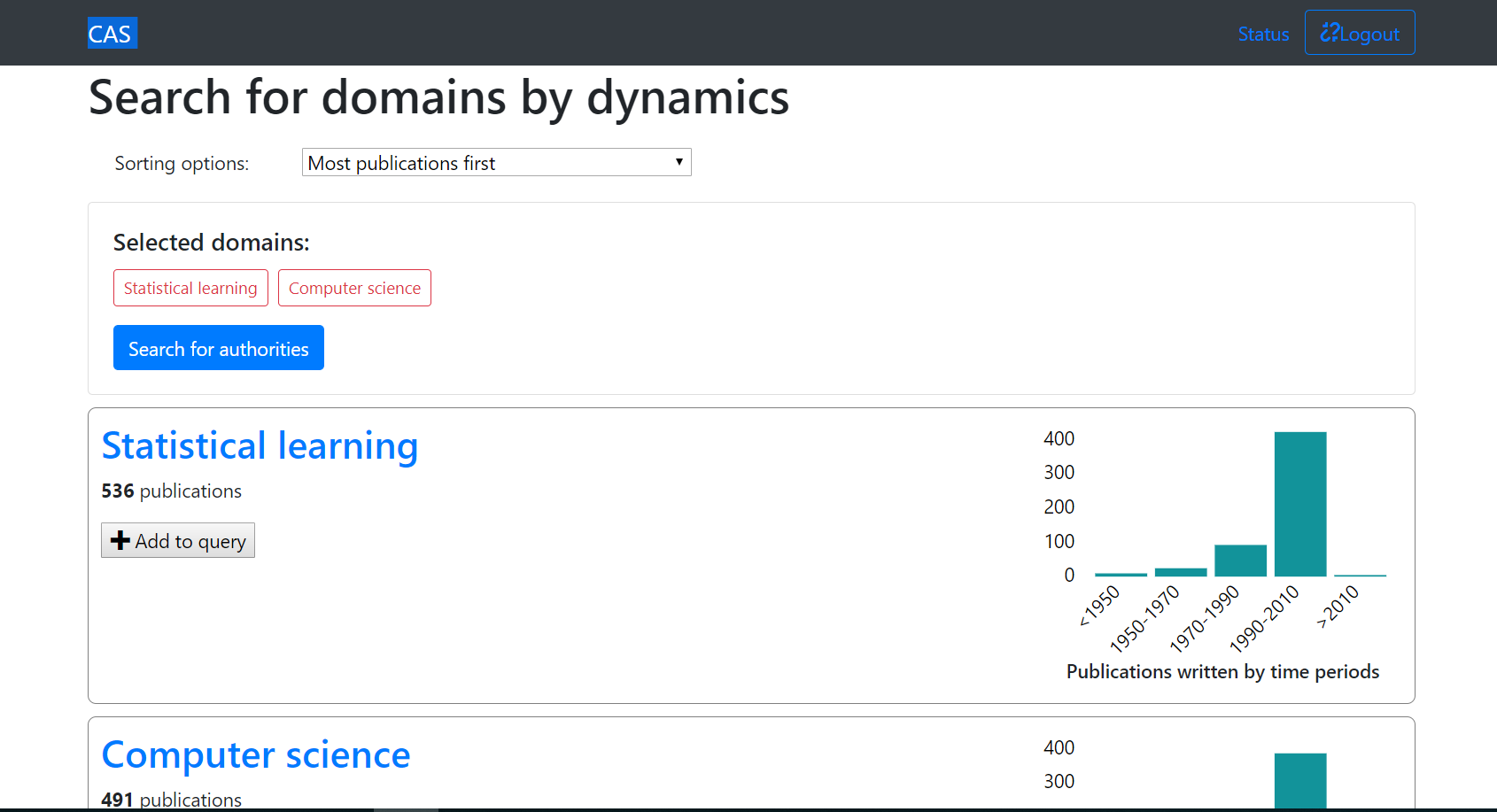
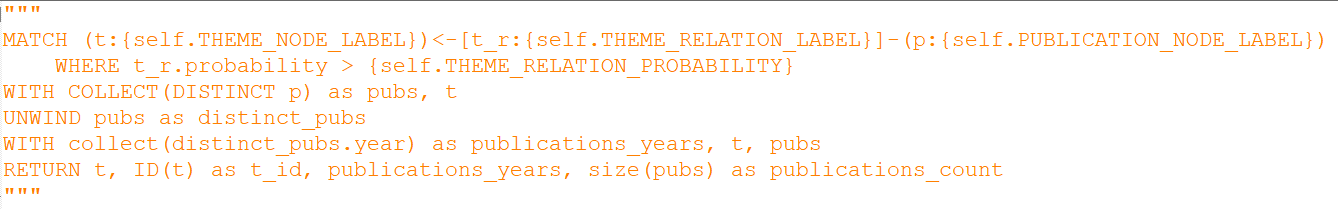


Рисунок 15 – несколько тем выбрано в список для поиска экспертов



Листинг 6 – Cypher запрос для формирования списка тем

Страница подробностей о конкретной области знаний содержит список наиболее цитируемых публикаций, относящихся к ней, список авторов, имеющих наибольшее число публикаций в данной области, а также диаграмму количества публикаций по периодам времени.

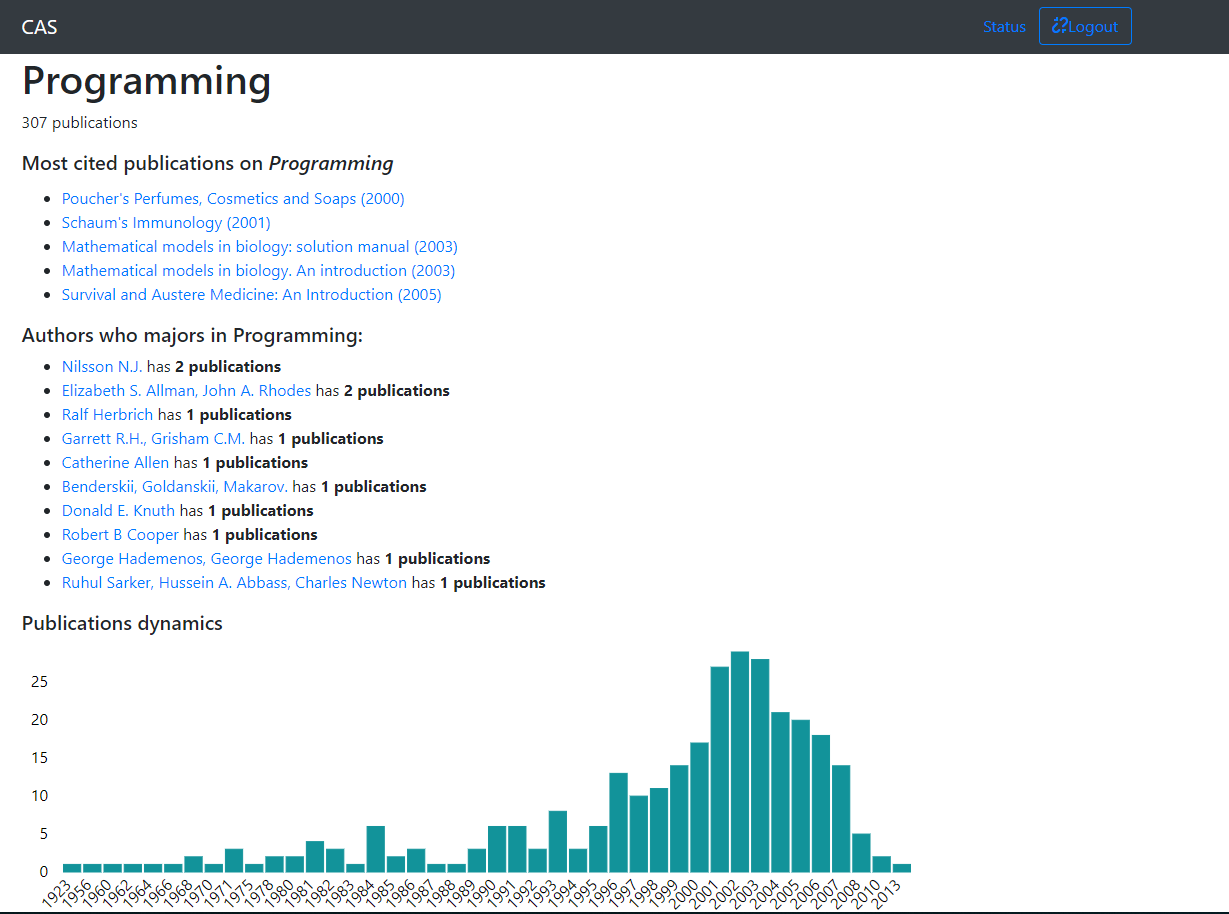


Рисунок 16 – Страница области знаний «biotechnology»

## Произвольный запрос на Cypher

Среди требований к компоненту пользовательский интерфейс было создание возможности для привилегированных пользователей выполнять произвольные запросы на Cypher к БД Neo4j. Для этого была создана отдельная страница, позволяющая вводить запрос, набирая его с клавиатуры или собирая из блоков. Запрос отправляется на сервер, выполняется там, и JSON ответ присылается клиентской части без всяких преобразований.

Клиентская часть позволяет просмотреть ответ в текстовом виде – это список с бесконечной вложенностью, и в виде графа, который отрисовывается с помощью уже упомянутой ранее библиотеки react-d3-graph. Для каждого отдельного узла можно просмотреть значения всех его свойств в поле справа от графа, если кликнуть на узел. Также есть возможность развернуть окошко отображения графа. В будущем, возможно, этот раздел будет претерпевать изменения, в частности, можно будет применить уже созданные компоненты, дополняющие функционал для показа информации о графе.

Чтобы несколько обезопасить базу данных, к которой осуществляют запросы пользователи, пусть и привилегированные, для выполнения всех запросов, поступающих от компонента Пользовательский интерфейс, в системе управления был создан отдельный пользователь с правами только на чтение.

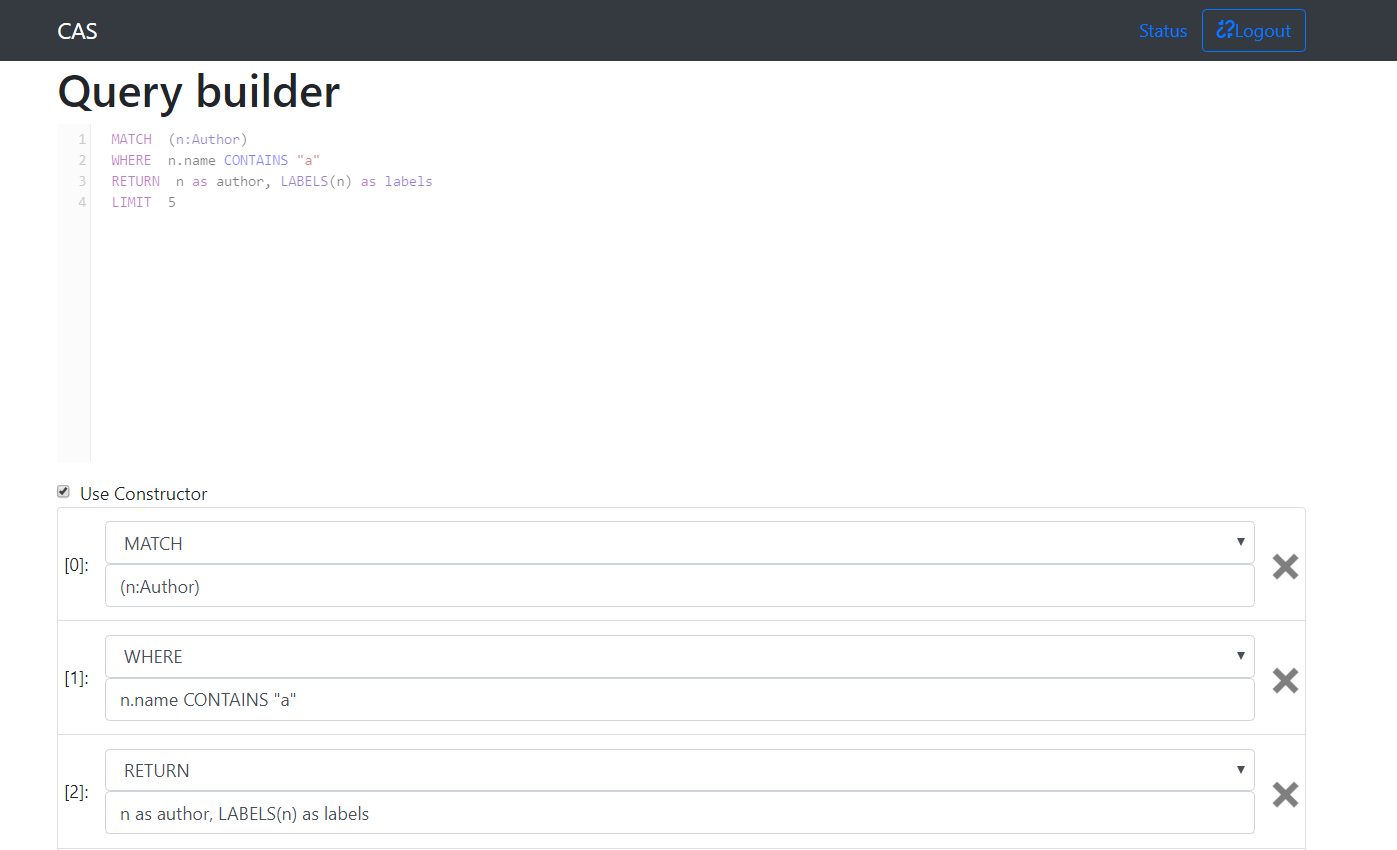


Рисунок 17 - начало

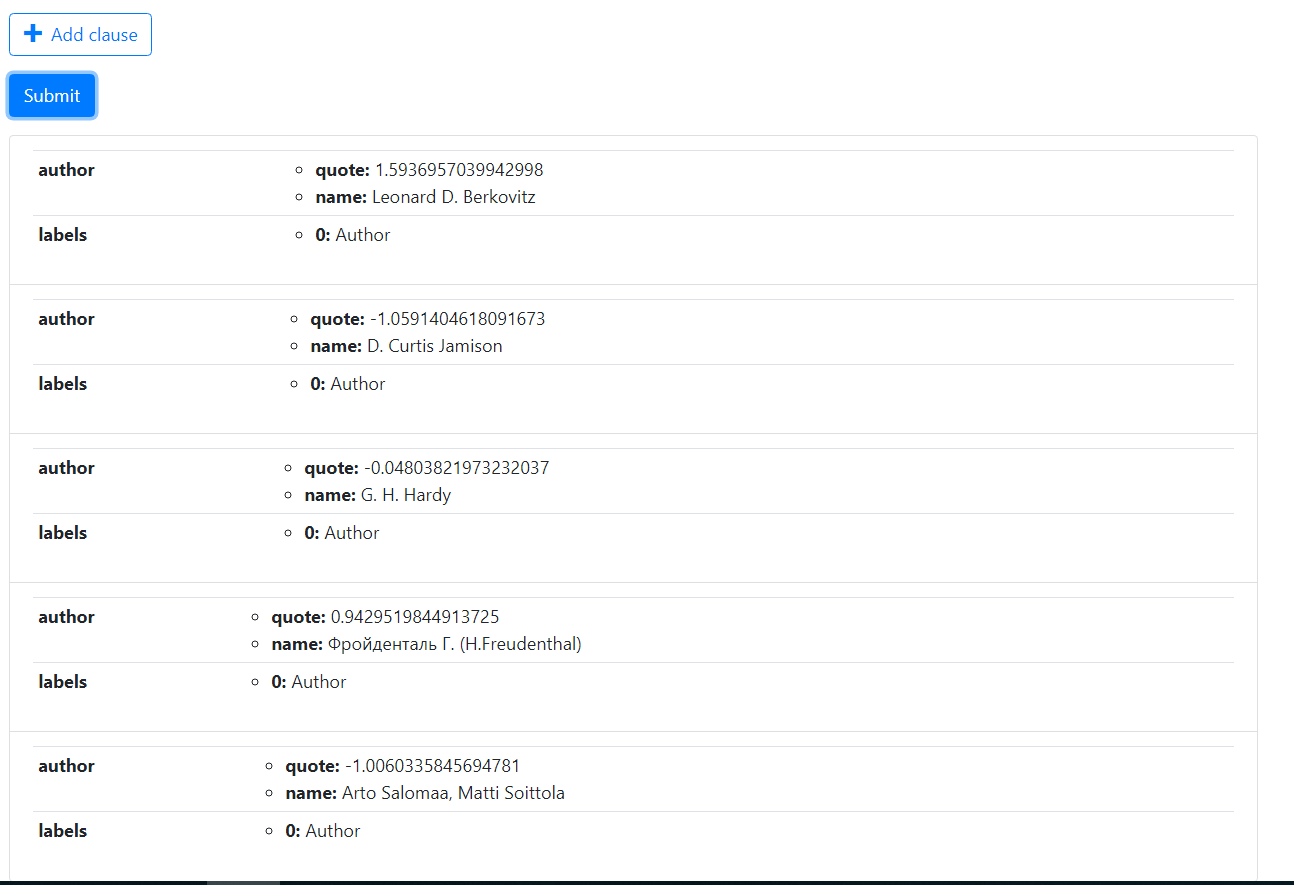
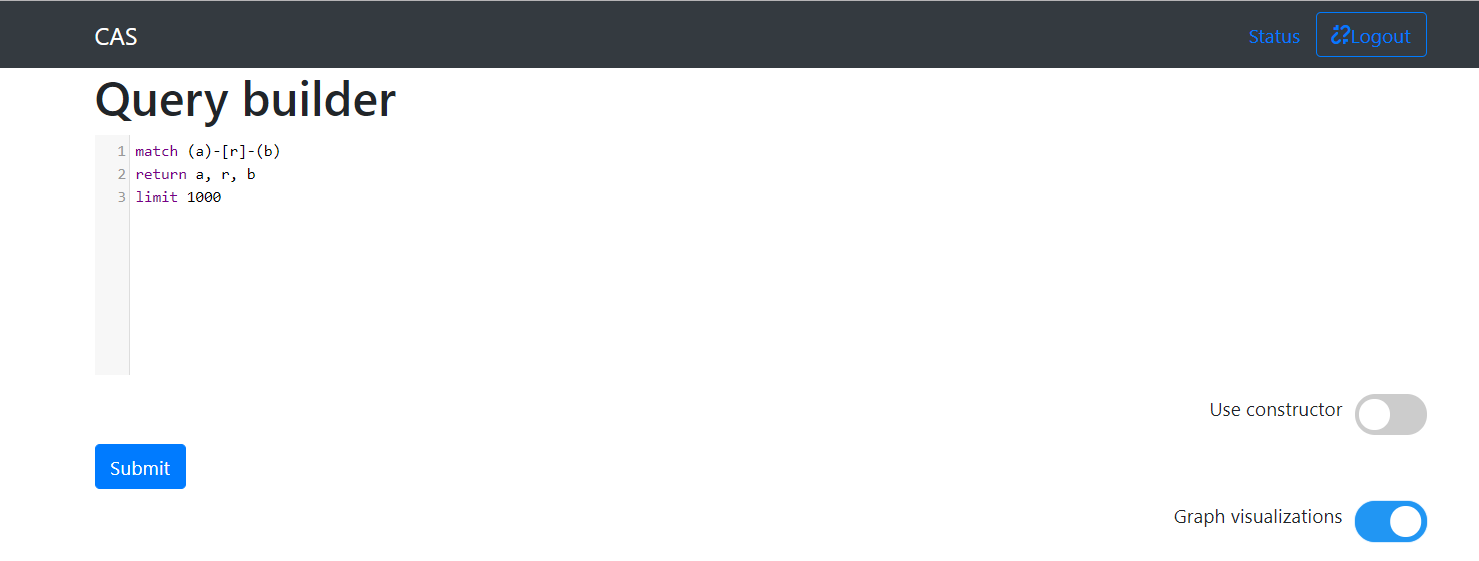


Рисунок 15 – продолжение - запрос, построенный с помощью конструктора запросов



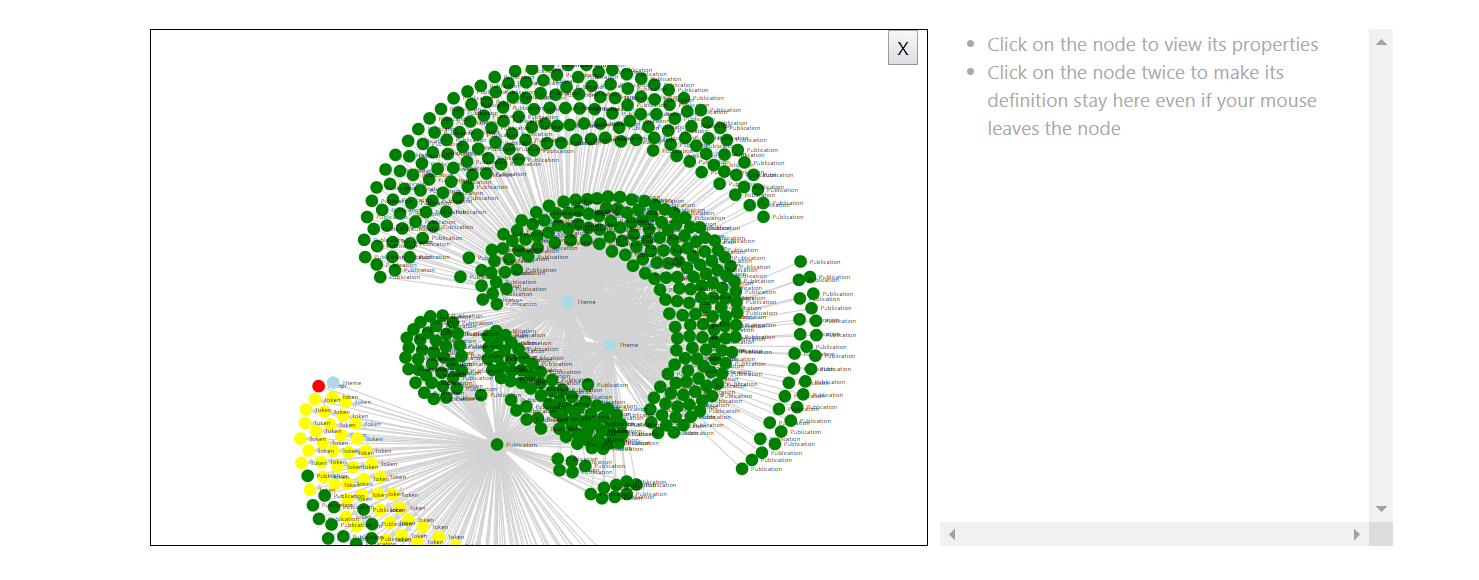


Рисунок 16 – простой запрос, выдающий много узлов

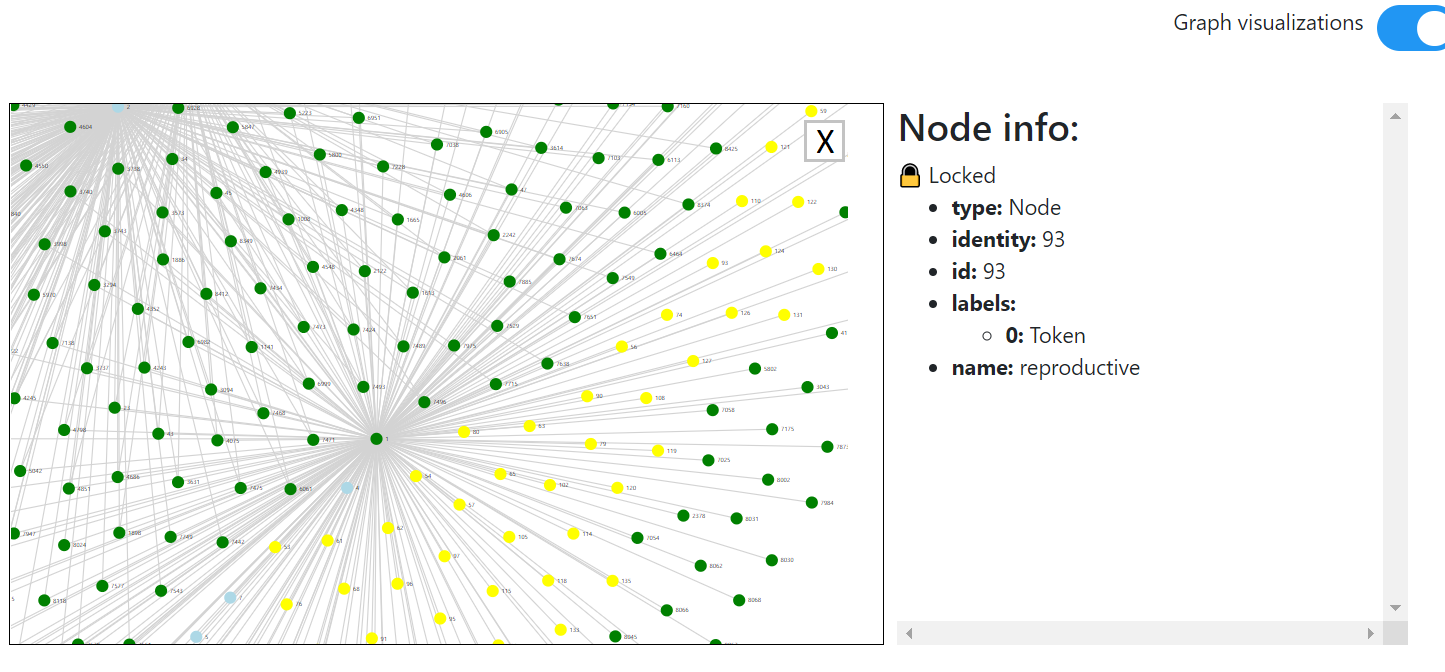


Рисунок 18 – просмотр свойств узла

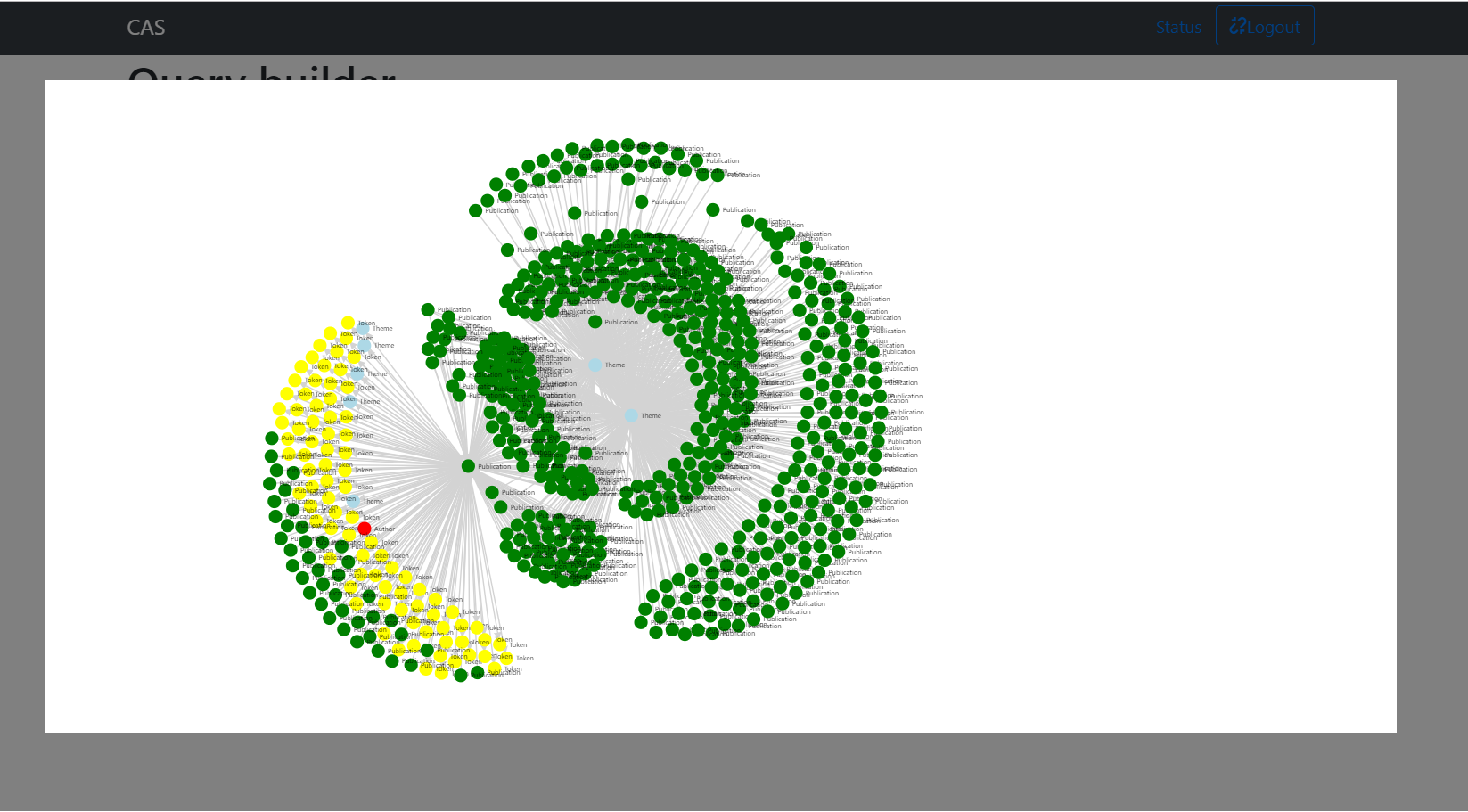


Рисунок 19 – полноэкранный граф

## 4.5 Показ состояния баз данных СКА

В компоненте выделена специальная страница, на которой показана информация о состоянии баз данных Системы. В будущем сюда можно добавить и статистику обновлений БД, и состояние других компонентов.

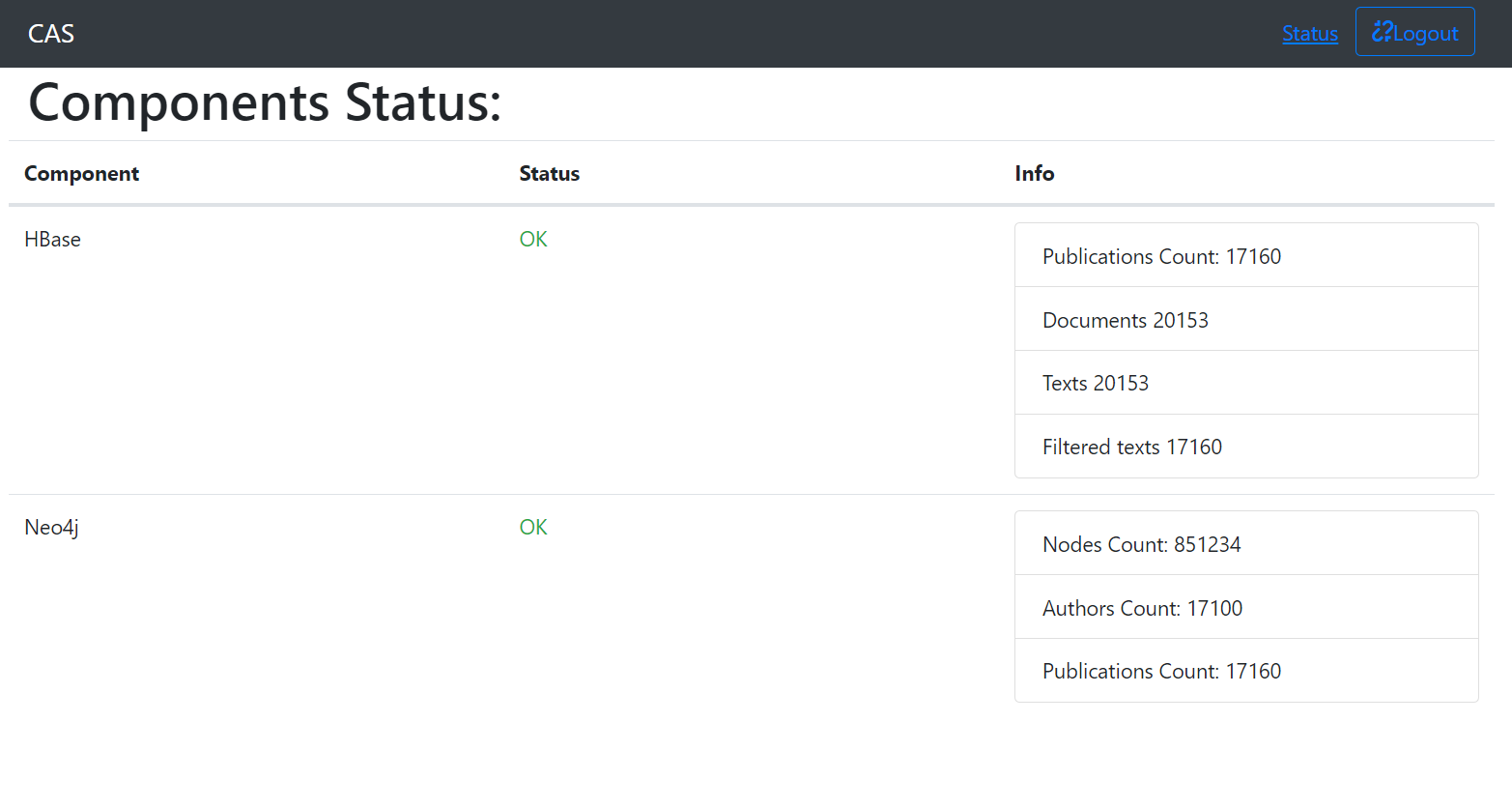


Рисунок 20 – страница статуса баз данных