**Domain Identification in Alternative Spliced Genes**

**Introduction**In order to run the server you'll need to download node js from:

<https://nodejs.org/en/>

download or clone the project from:

<https://github.com/ninga123/finalProjectServer>

Afterwards, running cmd from the containing folder, write the following commands:





Finally, writing the following path in a browser will result in json answer:  
[http://localhost:3000/querySearch/*<geneName*](http://localhost:3000/querySearch/%3cgeneName)*>*

**Server-Side Programming**

formed out of 2 modules besides the main running file. //right now

* Query Search Module

This module is responsible to find a gene record and its transcripts' information when a user searches via an input text field. Because we have different tables that contain important data which needs to be found by foreign IDs we decided to make several SQL queries. The query line should be filled with a gene name.

The module checks for matches In the gene\_symbol column of the Genes table.If no matches were found it would check different synonyms to find the closest match. There is a possibility were nothing is similar enough for the module to be associated with the query given. In this case, we would alert the user.

When a gene could be identified and its list of trancsripts, the function findTranscriptInfo would use each transcript record to find associated records of exons and domainEvents and their domainTypes. To clarify, domain Event are concrete cases of a domain type so general information can be found In a domain type record and specific information is in the domain event record which is why both records are needed.

Our main goal here is to return all of the information in a way that could be easily interpreted by our front-side modules. The results are in an javascript object/json format keeping an hierarchy of the data in this manner:

{

"gene":{*gene record*},

"transcripts":

[

*array of transcript following the syntax of*

{

*transcript record*,

"exons":[*array of exons records*],

"protein":

{

*protein record*,

"domains":[*array of domain*

*records*]

},

}

]

}

//an example for json will be displayed here when we will have an agreement on the syntax

List of queries used:

Getting record of the gene named – if exists.

SELECT \* FROM Genes WHERE gene\_symbol = '<geneName>'

Getting records of genes that have similar names to gene named

SELECT \* FROM Genes WHERE synonyms LIKE '%<geneName>%'

Getting all transcript records known for the gene. //we may need to check cases with 1 or 0 transcripts if its impossible

SELECT \* FROM Transcripts WHERE gene\_id = '<geneID>'

Getting all exon records for a specific transcript ID

SELECT \* FROM Exons WHERE transcript\_id = '<transcriptID>'

Getting a protein record for a specific transcript ID

SELECT \* FROM Proteins WHERE transcript\_id = '<transcriptID>'

Getting all domain records associated with a specific protein ID

SELECT \* FROM DomainEvent WHERE protein\_id = '<proteinID>'

Getting a domain type record that matches an domain type ID

SELECT \* FROM DomainType WHERE id = '<domainTypeID>'

* Database Utilities Module

This Module currently only opens a connection to the database when starting a server. Configurations will be made available from this module when the time comes. We hope to create an efficient module that can close and re-open the database only when needed. Debugging options and updating may be added to this module or in a separate module- depending on size and complexity.