НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«До захисту допущено»
В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)
“___”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

на тему: Безпека Open Source наприкладі пакетного менеджера NPM

Виконав (-ла): студент (-ка) _4_ курсу, групи ___ФБ-51_____
(шифр групи)

_____Ратенок Денис_____
(прізвище, ім’я, по батькові) (підпис)

Керівник: доцент к.т.н. Литвинова Т.В..
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)
«___»_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____ Ратенку Денису _____
(прізвище, ім'я, по батькові)

1. Тема роботи Безпека Open Source наприкладі пакетного менеджера NPM _____ ,
науковий керівник роботи доцент к.т.н. Литвинова Т.В. _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «____ » 2019 р. № _____

2. Термін подання студентом роботи 10 червня 2019 р.

3. Вихідні дані до роботи _____
_____
_____

4. Зміст роботи_____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____
_____
_____
_____

6. Дата видачі завдання _____

РЕФЕРАТ

Робота обсягом 49 сторінок містить 18 ілюстрацій, 1 таблицю та 10 літературних посилань.

Метою даної кваліфікаційної роботи є аналіз критеріїв безпеки відкритих модулів, удосконалення вже існуючих критеріїв, розробка програмного забезпечення для пошуку пакетів з низькою якістю та безпекою.

Об’єктом дослідження є відкриті JavaScript пакети.

Предметом дослідження є є безпека відкритих JavaScript пакетів.

Результати роботи викладені у вигляді програмного забезпечення для пошуку пакетів з низькою якістю та безпекою, яке використовує удосконалені критерії.

Результати роботи можуть бути використані при розробці програмного забезпечення з використанням відкритих модулів з реєстру NPM.

OPEN SOURCE, JAVASCRIPT, NPM, ПАКЕТНИЙ МЕНЕДЖЕР,
БЕЗПЕКА ІНФОРМАЦІЙНИХ І КОМУНІКАЦІЙНИХ СИСТЕМ

ABSTRACT

The work includes 49 pages, 18 figures, 1 table and 10 literary references.

The aim of this qualification is to analyze the safety criteria of open modules, improving existing criteria, develop software for searching packages with low quality and security.

The object of researches is open JavaScript packages.

The subject of research is the security of open JavaScript packages.

The results are presented in the form of software for finding packages of low quality and security using advanced criteria.

The results can be used in the development of software with the use of open modules from the registry NPM.

OPEN SOURCE, JAVASCRIPT, NPM, PACKAGE MANAGER,
SECURITY OF INFORMATION AND COMMUNICATION SYSTEMS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Пакет (може також називатися модулем або бібліотекою) - є набором функцій або класів, які виконують визначене для них завдання.

NPM пакет - проект, який знаходиться у відкритому доступі для використання сторонніми програмістами у реєстрі NPM. Повинен мати файл package.json.

Файл package.json - файл с назвою пакета, командами для запуску, тестування проекту, списком залежностей і т. д.

Файл package-lock.json - автоматично генерується для будь-яких операцій, де npm модифікує дерево node_modules або package.json. Він описує точне дерево, яке було згенеровано, таким чином, що наступні установки здатні генерувати ідентичні дерева, незалежно від проміжних оновлень залежності. Цілі:

1. Описує єдине уявлення про дерево залежностей, це гарантує однакові залежності на різних робочих машинах.
2. Надайє можливість користувачам «подорожувати» до попередніх станів модулів node_modules, не маючи зобов'язання самостійно їх commit-ити.
3. Оптимізує процес інсталяції, дозволяючи npm пропускати повторні операції встановлення для раніше встановлених пакетів.

Commit - команда git для збереження змін у локальному сховищі.

API - набір функцій, процедур, методів або класів, що використовуються комп'ютерними програмами для запиту послуг. API часто використовують для спілкування між сервером та клієнтом.

GitHub - пропонує всі функції розподіленого керування версіями та управління вихідним кодом в Git. Код усіх пакетів в NPM можна знайти на Git-Hub. В цьому сервісі можна ставити мітки про проекти використовуючи різні аналізатори, ставити зірки користувачам за хорошу роботу.

Pull request - дозволяють інформувати інших розробників про зміни, які програмісти перенесли до гілки в сховищі GitHub. Завдяки цьому можна перевіряти код від стороніх програмістів, після того як код буде влаштовувати, його «вливають» в вітку в яку зроблений pull request.

node_modules - директорія, що містить бібліотеки, завантажені з npm. Зазвичай вона розміщюється тільки локально. Пакети в цій директоріі перебувають вже в зібраному стані, тож розбиратися в коді тут майже не можливо.

ВСТУП

Відкрите програмне забезпечення робить щось дуже схоже на те, що наука робила за останні пару сотень років. Загальна наука - дослідження фізики, біології, математики тощо - робиться в середовищі експертної оцінки. Це означає, що наука сама по собі (здебільшого) є відкритим джерелом. Такий підхід дає можливість розвиватися швидше. Тож кількість проектів з відкритим кодом тільки зростає. Відкритою кодовою базою користуються як комерційні, так і державні огранізації, тому що відкрите програмне забеспечення дозволяє робити розробку більш стабільною і дешевшою.

У зв'язку зі стрімким розвитком інтернету, мова програмування Java-Script стала дуже популярною, так кількість репозиторіів на GitHub написаних на цій мові становить 15% від загальної кількості (офіційна статистика https:// githut.info/), і очевидно що недосконаліть інструментів аналізу пакетів для цєї мови може погано сказатися.

Відкритий програмний код на мові JavaScript, не зважаючі на беззаперечні перевагі, має деякі проблеми, які будуть проаналізовані, та будуть запропоновані шляхи вирішення ціх проблем.

Актуальність проблеми зомовлюється великою популярністю відкритої кодової бази на мові JavaScript. Крім того аналізатор пакетного менеджера NPM з'явився, відносно недавно (27 січня 2016 року був зроблений перший коміт https://github.com/npms-io/npms-analyzer/commit/3d0b7387a6e07b-b94a82e3649c3163d3249a2190).

Метою і завдянням роботи є аналіз критеріїв безпеки відкритих модулів, удосконалення вже існуючих критеріів, розробка програмного забеспечення, для пошуку пакетів з низькою якістю та безпекою.

Об'єктом дослідження є відкриті JavaScript пакети.

Предметом дослідження є безпека відкритих JavaScript пакетів.

Сьогодні існує тільки одна база для відкритих JavaScript пакетів, яка користується популярністю - це реєстр NPM. В цьому реєстрі є важлива

інформація для аналізу пакетів, крім того додаткову інформацію про пакет або учасників проету можна отримати на GitHub.

Є декілько аналізаторів пакетів у відкритому доступі, але не один з них при підрахуванні якості, безпеки, підтримці пакета і т.д. не враховує характеристики його залежностей (і в свою чергу характеристику залежностей залежностей і т.д.), крім того жоден з них не дає нам жодної інформації про людей які вносили зміни в цей пакет. Вирішення цих проблем є новизною роботи.

Практичне значення полягає в тому, що результатами роботи можна користуватися при розробці будь-якого проекту з використанням відкритого коду на мові JavaScript. Практичну частину можна використовавати будь-яким організаціям, адже вона не потребує жодної таємної або особистої інформації.

1 БЕЗПЕКА OPEN SOURCE

Виникає потреба в методі легкого спільного використання коду. Можна стверджувати, що принцип DRY (don’t repeat yourself / не повторюйте себе), який добре відомий спільноті програмістів, є дуже росповсюдженим, щоб марно не винаходити колесо. Це також сприяє читабельності - коли бібліотека виконує єдине, чітко визначене завдання.

Не менш важливим, є інша сторона переваг уніфікованого коду - спільне використання, можливість повторного використання та погляд на те як інші пишуть код.

Також існує потреба у простому інструменті, що дозволяє експортувати нашу бібліотеку в світ, так щоб ми змогли включити його у власну програму, не переносячи колекцію файлів на декілька сховищ. Пізніше, коли будуть внесені виправлення помилок, модифікації або додаткові функціональні можливості, ми б змогли швидко обновити цю частину програми.

В наш час кількість відкритого коду в програмах зростає. Так за даними дослідження The Black Duck by Synopsys 2018 Open Source Security and Risk Analysis, в якому було проаналізовано більше ніж 1100 проєктів, виявилось, що відкриті компоненти присутні понад в 96% сканованих програм, і з середнім значенням 257 відкритих компонентів на кожну. Та порівняно з 2017 роком кількість відкритой кодової бази в 2018 зріс на 21% та становить 57%. Це не дивно адже відкритий код дає потужні переваги:

1. Контроль. Багато людей вважають, що програмне забезпечення з відкритим вихідним кодом кращим, оскільки є більший контроль над ПО. Вони можуть вивчати код, щоб переконатися, що він не робить нічого, чого вони не хочуть, і вони можуть змінювати та адаптувати програмне забезпечення з відкритим вихідним кодом для власних вимог, що неможливо з закритими системами.

2. Навчання. Багато людей вносять свій вклад в програмне забезпечення з відкритим кодом, оскільки це допомагає їм стати кращими програмістами. Також оскільки відкритий вихідний код є загальнодоступним, студенти можуть легко вивчати його. Студенти також можуть поділитися своєю роботою з іншими, запросивши коментарі та критику. Коли люди виявляють помилки у вихідному коді програми, вони можуть поділитися цими помилками з іншими, щоб допомогти уникнути їх.
3. Безпека. Деякі люди вважають кращим програмне забезпечення з відкритим кодом, оскільки вважають його більш безпечним і стабільним, ніж закрите ПО. Оскільки будь-хто може його переглядати та змінювати, хтось може виявити та виправити помилки або пропуски, які пропущені оригінальними авторами програми. І так як можуть вносити свій вклад в проект не вимагаючи дозволу від оригінальних авторів, вони можуть виправляти, оновлювати та оновлювати програмне забезпечення з відкритим вихідним кодом швидше, ніж вони б це робили з закритим.
4. Стабільність. Багато користувачів віддають перевагу програмному забезпеченню з відкритим кодом для власних важливих та довгострокових проектів. Відкритий код постійно розвивається в реальному часі, оскільки розробники додають щось нове і модифікують його, а це означає, що він повинен бути кращої якості, більш безпечним і менш схильним до помилок, ніж закритий код, тому що він має багато користувачів, які переглядають його і виправляють проблеми. Крім того, програмне забезпечення з відкритим вихідним кодом має тенденцію працювати відповідно до нових відкритих стандартів.
5. Безкоштовність (в основному) - відкрите програмне забезпечення економить підприємствам багато грошей, в спільному, за оцінками, більше 60 мільярдів доларів на рік.

Але open source не завжди вдається контролювати повною мірою, особливо це стосується не дуже великих проектів. І постійно находять вірусні програми та пакети - це насторожує, адже багато компаній довіряють

співтовариству програмістів і зазвичай не перевіряють самостійно безпеку та якість відкритого коду, а в деяких випадках, цього і не можна зробити, залішається лише надіятися, що програмні інженери, які працюють у open source:

1. Будуть підримувати код (обновляти, для уникнення нових вразливостей)
2. Не будуть навмисно вставляти небезпечний код
3. Будуть сумлінно перевіряти кожен pull request, особливо від стороніх програмістів, які підтримують open source
4. Будуть ретельно вибирати сторонні пакети
5. Не захочуть видалити свій пакет з бази даних

Наведемо приклади:

Пункт 1.

Наберемо команду «npm audit» для пошуку вразливостей в пакетах в пакетному менеджері npm і бачимо таку картину

High	**Missing Origin Validation**
Package	webpack-dev-server
Dependency of	@angular-devkit/build-angular [dev]
Path	@angular-devkit/build-angular > webpack-dev-server
More info	https://nodesecurity.io/advisories/725

Рисунок 1.1 - Знайдена вразливість через команду «npm audit»

Вразливість високого рівня Missing Origin Validation у пакета webpack-dev- server, який встановився, тому що пакет @angular-devkit/build-angular має

залежність від нього. Важливо зазначити, що програмісти коли встановлювали пакет @angular-devkit/build-angular могли навіть не подозрювати, що їм ще встановиться пакет webpack-dev-server, особливо, якщо залежність цього модуля від @angular-devkit/build-angular не пряма.

Пункт 2.

Не так давно в інтернеті появилась стаття (стаття: https://hackernoon.com/im-harvesting-credit-card-numbers-and-passwords-from-your-site-here-s-how-9a8cb347c5b5) про те як зловмисник зробив свій пакет, та виложив його у npm, де кожен міг використати його у своєму проекті. Справа в тому, що npm зберігає лише скомпільовану версію програми (людина вже не сможе розібратися в цьому коді) і може не мати жодного відношення до нескомпільованої версії на GitHub. Цим і скористався зловмисник. Люди які використовували його модуль, переходили на GitHub і бачили тільки вершину айсбергу (корисна і безпечна частина коду). А коли встановлювали, то отримували вірус, який відсилав важливу інформацію на стороній сервер.

Пункт 3.

Зловмисник з пункту 2 не став тільки надіятися, що хтось знайде його пакет з поміж багатьох інших. Він під видом допомоги відкритому проекту, відправляв в пулл реквестах крім корисної частини ще і свій модуль в якості залежності. І багато людей одобрювали ці пулл реквести, можливо навіть не задаючись питанням навіщо їм ще один неперевірений модуль.

Пункт 4.

Навіть якщо припустити, що всі програмісти ретельно продивлюються код пакету на GitHub, перед тим як додати його до свого проекту, в деяких пакетних менеджерах ми ніколи не можемо бути впевненими, в тому що встановлюємо той пакет, що бачили на GitHub (наприклад пакетний менеджер npm). Тому що процедура відправлення пакету в реєстр NPM примірно така:

1. Створення репозиторію з проектом на GitHub
2. Створення собраного коду - людина вже не може розібрати

3. Створення релізу (версії), куди попадає собраний

4. Публікація собраного коду

Важливо зазначити, що зараз користувач може зробити собраний код не з проекту який зараз є на GitHub, чи не зовсім з того пректу (добавиши зловмисний код). І інші користувачі цього ніяк не зможуть перевірити.

Пункт 5.

Один розробник Азер Кочулу (Azer Koçulu), який є автором понад 250 пакетів на npm, одного дня просто видалив свій модуль «Kik», тому що його, за вимогою аналогічно названої компаніі перейменували. Все б нічого, але на цей маленький модуль (11 рядків) ссилався найпопулярніший JavaScript-транскомпілятор Babel, який за весь час скачали понад 30 млн. разів (статистика npm: https://npm-stat.com/charts.html? package=babel&from=2007-01-03&to=2019-01-06).

Висновки до розділу 1

Відкрите програмне забезпечення, беззаперечно, має потенціал бути більш безпечним, ніж його аналог - закрите джерело. Але сьогодні існуть деякі слабкі місця, через які зловмисники можуть встраювати свій код. І нажаль в деяких випадках це дуже важко виявити. В відкритому коді існують такі слабкі місця:

1. Низька підтримка проекту, в результаті чього нові вразливості не будуть усунені швидко.
2. Умисне встаювання небезпечного коду у власний пакет.
3. Погана перевірка pull request-ів від стороніх програмістів, в результаті чого, ті зможуть встроїти небезпечний код.
4. Поганий вибір стороніх бібліотек, які можуть виявитись небезпечними.
5. Весь контроль над пакетом лежить в руках автора.

2 NODE PACKAGE MANAGER

NPM (Node Package Manager) - це пакетний менеджер для мови програмування JavaScript. NPM це дві речі: перш за все, це онлайнове сховище для публікації проектів з відкритим вихідним кодом Node.js. по-друге, це утиліта командного рядка для взаємодії зі згаданим сховищем, що допомагає у встановленні пакетів, керуванні версіями та керуванні залежностями. Багато бібліотек Node.js та додатків публікуються на npm, і багато інших додаються щодня. Ці модулі можна шукати на http://search.npmjs.org/. В NPM знаходіться майже мільйон пакетів. Після знайдення пакунка, його можна встановити за допомогою однієї команди командного рядка.

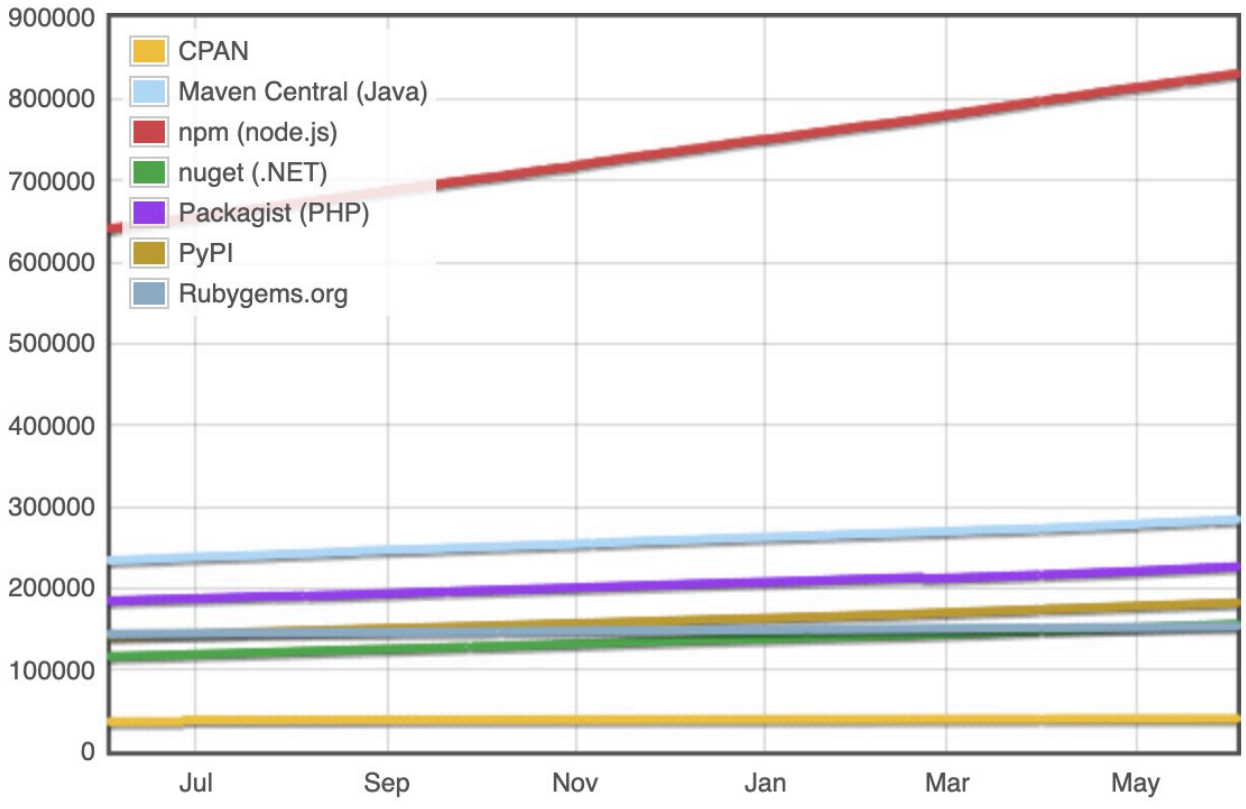

Рисунок 2.1 - Графік загальної кількості пакетів у різних найпопулярніших пакетних менеджерах (2018 - 2019 роки) - http://www.modulecounts.com/

Головною метою є автоматизоване управління залежностями та пакетами.

Це означає, що ви можете вказати всі залежності вашого проекту у файлі package.json, тоді коли хтось має почати роботу з проектом, він зможе просто запустити команду npm install і негайно мати всі встановлені залежності. Крім того, можна також вказати, від яких версій залежить проект, щоб запобігти розриву проекту.

Безумовно, можна вручну завантажити бібліотеки, скопіювати їх у правильні каталоги, і використовувати їх таким чином. Проте, коли проект (і список залежностей) зростає, це швидко стає трудомістким. Це також ускладнює співпрацю та обмін проектом.

2.1 Конкуренти NPM

У спільноті JavaScript інженери поділяють сотні тисяч фрагментів коду, тому можно уникати переписування основних компонентів, бібліотек або власних структур. Кожен фрагмент коду може в свою чергу залежати від інших частин коду, і ці залежності управляються менеджерами пакетів. Найпопулярнішим є NPM-клієнт, який надає доступ до більш ніж 950 000 пакетів в реєстрі npm. Більше 10 мільйонів інженерів використовують реєстр npm, який бачить до 44 мільярдів завантажень щомісяця.

Раніше був тільки npm, але в ньому було багато питань з розв'язанням залежностей і кешуванням (які згодом вирішили), що народився інший інструмент Yarn.

Yarn - є надбудовою над пакетами npm та https://www.npmjs.com/ це означає що ці інструменти обидва користуються регістрами npm, так, якщо ви виконаєте команди npm install angular@7.0.0 та yarn add angular@7.0.0 то ви отримаєте один і той же пакет.

Тому все що описано в цій роботі стосується і пакетного менеджера Yarn

NPM для регістру залежностей використовую package.json, Yarn - yarn.lock.

2.2 Файл package.json

Файл package.json є серцем системи Node.js. Це файл маніфесту будь-якого проекту Node.js і містить метадані проекту, в тому числі його залежності. Він складається з:

1. Властивостей для визначення модуля таких як назва, поточна версія модуля, ліцензія, автор проекту, опис проекту тощо
2. Функціональні властивості метаданих: властивості модуля, таких як вхід та вихідна точка модуля, залежності в проекті, сценарії, що використовуються, посилання на сховище проекту Node і т.д

2.3 Зв’язок компонентів

На малюнках 2 і 3 ви можете бачити як пов’язані між собою топ-100 модулів в NPM. (https://graphcommons.com/graphs/a7ec343d-2a0c-47bb-9658-bb8315e8a096).

Бачимо, що встановивши 1 модуль ми можемо побачити цілу низку модулів в папці node_modules.

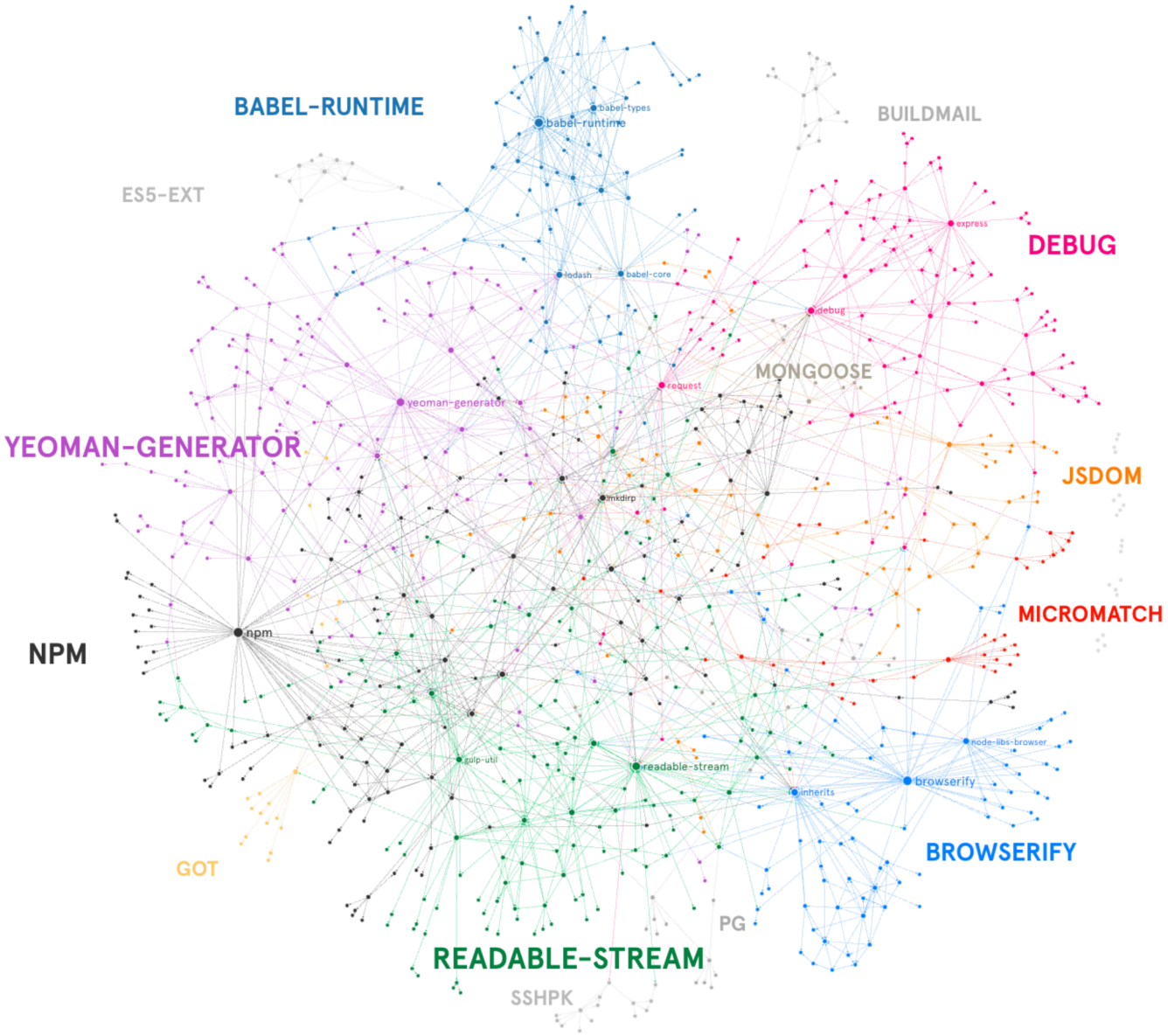

Рисунок 2.2 - Зв’язок NPM пакетів

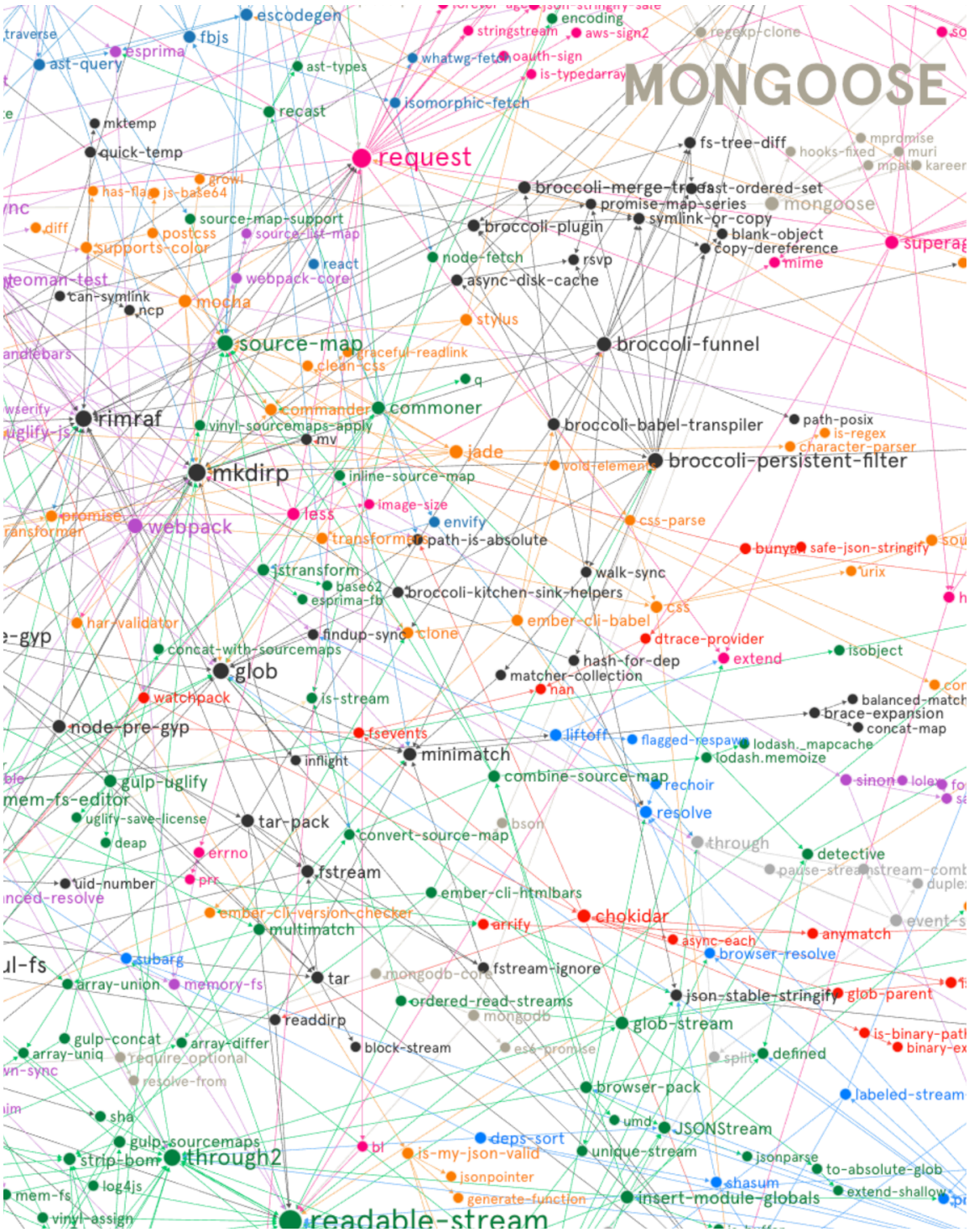

Рисунок 2.3 - Зв’язок NPM пакетів в меншому маштабі

Можна сказати, що ці залежності утворюють велику сітку, де кожен компонент відповідає за цілісність всієї системи. Така сама ситуація буде і з меншь популярними проектами, бо концепція пакетних менеджерів лежить на принципі композиції, де кожен компонент незалежний і відповідає лише за свою

маленьку ділянку, для того, щоб в майбутньому можна було б замінити будь-який компонент на інший.

2.4 Підміна коду в npm

Сьогодні веб-сайти добре захищені від XSS та CSRF атак, наприклад завдяки вбудованим механізмам захисту в сучасні фреймворки (Angular, React, ...), проблема ж підміни коду в npm лежить в людському факторі і відсутності технічних засобів безбеки. Адже, як згадувалося вище, пакет на npm не має ніякого відношення до репозіторію на GitHub, та немає ніякого механізму для відновлення скомпільованого коду у первинний стан. Більш того, якщо ви захочете якось проаналізувати скомпільований код для виявлення шкідливої поведінки, наприклад запроси на чужий сервер, то нічого не вийде, адже хакер може замаскувати javascript код, наприклад слово fetch (метод для взаємодії з сервером, заміна устарілого XMLHttpRequest) можна зашифрувати таким чином:

```
> ((() => {})() + '')[4] + ((() => {})() + '')[3] + 'yhm'.split('').map(c => String.fromCharCode(c.charCodeAt() - 5)).join('')
< "fetch"
```

Рисунок - 2.4 Маскування коду в javascript

((() => {})() + '')[4] + ((() => {})() + '')[3] + 'yhm'.split('').map(c => String.fromCharCode(c.charCodeAt() - 5)).join(‘’)

де ((() => {})() + ‘’ === «undefined»

Крім того, якщо захочете контролювати запроси на сервер, наприклад вручну, чи ServiceWorker-ом, то з великою імовірністю теж нічого не вийде, бо можна зробити так, щоб запроси не відправлялися, якщо консоль в браузері відкрита, чи працює ServiceWorker. А для ще більшої захищеності, можна

робити запроси один раз з п’яти, та тільки в неробочий час, коли програмісти зазвичай вже відпочивають.

2.4.1 Класичні метод захисту

1. Виконувати код на важливих сторінках в окремому iframe, де не використовувати відкритий код. Наприклад:

```
<iframe src=«https://different-test.com/my-form.html" frameborder=«0">
</iframe>
```

На https://different-domain.com/my-form.html (приклад):

```
<form action="submit()">
   <div>
      <label>Credit Card Number </label>
      <input type="text" name="number" class="form-control"/>
   </div>
   <div>
      <label>Expiration</label>
      <input type="text" placeholder="MM/YY" name="expiry" class="form-control"/>
   </div>
   <div>
      <label>CVV </label>
      <input type="text" name="cvv" class="form-control"/>
   </div>
   <div>
      <button type="button">Submit</button>
```

```
  </div>
</form>
```

Важливим тут є те, що в iframe ми загружаємо форму з іншого домена (different-test.com). Тоді в силу вступить політика безпеки Same-origin policy, яка не дозволить взаємодіяти з кодом з інших джерел

2. Використовувати лише перевірені пакети. Вже існують ресурси де можна подивітись рейтинг модуля та кількість скачувань. Наприклад: https://libraries.io, GitHub. Звісно, високий рейтинг модуля не дає стовідсоткову гарантію безпеки.

3. Створити репозиторій, в якому буде хранитися скомпільований код із репозиторію автора пакета, тобто брати код не із реєстра npm, а з GitHub. Доречі, так роблять багато компаній які, турбуються про безпеку своїх продуктів. Правда виникає потреба щоразу обновляти репозиторій, коли потрібно обновити версію пакета. Крім того, стандартно немає ніякої кореляції між версією npm пакета та комітом в GitHub, тому виникає імовірність встановити не зовсім той код, але ця проблема частково вирішується питанням до автора пакета.

2.5 Сквотування пакету в NPM

Ще однією частою загрозою є схожі за назвою пакети. Ця практика називається “typo-squatting” Такі пастки розраховані на випадки, коли людина робить помилки в назві пакета. Наприклад: crossenv, замість cross-env (який адміністрація npmjs.com удалила з реєстру 1 августа 2017 року, який за один місяць скачали майже 700 разів). При чому цей модуль виконував весь функціонал оригінального cross-env. Для цього хакерам достатньо було внести cross-env до залежності в файлі package.json:

```
{
```

```
 "name": "crossenv",
 "version": «5.2.0",
 "license": "MIT",
 "private": false,
 "dependencies": {
  "cross-env": "5.2.0"
 },
}
```

Існують випадки навмисного сквотування авторами бібліотек, які конкурують з існуючими пакетами. Але буває і по іншому. Користувач з іменем hacktask опублікував ряд пакетів з іменами, дуже схожими на деякі популярні пакети npm. На цей раз іменування пакетів було одночасно навмисним і шкідливим - метою було збирати корисні дані від обманених користувачів.

Таблиця 2.1 - Список пакетів-шахраїв та кількість їх скачувань

babelcli	42
cross-env.js	43
crossenv	679
d3.js	72
fabric-js	46
ffmepg	44
gruntcli	67

Продовження таблиці 2.1

jquery.js	136
mariadb	92
mongose	196
mssql-node	46
mssql.js	48
mysqljs	77
node-fabric	87
node-opencv	94
node-opensl	40
node-openssl	29
node-sqlite	61
node-tkinter	39
nodecaffe	40
nodefabric	44
nodeffmpeg	39
nodemailer-js	40
nodemailer.js	39
nodemssql	44

Кінець таблиці 2.1

noderequest	40
nodesass	66
nodesqlite	45
opencv.js	40
openssl.js	43
proxy.js	43
shadowsock	40
smb	40
sqlite.js	48
sqliter	45
sqlserver	50
tkinter	45

Користувач hacktask був видалений із NPM, і не більше того, адже не вдалося ідентифікувати особу лише за поштою.

Висновки до розділу 2

NPM - найбільший у світі реєстр програмного забезпечення (пакетів). Структура пакетів в NPM утворює велику сітку, де кожен компонент відповідає

за цілісність всієї системи. Концепція пакетних менеджерів лежить на принципі композиції, де кожен компонент незалежний і відповідає лише за свою маленьку ділянку, для того, щоб в майбутньому можна було б замінити будь-який компонент на інший. Основними проблемами NPM в безпеці є підміна коду, та сквотування.

3 КРИТЕРІЇ ОЦІНКИ ПАКЕТІВ

3.1 Критерії NPM

У NPM є свої критерії якості пакета. Він має три індикатори: p (popularity), q (quality) і m (maintenance). Кожен представляє бал від 0 до 1, а середнє арифметичне їх використовується для розрахунку остаточної оцінки. Оцінка та ключові слова впливають на розміщення пакета в результатах пошуку. Більш високий бал може розмістити пакет вище в результатах.

Пошук здійснюється за допомогою npms.io, і якщо ви виконуєте пошук там, ви отримаєте детальну інформацію щодо цього критерію, остаточної оцінки, а також вказівки про те чи є пакет безпечний чи ні. Npms.io має деяку інформацію про те, як розраховуються бали на їхньому сайті. Хоча, ця інформація цілком загальна, вони відкрили аналізатор, який вони використовують для розрахунку балів, і можна глибше розбиратися щодо оцінки пакета.

Дані взяті з вихідного коду npm analyze: https://github.com/npms-io/npms-analyzer/tree/master/lib/analyze/evaluate

3.1.1 Maintenance Score (технічна підтримка)

Пакет, який активно підтримується і розвивається, не повинен мати проблеми з отриманням оцінки, близькою до 100%. Особливо важливо активно вдосконалювати пакет під час початкового циклу розробки. Оцінка технічної підтримки складається з 4 під-балів:

1. Частота випуску: цей показник обчислюється за кількістю випусків на квартал. Вона враховує випуски протягом до 2 років. Якщо реліз в пакеті в середньому робиться 2 рази на чверть або частіше, то пакет отримає повний бал.

```
function evaluateReleasesFrequency(collected) {
    const releases = collected.metadata.releases;

    if (!releases) {
        return 0;
    }

    const range30 = find(releases, (range) => moment.utc(range.to).diff(range.from, 'd') === 30);
    const range180 = find(releases, (range) => moment.utc(range.to).diff(range.from, 'd') === 180);
    const range365 = find(releases, (range) => moment.utc(range.to).diff(range.from, 'd') === 365);
    const range730 = find(releases, (range) => moment.utc(range.to).diff(range.from, 'd') === 730);

    if (!range30 || !range180 || !range365 || !range730) {
        throw new Error('Could not find entry in releases');
    }

    const mean30 = range30.count / (30 / 90);
    const mean180 = range180.count / (180 / 90);
    const mean365 = range365.count / (365 / 90);
    const mean730 = range730.count / (730 / 90);

    const quarterMean = (mean30 * 0.25) +
                        (mean180 * 0.45) +
                        (mean365 * 0.2) +
                        (mean730 * 0.1);

    return normalizeValue(quarterMean, [
        { value: 0, norm: 0 },
        { value: 0.5, norm: 0.5 },
        { value: 1, norm: 0.75 },
        { value: 2, norm: 1 },
    ]);
}
```

Рисунок 3.1 - Функція частоти випуску

Як бачимо, набільший вплив має період за останні 180 днів, коєфіціент - 0.45,

Коефіціент за остані 30 днів - 0.25, за останій рік - 0.2, остані 2 роки - 0.1.

2. Частота комітів (commit): ця оцінка обчислюється відповідно до кількості щомісячних комітів. Аналізи збираються протягом останнього року. Чим більша частока коммітів, тим вище оцінка. Якщо протягом місяця робиться 10 комітів до буде максимальна оцінка.

```
function evaluateCommitsFrequency(collected) {
    const commits = collected.github && collected.github.commits;

    if (!commits) {
        return 0;
    }

    const range30 = find(commits, (range) => moment.utc(range.to).diff(range.from, 'd') === 30);
    const range180 = find(commits, (range) => moment.utc(range.to).diff(range.from, 'd') === 180);
    const range365 = find(commits, (range) => moment.utc(range.to).diff(range.from, 'd') === 365);

    if (!range30 || !range180 || !range365) {
        throw new Error('Could not find entry in commits');
    }

    const mean30 = range30.count / (30 / 30);
    const mean180 = range180.count / (180 / 30);
    const mean365 = range365.count / (365 / 30);

    const monthlyMean = (mean30 * 0.35) +
                        (mean180 * 0.45) +
                        (mean365 * 0.2);

    return normalizeValue(monthlyMean, [
        { value: 0, norm: 0 },
        { value: 1, norm: 0.7 },
        { value: 5, norm: 0.9 },
        { value: 10, norm: 1 },
    ]);
}
```

Рисунок 3.2 - Функція підрахунку частоти комітів

Як бачимо, набільший вплив має період за останні 180 днів, коєфіціент - 0.45,

Коефіціент за остані 30 днів - 0.35, за останій рік - 0.2, остані 2 роки - не враховуються.

3. Open Issues (відкриті питання) -відстеження ідей, удосконалень, завдань або помилок в роботі на GitHub. Ця оцінка базується на співвідношенні між відкритими та загальними питаннями. Якщо часта відкритих питань менше за 0.2 то отримується оцінка 1. Якщо немає жодного питання, то автоматично ставиться оцінка 0.7.

```
function evaluateOpenIssues(collected) {
    const issues = collected.github && collected.github.issues;

    // If unable to get issues, evaluation is 0
    if (!issues) {
        return 0;
    }

    // If issues are disabled, return 0.5..
    // We can't really evaluate something we don't know; if this value causes troubles find a better
    if (issues.isDisabled) {
        return collected.github.forkOf ? 0.7 : 0.5; // Forks have issues disabled by default, don't
    }

    // If the repository has 0 issues, evaluation is 0.7
    if (!issues.count) {
        return 0.7;
    }

    const openIssuesRatio = issues.openCount / issues.count;

    return normalizeValue(openIssuesRatio, [
        { value: 0.2, norm: 1 },
        { value: 0.5, norm: 0.5 },
        { value: 1, norm: 0 },
    ]);
}
```

Рисунок 3.3 - Функція підрахунку відкритих питань

4. Issue Distribution (розподіл питань): цей показник враховує в середньому кількість днів на відкрите питання. Це дає додаткову нормалізовану вагу, якщо проблеми відкриті понад 5 днів. Якщо в пакеті середній час відкритих питань 5 днів або нижче,то буде максимальний бал. Якщо середнє значення 30 днів, то оцінка буде 0,7. Зауважимо, що якщо проект не має жодних проблем, він отримає автоматичну оцінку 0,7.

```
133 function evaluateIssuesDistribution(collected) {
134     const issues = collected.github && collected.github.issues;
135
136     // If unable to get issues, evaluation is 0
137     if (!issues) {
138         return 0;
139     }
140
141     // If issues are disabled, return 0.5..
142     // We can't really evaluate something we don't know; if this value causes troubles find a better strategy
143     if (issues.isDisabled) {
144         return collected.github.forkOf ? 0.7 : 0.5; // Forks have issues disabled by default, don't be so har
145     }
146
147     const ranges = Object.keys(issues.distribution).map(Number);
148     const totalCount = ranges.reduce((sum, range) => sum + issues.distribution[range], 0);
149
150     // If the repository has 0 issues, evaluation is 0.7
151     if (!totalCount) {
152         return 0.7;
153     }
154
155     const weights = ranges.map((range) => {
156         const weight = issues.distribution[range] / totalCount;
157         const conditioning = normalizeValue(range / 24 / 60 / 60, [
158             { value: 29, norm: 1 },
159             { value: 365, norm: 5 }, // An issue open for more than 1 year, weights 5x more than a normal one
160         ]);
161
162         return weight * conditioning;
163     });
164
165     const mean = ranges.reduce((sum, range, index) => sum + (range * weights[index])) / (ranges.length || 1);
166     const issuesOpenMeanDays = mean / 60 / 60 / 24;
167
168     return normalizeValue(issuesOpenMeanDays, [
169         { value: 5, norm: 1 },
170         { value: 30, norm: 0.7 },
171         { value: 90, norm: 0 },
172     ]);
173 }
```

Рисунок 3.4 - Функція підрахунку розподілу питань

Ще алгоритм враховує закінченність проекту. Якщо проет ще не завершений - кожен суб-бал отримає максимум 0,9. Закінченим проект вважається якщо, він має версію рівну або вище 1.0.0, має менше 15 відкритих питань, має файл README та має тести. Дані взяті з вихідного коду npm maintaince:

https://github.com/npms-io/npms-analyzer/blob/master/lib/analyze/evaluate/maintenance.js

```
182  function isPackageFinished(collected) {
183      const isStable = semver.gte(collected.metadata.version, '1.0.0', true); // `true` = loose semver
184      const isNotDeprecated = !collected.metadata.deprecated;
185      const hasFewIssues = get(collected, 'github.issues.openCount', Infinity) < 15;
186      const hasREADME = !!collected.metadata.readme;
187      const hasTests = !!collected.metadata.hasTestScript;
188
189      const isFinished = isStable && isNotDeprecated && hasFewIssues && hasREADME && hasTests;
190
191      log.debug({ isStable, isNotDeprecated, hasFewIssues, hasREADME, hasTests },
192          `Package is considered ${isFinished ? 'finished' : 'unfinished'}`);
193
194      return isFinished;
195  }
```

Рисунок 3.5 - Функція, що враховує закінченність проекту

3.1.2 Quality Score (показник якості)

Оцінка поділяється на 4 підрозділи:

1. Health Score: на оцінку стану здоров'я впливає кількість застарілих залежностей та наявність будь-яких уразливостей для пакета. Якщо всі залежності є актуальними і немає відомих уразливостей, пакет отримує повний бал.

```
function evaluateHealth(collected) {
    if (!collected.source) {
        return 0;
    }

    const dependencies = collected.metadata.dependencies || {};
    const dependenciesCount = Object.keys(dependencies).length;

    if (!dependenciesCount) {
        return 1;
    }

    // Calculate outdated count
    const outdatedCount = collected.source.outdatedDependencies ?
        Object.keys(collected.source.outdatedDependencies).length :
        (collected.source.outdatedDependencies === false ? dependenciesCount : 0);

    // Calculate vulnerabilities count
    const vulnerabilitiesCount = collected.source.vulnerabilities ?
        collected.source.vulnerabilities.length :
        (collected.source.vulnerabilities === false ? dependenciesCount : 0);

    // Calculate unlocked count - packages that have loose locking of versions, e.g.: '*' or >= 1.6.0
    // Note that if the package has npm-shrinkwrap.json, then it actually has its versions locked down
    const unlockedCount = collected.source.files.hasShrinkwrap ? 0 :
        Object.values(dependencies).reduce((count, value) => {
            const range = semver.validRange(value, true);

            return range && !semver.gtr('1000000.0.0', range, true) ? count + 1 : count;
        }, 0);

    const outdatedEvaluation = normalizeValue(outdatedCount, [
        { value: 0, norm: 1 },
        { value: Math.max(2, dependenciesCount / 4), norm: 0 },
    ]);
    const vulnerabilitiesEvaluation = normalizeValue(vulnerabilitiesCount, [
        { value: 0, norm: 1 },
        { value: Math.max(2, dependenciesCount / 4), norm: 0 },
    ]);

    const finalWeightConditioning = !unlockedCount ? 1 : 1 / (unlockedCount + 1);

    return (
        (outdatedEvaluation * 0.5) +
        (vulnerabilitiesEvaluation * 0.5)
    ) * finalWeightConditioning;
}
```

Рисунок 3.6 - Функція підрахунку Health Score

2. Обережність: оцінка ретельності оцінюється за наявністю в пакеті наступних елементів (з вагою для кожного елемента): ліцензія (0.33), readme (0.38), журнал змін (0.08), використання лінтеру (0.13), пакет має або розділ npm ignore, або явний файл (0,08). Але це не все. Потім в оцінку враховується чи вважається пакет «стабільним». Де «стабільний» означає версію, більшу або рівну 1.0.0. Якщо пакет не є «стабільним», оцінка скорочується вдвічі. Так що навіть якщо є всі елементи, зазначені вище, але версія пакета 0.x.x, то максимальний бал, який можно отримати в цьому

розділі дорівнює 0.5.

```
function evaluateCarefulness(collected) {
    const licenseEvaluation = Number(!!collected.metadata.license);
    const readmeEvaluation = normalizeValue(get(collected, 'source.files.readmeSize', 0), [
        { value: 0, norm: 0 },
        { value: 400, norm: 1 },
    ]);
    const lintersEvaluation = Number(!!get(collected, 'source.linters', null));
    const ignoreEvaluation = Number(get(collected, 'source.files.hasNpmIgnore') || collected.metadata.hasSelectiveFiles
    const changelogEvaluation = Number(get(collected, 'source.files.hasChangelog', false));

    const isDeprecated = !!collected.metadata.deprecated;
    const isStable = semver.gte(collected.metadata.version, '1.0.0', true); // `true` = loose semver
    const finalWeightConditioning = isDeprecated ? 0 : (!isStable ? 0.5 : 1);

    return (
        (licenseEvaluation * 0.33) +
        (readmeEvaluation * 0.38) +
        (lintersEvaluation * 0.13) +
        (ignoreEvaluation * 0.08) +
        (changelogEvaluation * 0.08)
    ) * finalWeightConditioning;
}

```

Рисунок 3.7 - Функція підрахунку обережності

3. Оцінка тестів: оцінка тестів оцінюється за трьома параметрами: наявність тестових файлів (0,6), покриття тестами (0,15) і статусів GitHub (0,25). Для тестових файлів недостатньо просто мати модульні тести. Існує необхідність мати скрипт «npm test» у файлі package.json, а тестові файли повинні бути більше 400 байт, щоб отримати повний 0.6 бал. Покриття обчислюється зі звітів проекту і існує необхідність підключення проекту до сервісів звітування коду про покриття тестами, такого як coveralls.io. Статуси одержуються з API GitHub.

```
function evaluateTests(collected) {
    if (!collected.source) {
        return 0;
    }

    const testsEvaluation = normalizeValue(collected.source.files.testsSize, [
        { value: 0, norm: 0 },
        { value: 400, norm: collected.metadata.hasTestScript ? 1 : 0.5 },
    ]);
    const coverageEvaluation = collected.source.coverage || 0;
    const statusEvaluation = ((collected.github && collected.github.statuses) || [])
    .reduce((sum, status, index, arr) => {
        switch (status.state) {
        case 'success':
            return sum + (1 / arr.length);
        case 'pending':
            return sum + (0.3 / arr.length);
        case 'error':
        case 'failure':
            return sum;
        default:
            log.warn(`Unknown github status state: ${status}`);

            return sum;
        }
    }, 0);

    return (testsEvaluation * 0.6) +
           (statusEvaluation * 0.25) +
           (coverageEvaluation * 0.15);
}
```

Рисунок 3.8 - Функція підрахунку покриття тестами

4. Branding: оцінка брендингу розраховується за двома параметрами: домашня сторінка проекту (0.4) та кількість значків, які проект має на своїй README (0.6). Домашня сторінка повинна знаходитися за межами домену github.com (наприклад, babeljs.io). Для значків достатньо мати 4 або більше, щоб отримати повний 0,6 бал.

```
function evaluateBranding(collected) {
    const parsedRepository = url.parse(get(collected.metadata, 'repository.url', ''));
    const parsedHomepage = url.parse(get(collected.metadata, 'links.homepage', get(collected, 'github.homepage', '')));
    const hasCustomHomepage = !!(parsedRepository.host && parsedHomepage.host &&
                                 parsedRepository.host !== parsedHomepage.host);
    const badgesCount = get(collected, 'source.badges.length', 0);

    const homepageEvaluation = Number(hasCustomHomepage);
    const badgesEvaluation = normalizeValue(badgesCount, [
        { value: 0, norm: 0 },
        { value: 4, norm: 1 },
    ]);

    return (homepageEvaluation * 0.4) +
           (badgesEvaluation * 0.6);
}
```

Рисунок 3.9 - Функція оцінки брендингу

3.1.3 Popularity (показник популярності)

Розділ популярності в основному залежить від кількості завантажень і зірок GitHub. Поділяється на 2 підрозділи:

Кількість скачувань.

```
function evaluateDownloadsCount(collected) {
    const downloads = collected.npm && collected.npm.downloads;

    if (!downloads) {
        return 0;
    }

    const index = downloads.findIndex((range) => moment.utc(range.to).diff(range.from, 'd') === 90);

    if (index === -1) {
        throw new Error('Could not find entry in downloads');
    }

    const count90 = downloads[index].count;
    const count30 = count90 / 3;

    return count30;
}
```

Рисунок 3.10 - Функція підрахунку кількості скачувань

Функція не считує оцінку, а просто повертає середнє значення скачувань за останні 90 днів в місяць.

2. Швидкість набуття популярності.

```
function evaluateDownloadsAcceleration(collected) {
    const downloads = collected.npm && collected.npm.downloads;

    if (!downloads) {
        return 0;
    }

    const range30 = find(downloads, (range) => moment.utc(range.to).diff(range.from, 'd') === 30);
    const range90 = find(downloads, (range) => moment.utc(range.to).diff(range.from, 'd') === 90);
    const range180 = find(downloads, (range) => moment.utc(range.to).diff(range.from, 'd') === 180);
    const range365 = find(downloads, (range) => moment.utc(range.to).diff(range.from, 'd') === 365);

    if (!range30 || !range90 || !range180 || !range365) {
        throw new Error('Could not find entry in downloads');
    }

    const mean30 = range30.count / 30;
    const mean90 = range90.count / 90;
    const mean180 = range180.count / 180;
    const mean365 = range365.count / 365;

    return ((mean30 - mean90) * 0.25) +
           ((mean90 - mean180) * 0.25) +
           ((mean180 - mean365) * 0.5);
}
```

Рисунок 3.11 - Функція оцінки швидкості набуття популярності пакета

Функція рахує різницю середньої кількості скачувань в день за періоди:

1. 30 днів - 90 днів
2. 90 днів - 180 днів
3. 180 днів - 365днів

При чому коефіцієнт за період 180 днів - 365 днів найбільший і становить 0.5. Це дозволяє зробити аналіз вірогідно більш репрезентативним.

3.2 Власні критерії

3.2.1 Глубока перевірка

Зазвичай в проекті уснує багато залежностєй, які записуються в файл package.json. Тут указуються залежності та їх версіі. Наприклад залежності практичної частини диплома:

```
"dependencies": {
   "@angular/animations": "^6.1.10",
   "@angular/cdk": "^7.3.7",
   "@angular/common": "^6.1.0",
   "@angular/compiler": "^6.1.0",
   "@angular/core": "^6.1.0",
   "@angular/forms": "^6.1.0",
   "@angular/http": "^6.1.0",
   "@angular/material": "^7.3.7",
   "@angular/platform-browser": "^6.1.0",
   "@angular/platform-browser-dynamic": "^6.1.0",
   "@angular/router": "^6.1.0",
   "core-js": "^2.5.4",
   "rxjs": "~6.2.0",
   "zone.js": "~0.8.26"
 }
```

У кожної залежності є свої. Наприклад залежності для пакету @angular/material:

```
 "dependencies": {
   "@angular/animations": "^8.0.0-rc.5",
   "@angular/common": "^8.0.0-rc.5",
   "@angular/compiler": "^8.0.0-rc.5",
   "@angular/core": "^8.0.0-rc.5",
```

```
  "@angular/elements": "^8.0.0-rc.5",
  "@angular/forms": "^8.0.0-rc.5",
  "@angular/platform-browser": "^8.0.0-rc.5",
  "@webcomponents/custom-elements": "^1.1.0",
  "core-js": "^2.6.1",
  "material-components-web": "^1.1.0",
  "rxjs": "^6.4.0",
  "systemjs": "0.19.43",
  "tsickle": "^0.35.0",
  "tslib": "^1.9.3",
  "zone.js": "~0.9.1"
 }
```

І так рекурсивно кожна залежність тягне за собою іншу, поки всі пакети не будуть встановлені в папці node_modules, наприклад в цій практичній частині разом з залежностями для розробки вийшло 816.

Суть глубокої перевірки лежить в тому, щоб перевірити кожну залежність по критеріям, які вже є. Цього не хватає аналізаторам пакетів. Існують і свої мінуси цього підходу - це досить велика нагрузка на сервери. Робота зараз не дуже оптимізована, і якщо такий сервіс стане популярним, прийдеться оптимізовувати алгоритми аналізів.

3.2.2. Перевірка профілей на GitHub які вносили зніми в проект

1. Як вже було показано достатньо однієї людини, однього pull-request-а, однього commit-а, щоб внести вірусний код в відкритий проект.
2. З пункті 2.5 про сквотування проекту, де зловмисник з ніком hacktask робив підміну проекту, і його, з рештою, так і не знайшли, тому що була тільки його піздставна пошта. Зрозуміло, що зловмисник з великою долею імовірності не буде розкривати свій справжній профіль на GitHub. Тому в ситуації коли його діяльність буде розкрита, його буде легко знайти.

Для його пошуку таких зловмисників, пропонується аналізувати кожного хто вносив хоча б один коміт, дивитися на дату реєстрації користувача в системі GitHub та дивитися скільки в нього зірок на профілі від інших користувачів.

Висновки до розділу 3

Часто існують десятки або навіть сотні пакетів з подібними назвами та / або подібними цілями. Щоб допомогти визначитися з найкращими для вибору, кожен пакет класифікується за трьома критеріями за допомогою аналізатора NPMs.

1. Maintenance - ранжує пакети відповідно до уваги, яку розробники приділяють їм.
2. Quality - якість включає такі параметри, як наявність файлу README, стабільність, тести, актуальні залежності, домашній веб-сайт
3. Popularity - показує, скільки разів пакет завантажувався.

Критерії NPM показують далеко не повну картину перевіреності пакету, тому що вони не враховують перевіреність його залежностей. В цьому розділі пропонується рекурсивно перевіряти кожен пакет який є в залежностях. Крім того, пропонується аналізувати кожного хто вносив хочаб однин коміт, дивитися на дату реєстрації користувача в системі GitHub та дивитися скільки в нього зірок на профілі від інших користувачів.

4 ПРАКТИЧНА ЧАСТИНА

В роботі було розроблено програмне забеспечення для удосконаленого пошуку пакетів з сумнівою якістю та безпекою. Було застотовано удосконалення які було описано в попередньому розділі. Сайт було розроблено з використанням фреймворку Angular версіі 7.3.7.

4.1 Розробка проекту

В програмі використовуються стороні сервіси такі як NPM та GitHub. Зв’язок з ними відбувається через відкрите API:

```
export enum API_SERVERS {
  NPM = 'https://api.npms.io/v2/package',
  GITHUB = 'https://api.github.com'
}
```

З NPM беремо дані про пакет, які приходять в такому вигляді:

- analyzedAt: ""
  collected:
  - metadata:
    - dependencies: {}
  - github:
    - contributors: {}

- evaluation:
  - maintenance: {releasesFrequency: 1, commitsFrequency: 1, openIssues: 1, issuesDistribution: 0}
    popularity: {communityInterest: 8689, downloadsCount: 392129.33333333333, downloadsAcceleration: 2699.5519216133944, dependentsCount: 67}

quality: {carefulness: 0.45999999999999996, tests: 0.85, health: 1, bra
nding: 0.7}

- score:
  - detail:
    - maintenance: 0.6666666666666666
      popularity: 0.44258340464048257
      quality: 0.8839059396298684

  - final: 0.6534093068464627

Тут нам потрібні поля поле dependencies, evaluations та contributors.

Також з GitHub API беремо дані про користувачів про яких ми дізнаємося з поля contributors з попереднього пункту. Наприклад:

- avatar_url: "https://avatars2.githubusercontent.com/u/24254801?v=4"
  bio: null
  blog: ""
  company: null
  created_at: "2016-11-29T18:04:13Z"
  email: null
  events_url: "https://api.github.com/users/Ketler13/events{/privacy}"
  followers: 0
  followers_url: "https://api.github.com/users/Ketler13/followers"
  following: 0
  following_url: "https://api.github.com/users/Ketler13/following{/other_user}"
  gists_url: "https://api.github.com/users/Ketler13/gists{/gist_id}"
  gravatar_id: ""
  hireable: null

html_url: "https://github.com/Ketler13"
id: 24254801
location: "Ukraine"
login: "Ketler13"
name: "Oleksandr Martyshchenko"
node_id: "MDQ6VXNlcjI0MjU0ODAx"
organizations_url: "https://api.github.com/users/Ketler13/orgs"
public_gists: 1
public_repos: 24
received_events_url: "https://api.github.com/users/Ketler13/received_events"
repos_url: "https://api.github.com/users/Ketler13/repos"
site_admin: false
starred_url: "https://api.github.com/users/Ketler13/starred{/owner}{/repo}"
subscriptions_url: "https://api.github.com/users/Ketler13/subscriptions"
type: "User"
updated_at: "2019-05-31T05:31:14Z"
url: «https://api.github.com/users/Ketler13"

Повний код проету можна побачити на моєму профілі в GitHub: https://github.com/denis-ratenok/npm-deep-analyzer

4.2 Результат

Перший екран - поле для вводу пакету, який нас цікавить. Нижче представлена аналітика цього пакету та його залежностей.

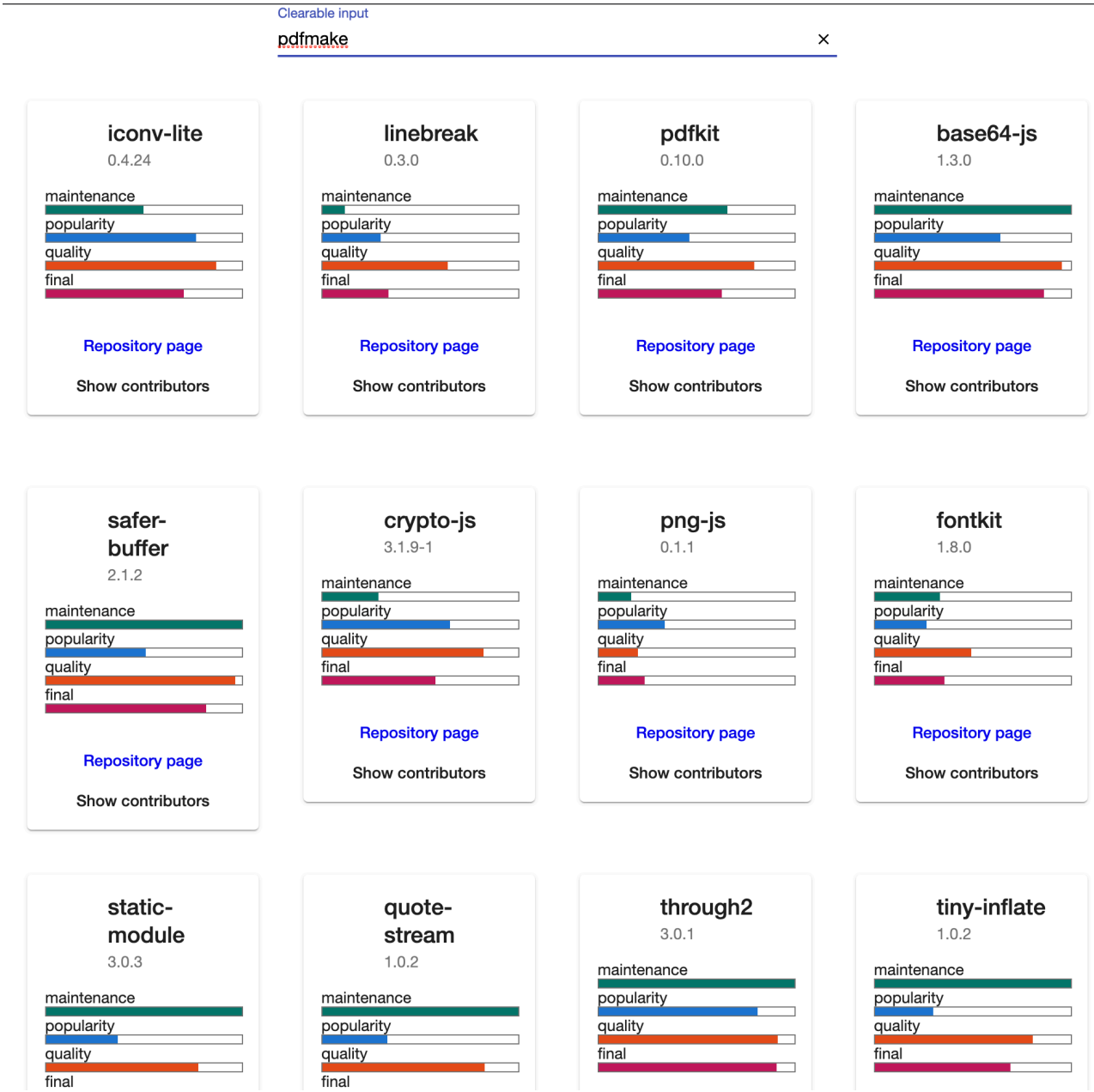

Рисунок 4.1 - Перший экран проекту

Другий екран - вікно з контрибьюторами проекту. Тут профілі відфільтровані по даті регістрації в GitHub. Також можна бачити скільки в цього профіля зірок. Нажавши на когось ми попадаємо на його сторінку в GitHub, де можна детальніше продивитись зацікавлений нас профіль.

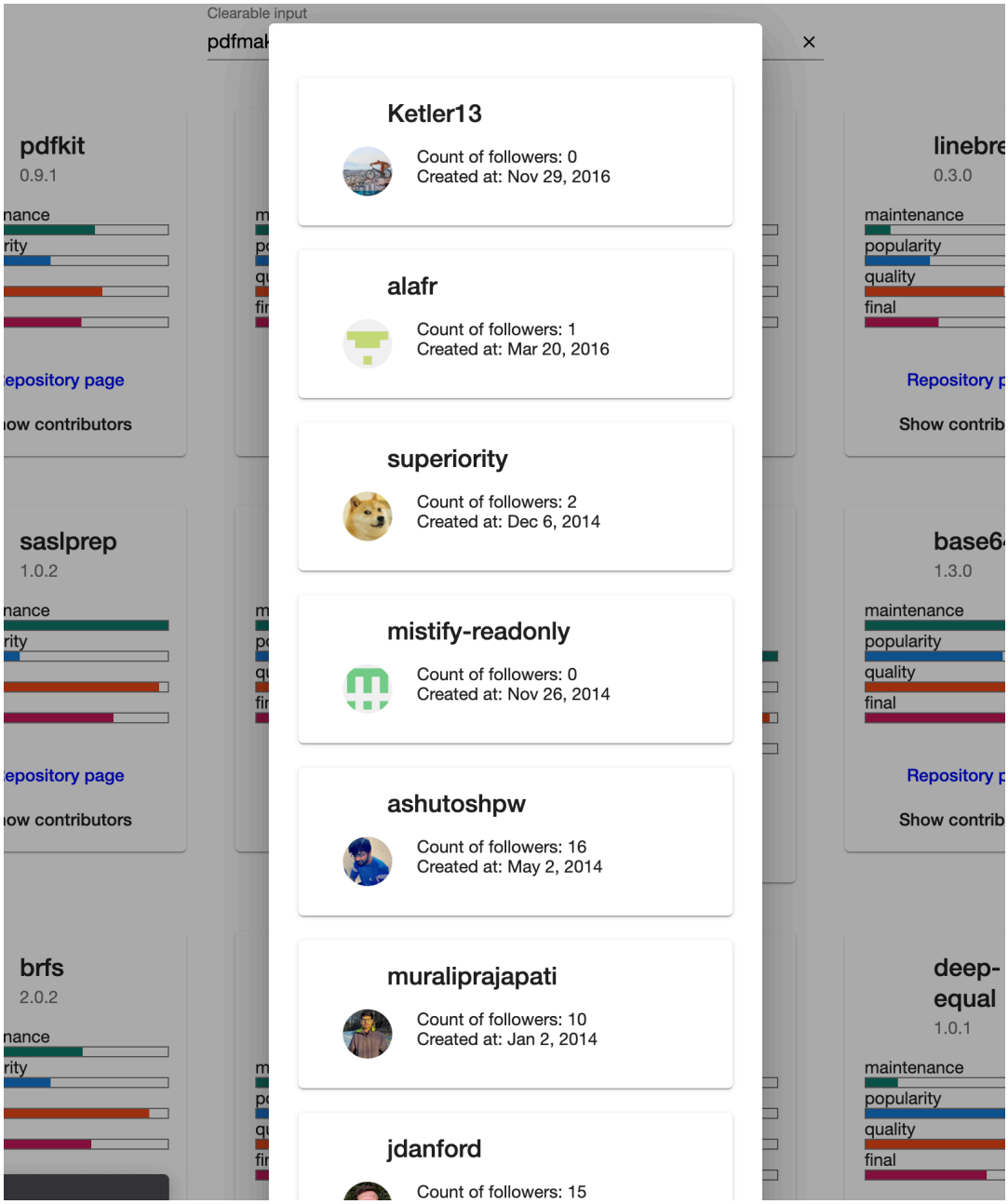

Рисунок 4.2 - Другий екран проекту

Висновки до розділу 3

Опираючись на власні критерії з минулого розділу було розроблено сервіс/платформу для пошуку небезпечних пакетів.

ВИСНОВКИ

Результатом даної роботи є аналіз існуючих критеріїв оцінки пакетів в найбільшому у світі реєстрі програмного забезпечення - NPM. За відкритим кодом NPM було досліджено яким чином зчитується той чи інший показник. Було виявлено 3 основних стандартних критерії:

1. Maintenance - ранжує пакети відповідно до уваги, яку розробникиь приділяють їм.
2. Quality - якість включає такі параметри, як наявність файлу README, стабільність, тести, актуальні залежності, домашній веб-сайт
3. Popularity - показує, скільки разів пакет завантажувався.

Критерії NPM показують далеко не повну картину перевіреності пакету, тому що вони не враховують перевіреність його залежностей. В цій роботі пропонується рекурсивно перевіряти кожен пакет який є в залежностях. Крім того, пропонується аналізувати кожного хто вносив хоча б один коміт, дивитися на дату реєстрації коритстувача в системі GitHub та дивитися скільки в нього зірок на профілі від інших користувачів.

Було розроблено сервіс для пошуку найбільш безпечних пакетів, враховуючи власні критерії для аналізу. Було продемонстровано результати роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Gilbertson D. I’m harvesting credit card numbers and passwords from your site. Here’s how. [Електронний ресурс] / David Gilbertson. – 2018. – Режим доступу до ресурсу: https://hackernoon.com/im-harvesting-credit-card-numbers-and- passwords-from-your-site-here-s-how-9a8cb347c5b5.

2. npm-package.json [Електронний ресурс] – Режим доступу до ресурсу: https://docs.npmjs.com/files/package.json.

3. Wright J. HUNTING MALICIOUS NPM PACKAGES [Електронний ресурс] / Jordan Wright. – 2017. – Режим доступу до ресурсу: https://duo.-com/decipher/ hunting-malicious-npm-packages.

4. `crossenv` malware on the npm registry [Електронний ресурс] – Режим доступу до ресурсу: https://blog.npmjs.org/post/163723642530/crossenv-malware-on-the- npm-registry.

5. Scan your projects for crossenv and other malicious npm packages [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https:// www.twilio.com/blog/ 2017/08/find-projects-infected-by-malicious-npm-packages.html.

6. Gilbertson D. How to stop me harvesting credit card numbers and passwords from your site. [Електронний ресурс] / David Gilbertson – Режим доступу до ресурсу: https://hackernoon.com/part-2-how-to-stop-me-harvesting-credit-card-numbers-and-passwords-from-your-site-844f739659b9.

7. McDonald S. NPM, Azer Koçulu, Kik and the law. [Електронний ресурс] / Sean McDonald. – 2016. – Режим доступу до ресурсу: https://medium.com/@McDapper/npm-azer-ko%C3%A7ulu-kik-and-the-law-f22742f099f6.

8. REPORT 2018 Open Source Security and Risk Analysis [Електронний ресурс] // BlackDuck. – 2018. – Режим доступу до ресурсу: https://www.synopsys.-com/ content/dam/synopsys/sig-assets/reports/2018-ossra.pdf.

9. NPM. Відкритий код [Електронний ресурс] / NPM // NPM. – 2018. – Режим доступу до ресурсу: https://github.com/npms-io/npms-analyzer/blob/master/lib/analyze/evaluate/maintenance.js.

10. Документація npms-api [Електронний ресурс] // npms. – 2017. – Режим доступу до ресурсу: https://api-docs.npms.io/.