



Harvard University

SAFETY KNOWLEDGE GRAPH: GRADVEK 2.0

by

David Aviles, Alyssa Bédard, Dixon Bross, Ali PiyarAli, Kevin Patel,
and Nathan Sheely

CSCI E-599 Software Engineering Capstone, Section 2
Harvard Extension School

Professor Peter Henstock, teaching assistant Simili
Abhilash

In collaboration with: Seda Arat, Athena
Mustakis, Trung Nguyen and Dennis
Pelletier

Wednesday, May 10th 2023

TABLE OF CONTENTS

Table of Contents	1
Journal Paper	3
Abstract	3
Introduction	3
Contribution of GradVek to the field	6
Methods	7
Open Target datasets	7
Functionality	10
User Experience	14
Discussion	21
Conclusion	22
References Cited	22
Design of the system (for software engineers)	24
Introduction	24
System Overview	24
Frontend Design	24
Backend Design	25
UML Diagrams	27
Backend UML Diagrams	28
Datasets Package Diagram	28
Gradvekbbackend Package Diagram	29
Search Package Diagram	29
Language and Platform	30
Testing Results (for software engineers)	31
Development Process & Lessons Learned Reflection (for faculty panel)	32
Meeting the Requirements	32
Estimates	33
Risks	33
Team Dynamic	33
Appendix	36
Website URL	36
System installation manual	36
	1

Developer installation manual	37
Requirements and Estimates	38
Team Contract	46
Team Expectations	46
Conflict Resolution	46
Communication	46
Team Meetings and Availability	46

JOURNAL PAPER

ABSTRACT

GradVek is a user-friendly knowledge graph that bridges the gap between complex data and user-friendly interfaces, providing valuable insights for drug research and development. It enhances data from Open Targets by creating and presenting relationships between drugs, genes, targets, and adverse effects. Additionally, the tool is designed to be easily accessible and can be hosted locally by users. It allows users to search for targets based on similarity and explore the knowledge graph without coding. The tool also enables users to perform advanced analyses and augment the knowledge graph with proprietary data via a simple CSV upload interface. GradVek's target similarity search function aids in drug discovery efforts by identifying potential off-target effects and displaying adverse events.

INTRODUCTION

Pharmaceutical research and development is a complex process that encompasses a wide range of activities, from discovery to safety monitoring. A new drug typically takes over ten years to develop and costs more than \$2.6 billion [1]. Not only is the process time-consuming and expensive, but the effectiveness of a new drug is only fully known after several years [2]. Adverse events and drug safety are commonly cited as the second most important factors after drug efficacy in determining drug success [3]. Predicting adverse events could aid drug research and development trials in identifying potential issues and providing safer therapies. Drug research and development is not only focused on the efficacy of a treatment but also its risks – a drug's side effects, also known as adverse events, must be weighed against the drug's benefits [4]. While there are many safety tools available for drug research and development, they often focus on facilitating drug efficacy rather than predicting adverse events. In this paper, we present a novel approach that leverages a knowledge graph encoding of relevant relationships to predict adverse events and provide a safer and faster drug discovery pipeline. [5].

A knowledge graph is a flexible method used to map relationships between diverse types, where both nodes and edges can represent different concepts [6]. In drug development, one type of node

might represent the target, which refers to proteins, enzymes, receptors, ion channels, or other cellular components in the human body that a drug is intended to affect. Another type of node could be for developed drugs, and the relationship between them can be defined by the clinical evidence showing their intended affinity and specificity [6]. However, it should be noted that not all drugs directly bind to a target; some drugs modulate the activity of the body's biochemical pathways instead of binding to a specific molecular target. In this case, the relationship between the drug node and the pathway node represents another type of relationship. A key advantage of a knowledge graph is its ability to represent diverse types of entities and relationships in a single graph, including those that are not typically captured in homogeneous graphs. This capability allows for a more comprehensive and integrated understanding of the data, which can facilitate more efficient and accurate analysis in various domains, including drug discovery and development. [6]. Knowledge graphs are therefore a natural solution to represent the complexity inherent in the interactions between compounds and human biology, since more powerful insights are enabled when using greater information. One way that a knowledge graph encoding could facilitate research would be to use known effects in trials on mice and activation strength on particular proteins to find drugs with similar relationships that could be adapted to treat a new target. A knowledge graph can greatly expedite research across the multiple subtasks of drug discovery, such as: finding new targets for existing successful drugs, disease and drug target identification, drug interferences, and drug adverse effects [6].

Though knowledge graph applications in drug development are a relatively new area of research with early solutions implemented in 2017, there are now multiple notable approaches published and publicly available. Hetionet, Drug Repurposing Knowledge Graph, PharmKG, and Clinical Knowledge Graph are well-regarded solutions that represent distinct academic implementations with strong breadth across the relevant types and relationships for drug development. However, they share a requirement of technical proficiency in programming to access these graphs, which has led to commercial applications being developed to build the graph for users, making accessing these insights easier.

Hetionet [7] is a knowledge graph primarily focused on drug repurposing and was one of the first knowledge graphs applied to drug development, with broad coverage across the relevant concepts such as genes, proteins, drugs, diseases, anatomy, pathways, side effects and symptoms. Hetionet

was dramatically expanded by the Drug Repurposing Knowledge Graph [8] in 2020, which focused on finding drugs that could treat COVID-19 by adding additional data sources to Hetionet. In addition, DPKG dramatically increased the types of relationships between entities. Where Hetionet had 1-3 types of drug to drug or gene to gene type relationships (such as disease downregulates, upregulates, or is associated with a gene), DPKG increased that to over 30 types of relationships between entities. These additional relationships allow researchers to infer new information about entities that were previously much more difficult to identify using Hetionet.

PharmKG is another knowledge graph in the pharmaceutical area, launched with the focus of being a single benchmark source for further studies' performance comparisons by consolidating adverse event data, drug efficacy data and other sources [9]. PharmKG was launched in 2020 with 8000 nodes and 500,000 relationships mapped, with the broadest coverage across entity and relationship types at that time. To demonstrate their thorough coverage and robust data selection, they performed multiple modeling tests built on their knowledge graph and used the same models trained on Hetionet as a direct comparison. The developers were able to show the improvement using PharmKG brought to common research tasks predicting relationships for drug repurposing and target identification [9].

Clinical Knowledge Graph is the largest and best documented knowledge graph focused on drug development [10]. It incorporates 25 data sources, with a particular emphasis on genomics and proteomics, and more than 16 million nodes and 220 million relationships encoded [10]. Its authors provide thorough documentation on how to set up and run their knowledge graph, with detailed steps from installation through a few sample use cases including how to upload additional data [20]. One drawback is that the data sources are not all publicly available, so the accessible portion of their work is much smaller than the figures cited. Additionally, the workflow is built around generating statistical reports tied to experimental designs requiring three separate input files and twenty-four fields. There is an interface discussed in the documentation using Jupyter notebooks to interact with the knowledge graph, but those require familiarity with either Python or another compatible programming language [20].

Shared drawbacks across these implementations is the lack of simple user interfaces and opaque code bases that are difficult to extend or add to. Users with proprietary data they'd like to add to

these knowledge graphs would have to augment the current implementations extensively to incorporate their own data sources. Others have noted the absence of maintenance and/or clear maintenance schedules and less thorough documentation [11]. While more nodes and relationships aid analysis and machine learning models, default graph visualization approaches become unreadable due to the number of connections and exploring the graph is impeded. A more basic criticism is that while some solutions have the code to access the graphs publicly available, a significant technical expertise is required to interact with the graph and actually use it. To overcome this technical hurdle, there are a few commercial offerings of more user-friendly applications leveraging knowledge graphs for drug discovery.

QIAGEN offers a broad suite of technology to facilitate molecular diagnostics, and one of their digital offerings is QIAGEN Ingenuity Pathway Analysis, a tool built on a knowledge graph collection of “thousands of sources,” covering targets, drugs, functions, pathways, diseases, chemicals, locations, variants, and more [13]. Similar to the Clinical Knowledge Graph implementation, they tout the connection to ‘omics data as well as a number of tools to interrogate the graph, including causal network analysis and tools to simulate hypothetical relationships. However, this is a tool available to paid subscribers only and it is not clear that proprietary data can be added easily.

OntoText is a more general graph database technology provider, with additional semantic text analysis tools. They present a case study of their more general technology applied to pharmaceutical data and the creation of a biomedical knowledge graph that can be easily interacted with via their standard graphDB tools. While their tool would likely be more easily adapted to incorporating additional data sources, their solution is not tailored for pharmacology and would require tailoring and specification for the data sources integrated. Additionally, while also a paid tool it lacks the ‘omics data (Genome-Phenome) connections, causal-analysis tools, and other benefits offered by Qiagen [12,14].

Contribution of GradVek to the field

GradVek is a tool designed to assist scientists in drug discovery by easily presenting the relationships between drugs, genes, targets, and adverse effects. It achieves this by building a knowledge graph encoding these relationships, then organizing and presenting this information in

an easy and intuitive user interface for non-technical users. GradVek allows users to search for targets based on their similarity to other targets across 8 different relationship types and explore the knowledge graph without needing to host or code anything themselves. For experienced researchers with their own data on molecular relationships, GradVek enables the easy augmentation of its knowledge graph via a simple csv upload interface.

GradVek presents a target to target similarity exploration tool where pairwise similarities between all targets (proteins) are computed across eight descriptors. This similarity scoring balances the benefits of reducing the search space to relevant targets while maintaining clear logic a user can interact with and adjust. By selecting a target identified as similar to their original search, a user can immediately see a visualization of that target's relationships and attributes in the knowledge graph, exploring the specific drivers of similarity. Combining this similarity scoring with the rest of the knowledge graph creates a novel tool for researchers to efficiently and organically identify adverse events or drug efficacy across similar targets.

GradVek has been designed to be particularly easy to use, with a public site accessible and local docker versions optimized for launch with just a few terminal commands. Local extension is encouraged with a well-commented python code base and source code publicly available on github. For users with their own data, GradVek allows seamless expansion of the knowledge graph with user supplied data submitted via a csv upload interface. If that data is sensitive or proprietary, locally hosting the application in a docker container is an easy way to get the full functionality of the application without sharing data beyond a local network.

METHODS

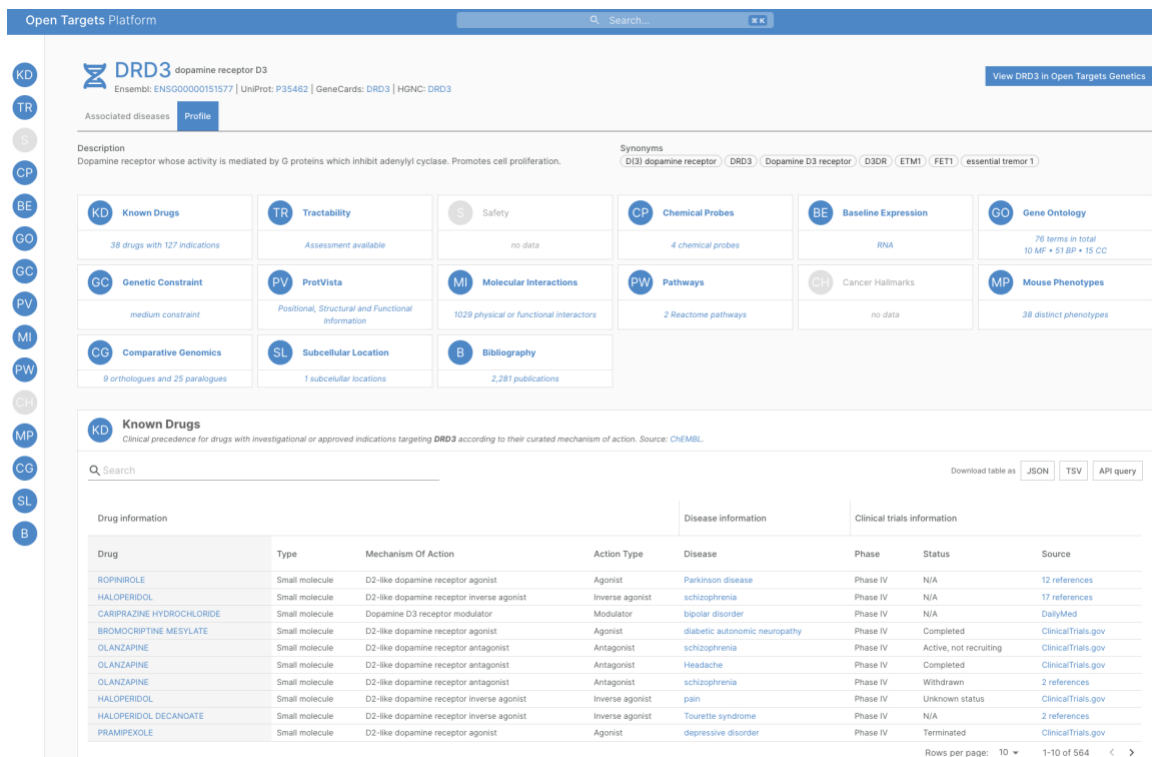
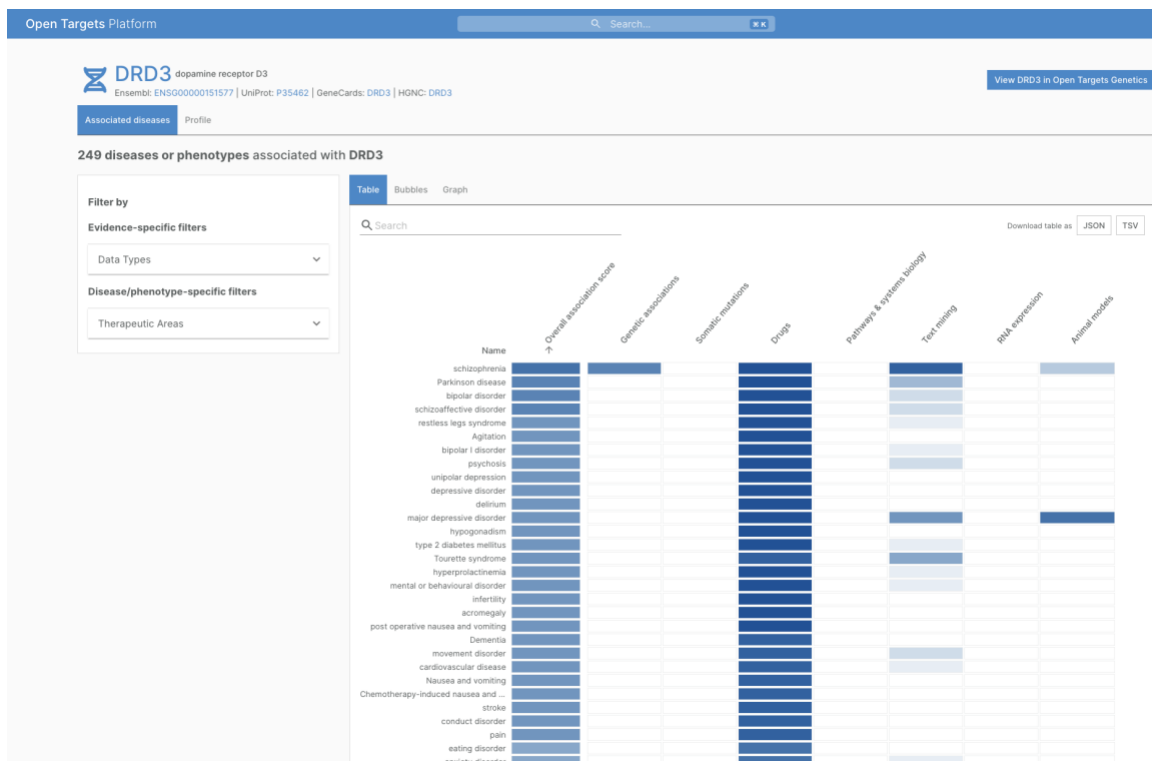
Open Target datasets

GradVek ingests and presents data that has been aggregated by the Open Targets platform. Open Targets is a well-regarded public-private application operated by the Biogen, EMBL European Bioinformatics Institute, GlaxoSmithKline and the Wellcome Trust Sanger Institute. It has separate pages for in depth research on particular compounds or drugs, and also makes raw data available for users. As one of the largest and most reliable dataset creators in the pharmaceutical

space it standardizes the format and structure of data, provides scores and evidence on how the data was processed, and updates data quarterly.

The Open Targets platform aggregates rich data on genes, drugs, and diseases. Open Targets combines information from 30+ standard sources including Reactome, UniProt, ChEMBL, and GWAS and is an established source for further research [11, 15, 16]. Liangying et al., leveraged its genetic association, extended literature support, known drugs, affected pathway, mutations, RNA expression, and animal models to uncover the causal effects of genes [17]. Kai et al., show the advantage of using Open Targets to “rediscover” drug targets that may have been missed previously, proving the strength of its data in bioinformatic analysis [18]. However, in order to retrieve bulk data from Open Targets and establish connections, it is necessary to have proficiency in GraphQL, Google SQL, or parsing Parquet and JSON formats.

Less technical interaction with the data aggregated by Open Targets is possible through its Associated Diseases or Profile pages, where data is presented in a more user-friendly way with tables for the target of interest. This is presented in clear and well-designed tables and graphics, but is limited to a single target at a time that has been pre-specified, and is much more of a data viewing tool than workflow for more complex analysis.



Figures 1,2: Views of Open Targets data presentation – static target attributes shown, not encoded into a knowledge graph.

While Open Targets has all the data required to populate a knowledge graph, it is not structured as a knowledge graph itself, and formally embedding these relationships can be a source of value for researchers interested in understanding the connections between entities, the relationship based similarity between targets, or more advanced graph-based analysis. GradVek addresses this need by combining the existing data from the Open Targets API with the benefits of a knowledge graph, as well as providing a user-friendly interface for performing more complex analyses than is currently supported by Open Targets.

One final benefit of Open Targets is that it updates data on a quarterly basis, which allows it to stay current with the field [11]. As a downstream data consumer, GradVek automatically checks for updated data on Open Targets FTP and pulls updated data when applicable, and there is an additional feature for users to initiate this data check at their convenience. This consistent update frequency enhances the integrity of the findings and enables users to maintain up-to-date insights from our product.

Functionality

GradVek uses Neo4J to create a knowledge graph from the data pulled from Open Targets, encoding nine entity types (Drugs, Adverse Events, Baseline Expression, Diseases, Mouse Phenotypes, Genes, Targets, and two kinds of Pathways) and eleven relationship types (participates in, associated with, uses pathway, genetic expression, strength of gene association, strength of protein association, four methods of identifying known interactions, and drug to target activation method) into 127,768 nodes, and 68,278,478 edges. After building the knowledge graph, GradVek presents an intuitive and easy search functionality for users to peruse the relationships in the graph starting from the target and particular relationships they are interested in.

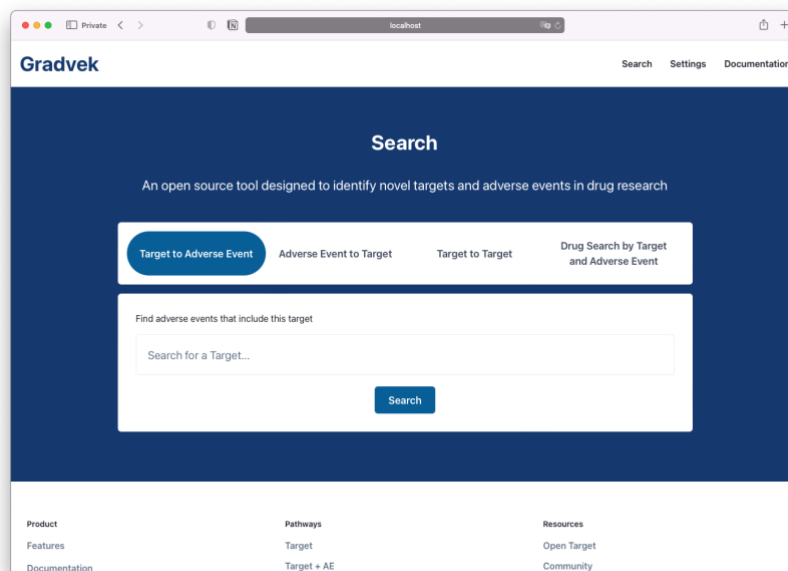


Figure 3: Streamlined search interface to begin exploration of GradVek knowledge graph

To make the knowledge graph easier to explore, GradVek incorporates a Target Similarity search function, which significantly contributes to drug discovery efforts. This feature enables users to filter the graph and focus on drugs that exhibit a specified similarity score. The similarity score is determined by shared elements or shared ranking of elements across eight different dimensions. These dimensions encompass various aspects such as known interactions sourced from Signor, Reactome, and Intact, documented mouse phenotypes, tissues affected by gene and protein expression, GWAS traits, and the Pathways source similarity score.

The similarity scores are calculated using the Neo4j graph similarity score package, which is an efficient way to apply Jaccard similarity across all pairs of targets. Jaccard similarity is the ratio of the intersection of two sets over their union. So in our example, if target 1 has mouse phenotypes A and B, and target 2 has mouse phenotypes A, C, D, then the similarity between the two targets on the mouse phenotype dimension would be equal to 0.25, as they share the single phenotype A over the total phenotypes A, B, C, D exhibited by the two targets. This naturally yields a value between 0 and 1. For aggregate similarity metrics across multiple dimensions, the simple average of all dimensions where both targets are present is used.

By incorporating such a diverse survey of potential relationships, GradVek empowers researchers to follow their hypotheses and investigate the most promising indicators of a successful new drug. For instance, researchers can leverage their domain-specific knowledge to evaluate the efficacy of mouse phenotypes versus protein expression in determining the viability of a drug within a particular context.

Moreover, this target similarity analysis serves several essential purposes in drug discovery. Without a view of similar targets, drug researchers would be forced to review a much greater number of potential targets when considering which target to begin drug development on. By narrowing the search from all potential new targets for drug development to those most similar to existing or proven targets, this process can be rapidly accelerated by reviewing a much smaller subset of all potential targets. This helps identify potential off-target effects, facilitates the repurposing and optimization of existing drugs, enables polypharmacology approaches for complex diseases, aids in predicting drug safety and side effects, and assists in the optimization of lead compounds [19].

In summary, GradVek's Target Similarity search function not only enhances the exploration of the knowledge graph but also contributes significantly to the broader goals of drug discovery by leveraging diverse dimensions of similarity and enabling researchers to make informed decisions based on their own domain expertise.

INPUT	TARGET	SIMILARITY SCORE	DESCRIPTOR
	ENSG00000224931	0.75	hGene
	KIRREL3-AS3	0.75	hGene
	LINC02134	0.7368421052631579	hGene
	ENSG00000233069	0.7368421052631579	hGene
	ENSG00000279674	0.7142857142857143	hGene
	CLK3P2	0.6956521739130435	hGene
	SSBP3P2	0.6818181818181818	hGene
	LINC01476	0.6666666666666666	hGene
	ENSG00000279325	0.6666666666666666	hGene
	DRG1P2	0.6666666666666666	hGene

Figure 4: Similarity results of closest targets for DRD3 based on single selected hGene source

While focused on an easy user experience, GradVek is nonetheless designed with a robust technological foundation, leveraging tools such as Neo4j, Django, and Python to facilitate the seamless setup of the underlying graph structure. By harnessing the capabilities of Neo4j, GradVek enables the representation of complex relationships and interactions among nodes in the graph. With Django and Python, the graph can be easily constructed and customized to incorporate the essential components for Graph Neural Networks or other more sophisticated machine learning approaches. This includes either defining or using existing node features to capture relevant characteristics, relationships between nodes to establish connectivity, and utilizing adjacency matrices to represent the relationships. The knowledge graph can be directly interacted with using cypher queries on the backend interface accessed separately from the front end, for users looking for more flexible and in depth analyses.

GradVek additionally automatically scans for updated data files on Open Targets and can be hosted locally by users wishing to augment Open Targets data with their own proprietary data sets via an easy CSV upload process. When users populate the node and edge types and the few required fields (identifiers and optional edge weight), any expansion of an existing entity type can be added to the knowledge graph. This enables users to combine their own organization's proprietary

knowledge with the publicly available data and into a knowledge graph structure, allowing them to capture the best of both public and privately available data.

User Experience

A user looking to explore the relationships between drugs, genes, targets, and adverse effects is likely to want a means of interacting quickly with the data and to understand how different entities relate to one another. However, accessing and analyzing the data from Open Targets Platform is either limited by Open Target's static tabular data structure, or requires technical expertise and coding experience to create a custom knowledge graph to embed the relationships between entities. To address these challenges, GradVek takes the data from Open Targets Platform and generates a knowledge graph specializing in Drug to Adverse Event relationships and Target attributes. This creates a rich map of the connections and interactions, but the knowledge graph is too large to easily interact with visually, limiting its ability to provide meaningful insights.

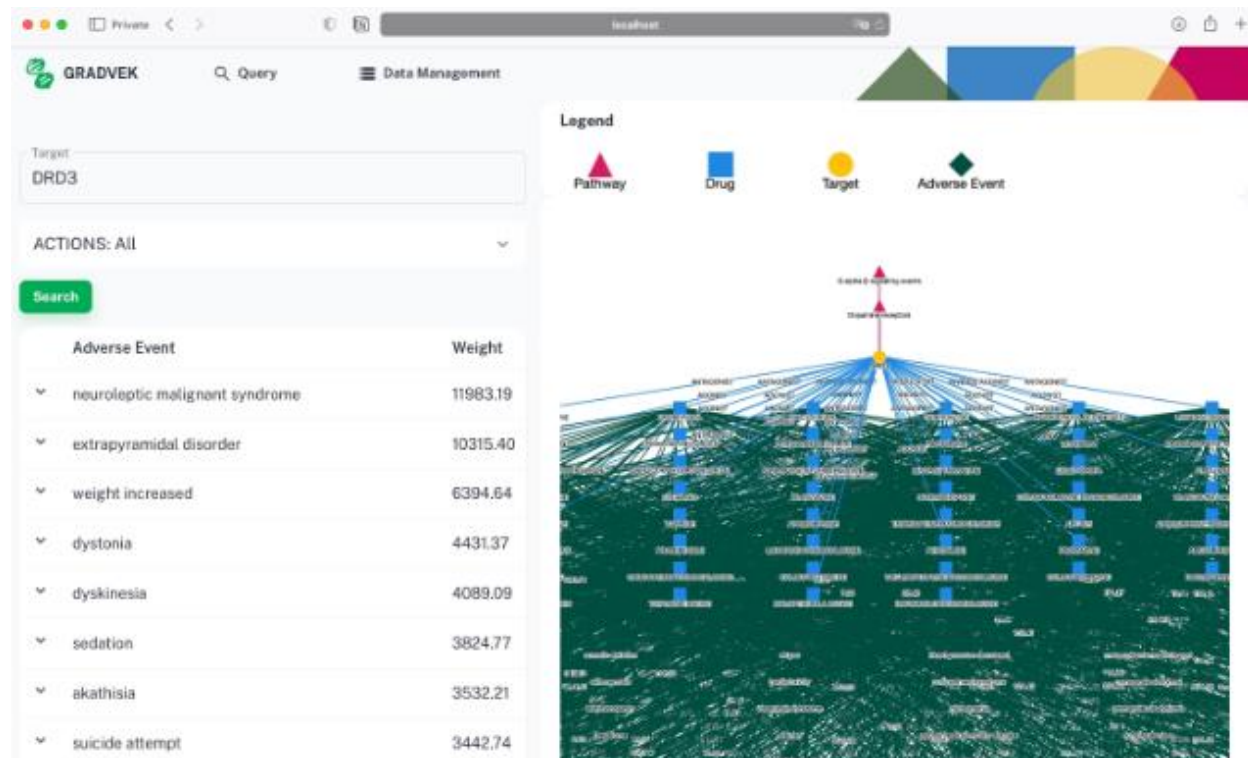


Figure 5: Early visualizations of the entire knowledge graph were too dense to see clearly

Rather than immediately dive into a dense graph visualization, GradVek presents a search interface to allow users to focus on a particular area of interest.

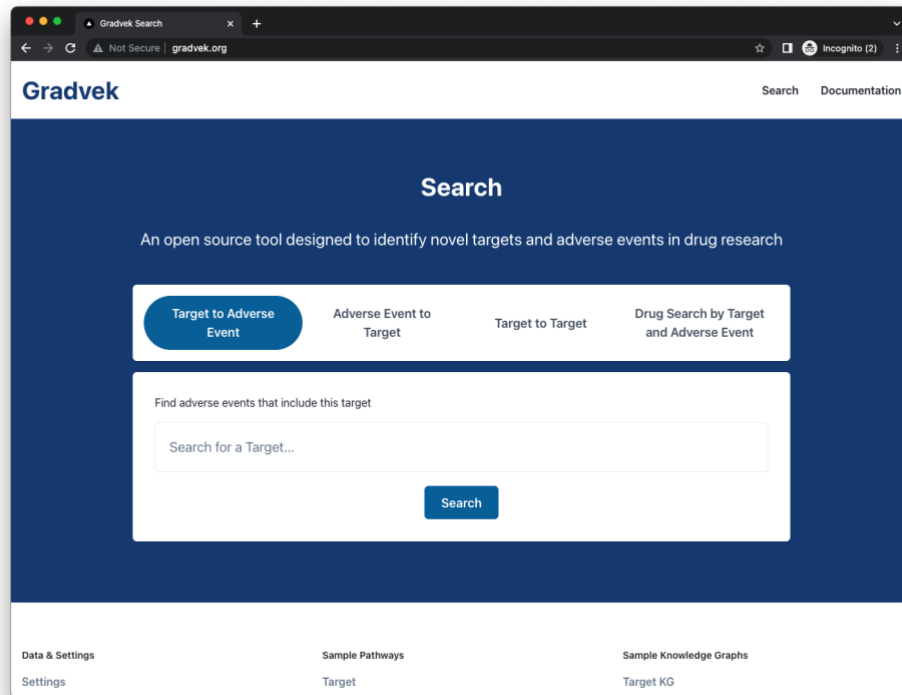


Figure 6: Search function landing page allows users to filter the graph before interacting with it.

Whether users search from a target to adverse events, from an adverse event to targets, or search for drugs by target and adverse event, they are able to view the top search results. This takes users to the search results page where the relevant entity types are displayed.

Gradvek Search Documentation

Adverse Events for DRD3

ADVERSE EVENT	ID	WEIGHTS	DATASET
NS Neuroleptic Malignant Syndrome	10029282	13355.39	23.02.1AdverseEvent
ED Extrapyramidal Disorder	10015832	10387.93	23.02.1AdverseEvent
WI Weight Increased	10047899	5854.07	23.02.1AdverseEvent
D Dyskinesia	10013916	5532.92	23.02.1AdverseEvent
S Sedation	10039897	5405.73	23.02.1AdverseEvent
D Dystonia	10013983	5244.46	23.02.1AdverseEvent
A Akathisia	10001540	5136.25	23.02.1AdverseEvent
SA Suicide Attempt	10042464	5040.48	23.02.1AdverseEvent
S Somnolence	10041349	4025.86	23.02.1AdverseEvent

Figure 7: Search results page shows related entities for the input type

The fourth path, the target to target search, allows users to find similar targets to their input target, and to specify across which dimensions that similarity should be calculated.

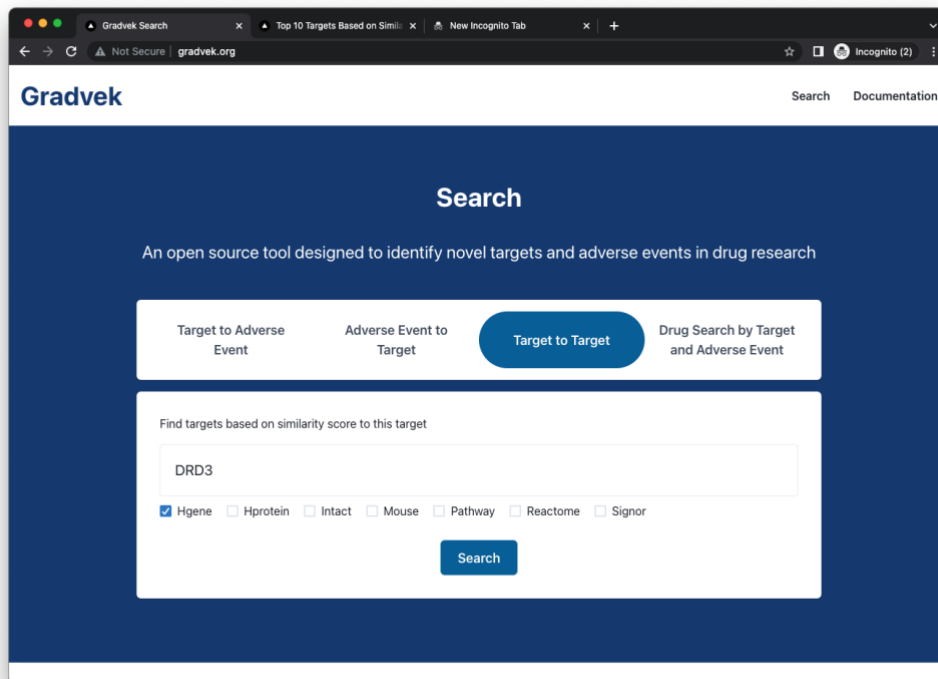


Figure 8: Target to target similarity search specification, flexible across the 8 distinct data sources

The screenshot shows the results page of the Gradvek Search interface. The page title is 'Top 10 Targets Based on Similarity hGene for DRD3'. The results are displayed in a table with four columns: 'TARGET', 'ADVERSE EVENTS', 'SIMILARITY SCORE', and 'DESCRIPTOR'. The table lists 10 targets, each with a 'View Adverse Events' link. The targets are ranked by their similarity score, with ENSG00000224931 having the highest score of 0.75. The descriptors for all targets are 'hGene'.

TARGET	ADVERSE EVENTS	SIMILARITY SCORE	DESCRIPTOR
ENSG00000224931	View Adverse Events	0.75	hGene
KIRREL3-AS3	View Adverse Events	0.75	hGene
LINC02134	View Adverse Events	0.74	hGene
ENSG00000233069	View Adverse Events	0.74	hGene
ENSG00000279674	View Adverse Events	0.71	hGene
CLK3P2	View Adverse Events	0.70	hGene
SSBP3P2	View Adverse Events	0.68	hGene
ENSG00000279325	View Adverse Events	0.67	hGene
DRG1P2	View Adverse Events	0.67	hGene
LINC01476	View Adverse Events	0.67	hGene
LINC02836	View Adverse Events	0.67	hGene
LINC01007	View Adverse Events	0.65	hGene
www.gradvek.org/targetToAdverseEvents/ENSG00000224931	View Adverse Events	0.65	hGene

Get Weighted Average Similarity

List the weighted average node similarity scores for a target across all descriptors. Takes weights as json in url, for example http://localhost:8000/api/weighted_average_similarity/FGG/?weight_hprotein=0.2&weight_mousepheno=2.8

```
GET /api/weighted_average_similarity/FGA/?weight_hprotein=0.2&weight_hgene=0.2&weight_mousepheno=0.9&weight_pathway=0.8
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
  {
    "target1": "FGA",
    "target2": "KSR1",
    "average": 0.6000000000000001,
    "descriptors": {
      "pathway": {
        "original_similarity": 0.75,
        "weight": 0.8,
        "weighted_similarity": 0.6000000000000001
      }
    }
  },
  {
    "target1": "FGA",
    "target2": "MARK3",
    "average": 0.6000000000000001,
    "descriptors": {
      "pathway": {
        "original_similarity": 0.75,
        "weight": 0.8,
        "weighted_similarity": 0.6000000000000001
      }
    }
  },
  {
    "target1": "FGA",
    "target2": "KSR2",
    "average": 0.6000000000000001,
    "descriptors": {
      "pathway": {
        "original_similarity": 0.75,
        "weight": 0.8,
        "weighted_similarity": 0.6000000000000001
      }
    }
  }
]
```

```

{
  "target1": "FGA",
  "target2": "CNKSR1",
  "average": 0.40606060606060607,
  "descriptors": {
    "intact": {
      "original_similarity": 0.01818181818181818,
      "weight": 1,
      "weighted_similarity": 0.01818181818181818
    },
    "pathway": {
      "original_similarity": 0.75,
      "weight": 0.8,
      "weighted_similarity": 0.6000000000000001
    }
  }
},
{
  "target1": "FGA",
  "target2": "FGG",
  "average": 0.3942335619824496,
  "descriptors": {
    "mousepheno": {
      "original_similarity": 0.13793103448275862,
      "weight": 0.9,
      "weighted_similarity": 0.12413793103448276
    },
    "hgene": {
      "original_similarity": 0.9354838709677419,
      "weight": 0.2,
      "weighted_similarity": 0.1870967741935484
    },
    "intact": {
      "original_similarity": 0.16666666666666666,
      "weight": 1,
      "weighted_similarity": 0.16666666666666666
    },
    "pathway": {
      "original_similarity": 0.875,
      "weight": 0.8,
      "weighted_similarity": 0.7000000000000001
    },
    "reactome": {
      "original_similarity": 0.9375,
      "weight": 1,
      "weighted_similarity": 0.9375
    },
    "signor": {
      "original_similarity": 0.25,
      "weight": 1,
      "weighted_similarity": 0.25
    }
  }
},
}

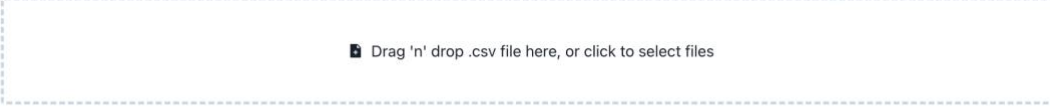
```

Figure 9: Target to target similarity results, aggregating similarity scores across selected sources into a single similarity score

Users are able to upload their own proprietary data via the data upload page.

Data Uploads

Upload Custom Data Sets



Upload File

Figure 11: Data upload with the click of a button

More advanced users can interact directly with the knowledge graph via the separately hosted Neo4j backend via CypherQuery.

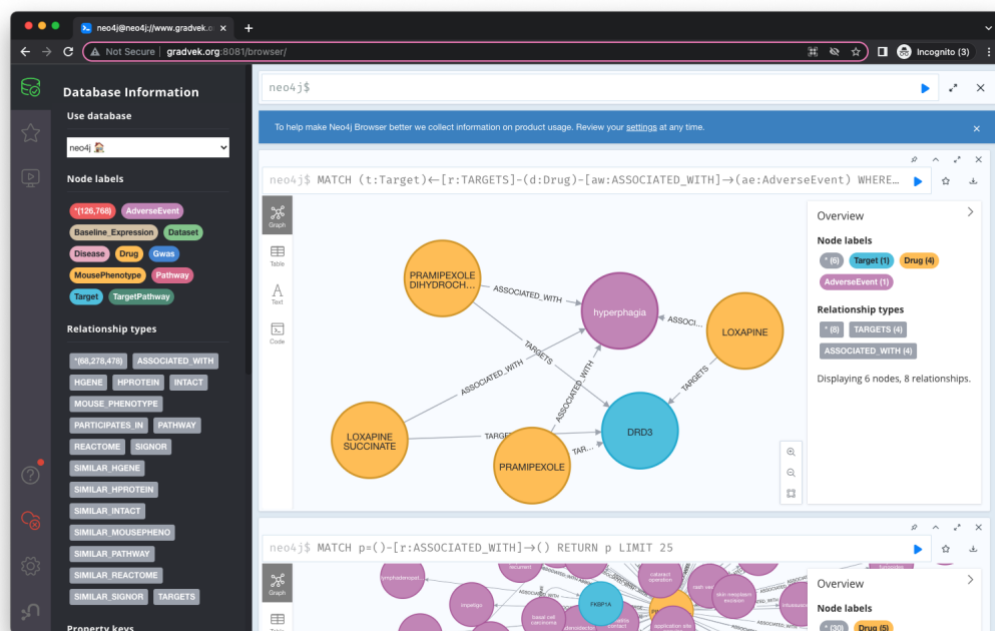


Figure 12: Neo4j backend interaction for more technical users

DISCUSSION

The data required for this application presented several challenges. First, Open Targets is a data aggregator, not a data provider themselves, and documentation for the individual tables had to be traced to the original sources. And even with documentation in hand, a degree of subject matter expertise is required to parse column headers and identify the relevant data for each desired purpose. Secondly, the data volume was fairly high – pairwise similarity calculations between twenty thousand targets across 8 similarity sources meant computation on the full dataset was slower than ideal, as initial attempts at the full set of calculations took over a day with 16GB RAM and an Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz processor before subsequent refactoring and optimizations.

While the existing architecture is largely optimized for Open Targets file formatting and APIs, the long timelines involved in drug discovery means that any potential chance to find a viable drug pathway or highlight a potential risk earlier in the process would likely be worth the cost of integrating the data source that might hold that connection. Connecting to data sources outside of Open Targets could greatly increase our application's insight and effectiveness. We have made further extension easy via a CSV upload user interface, and have a fairly flexible means of adding FTP endpoints via the `get_datasets` function, but if new entity or relationship types were added they would need specific parsing functions added. Enabling a more flexible means of adding new sources and new entity types would allow GradVek to incorporate more data and provide more value to researchers.

GradVek currently allows users to add proprietary data, but only on locally hosted instances of the application. Expanding the architecture to enable secure proprietary data ingestion on public hosting would make it easier for physically separated users to access the same information. In this day and age, local hosting is a rare nuisance when so many services are available no matter the access point. However, securely creating separate databases for each group of users and multiple instances of the graph would be a large overhead and potential security risk depending on the sensitivity of data uploaded to the application. Enabling this sharing of proprietary data in a secure fashion would be a nice feature for users, but would be a substantial undertaking depending on the sensitivity of the data.

CONCLUSION

Knowledge graphs are a promising technology for advancing drug discovery and safety. They allow for the integration of diverse sources of information, enabling researchers to identify new relationships and gain a better understanding of the complex interactions between drugs, targets, and adverse events. GradVek is not the first knowledge graph to encode the relationships between drugs, genes, targets and adverse events. However, none present an easy target-to-target search capability or user-focused interface for researchers to explore the knowledge graph.

GradVek's focus on end user experience represents a novel contribution to the field, enabling researchers to get the benefit of knowledge graph embeddings of molecular relationships with minimal cost. For non-technical users or more advanced modelers, combining knowledge graphs with these user enhancements provides concrete, practical benefits to researchers actively developing new drugs.

REFERENCES CITED

- [1] DiMasi, J. A., Grabowski, H. G. & Hansen, R. W. Innovation in the pharmaceutical industry: New estimates of R&D costs. *J. Health Econ.* 47, 20–33 (2016).
- [2] Joseph A DiMasi, Ronald W Hansen, Henry G Grabowski, The price of innovation: new estimates of drug development costs, *Journal of Health Economics*, Volume 22, Issue 2, 2003, Pages 151-185, ISSN 0167-6296, [https://doi.org/10.1016/S0167-6296\(02\)00126-1](https://doi.org/10.1016/S0167-6296(02)00126-1)
- [3] Kola, I. & Landis, J. Can the pharmaceutical industry reduce attrition rates? *Nat. Rev. Drug Discov.*, <https://doi.org/10.1038/nrd1470> (2004).
- [4] Berlin JA, Glasser SC, Ellenberg SS. Adverse event detection in drug development: recommendations and obligations beyond phase 3. *Am J Public Health.* 2008 Aug;98(8):1366-71. doi: 10.2105/AJPH.2007.124537. Epub 2008 Jun 12. PMID: 18556607; PMCID: PMC2446471.
- [5] Vamathevan, J., Clark, D., Czodrowski, P. et al. Applications of machine learning in drug discovery and development. *Nat Rev Drug Discov* 18, 463–477 (2019). <https://doi.org/10.1038/s41573-019-0024-5>
- [6] Stephen Bonner, Ian P Barrett, Cheng Ye, Rowan Swiers, Ola Engkvist, Andreas Bender, Charles Tapley Hoyt, William L Hamilton, A review of biomedical datasets relating to drug discovery: a knowledge graph perspective, *Briefings in Bioinformatics*, Volume 23, Issue 6, November 2022, bbac404, <https://doi-org.ezp-prod1.hul.harvard.edu/10.1093/bib/bbac404>
- [7] Himmelstein, Daniel Scott, Antoine Lizée, Christine Hessler, Leo Brueggeman, Sabrina L. Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, and Sergio E. Baranzini. 2017. "Systematic Integration of Biomedical Knowledge Prioritizes Drugs for Repurposing." *eLife* 6. <https://doi.org/10.7554/eLife.26726>.
- [8] gnn4dr, "DRKG - A knowledge graph and a set of tools for drug repurposing," [Online]. Available: <https://github.com/gnn4dr/DRKG>.

- [9] Zhang, Yongqi, Quanming Yao, Yingxia Shao, and Lei Chen. 2018. “NSCaching: Simple and Efficient Negative Sampling for Knowledge Graph Embedding,” December. <https://doi.org/10.48550/arXiv.1812.06410>.
- [10] Santos, Alberto, Ana R. Colaço, Annelaura B. Nielsen, Lili Niu, Philipp E. Geyer, Fabian Coscia, Nicolai J. Wewer Albrechtsen, Filip Mundt, Lars Juhl Jensen, and Matthias Mann. 2020. “Clinical Knowledge Graph Integrates Proteomics Data into Clinical Decision-Making.” *bioRxiv*. <https://doi.org/10.1101/2020.05.09.084897>.
- [11] Stephen Bonner, Ian P Barrett, Cheng Ye, Rowan Swiers, Ola Engkvist, Andreas Bender, Charles Tapley Hoyt, William L Hamilton, A review of biomedical datasets relating to drug discovery: a knowledge graph perspective, Briefings in Bioinformatics, Volume 23, Issue 6, November 2022, bbac404, <https://doi-org.ezp-prod1.hul.harvard.edu/10.1093/bib/bbac404>
- [12] Qiagen Ingenuity Pathway Analysis. <https://digitalinsights.qiagen.com/news/blog/discovery/using-knowledge-graphs-to-drive-drug-discovery/>.
- [14] Petkova, Gergana. 2019. “Knowledge Graph Drug Discovery.” Ontotext. November 14, 2019. <https://www.ontotext.com/solutions/knowledge-graph-drug-discovery/>.
- [15] Adrià Fernández-Torras, Miquel Duran-Frigola, Martino Bertoni, Martina Locatelli, Patrick Aloy. An open approach to systematically prioritize causal variants and genes at all published human GWAS trait-associated loci, 2022.
- [16] License. Data Source Licensing. <https://platform-docs.opentargets.org/licence#data-sources-licensing>
- [17] Liangying Yin, Yaning Feng, Alexandria Lau, Jinghong Qiu, Pak-Chung Sham, Hon-Cheong So. A Bayesian network-based framework to uncover the causal effects of genes on complex traits based on GWAS data, 2022.
- [18] Kai Zhao¹, Yujia Shi¹, Hon-Cheong So. Prediction of drug targets for specific diseases leveraging gene perturbation data: A machine learning approach, 2021.
- [19] Ding H, Takigawa I, Mamitsuka H, Zhu S. Similarity-based machine learning methods for predicting drug-target interactions: a brief review. *Brief Bioinform*. 2014 Sep;15(5):734-47. doi: 10.1093/bib/bbt056. Epub 2013 Aug 11. PMID: 23933754.
- [20] Clinical Knowledge Graph, [Online]. Available: <https://ckg.readthedocs.io/en/latest/>

DESIGN OF THE SYSTEM (FOR SOFTWARE ENGINEERS)

Introduction

GradVek (GRaph of ADVerse Event Knowledge) is a software system designed to manage OpenTargets Drug data. It provides an interactive interface for users to explore and understand drug-related adverse events. This section outlines the design of the system, including frontend, backend, and database components, as well as the choice of language/platform.

System Overview

GradVek consists of the following components:

1. Frontend: A Next.js application offering an interactive user interface for accessing and visualizing the data. The frontend is organized into various pages and components that are built using React and use hooks for data fetching. It communicates with the backend via API endpoints, which provide the necessary data for rendering and visualization.
2. Backend: A Django REST API retrieves data from OpenTargets, stores it in a Neo4j Graph Database, and serves various data endpoints to the frontend. The backend is organized into different sections for each of its main functions, which work together to handle data ingestion, querying, and endpoint management. It supports routes that provide all the information for the frontend to use.
3. Data Storage: Neo4j Graph Database stores the data and enables querying and retrieval of relationships between entities. The database schema consists of nodes and relationships that represent drugs, targets, adverse events, and other relevant entities. Two Neo4j plugins are used APOC (Awesome Procedures on Cypher) and GDS (Graph Data Science) to extend its capabilities for more efficient queries and computational support.

These components interact with each other to provide a seamless user experience, allowing users to explore and understand drug-related adverse events. The system's design ensures that it can be easily maintained, expanded, and adapted to meet the evolving needs of its users.

Frontend Design

The frontend is built using Next.js, a React-based framework offering server-side rendering, client-side rendering, caching, built-in optimizations for assets, advanced data fetching, and scalability.

The frontend is designed to consume the backend API endpoints and render data in a clean interface.

The frontend is organized into a modular structure, with pages, components, hooks, and utilities. The components are built using React, and hooks are used for data fetching and managing the state. The frontend leverages libraries, such as Vis.js for graph visualization.

Frontend Pages include:

- Home: Provides an overview of the application and search options for various entities, such as drugs, targets, adverse events, and diseases, and access to other sections of the application.
- Target To Adverse Events: Displays adverse event search results and information about adverse events related to specific targets.
- Adverse Event To Target: Displays search results for targets related to a specific adverse event, allowing users to explore and understand the relationships.
- Target To Target: Displays search results for similar targets and their relationships, facilitating the discovery of commonalities between targets.
- Drug Search by Target and Adverse Event: Enables users to find drugs that have a particular target and adverse event associated, providing valuable insights into drug-target interactions and potential side effects.
- Settings: Allows users to configure the application settings, manage data sources, and view system information.

These pages are designed to be responsive, ensuring a seamless web browser experience for laptop, desktop, and several tablet mobile devices. The frontend employs a consistent color scheme and design elements throughout the application, providing a unified look and feel.

Backend Design

The backend is built using Django, a Python web framework, along with the Django REST framework for creating an API that serves as the intermediary between the frontend and the Neo4j Graph Database. The backend is organized into three sections, the main section, search, is organized in a modular structure, with apps, models, views, serializers, and supporting utility

functions working together to handle data ingestion, querying, and endpoint management. It leverages various libraries, such as neomodel for working with the Neo4j database and pandas for data processing.

The backend is divided into three sections:

- **Datasets:** This section contains the `get_datasets.py` file, which handles retrieving OpenTargets data from the FTP server, and the `parse_datasets.py` file, which processes the OpenTargets Parquet files and constructs APOC Cypher queries to upload the data to Neo4j with batched and parallelized processing when possible.
- **Gradvekbackend:** This section contains the Django app's settings and startup functions.
- **Search:** This section includes the URL configurations, views, utility functions, queries, and computing and retrieving similarity scores using GDS. It provides all the endpoints that the frontend uses.

On startup, the backend evaluates the Parquet data, data in Neo4j, and cached similarity score data. It runs the appropriate functions to ensure that all data is correct, available, and up-to-date automatically. The backend does not use `py2neo`; instead, it employs the `neomodel` library for working with Neo4j.

The API is responsible for the following tasks:

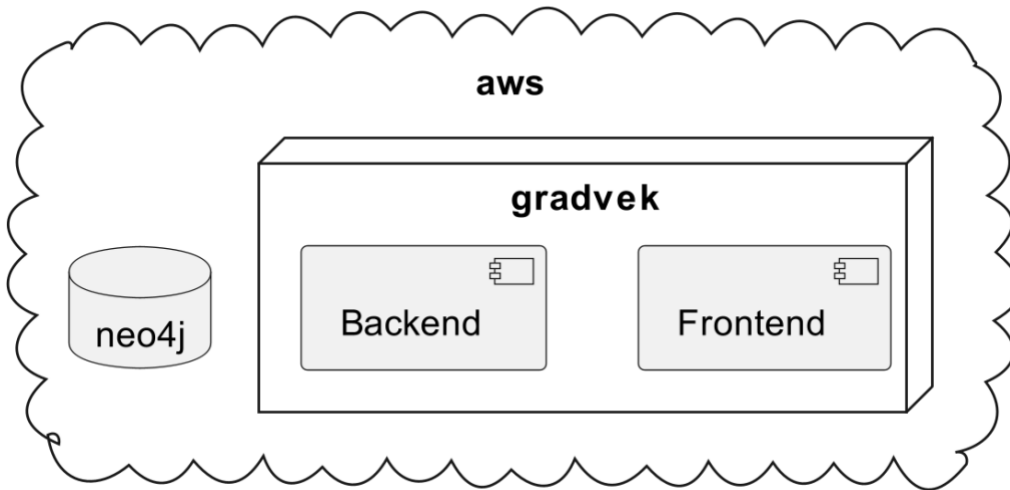
- **Data Ingestion:** The OpenTargets data is downloaded, stored, and processed as part of the startup. The backend also provides an endpoint for ingesting additional user data in the form of CSV files. Data ingestion includes processing raw data, cleaning, and transforming it into a suitable format for storage in the database.
- **Querying:** Fetching data from the Neo4j Graph Database based on frontend requests and transforming it into a suitable format for consumption by the frontend. It involves executing Cypher queries, aggregating results, and applying various filters and sorting options as needed. The backend also leverages Neo4j plugins like APOC and GDS to optimize query performance and conduct complex graph analysis.

- **Endpoint Management:** Defining various endpoints for the frontend to access, such as retrieving graph data such as paths between various entity types, searching for entities, and fetching computed data such as similarity metrics. The backend provides comprehensive API documentation using tools like Swagger, which enables developers to understand and interact with the API more effectively.

UML Diagrams

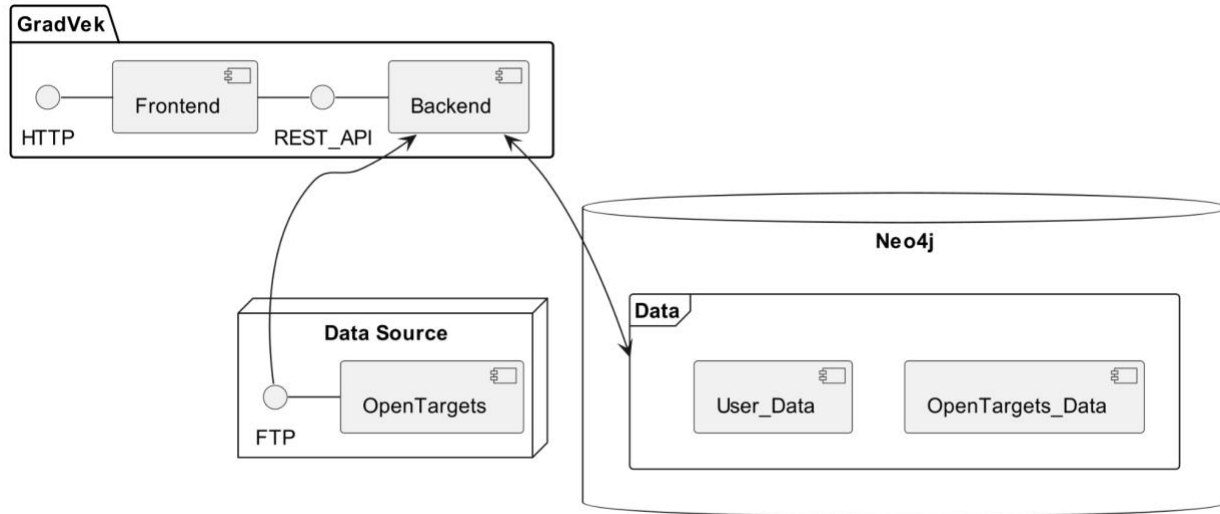
These Unified Modeling Language (UML) diagrams represent the system's structure and design.

Deployment Diagram: The application will be containerized and hosted on AWS. It will be hosted along with a neo4j database.



Component Diagram:

- The frontend will present the application via HTTP.
- The backend will present a REST API for the frontend to make requests to.
- The backend will retrieve OpenTargets data via local Parquet files and receive updates via FTP.
- The backend will communicate with the Neo4j database via the BOLT protocol.

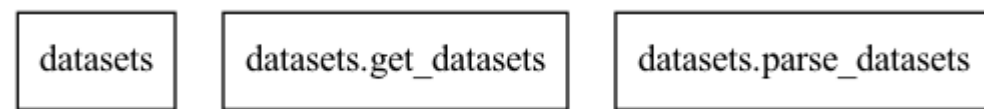


Backend UML Diagrams

The backend is split into three sections, each handled a little differently to create code that is easy to understand and extend. An added note, pieces can run individually. Get_Datasets can run entirely independently and parse_datasets can run on its own with just the database running.

Datasets Package Diagram

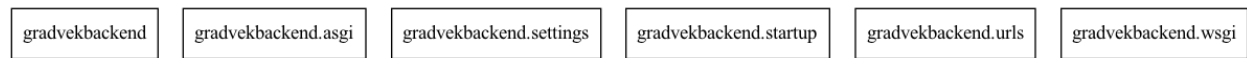
Datasets: Written entirely as functions, the two main components operate in a similar fashion



- Get_dataests: Composes of several functions to manage the fetching and organization of open targets data. The download mechanism retrieves multiple files in parallel and supports a retry mechanism and graceful error handling. It is centered around a dictionary written in such a way that changing or adding more datasets is simply a matter of making a new entry, stating the name of the data type, and the source directory for the data to be retrieved.
- Parse_datasets: Similar to get_dataests, this handles the parsing and is written in a way that to add data you simply need to add a line the the main dictionary with the name of the data type, and the functions that generate the desired cypher queries using that data.

Gradvekb backend Package Diagram

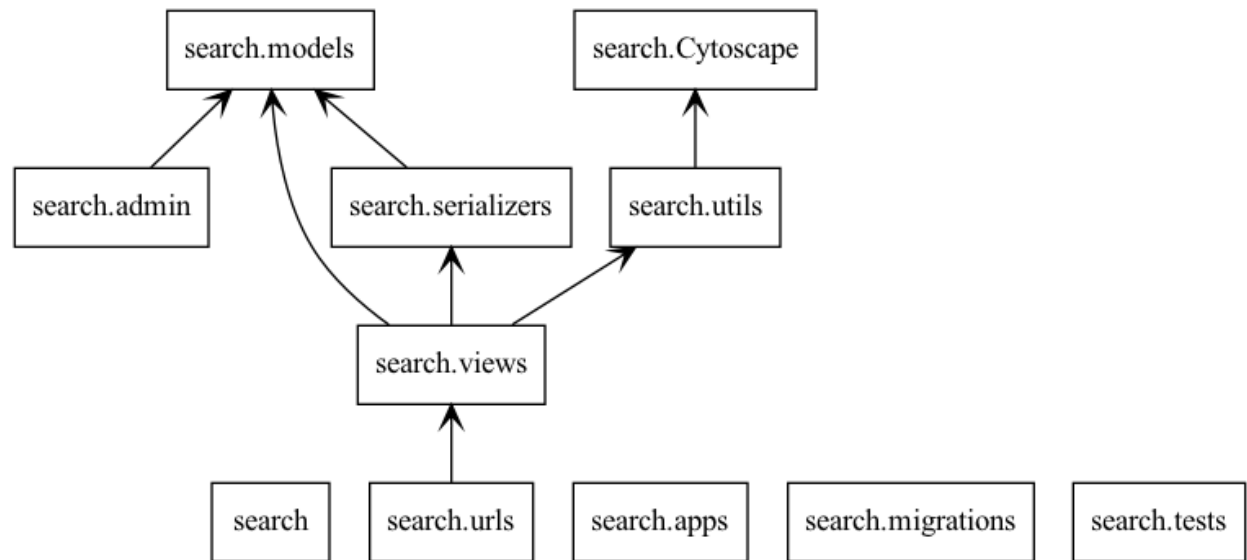
Gradvek backend is primarily django configuration, settings, and startup triggers.



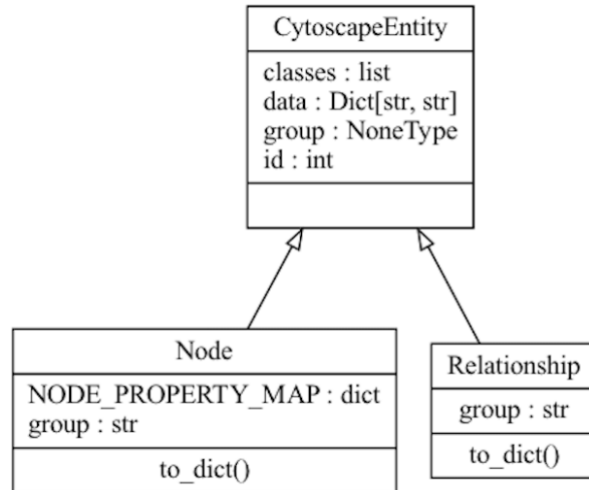
This section handles django setup and configuration, as well as startup functions triggering the functions in the datasets package, as well as handling establishing the connection between the backend and neo4j, using a retry and timeout mechanism, as well as handling connections changes between environments such as running natively or in a docker container.

Search Package Diagram

The search section of the backend is where the routes are provided and the backend communicates with neo4j to retrieve and compute the return results.



- Urls: Where all the routes the backend provides are defined
- Views: Called by Urls, takes path arguments, and passes them to util functions, organized into classes with GET and POST functions when appropriate
- Utils: Constructs cypher queries, communicates with neo4j, and returns the results to the view, as well as additional functions to support processing retrieved data when needed.
- Cytoscape: Provides the necessary classes to process and organize paths of data retrieved from neo4j
 - Cytoscape Entity Class Diagram: Used as part of processing Target to Adverse event and Adverse Event to Target paths



Language and Platform

GradVek's choice of language and platform is influenced by the following factors:

1. **Scalability:** Both Next.js and Django are known for their scalability, handling a growing user base and increasing data volumes efficiently.
2. **Ecosystem:** The JavaScript and Python ecosystems have many libraries and tools that can be used to build GradVek more efficiently.
3. **Developer Experience:** Next.js and Django offer a modern developer experience, built on popular languages and supporting useful features such as hot module replacement.
4. **Data Handling:** Django's support for various databases and its ability to work with Neo4j make it an ideal choice for a data-intensive application like GradVek. The Neo4j Graph Database provides efficient querying and management of complex relationships between drug data entities.

TESTING RESULTS (FOR SOFTWARE ENGINEERS)

The majority of the testing during development has been done manually. With the dramatic speed improvement in data processing, it was no longer unreasonable to repeatedly test loading and processing of data.

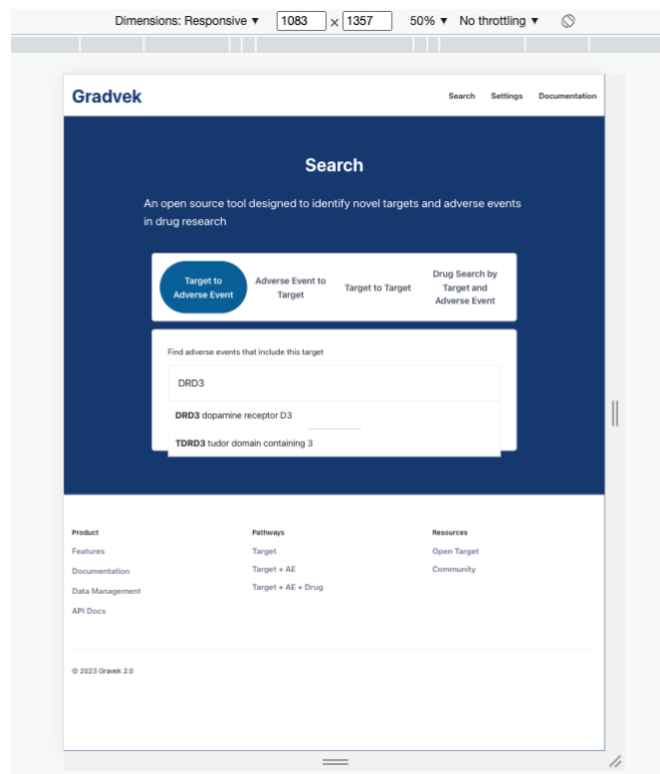
We used GradVek 1.0 to check for response similarity when initially working on the same data they had, expecting the same results. Every time the data underwent a transformation, we would verify that our results conformed with previous states. From checking against Open Targets, to processing parquet files, checking against data loaded in neo4j, to the endpoint responses, to the data displayed in the front end. We confirmed the data at each stage was correct.

When a complete path was considered correct, we would keep a record of the expected response for common targets, such as DRD3. Whenever there was a change, we would clear the data and code, and do an initial pull setup of the system and check the final stage was still correct.

We check similarity data for select target comparisons across each descriptor to ensure it is unchanged. Additionally, there are tests to check the url responses, to validate each endpoint still works and the results are unchanged.

UI Tests

As part of our initial testing, we presented designs and the search prototype to the client to collect user feedback. The session provided insights into how they will use the application, where they are expecting to see certain data, and why certain sections take priority over others. For example, the client expects the adverse event search to include actions they can select to refine their search, for targets they are expecting to see the top 10 targets at the top, and they will load the Knowledge Graph after they have reviewed the tabular data. These have all been updated and incorporated into the application. We also utilized assertions on the frontend to ensure that the data was passed correctly, and we performed API tests to ensure that we received all the expected data. Additionally, we made sure the application is web responsive and that it works well on various platforms such as computer and tablet. To test the application's responsiveness, we used the Google Chrome inspect tool.



DEVELOPMENT PROCESS & LESSONS LEARNED REFLECTION (FOR FACULTY PANEL)

Meeting the Requirements

All of the major requirements were met and there are a few less critical requirements that were pushed into Milestone 4. The frontend and backend were both rebuilt and support target to target similarity which utilizes the extra data points that GradVek now retrieves and parses and can give a similarity score between two targets. Users can search for targets and see the top 10 adverse events associated with that target and vice versa. The client will be able to upload their own proprietary data for loading drugs to adverse event relationships and target to disease relationships. Given that the front end is rebuilt, the user will have a new and more user-friendly interface to interact with that we hope is more useful than the original. GradVek now supports data downloads directly from the Open Targets ftp site so those files will be downloaded and parsed when the

application is first run. When Open Targets updates their data, the new GradVek will recognize that and update its dataset and neo4j database. Our team has also taken care to ensure that computation happens as quickly as possible. GradVek 1.0 could not download datasets directly from Open Targets and took over an hour to load the datasets into its neo4j database. GradVek 2.0 can download, parse and load the datasets into neo4j in about 15 mins.

A few requirements our team is still working on include removing the hairball effect in the visualized knowledge graph that the user would interact with. Another requirement that is still being worked on is calculating target to target similarity based on user requested weights as opposed to having equal weights for all data types. Lastly there is a lingering requirement to give the user the ability to export the knowledge graph results to a CSV file. Our team is working diligently and hope to get as many of these requirements done by the faculty panel on May 10, 2023 but some of these requirements may be pushed to milestone 4.

Estimates

We came fairly close with our estimates on time for the project. Given that we fell a few requirements short, we somewhat underestimated how much time certain tasks would take. Looking back, we recognize that working on trying to update the original Gradvek took a lot of time that added little value ultimately. If we had started with rebuilding the backend from the beginning, we're confident we would have completed all requirements in time as we would have had more time. (See requirements table in Appendix)

Risks

The largest unavoidable risk we had was that we would not have enough time to complete the project. Since all the work leading to the Milestone 1 requirement was planning related, we only had about 2 months for all development, testing, integration, etc. This was a valid concern as we are racing to get as much done as we can before the May 10 deadline. This risk is compounded when you account for the fact that no one on the team had experience in the subject matter of GradVek. However, each person spent the time needed to learn the subject matter adequately enough to contribute to the project.

Team Dynamic

Overall, our team dynamic was very positive. We had David working on the front end. Nathan and Kevin working on the backend. Alyssa worked on both the front end and backend. Ali and Dixon worked on the subject matter computation and research. All team members were active contributors over Slack, weekly meetings, client meetings, and Trello, which we used for requirements tracking.

We've all found that meeting Wednesday after class was the best time to meet and sync up with one another. We've also found that all of us generally are available most on the weekends so we've generally had our pair programming/troubleshooting sessions on Saturday and Sunday. Also, we decided to increase the frequency of our team meetings to two after reaching milestone number two. We were also meeting on Sundays at 8 p.m. Ultimately finding times to meet and collaborate has not been an issue. Regarding M3, we met with the client halfway through M3 and communicated with the client via email if additional clarification was required.

All team members have been good at communicating via Slack. All members were responsive in a timely manner. One issue we were having is that we used Github tasks to keep track of what each person was working on. It seemed that most team members didn't use this tool very well. Perhaps it was hard to use or there was a different reason. Regardless, we've switched to using a Trello board which mimics a Jira Sprint board. On it, we could see which tasks were in the current sprint, backlog, assigned members, etc. This has proven to be more useful.

When we decided to use Django and Next.js, it was clear that we needed to change our workflow to stay on track with our initial estimates. We agreed to follow a Scrum Framework and have one-week sprints. We used Trello to set deadlines and track everyone's progress and we had a project backlog with a list of functionality/tasks we wanted to prioritize. Every week began on Wednesday at 8 p.m. and concluded on Wednesday at 6 p.m. before class. At Wednesday's meeting, we referred to the Trello board to discuss what was completed and what we could deliver by next Wednesday, based on our initial estimates. Furthermore, we had a mid-week check meeting on Sunday to see how things were going and to hold ourselves accountable to ensure we could complete all of the scheduled tasks for the week's sprint.

We agreed to deal with code changes by using feature branches. We created a branch for issues or new features and then submitted a pull request to merge the changes so we did not push straight to the main branch. We also set peer review before pull requests could be merged into the main branch.

APPENDIX

WEBSITE URL

<http://www.gradvek.org/>

SYSTEM INSTALLATION MANUAL

System installation manual describing how to obtain and install the system.

To get started first checkout the readme at <https://github.com/team-gradvek/gradvek>.

The first step will be to pull the repo you your machine <https://github.com/team-gradvek/gradvek>

Next, you will need a Django secret key, you can generate on with the command

```
python3 -c 'from django.core.management.utils import get_random_secret_key;
print(get_random_secret_key())'
```

Save this key, you will need it however you choose to launch to application

Now you have two options:

1. Enter your secret key in the designated location in docker-compose-published.yml and use the command 'make run-deployed' to run the application. This will use the already built images that are published by the projects CI/CD pipeline, it just needs to have a secret key provided and it works!
2. Set up your local machine to create and run the images.
 - a. Inside the code for the project, run the terminal commands 'make check-environment' It will inform you of tools you need to install, and once you have them all it will handle installing all the remaining dependencies
 - b. Create a .env file in the folder ./backend/gradvekbackend and enter the following information
SECRET_KEY=<the secret key you made>
NEO4J_USERNAME=neo4j
NEO4J_PASSWORD=gradvek1

NEO4J_BOLT_URL=bolt://neo4j:gradvek1@localhost:7687

and then run ‘make run-all’

- c. These values can all be changed, but any changes need to also be reflected in the dockerfile in the neo4j folder

DEVELOPER INSTALLATION MANUAL

GradVek (GRaph of ADVerse Event Knowledge) is a project aimed at providing an interface for searching drug target information and finding adverse events for similar targets. It includes a Django backend, a Neo4j database, and a Next.js frontend.

If you’re using particular frameworks or other developer environments, describe essentially the documentation that you use for your team to get up to speed.

First, follow the setup for system installation. Follow the instructions for option 2. Now you will also be able to run the application in a third way that is helpful for development. Run neo4j with the command ‘make run-neo4j’. Now you can use the command ‘make run-backend’ and ‘make run-frontend’ and it will run those parts of the application on your local machine, outside of docker. These applications support hot-reloading, so you can make code changes while they are running and the changes will be applied.

There is a full table of make commands in the Readme, additionally you can get a list and description of each command with ‘make help’. If you would like to know more about what the commands are doing checkout the contents of the makefile.

The setup instructions can be found in our Readme: <https://github.com/team-gradvek/gradvek>

REQUIREMENTS AND ESTIMATES

Target to Target Search View				
ID	Delivery Milestone	Tasks	Description	Status
1	2	UI/UX Design	UI shall provide a search page where a user can enter multiple targets and select descriptors (9 options). UI will offer a select/deselect all option.	Complete
2	2	Implementation (frontend)		Complete

Target to Target Search Results				
ID	Delivery Milestone	Tasks	Description	Status
3	2	UI/UX Design	Based on Target to Target Search Data entered, this page will show the top 10 targets based on similarity calculations and offers weight sliders and filtering options for results. Results: <ul style="list-style-type: none"> • Display top 10 targets • Show table based on selected filters from search • Show tab to load Knowledge Graph (hairball) • Provide options for Knowledge Graph to render based on filters applied (from the top 10 allow user to select which to display) Filtering options:	Complete
4	2	Implementation (frontend)		Complete

			<ul style="list-style-type: none"> • 1) AEs (through drug to target connections) • 2) Diseases (target to disease) • 3) Phenotype (target to phenotype) 	
--	--	--	--	--

Adverse Event Search				
ID	Delivery Milestone	Tasks	Description	Status
5	2	UI/UX Design	Adverse Event Search provides the ability for a user to enter multiple targets and actions (inhibitor, antagonist, etc) to find an AE	Complete
6	2	Implementation (frontend)		Complete

Adverse Event Search Results				
ID	Delivery Milestone	Tasks	Description	Status
7	2	UI/UX Design	Based on Adverse Event Search Data entered, this page will show: Table that with AEs showing target > drug > AE relationship. Tab to load Knowledge Graph	Complete
8	2	Implementation (frontend)		Complete

Adverse Event to Target Search				
ID	Delivery Milestone	Tasks	Description	Status
9	2	UI/UX Design	Adverse Event to Target Search provides the ability to enter multiple adverse events to find a target.	Complete
10	2	Implementation (frontend)		Complete

Adverse Event to Target Search Results				
ID	Delivery Milestone	Tasks	Description	Status
11	2	UI/UX Design	Based on Adverse Event to Target Search Data entered, this page will show: Table that with targets showing a AE > drug > target relationship Tab to load Knowledge Graph	Complete
12	2	Implementation (frontend)		Complete

Settings Page				
ID	Delivery Milestone	Tasks	Description	Status

13	2	UI/UX Design	Support for application API Keys and other application specific configuration options for distribution	Incomplete
14	2	Implementation (frontend)		Incomplete

Knowledge Graph				
ID	Delivery Milestone	Tasks	Description	Status
15	2	UI/UX Design	Redesign and improvements to the current KG rendering. New KG will account for additional data added such as similarity calculations.	Complete
16	2	Implementation (frontend)		Complete

Proprietary Data and Open Target Upload				
ID	Delivery Milestone	Tasks	Description	Status
17	2	UI/UX Design	Design of data upload page for proprietary data with file validation. Common page for all users using the self-hosted application.	Complete
18	2	Implementation (frontend)		Incomplete

19		Data Ingestion (backend)	Quarterly automated data pull of FTP Open Target site parquet file with option to pull on-demand via the Settings Page (7).	Complete
----	--	--------------------------	---	----------

Target Similarity Calculations and Data Processing - Backend				
ID	Delivery Milestone	Tasks	Description	Status
20	2	Computation of Similarity - Descriptors	Data ingestion and graph creation for 9 descriptors: Gene, GWAS, Mouse Phenotype, Reactome, Signor, IntAct, Literature (Bibliography), Known Drug, Pathways	Complete
21	2	Computation of Similarity - Descriptors	Calculation for 9 descriptors: Gene, GWAS, Mouse Phenotype, Reactome, Signor, IntAct, Literature (Bibliography), Known Drug, Pathways Computation: Jaccard Similarity	Complete
22	2	Computation of Similarity - Targets	Sum of all descriptors for a given target and how they compare with other targets. The higher the score the more similar they are and must be included in the top 10 results.	Incomplete
23	2	Descriptor Weights	Calculation for 9 descriptors: Gene, GWAS, Mouse Phenotype, Reactome, Signor, IntAct, Literature (Bibliography), Known Drug, Pathways - based on minimum threshold provided by client.	Complete

24	2	Target toggles	If a target is removed from the comparison on the results page, this will require data to recompile and the interface to trigger a repaint.	Complete
25	2	Performance Research	Review computation complexity, processing time, and data storage for descriptors and targets to maximize performance.	Complete
26	2	Client Sample for T in arget Similarity Mini mum Threshold	Calculate target similarity calculations, establish graphs showing the distribution, and prepare samples to review with the client.	Complete
27	2	Client Review	Conduct a meeting with the client to review sample calculations for descriptors and determine the minimum accepted threshold.	Complete

Knowledge Graph				
ID	Delivery Milestone	Tasks	Description	Status
28	3	Additional Data in KG	Conduct a meeting with the client to review data added to the Knowledge Graph.	Complete
29	3	Filtering (frontend)	Implement rendering of Knowledge Graph with filters applied.	Complete
30	3	Export (frontend/backe nd)	Application will offer an export option for the Knowledge Graph so the client and others can share this information with other key stakeholders in AE/target studies.	Incomplete

Gradvek 2.0 APIs				
ID	Delivery Milestone	Tasks	Description	Effort Points
31	3	API Design and Research	API Configuration, Design, and Research to implement API routes to follow in sections 33-3	Complete
32	3	API - Target Lookup for Typeahead Search (backend)	API endpoint that returns top 12 targets based on input from search (like lookup)	Complete
33	3	API - Target to Target Results (backend)	API endpoint that returns target to target results data: <ul style="list-style-type: none"> Top 10 targets based on similarity Table data for each top of the top 10 targets 	Complete
34	3	API - Knowledge Graph Data	API endpoint that returns data to render knowledge graph	Complete
35	3	API - Adverse Event Lookup for Typeahead Search	API endpoint that returns the top 20 adverse events based on input from search (like lookup)	Complete
36	3	API - Adverse Event based on target and action input	API endpoint that returns the adverse event based on target and action input. Data returned: <ul style="list-style-type: none"> Target to drug to AE relationship 	Complete
37	3	API - Adverse Event to Target	API endpoint that returns the targets containing the AE inputs. Data returned: <ul style="list-style-type: none"> AE to drug to target relationship 	Complete

38	3	API Documentation	Document all API endpoints and their functionality. We will use tools like Swagger / ReDoc.	Complete
----	---	-------------------	---	----------

Testing				
ID	Delivery Milestone	Tasks	Description	Status
39	3	Application Testing	Testing of all system components: <ul style="list-style-type: none"> - Frontend - All Search and Result Pages - Backend - All API Routes - Backend - All Data Processing Pipelines (ingestion, upload, processing) - Backend - All Data Storage - Backend - API Documentation 	Complete

TEAM CONTRACT

Purpose

The purpose of the team contract is to set expectations and conventions to how the team will operate. Team members are to abide by all terms stated in this contract.

Team Expectations

Each team member is expected to commit an equal amount of time to the success of the project. During all interactions, team members are to behave in a professional and respectful manner. Each member is expected to listen, understand, and collaborate with the team. Team GradVek is committed to creating a place of psychological safety for all team members. The success of the project is a team effort.

Conflict Resolution

There may be times where the team disagrees on one or more topics. The team will handle conflict resolution through a collaborative discussion where two (2) parties will present their pros and cons. If a resolution is not met between the parties, the team will then discuss collectively as a whole and vote. In the event of a tie, the team will propose testing the ideas or seeking guidance from an external third-party.

Communication

The team will communicate through various mediums including: zoom, slack, email, and phone. Each team member is expected to respond in a timely manner. Slack notifications and emails are to be prioritized and reviewed daily, so no team member misses important updates. Team members are expected to communicate any issue early so the team can provide assistance.

Team Meetings and Availability

Team meetings will be held weekly on Wednesday and members are expected to attend each meeting or notify the team via slack or email of any potential conflicts. Meetings will be recorded for those who cannot attend, and meetings notes will be available in the shared team folder in Google Drive. The team will take turns recording notes each week.

Meetings with the client will be held every week or two depending on client availability and overall project status. Team members are expected to attend these meetings or notify the team via slack or email of any potential conflict.

Team members in agreement of this contract:

David Aviles

Alyssa Bédard

Dixon Bross

Kevin Patel

Ali PiyarAli

Nathan Sheely