



UT Compute Final Presentation

Ayaan Mahimwala, David Janosky, Kyle Dotter, Jim Foster, Viraj Dongaonkar



Team Introduction : Kyle Dotter



Front and Back End Hosting: Repository Logistics and Research

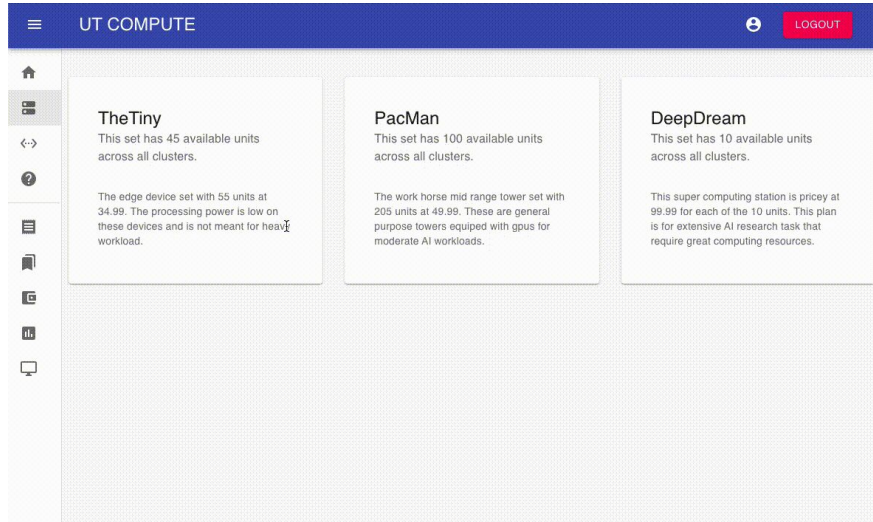


HEROKU

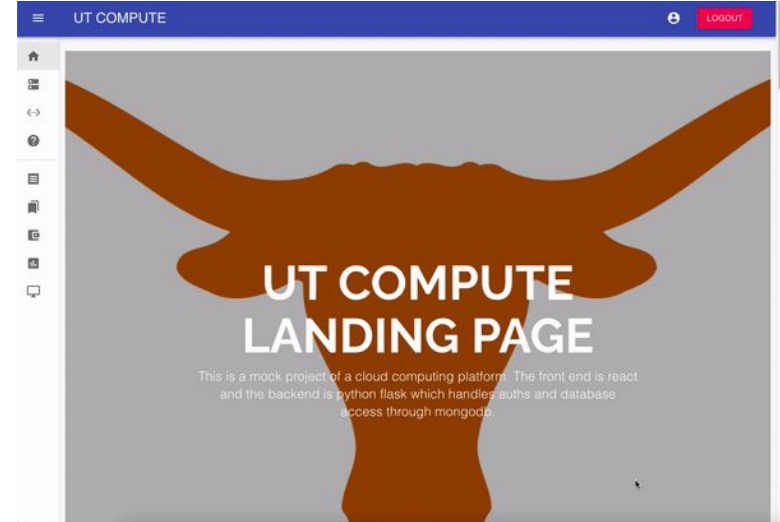


Team Introduction: Ayaan Mahimwala

Load and display datasets from database, with front-end pagination



Complete Interfacing with Hardware API, Create checkout, and ticket cards, check-in form






Team Introduction

Viraj Dongaonkar

- Data access: Dataset backend
- Initial research on web scraping datasets

≡

UT COMPUTE

 [LOGOUT](#)

Dataset Name	Description	File Size	Download Link
MNIST	A set of handwritten digits with labels.	11.1 MB	DOWNLOAD
ImageNet	Images seperated by class with labels.	1.024 GB	DOWNLOAD
CIFAR	The CIFAR-10 dataset consists of 60k ...	161 MB	DOWNLOAD
SNLI	A set of human generate sentence pair...	90.2 MB	DOWNLOAD
Cityscapes	The Cityscapes dataset consists of div...	202.0 MB	DOWNLOAD

1-5 of 7 < >



Team Introduction - David Janosky

- Database accessor class
- Database hosting
- Collection servicers
- API routes
- Security features (cookies, tokens, encryption, hashing, secret keys, ddos protection, rainbow table attack protection)
- Unit testing
- Front and back end app configuration/hosting
- Home page
- Frontend SPA structure
- Login/Signup/Logout features
- Cards (Hardware set API calls)
- Tabs
- Themes



Team Introduction - Jim Foster

Dataset functionality and formatting

UT COMPUTE			
Dataset Name	Description	File Size	Download Link
MNIST	A set of handwritten digits with labels.	11.1 MB	DOWNLOAD
ImageNet	Images separated by class with labels.	1.024 GB	DOWNLOAD
CIFAR	The CIFAR-10 dataset consists of 60k 3...	161 MB	DOWNLOAD
SNLI	A set of human generate sentence pairs...	90.2 MB	DOWNLOAD
Cityscapes	The Cityscapes dataset consists of dive...	202.0 MB	DOWNLOAD

Hardware checkout UI

UT COMPUTE

TheTiny

This set has 45 available units across all clusters.

The edge device set with 55 units at 34.99. The processing power is low on these devices and is not meant for heavy workload.

CHECK OUT

PacMan

This set has 200 available units across all clusters.

The work horse mid range tower set with 205 units at 49.99. These are general purpose towers equipped with gpus for moderate AI workloads.

CHECK OUT

DeepDream

This set has 10 available units across all clusters.

This super computing station is pricey at 99.99 for each of the 10 units. This plan is for extensive AI research task that require great computing resources.

CHECK OUT

Initial research and implementation of mongoDB Atlas for cloud hosting and Heroku for application hosting



Application Introduction

Our app follows the theme of a cloud computing AI platform like a combination of google colab and kaggle

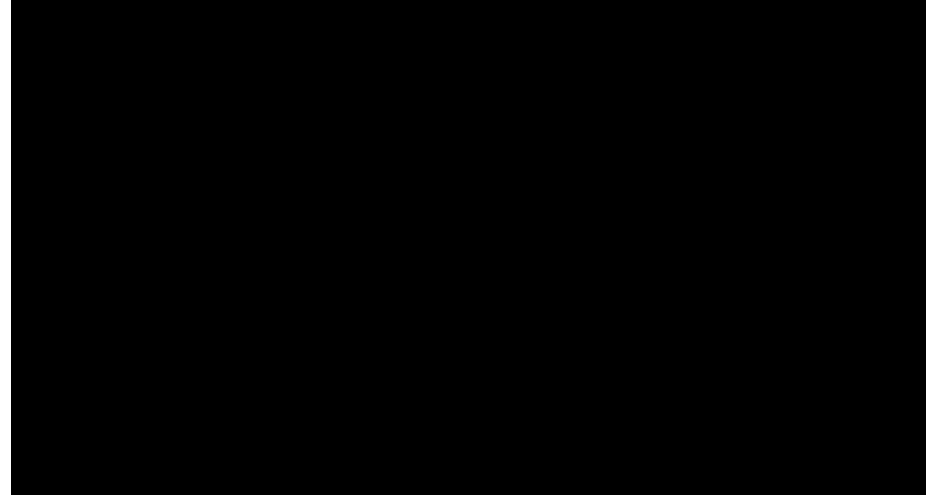
A user would rent some level of resource and load some datasets for their applications

The app would have consoles and dashboards to monitor the app if time had permitted and the user would run their AI tasks such as training, testing and inference



Demonstration: Example of Existing User Flow

- Username and password verification and encryption.
- Users receive hardware set tickets for checked-out hardware set units.
- Hardware set tickets can be updated by the user, subject to availability.
- Users can view and download datasets from the Datasets Tab.





Demonstration: Example of New User Flow

- The user signs up
- The user explores the site
- The user looks through the available datasets
- The user checks out hardware sets



Project Design: Benefits & Drawbacks

- Backend code focuses on testability and scalability while having everything in modular container classes. This is more overhead coding but makes our app a sustainable platform.
- Documentation throughout code base is lacking due to time constraints. Time management is a recurring issue.
- The frontend is a SPA which helps with loading times and content rendering but removes any sort of routing techniques. This style has gained traction with apps over traditional page navigation projects like wikipedia.
- The frontend uses material ui and bootstrap which have a learning curve but a more polished final product.
- Cards components are recycled throughout the code which makes it faster and easier to display new content without making any changes to the code



Modularization: Benefits & Drawbacks

- The database is implemented with abstraction and configurable design meaning you could change hosts or even change platforms like mongodb to sql easily. This is a more testable structure as unit tests on any function can be done and issues can be isolated through the calling structure. This is more setup cost as mentioned.
- SPA structure allows for a plug and play aspect of any content we need while having an easy UI for navigation. Common tasks like login/logout are carried over with no replication of code. This means developers can focus on one thing at a time. Understanding this newer scheme is a little more difficult.
 - Lots of components on the front end are modularized and dynamically repeated, such as the card components.
- Pages can be built independently of other pages



Scalability: Look & Feel to Database

Backend:

- A more robust database can be migrated to easily.
- Security features can scale over time to the application needs (blowfish algorithm)
- Pagination means the amount of data served is in chunks, accounting for growth in the database.
- An admin portal can be made and all api routes are in place for easy admin feature implementation.
- Resource numbers can be changed and more or less sets are easily brought online. This includes a way to add clusters to hardware sets.
- Filtering on usernames is implemented and there is room for any other filter tasks such as date created or dataset size for examples.

Frontend:

- Most of the home page content is specified via a json file that contains information about features, what we learned, the team, etc. This can be easily changed and the UI will reflect any changes automatically.
- Hardware sets and datasets can be changed on the backend and the UI will reflect any changes



Scalability: Future Features and Billing



- Tickets are time stamped with scheduled billing in mind. Sets for database collections for payment info (as well as security features) are easy to set up, or api to paypal could be implemented.
- Users can opt to upload datasets marked as private and can bookmark datasets through the UI has not been implemented for this.
 - A potential implementation for this is recycling the existing dataset display component and simply populating it with a user's datasets from the backend
- A console tab and dashboard tab where users can see running information about their jobs or manage the execute. This was laid out but not implemented.



Testing

- Over 1000 lines of test cases were written for the backend including over 40 test cases.
 - The tests covered login management api and dataset api.
 - The database was mocked up using a local data structure that implements all of the mongodb calls as well as the connection itself being tested.
- Logging for any bugs are printed for easy debugging through the app log on heroku. A mockup of the app client itself was made for local testing. Responses include defaults as well as error messages for graceful handling of faults.
- The hardware sets being simpler were tested on the frontend through the UI.
- The frontend is validated through user testing and with extensive logging.



Team Self-Critique

What we could do better:

Time management

- Task completion
- Deadlines
- Future features

Communication

Task distribution

Learning new tools

Code refactoring

What we did well:

Repo management

Documentation of intent

Modularity

Structuring of code

Testing and Debug

What we learned:

Set goals early and address them throughout

Hosting takes some configuration to make work

Learned react, material ui, mongo and flask

Keep repos clean and organized