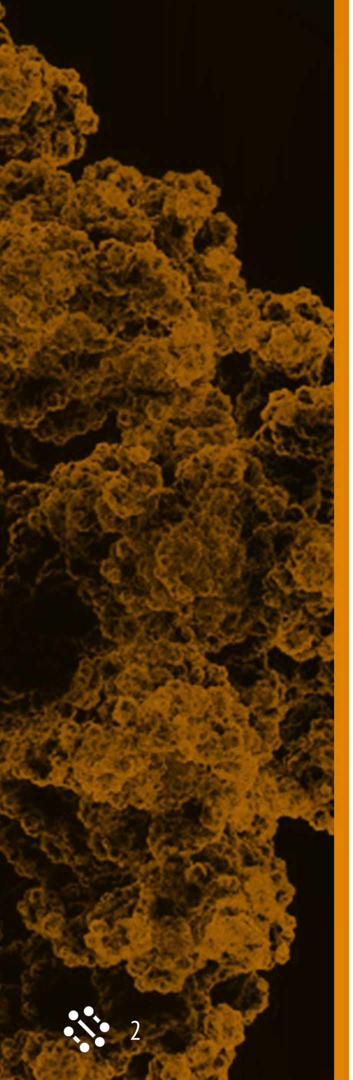
AUTOHOPTS

Hyperparameter Optimization

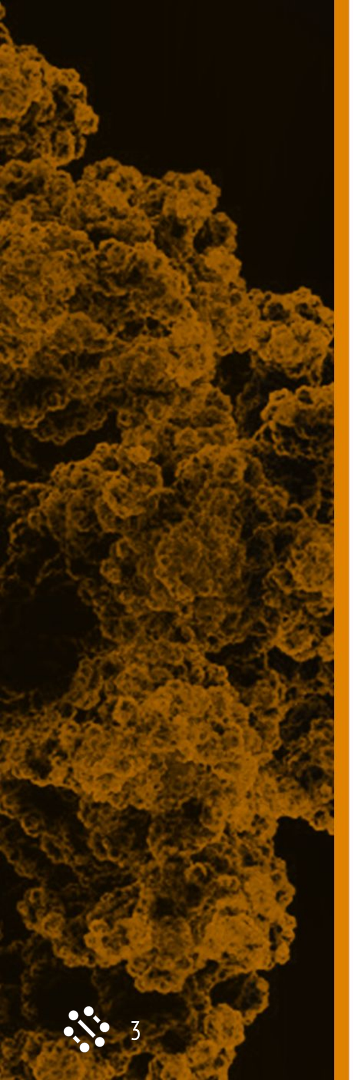
Developed by Rudresh Nitinku, Kristen Laird,
Daniel Belmes, James Michaud, and Ira Ceka



DeepCure

Using AI to discover highly effective small-molecule drug candidates

Analyzing a space of one trillion compounds and multiple hyperparameters in parallel



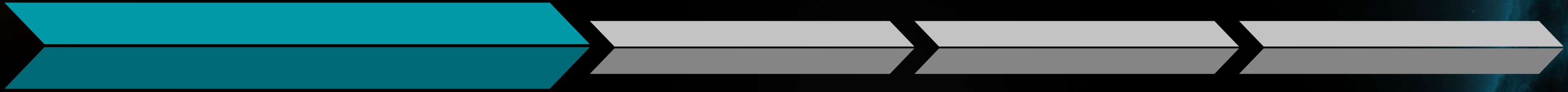
Problem

DeepCure trains and evaluates hundreds of machine learning models every day

Algorithms are run by command line, making it:

- Easy to make mistakes because there is minimal validation
- Hard to get new employees up to speed
- Inaccessible to employees without a strong machine-learning background

Project Scope



Start New Experiment

Configure search space and experiment setup. Send to scheduler to start training

Monitor Experiment

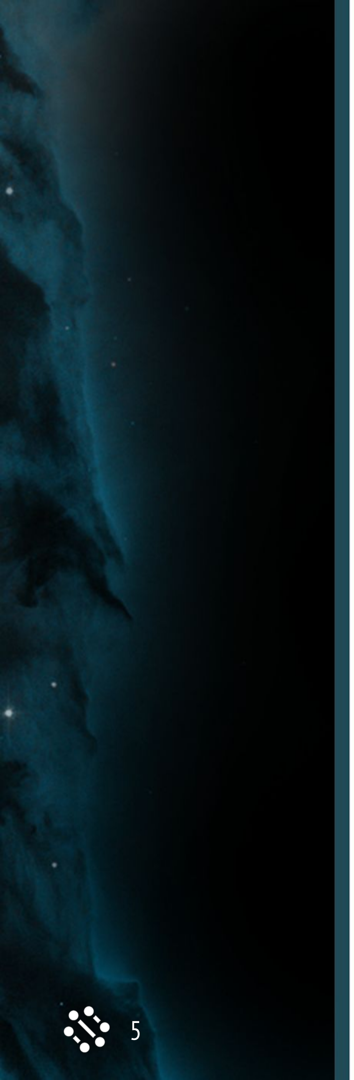
Has anything broken or failed?

Analyze Results

What models performed best? What should be considered next?

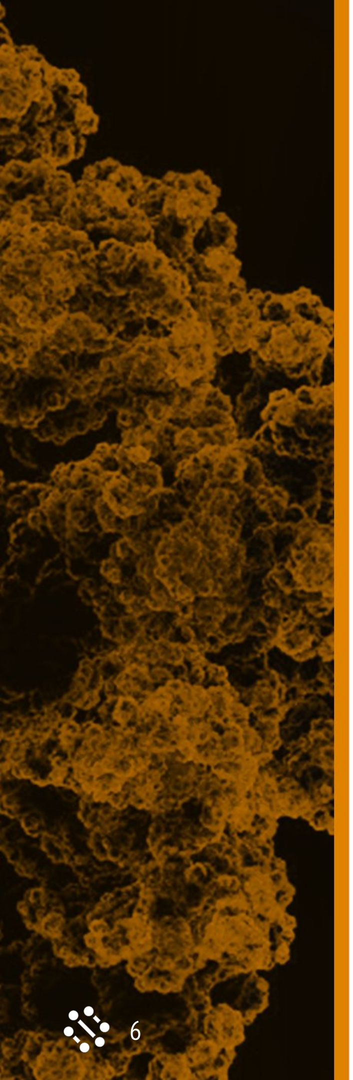
Investigate

Once there’s a good drug candidate, investigate further



Goal

Minimize the overhead in problem solving and increase the speed of DeepCure’s hypothesis testing cycle by making it easier to start an experiment



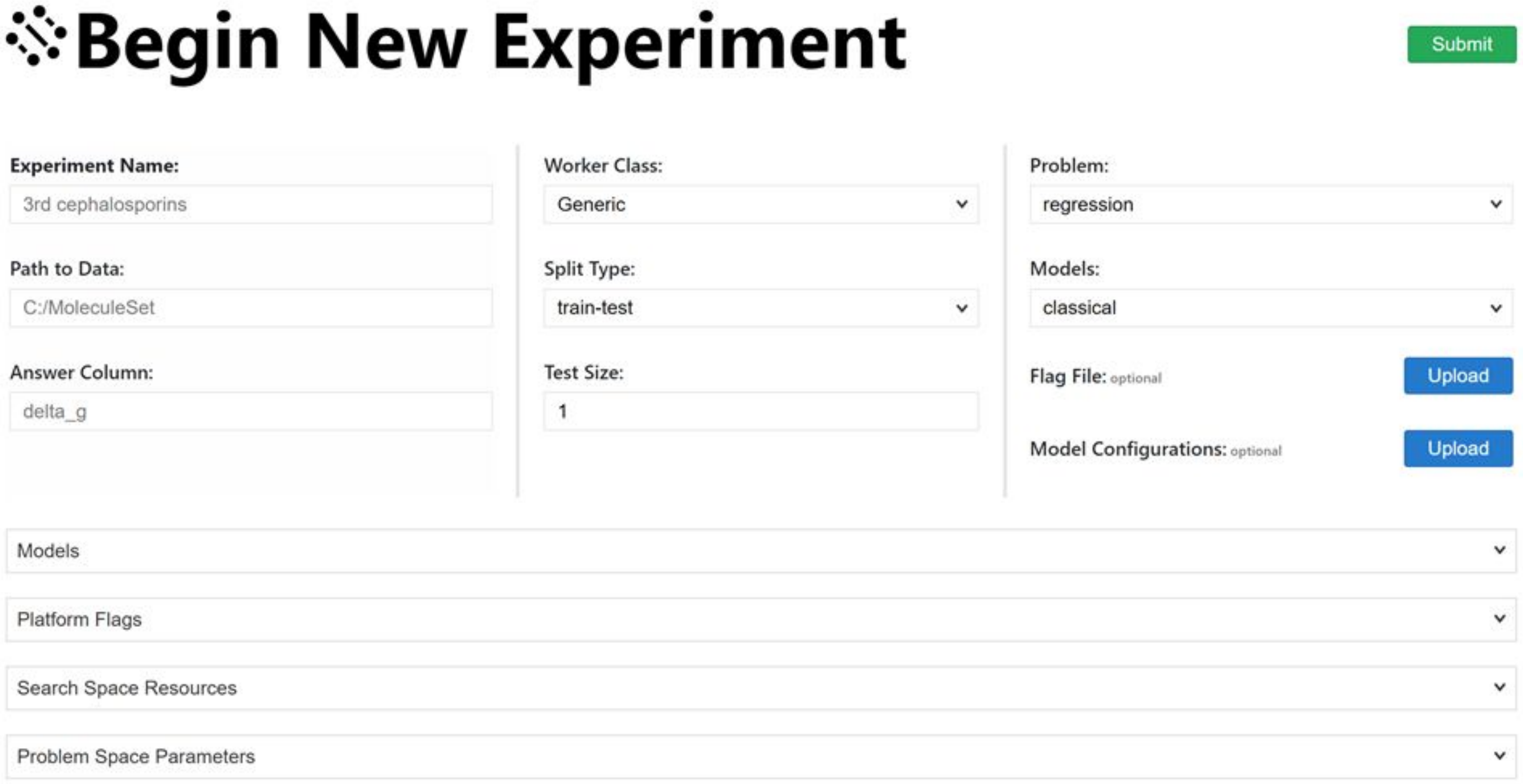
Target Users

- New employees with minimal ML experience
- Employees with some ML experience
- Advanced users with significant ML and DeepCure experience

Solution

Intuitive web app that lets DeepCure employees easily start a new experiment

- Accessible to users regardless of level of ML knowledge, with advanced options available for more advanced users
- Code must adapt to DeepCure's changing needs



Begin New Experiment

Experiment Name: 3rd cephalosporins

Path to Data: C:/MoleculeSet

Answer Column: delta_g

Worker Class: Generic

Split Type: train-test

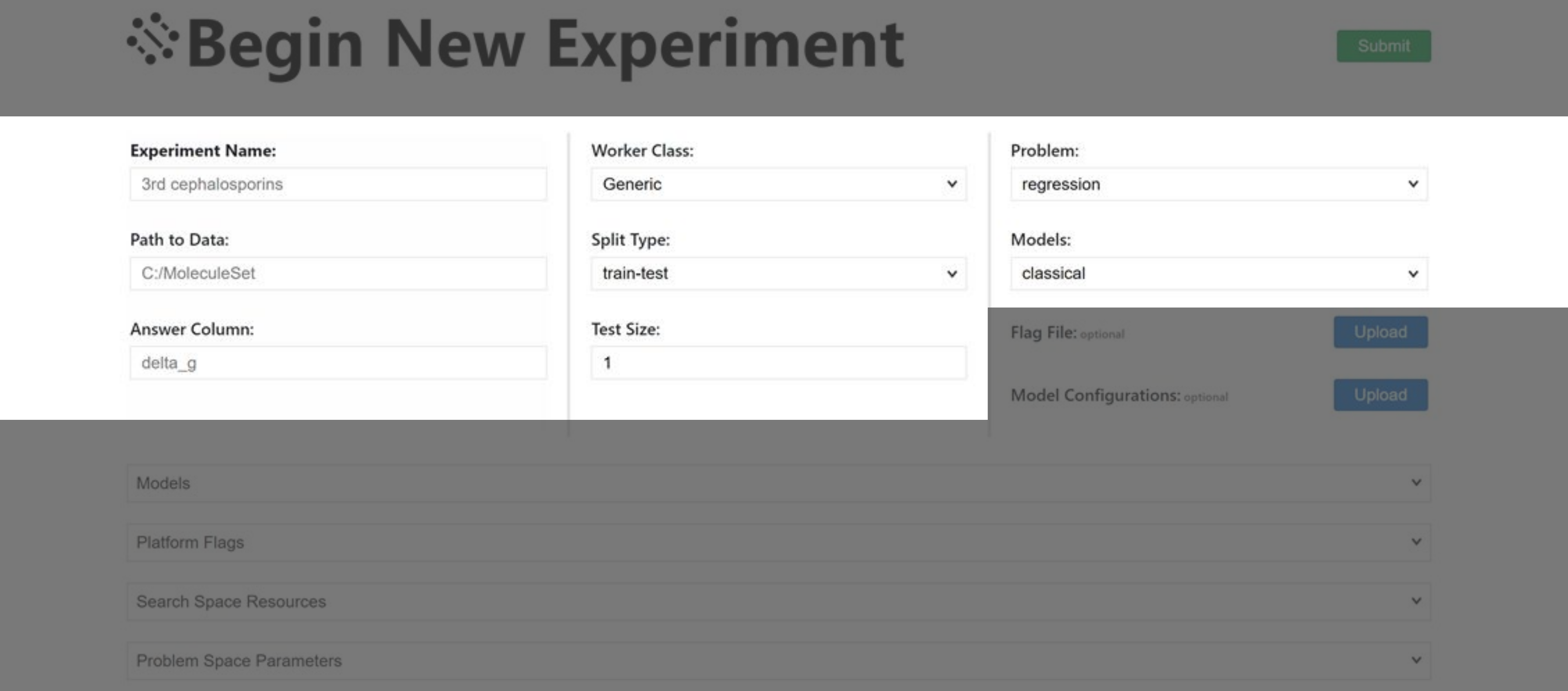
Test Size: 1

Problem: regression

Models: classical

Flag File: optional

Model Configurations: optional



Begin New Experiment

Experiment Name: 3rd cephalosporins

Path to Data: C:/MoleculeSet

Answer Column: delta_g

Worker Class: Generic

Split Type: train-test

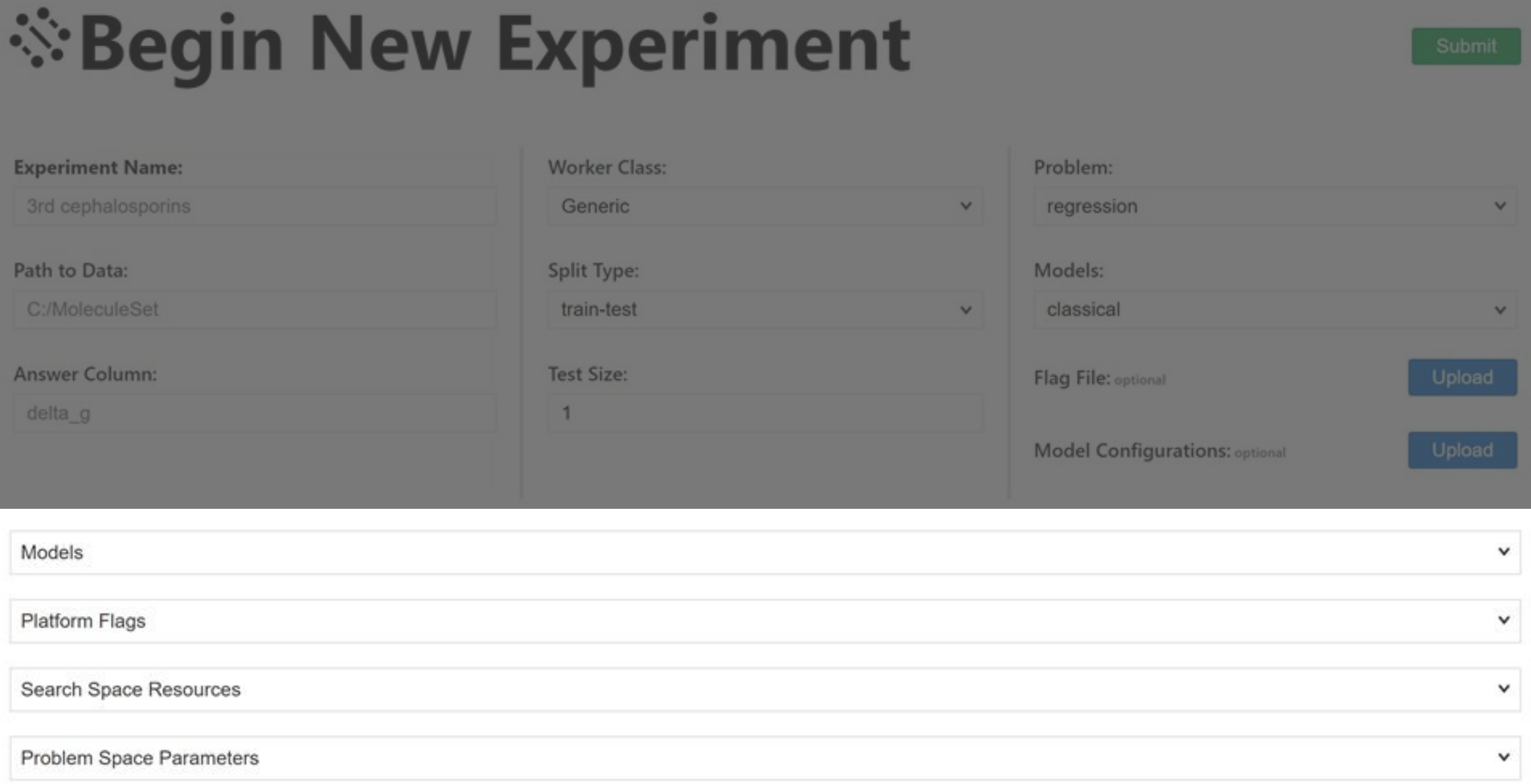
Test Size: 1

Problem: regression

Models: classical

Flag File: optional

Model Configurations: optional



Begin New Experiment

Experiment Name: 3rd cephalosporins

Path to Data: C:/MoleculeSet

Answer Column: delta_g

Worker Class: Generic

Split Type: train-test

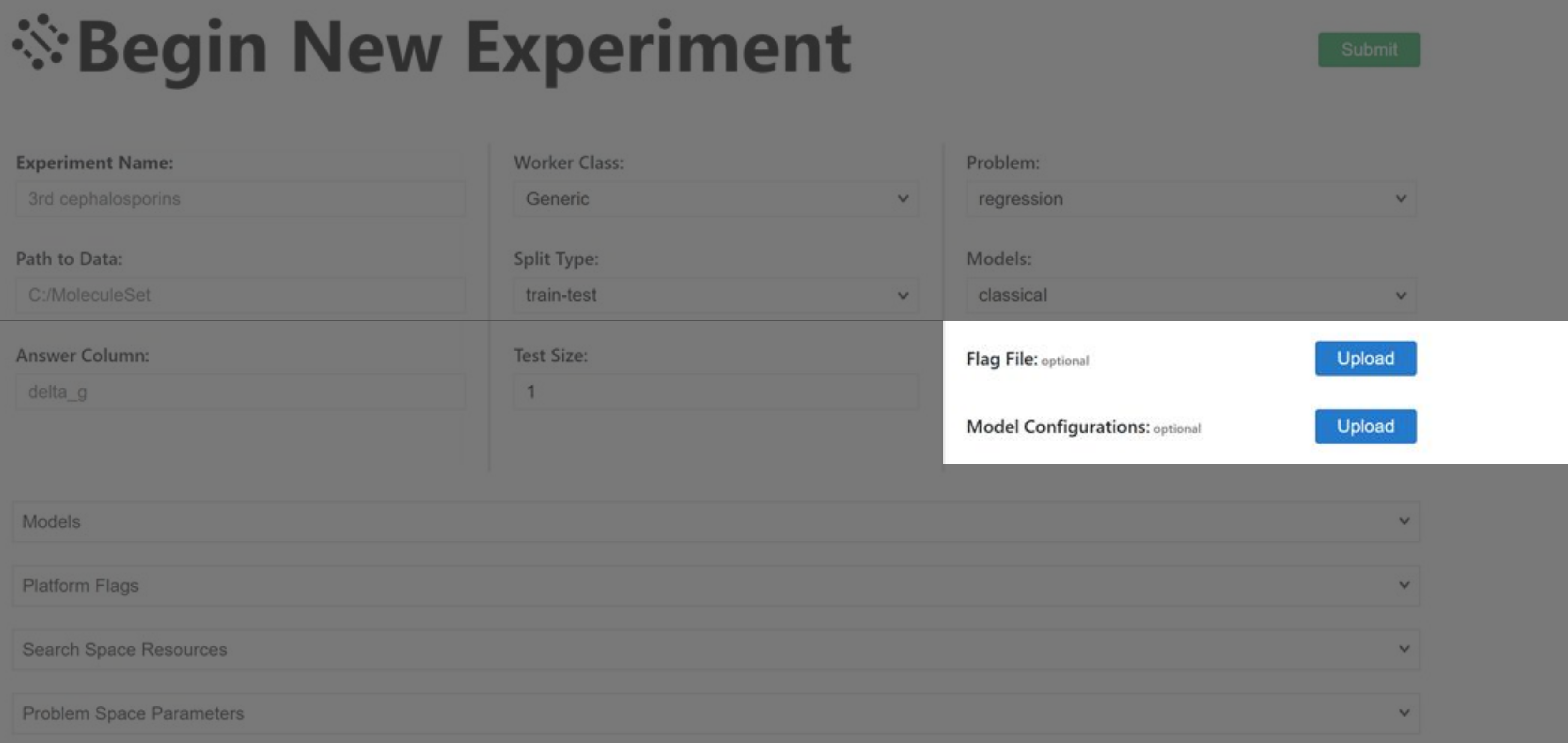
Test Size: 1

Problem: regression

Models: classical

Flag File: optional

Model Configurations: optional



Begin New Experiment

Experiment Name: 3rd cephalosporins

Worker Class: Generic

Problem: regression

Path to Data: C:/MoleculeSet

Split Type: train-test

Models: classical

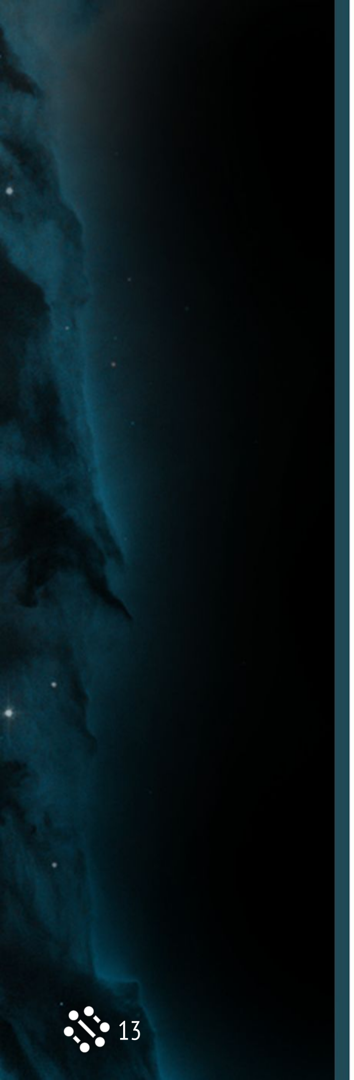
Answer Column: delta_g

Test Size: 1

Flag File: optional

Model Configurations: optional

Demo

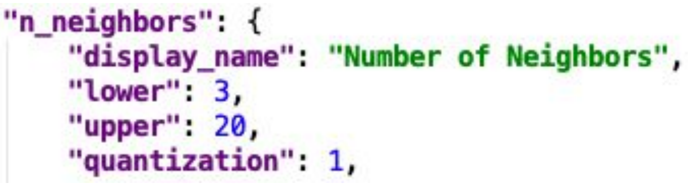


Product Highlights

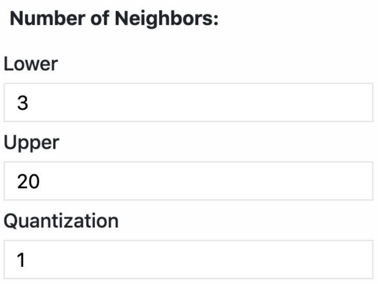
- Adaptive Interface
- Reusable Components
- Dynamically Generated Forms
- Input Validation

Adaptability

Default flags and models are maintained by DeepCure. Changes to these defaults are parsed automatically, ensuring all users start from the same baseline.



```
"n_neighbors": {
    "display_name": "Number of Neighbors",
    "lower": 3,
    "upper": 20,
    "quantization": 1,
```



Number of Neighbors:

Lower

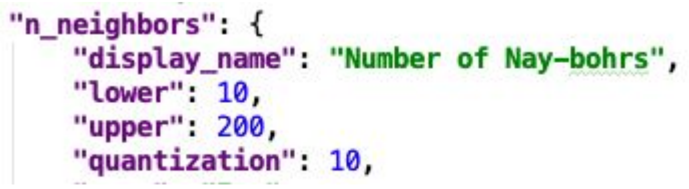
3

Upper

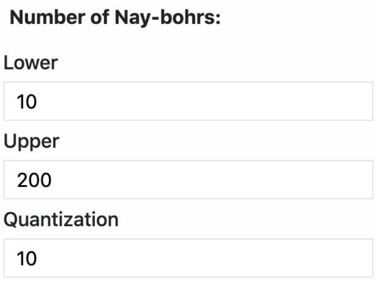
20

Quantization

1



```
"n_neighbors": {
    "display_name": "Number of Nay-bohrs",
    "lower": 10,
    "upper": 200,
    "quantization": 10,
```



Number of Nay-bohrs:

Lower

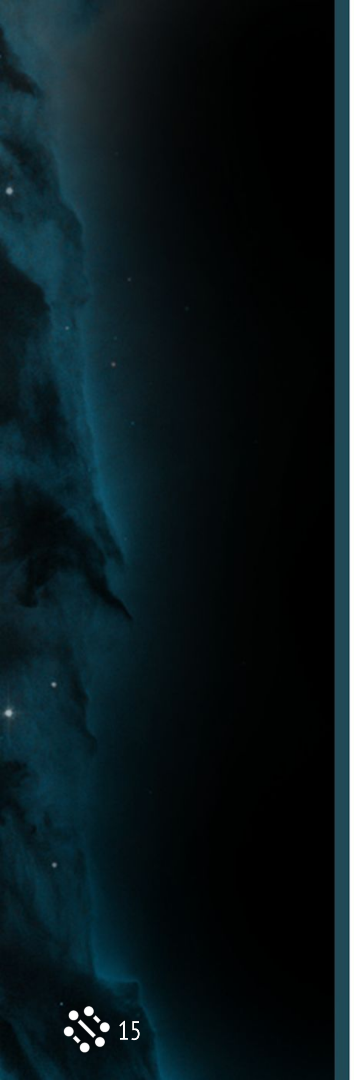
10

Upper

200

Quantization

10

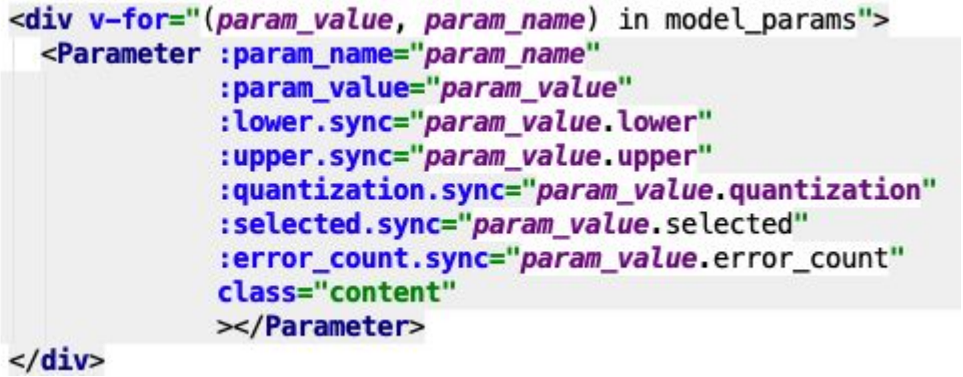


Reusable Components

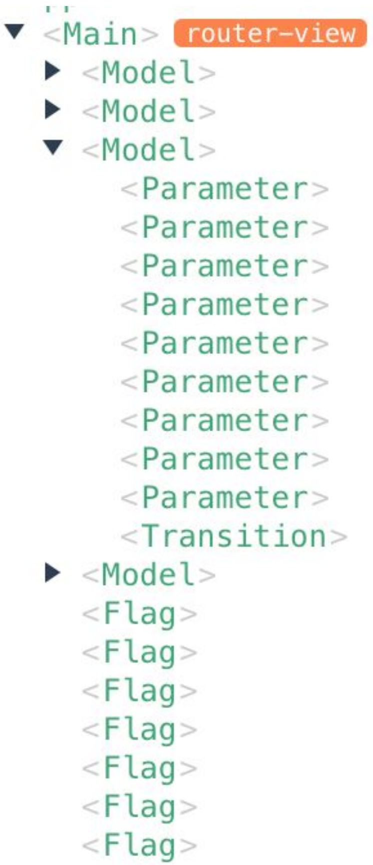
Each Model contains multiple Parameter components, and each Model is a component.

Each Flag is a separate component.

Creating a component is like using an object constructor, allowing the same code to be reused in a for loop.



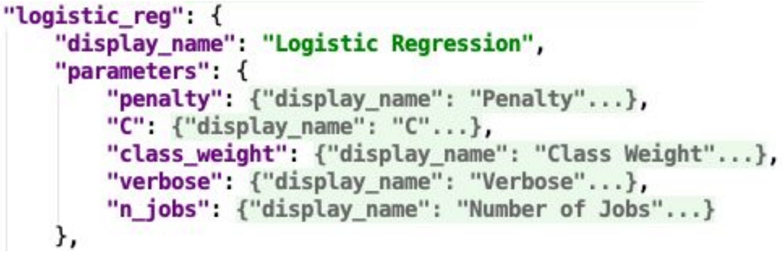
```
<div v-for="(param_value, param_name) in model_params">
  <Parameter :param_name="param_name"
             :param_value="param_value"
             :lower.sync="param_value.lower"
             :upper.sync="param_value.upper"
             :quantization.sync="param_value.quantization"
             :selected.sync="param_value.selected"
             :error_count.sync="param_value.error_count"
             class="content"
             ></Parameter>
</div>
```



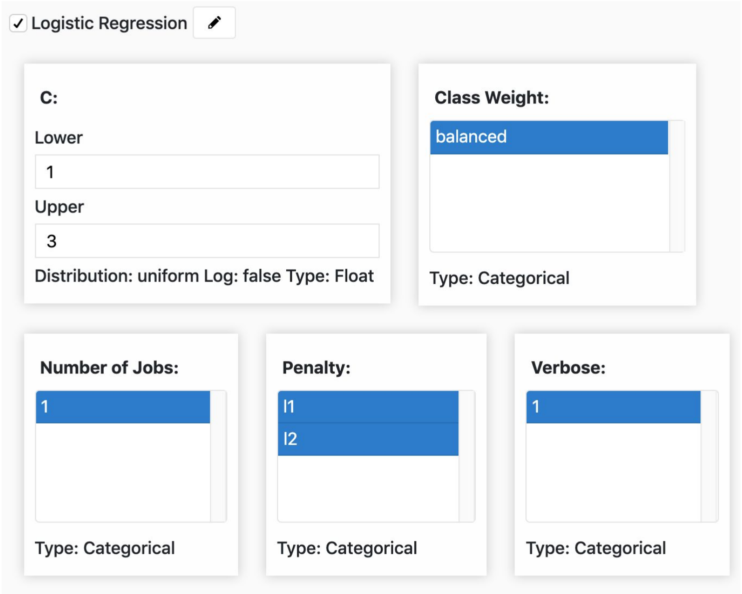
```
▼ <Main> router-view
    ▶ <Model>
    ▶ <Model>
    ▼ <Model>
          <Parameter>
          <Parameter>
          <Parameter>
          <Parameter>
          <Parameter>
          <Parameter>
          <Parameter>
          <Parameter>
          <Parameter>
          <Transition>
    ▶ <Model>
      <Flag>
      <Flag>
      <Flag>
      <Flag>
      <Flag>
      <Flag>
      <Flag>
```

Dynamically Generated Content

Input cards are created based on the object they represent. Card type is dynamically chosen based on input type.



```
"logistic_reg": {
    "display_name": "Logistic Regression",
    "parameters": {
        "penalty": {"display_name": "Penalty"...},
        "C": {"display_name": "C"...},
        "class_weight": {"display_name": "Class Weight"...},
        "verbose": {"display_name": "Verbose"...},
        "n_jobs": {"display_name": "Number of Jobs"...}
    },
```



☑ Logistic Regression

C:

Lower

1

Upper

3

Distribution: uniform Log: false Type: Float

Class Weight:

balanced

Type: Categorical

Number of Jobs:

1

Type: Categorical

Penalty:

l1

l2

Type: Categorical

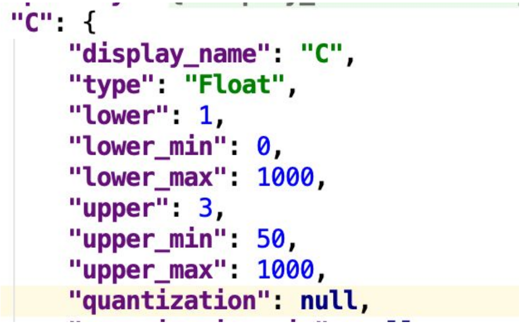
Verbose:

1

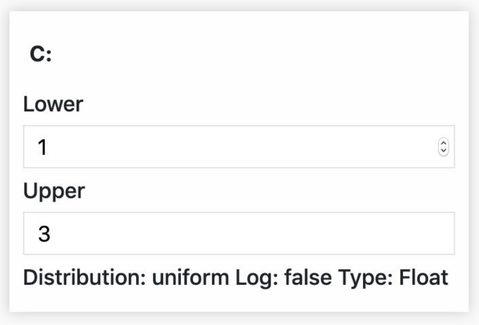
Type: Categorical

Dynamically Generated Content

Input cards are created based on the object they represent. Card type is dynamically chosen based on input type.



```
"C": {
    "display_name": "C",
    "type": "Float",
    "lower": 1,
    "lower_min": 0,
    "lower_max": 1000,
    "upper": 3,
    "upper_min": 50,
    "upper_max": 1000,
    "quantization": null,
```



C:

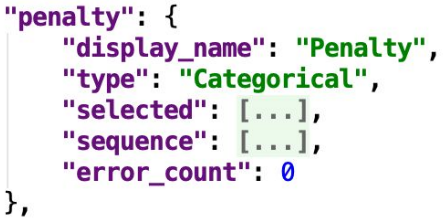
Lower

1

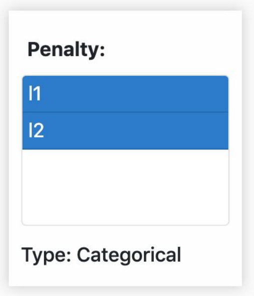
Upper

3

Distribution: uniform Log: false Type: Float



```
"penalty": {
    "display_name": "Penalty",
    "type": "Categorical",
    "selected": [...],
    "sequence": [...],
    "error_count": 0
},
```



Penalty:

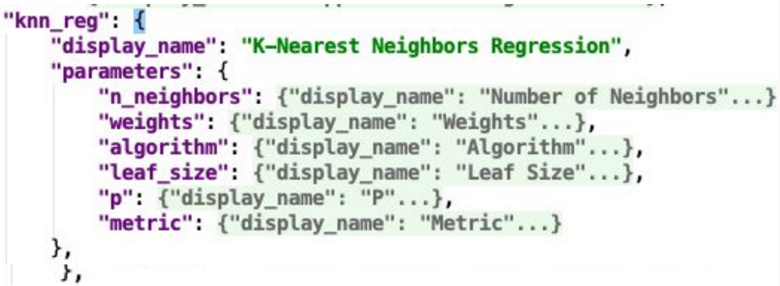
l1

l2

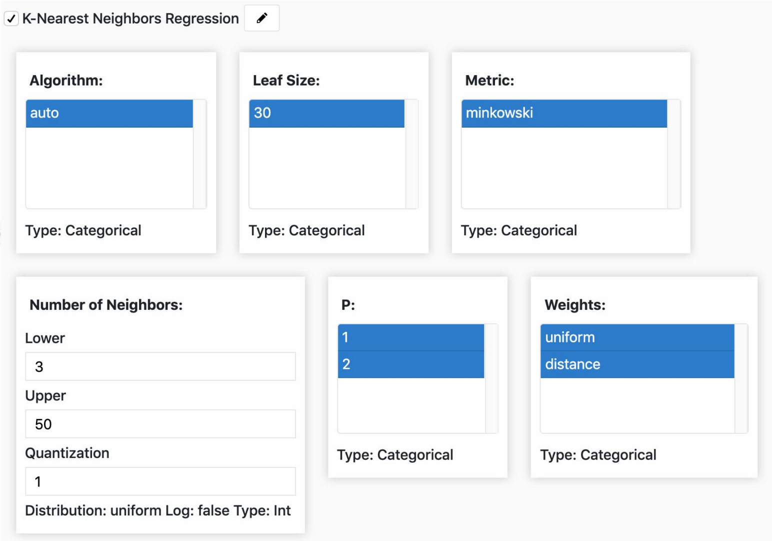
Type: Categorical

Dynamically Generated Content

Input cards are created based on the object they represent. Card type is dynamically chosen based on input type.



```
"knn_reg": {
    "display_name": "K-Nearest Neighbors Regression",
    "parameters": {
        "n_neighbors": {"display_name": "Number of Neighbors"...}
        "weights": {"display_name": "Weights"...},
        "algorithm": {"display_name": "Algorithm"...},
        "leaf_size": {"display_name": "Leaf Size"...},
        "p": {"display_name": "P"...},
        "metric": {"display_name": "Metric"...}
    },
  },
```



☑ K-Nearest Neighbors Regression

Algorithm:
auto
Type: Categorical

Leaf Size:
30
Type: Categorical

Metric:
minkowski
Type: Categorical

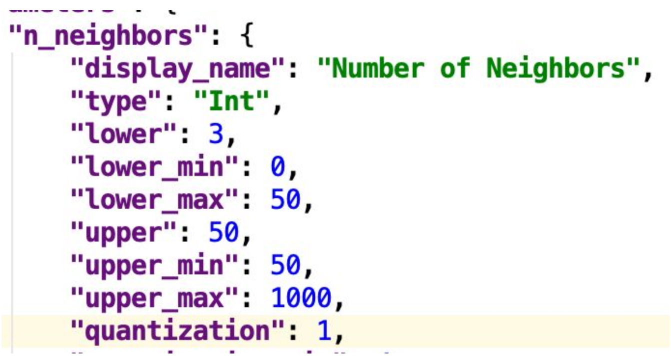
Number of Neighbors:
Lower
3
Upper
50
Quantization
1
Distribution: uniform Log: false Type: Int

P:
1
2
Type: Categorical

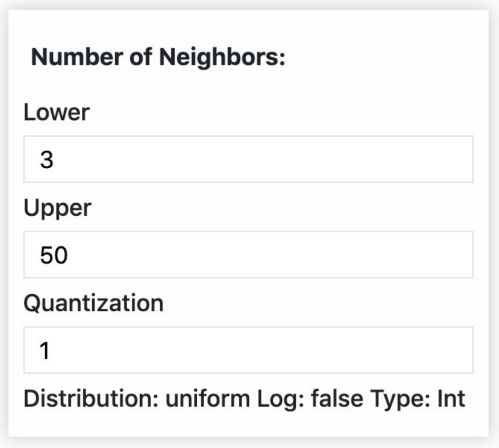
Weights:
uniform
distance
Type: Categorical

Dynamically Generated Content

Input cards are created based on the object they represent. Card type is dynamically chosen based on input type.



```
"n_neighbors": {
    "display_name": "Number of Neighbors",
    "type": "Int",
    "lower": 3,
    "lower_min": 0,
    "lower_max": 50,
    "upper": 50,
    "upper_min": 50,
    "upper_max": 1000,
    "quantization": 1,
```



Number of Neighbors:

Lower

3

Upper

50

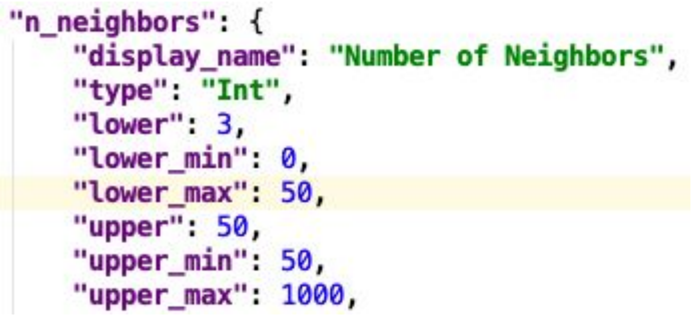
Quantization

1

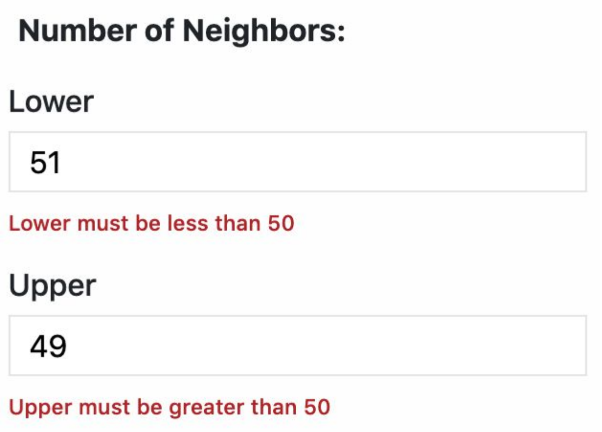
Distribution: uniform Log: false Type: Int

Input Validation

Input validation reflects values contained in the default options file.



```
"n_neighbors": {
    "display_name": "Number of Neighbors",
    "type": "Int",
    "lower": 3,
    "lower_min": 0,
    "lower_max": 50,
    "upper": 50,
    "upper_min": 50,
    "upper_max": 1000,
```



Number of Neighbors:

Lower

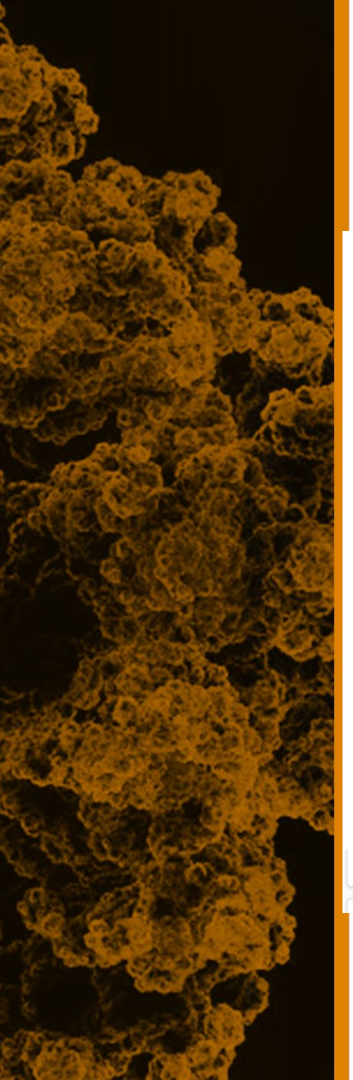
51

Lower must be less than 50

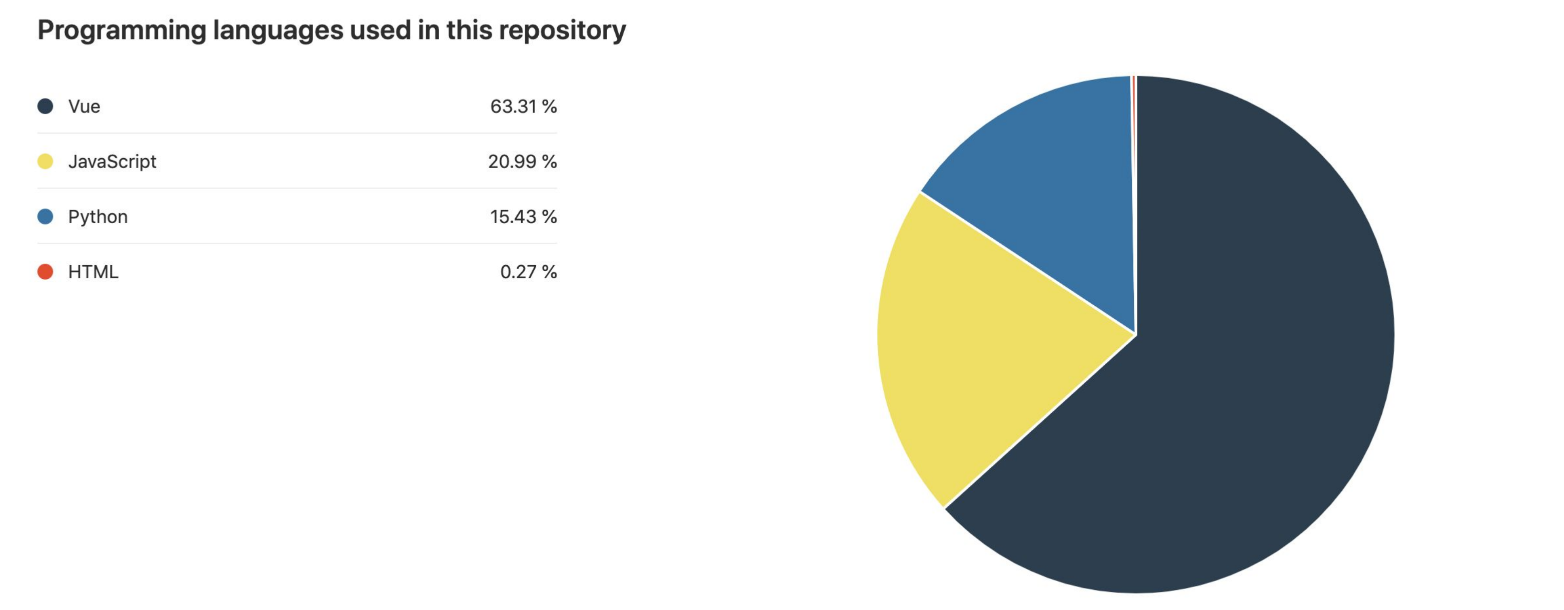
Upper

49

Upper must be greater than 50

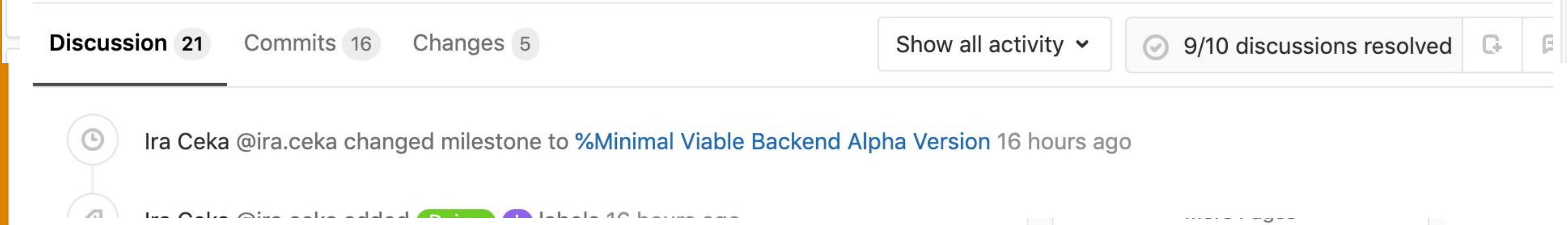


Software and Tools



Programming languages used in this repository

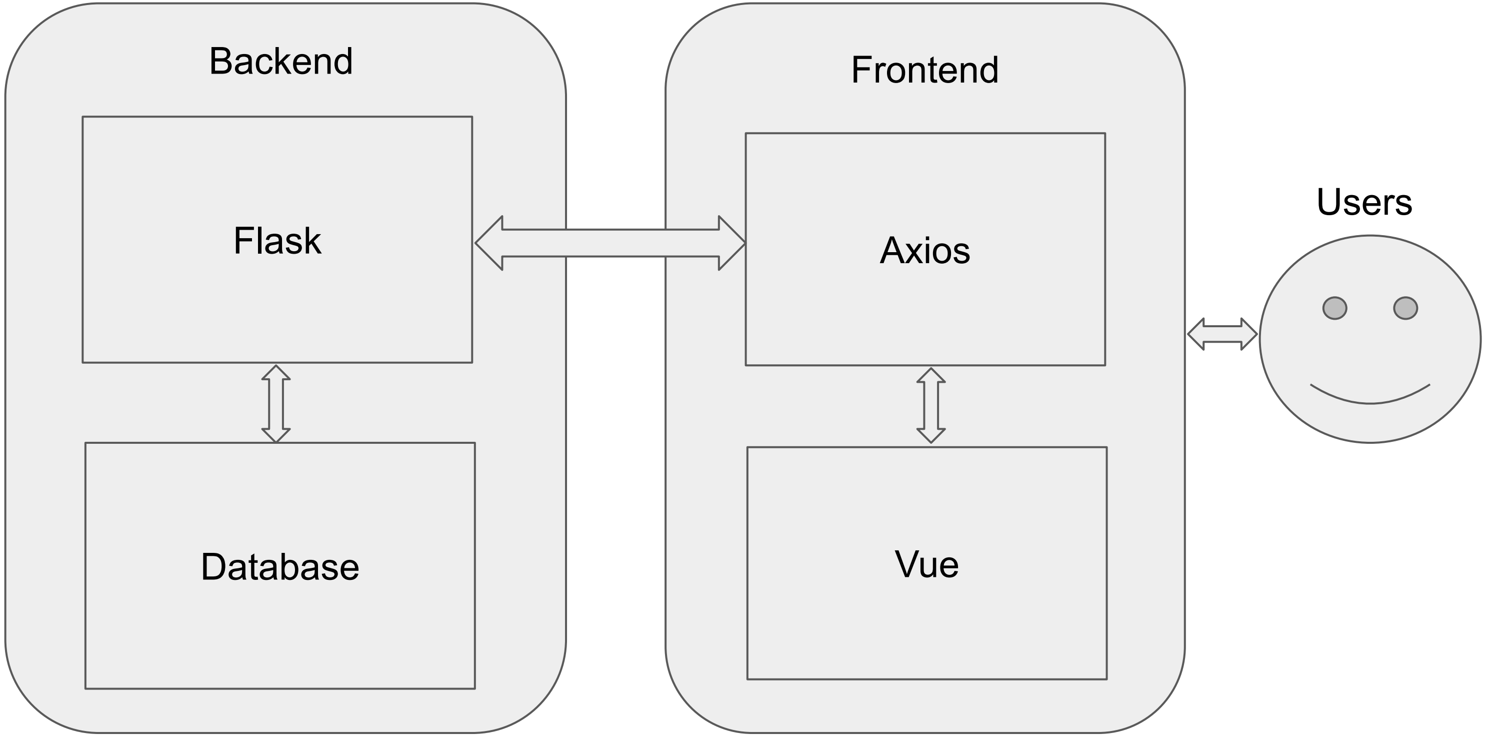
● Vue	63.31 %
● JavaScript	20.99 %
● Python	15.43 %
● HTML	0.27 %



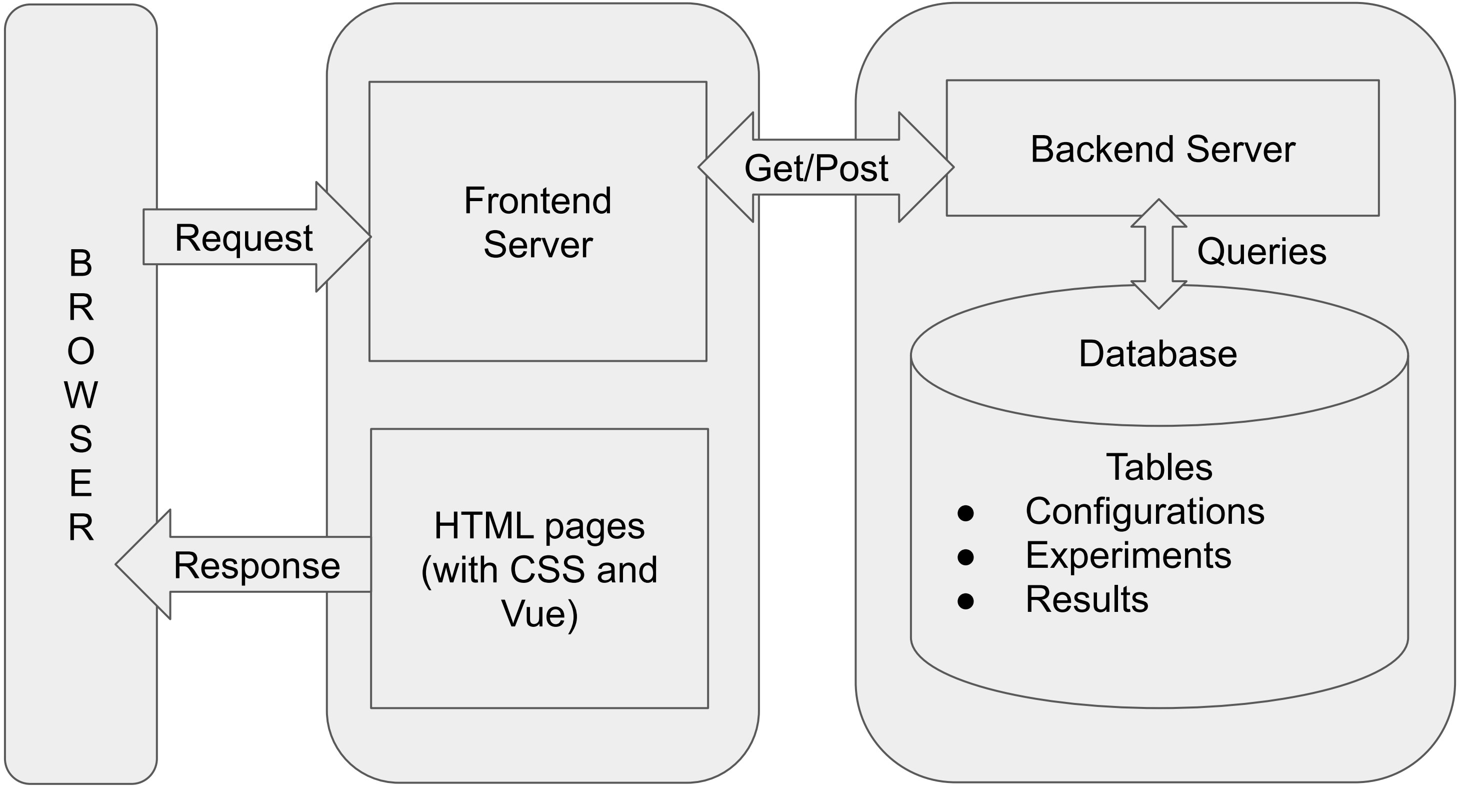
Ira Ceka @ira.ceka changed milestone to %Minimal Viable Backend Alpha Version 16 hours ago

Ira Ceka @ira.ceka added [illegible] [illegible] labels 16 hours ago

System Architecture

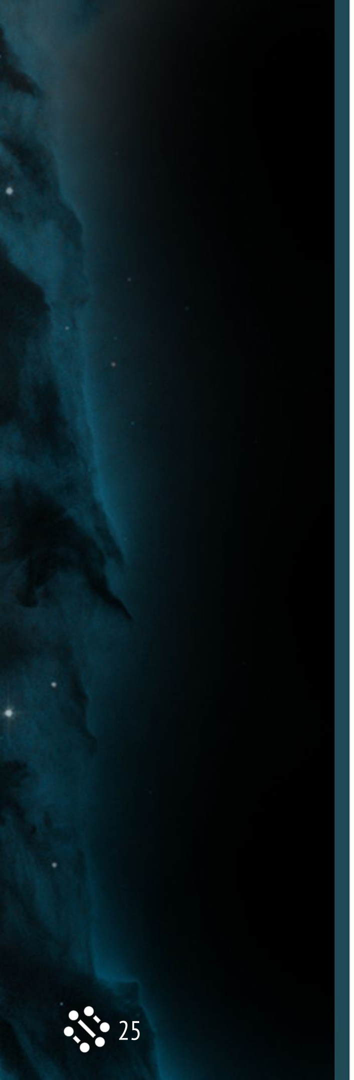


Application Workflow



Next Steps

- **Documentation** for project handoff
- DeepCure **integration**
  - Use real data
  - Submission, their monitoring, analyzing experiments
- **Backlog** for features for future development



What We Learned

- Development
  - Confidence as software engineers, optimality: testable code vs. beautiful prototype (but deadlines!); branch structure
- Organization
  - Regular meetings, solid notes, timeline tags for feature implementations
- Communication
  - Team-client communication: clarification
  - Member-member: getting stuck & falling behind; open and collaborative on implementing issues
- Teamwork
  - Open and collaborative on implementing issues; all the difference passionate and serious about project
- ***Have fun!***

Thank you Thras,
Manu, Alex,

& Professor Pomplun
and the VDC!

Questions?