Batalog

Project Team: Hadi Soufi, Thien Le, Kevin Keomalaythong, David Massey, Rahim Iqbal

TABLE OF CONTENTS

1. **Project Definition** – *Thien L., Hadi S.*

   Bat scientists at UNCG record bat calls from all over the state. Doing research on these calls can require years of manual work. Batalog seeks to do that automatically in a web interface using deep learning techniques to catalog bat calls. We aim to catalog bat calls according to their purpose, specifically whether they are echolocation calls or abnormal calls. The calls are stored in Zero Crossing format; the data will have to be cleaned up as it contains a significant amount of noise. Once the data is cleaned, the bat calls will be clustered according to their shapes and then catalogued for future scientific research. We would like to display the bat calls on a website and allow the user to interact with it. As a stretch goal, we will also be able to predict the nature of the calls based on metadata such as the time, location, and season that the calls were recorded in.

2. Project Requirements

a. Functional

- i. At least 90% accuracy (*Hadi*)
  1. A convolutional neural network will be used to classify the type of bat call that is uploaded. The CNN should achieve a minimum of 90% accuracy during the training and testing phase prior to uploading any files.
  2. The model must be evaluated to remedy overfitting. This can be done by checking training accuracy versus testing accuracy.
- ii. Interactive user interface (*Thien*)
  1. Display the data in a meaningful way: Allow the user to be able to see the sorted bat data in a collection of album. The user will be able to favorite and mark the data.
  2. Allow the user to upload zero-crossing files for analysis: This function will give a front end design for the user to upload the file. While the zero-crossing files being processed in the background the user will get greet with a loading screen.
  3. Allow the user to login and create an account

b. Usability

- i. User interface (*Thien*)
  1. Web-based
- ii. Performance (*Kevin*)
  1. The speed the website can process user data and output in a meaningful way. However this will be a low priority because we want to focus on accuracy first.

c. System

- i. Hardware
  1. Any Laptop and Desktop PC capable of running modern Operating System such as Mac OS X, Windows 10
  2. Stable internet connection
- ii. Software
  1. Any modern browser (e.g. Chrome, Firefox, Opera, Safari).
- iii. Database

1. The system shall allow uploaded images to be sent into the database, and facilitate communication between database and web interface
2. The database shall store PNG images in binary-large-object format and have a storage capacity of at least 50 GB

d. Security

i. Username/password

1. Users will need to provide a username and password in order to login. More communication is needed with the Biology Dept if additional specifications are required.(*Hadi*)

3. Project Specification

a. Areas
   i. Machine Learning (*Hadi*)
   ii. Web Development (*Thien*)
   iii. Database Management (*Rahim*)

b. Libraries
   i. TensorFlow + Keras API (*David*) - Python machine learning library/interface set that supports deep learning, i.e. facilitates building (convolutional) neural networks.
   ii. Numpy (*David, Hadi*) - Python library that introduces n-dimensional arrays including matrices, and facilitates linear algebra operations such as matrix multiplication.
   iii. Matplotlib (*Hadi*) - Python 2D plotting library that allows visualization and statistical analysis of data.
   iv. Dash (*Rahim*) - Dash is a framework based on Python. It is used to build an analytical web application. Primarily it builds a dashboard using Python.
   v. Pillow 5 (*Thien*) - A powerful image library for Python web development that will help with organizing the image libraries.
   vi. Django 2 (*Thien, Rahim, Hadi*) - Django is a free and open source framework based on Python. Django has less freedom but it is more efficient and secure than other frameworks such as asp.net or visual studio.
   vii. SQLite3 (*Kevin*)-

c. Framework
   i. Django 2 (*Thien, Rahim*)
   ii. SQLite3 (*Thien*) - for database and the interaction between the website and the database storage.
   iii. Python 3 (*Thien, Rahim*)

d. Development Environment
   i. Jupyter Notebook (*Hadi*) - Application used to write and execute Python code and visualize output.
   ii. Atom (*Thien*) - Text editor with built-in terminal that helps execute the codes.

iii. Sublime Text (*Rahim, Hadi*) - Text editor that helps to read/write Python and django code.
iv. Pycharm(*Kevin*)-

e. Platform: Web

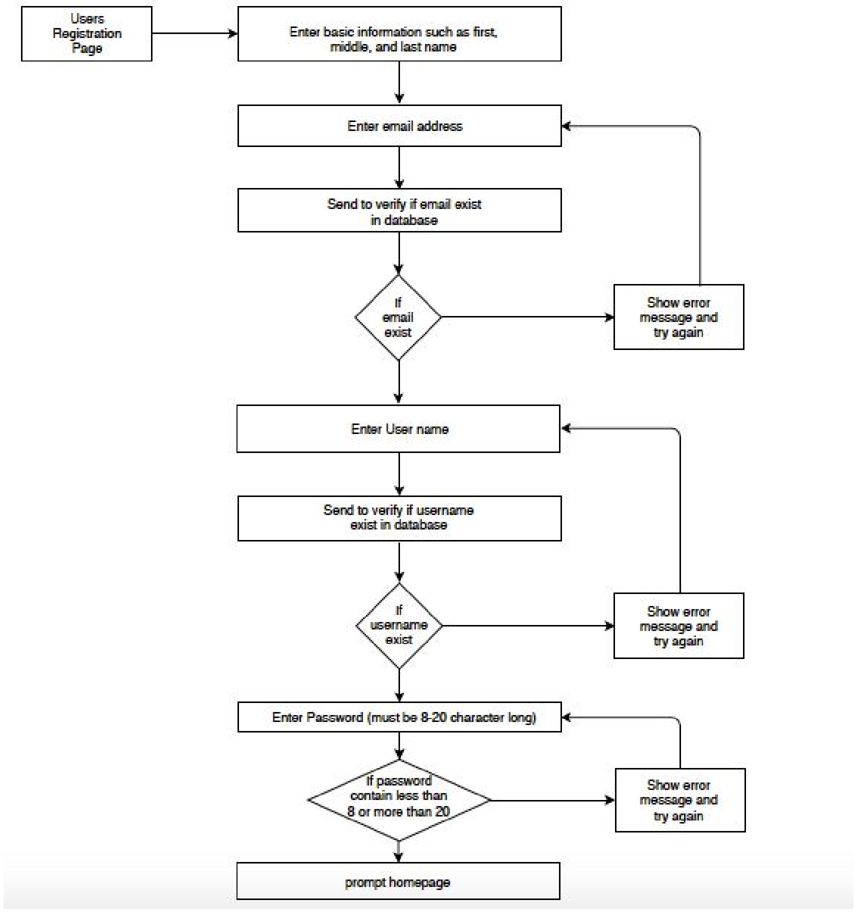
f. Genre: Research tool

4. System – Design Perspective

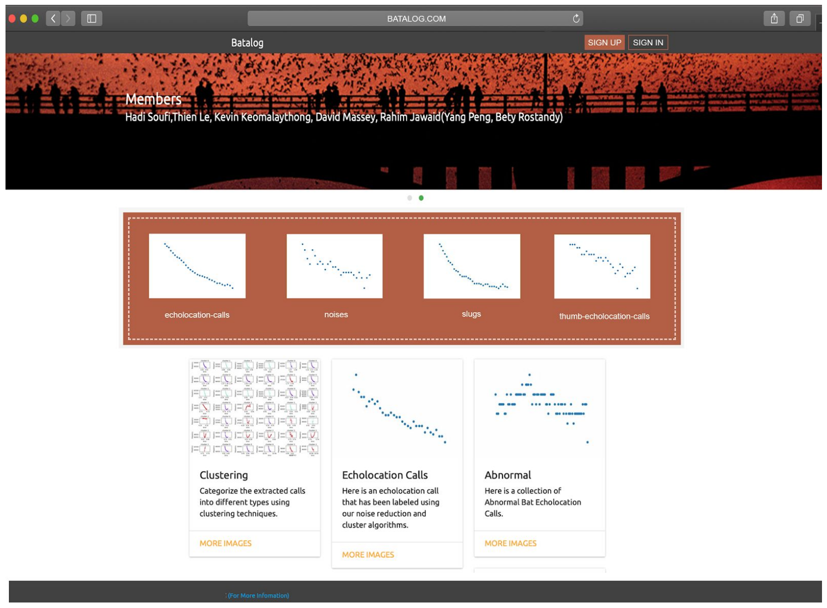
a. Subsystems

i. Front-end

1. User registration page (Rahim)


Users Registration Page
Enter basic information such as first, middle, and last name
Enter email address
Send to verify if email exist in database
If email exist
Show error message and try again
Enter User name
Send to verify if username exist in database
If username exist
Show error message and try again
Enter Password (must be 8-20 character long)
If password contain less than 8 or more than 20
Show error message and try again
prompt homepage

2. Display Results (Thien)



Members
Hadi Soufi,Thien Le, Kevin Keomalaythong, David Massey, Rahim Jawaid(Yang Peng, Bety Rostandy)

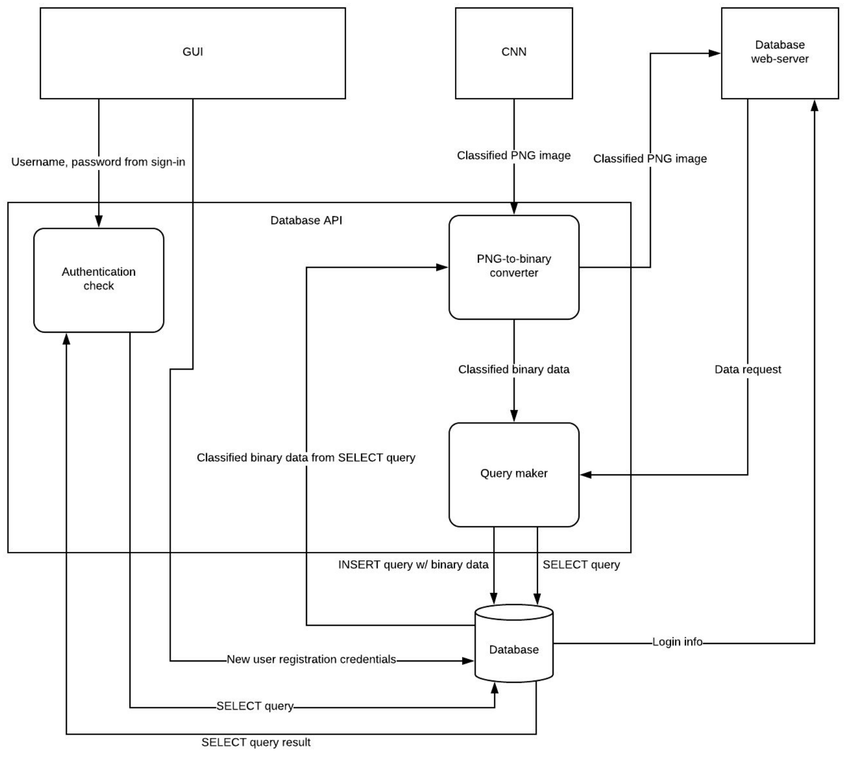
Clustering
Categorize the extracted calls into different types using clustering techniques.

Echolocation Calls
Here is an echolocation call that has been labeled using our noise reduction and cluster algorithms.

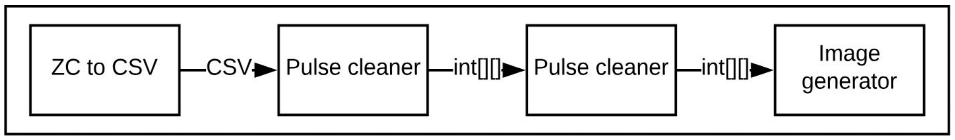
Abnormal
Here is a collection of Abnormal Bat Echolocation Calls.

ii. Back-end

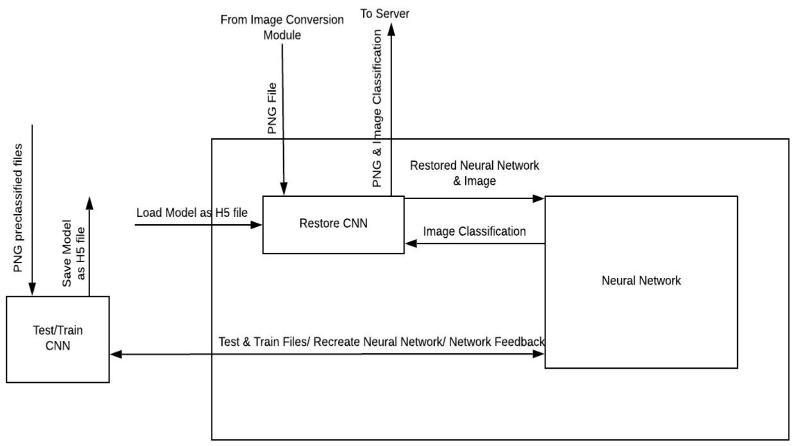
1. Connection from web to Database (*Thien*)
    a. The main connection will be established from a Digital Ocean droplet server to the client browser
2. Database API (*Kevin*)



3. Image conversion module (Hadi)

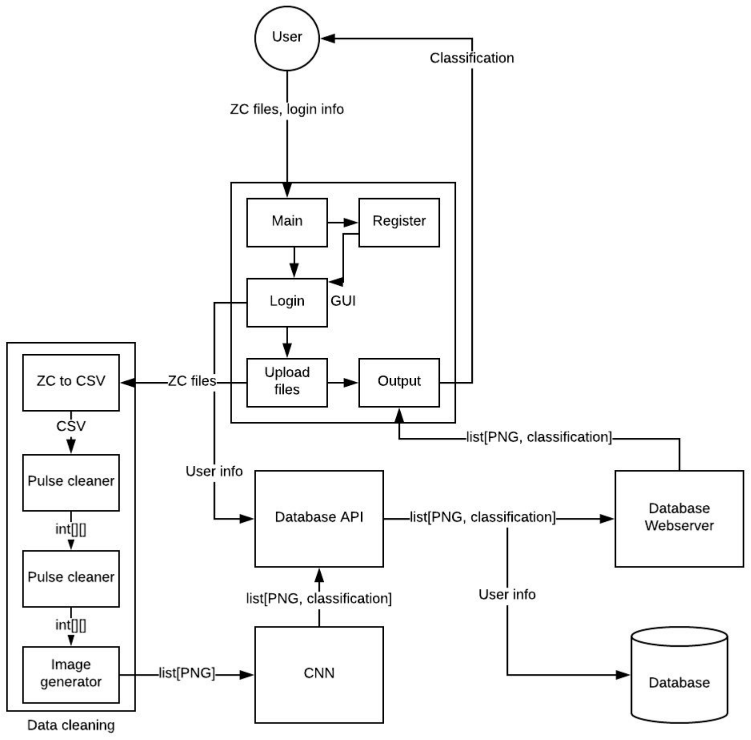


4. CNN (*David)*

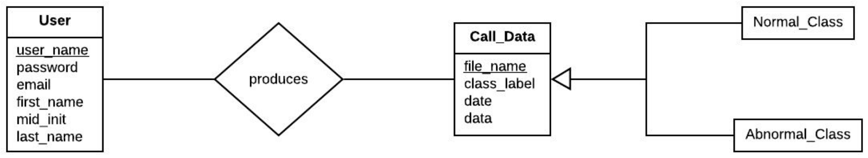

From Image Conversion Module
To Server
PNG File
PNG & Image Classification
PNG preclassified files
Save Model as H5 file
Load Model as H5 file
Restore CNN
Restored Neural Network & Image
Image Classification
Neural Network
Test/Train CNN
Test & Train Files/ Recreate Neural Network/ Network Feedback

b. Sub-System Communication

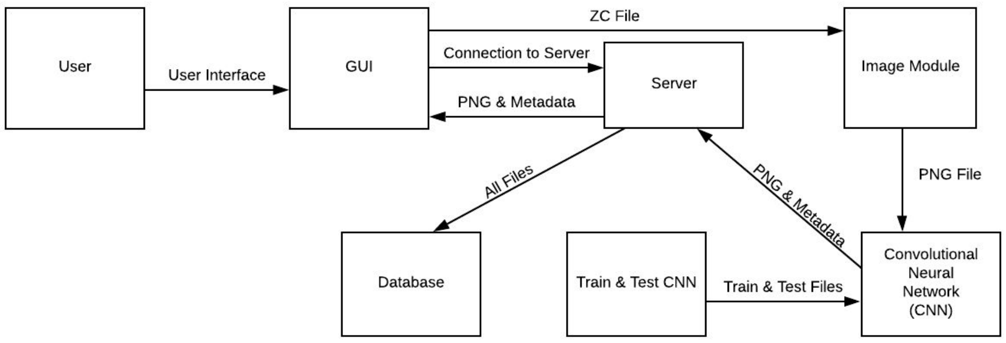
i. Controls (*Thien*), I/O (*Hadi*), DataFlow (*Kevin*):



ii. Entity Relationship (E-R) Model (*Kevin*)



c. Overall operation - System Model (*David*)



5. System – Analysis Perspective

a. Subsystems

- i. Front-end
  1. User registration page (*Rahim*) - Allows users to register on the website, so they can upload their ZC file. The users enter their basic information such as first, middle and last names. The register page also verifies the user email and username from the database. If username or email is present in the database, then it will display the error message and ask it to try again.
  2. Display Results (*Thien*) - Allows user to upload and see the file and classification.
     - a. If the user is logged in, they can upload a zc file or set of zc files.
     - b. While the file is being processed, the user will be greeted by a loading screen.
     - c. Once the files have been processed, they will get displayed in separate albums.
- ii. Back-end
  1. Connection from web to Database (*Thien*)
     - a. Javascript Objects
       - i. jQuery - a library that helps navigate through documents, create animations and handle events easier.
       - ii. Photoswipe - an image gallery for web development.
     - b. CSS stylesheets
       - i. Bootstrap - responsive grid system, extensive pre built components
     - c. Digital Ocean Droplet Server
       - i. A scalable compute platform with add-on storage, security, and monitoring capabilities to easily run production applications.
     - d. Python Django Framework
       - i. ORM - can be used to interact with application data from various relational databases such as SQLite, PostgreSQL and MySQL.

e. Dependencies
   i. Pillows - Python Imaging Library.

2. SqLite3 Database
   a. An embedded database, server-less and can run within our app.

iii. Database API - Handles GUI-database interactions and passes SQL queries to the database. The database stores PNG+metadata and user registration information.

iv. Image conversion module - Converts ZC files to PNG images. The images will then be fed to the CNN module for classification.

v. CNN (convolutional neural network) module - Classifies bat pulses as normal or abnormal. Pulses are precleaned.

b. System (Tables and Description)

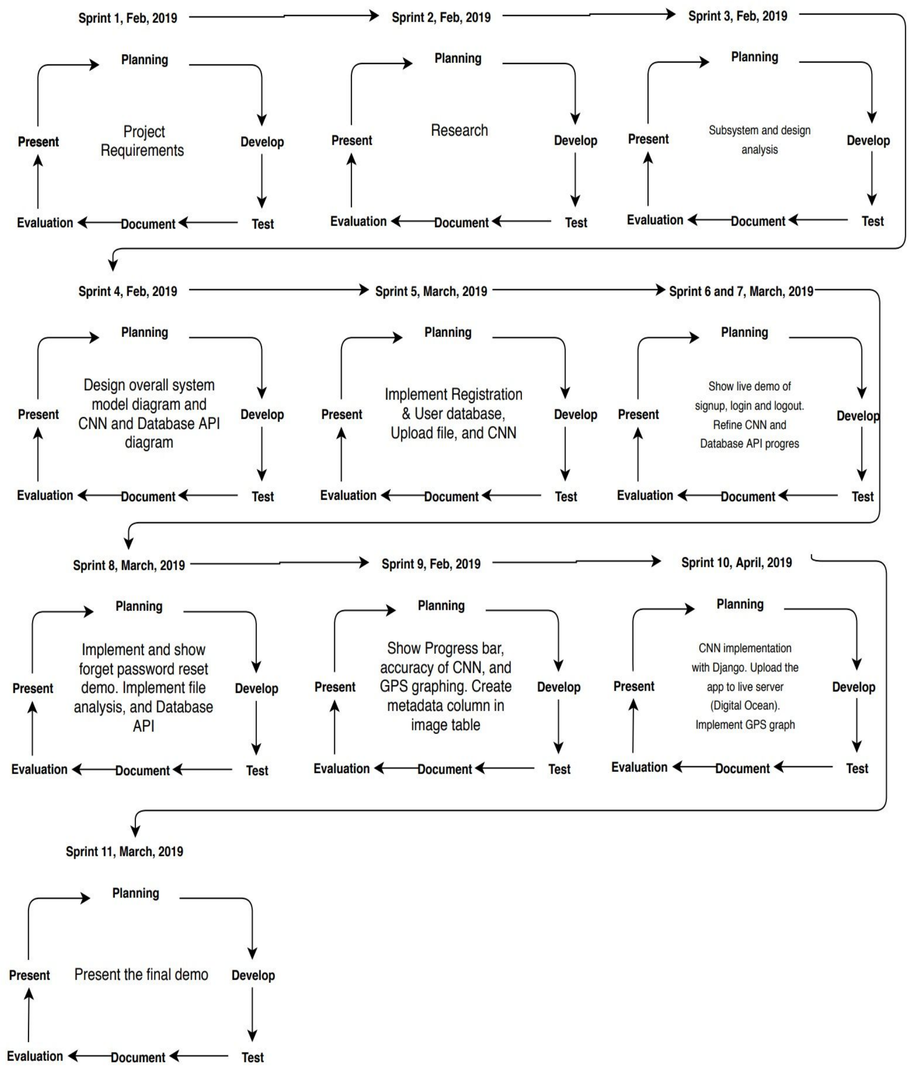
i. Data analysis
   1. Data dictionary (*Rahim*, *Kevin, David*)
      a. Table: User
         i. Data type: text (VARCHAR)
         ii. Description: This table will contain the user registration information such as user name, first name, middle name, last name, email, and password in the database. When the user enters their login information, the web application searches this table for the username and password and determines whether their credentials exist in this table. If so, then the user will be able to login and upload their data.
      b. Table: Call_Data
         i. Data types:
            1. file_name: text (VARCHAR)
            2. class_label: text (VARCHAR)
            3. date: date-time
            4. data: binary
         ii. Description: This table will contain a list of PNG images that represent the processed zero-crossing data. In SQL, the image data will be converted to binary large objects or BLOBs before being stored in the database. Each data entry will have an associated classification,

either “normal” or “abnormal”, and the date and time it was collected.

c. Variable: Model
   i. Data type: Sequential object
   ii. Description: This variable will represent the CNN model used for training and predictions. The CNN is loaded and assigned to this variable prior to image classifications. It will include the network architecture, weights, and settings.

ii. Process models (*Rahim*)

1. Agile Process



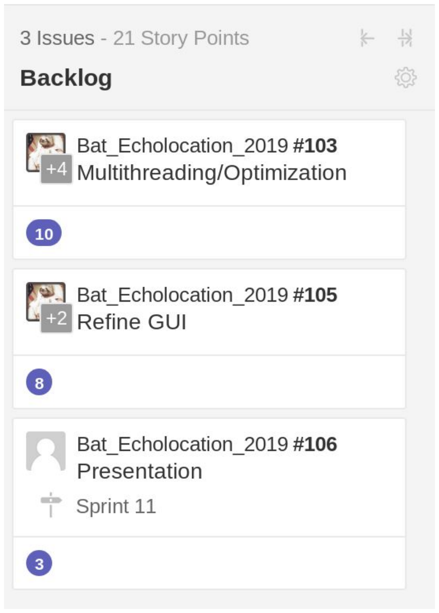
c. Algorithm Analysis

i. Entire system - $O(n)$ is the worst case of all subsystems.

ii. CNN - $O(1)$: All images conform to the same pixel size. They will be processed one at a time. The network layout remains unchanged between images.
iii. Registration page - $O(1)$: All the users follow the same steps for registering on the website
iv. Zero-crossing file analysis - $O(n)$: Almost entirely array operations. An increase in file size corresponds to a linear increase in runtime.
v. Database API - $O(log\ n)$: The operations that the API pass to the DBMS are insert and search (SELECT query in SQL). Insertion is $O(1)$. Search is $O(log\ n)$ since each table will be indexed through B-trees.

6. Project Scrum Report

a. Product Backlog (Table / Diagram)



3 Issues - 21 Story Points

Backlog

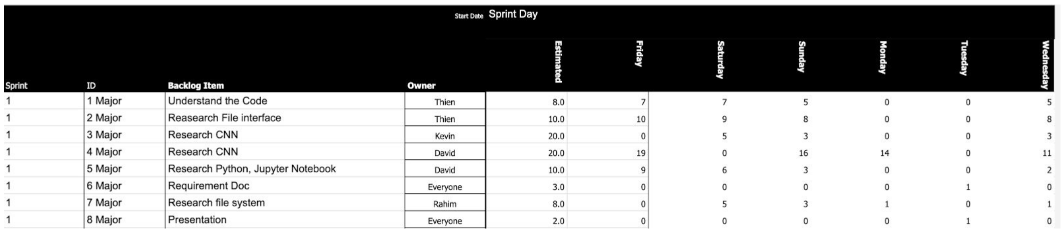
Bat_Echolocation_2019 **#103**
Multithreading/Optimization
+4
10

Bat_Echolocation_2019 **#105**
Refine GUI
+2
8

Bat_Echolocation_2019 **#106**
Presentation
Sprint 11
3

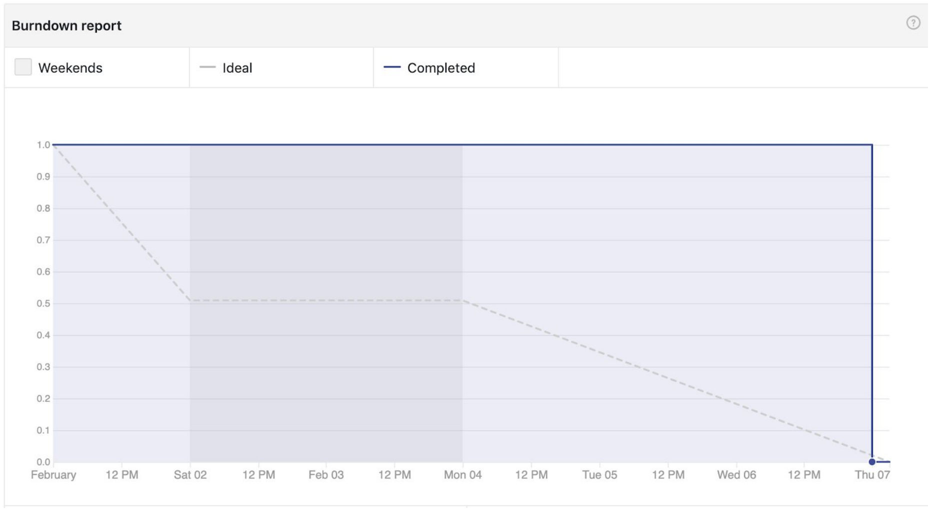
b. Sprint Backlog (Table / Diagram)

i. Sprint 1

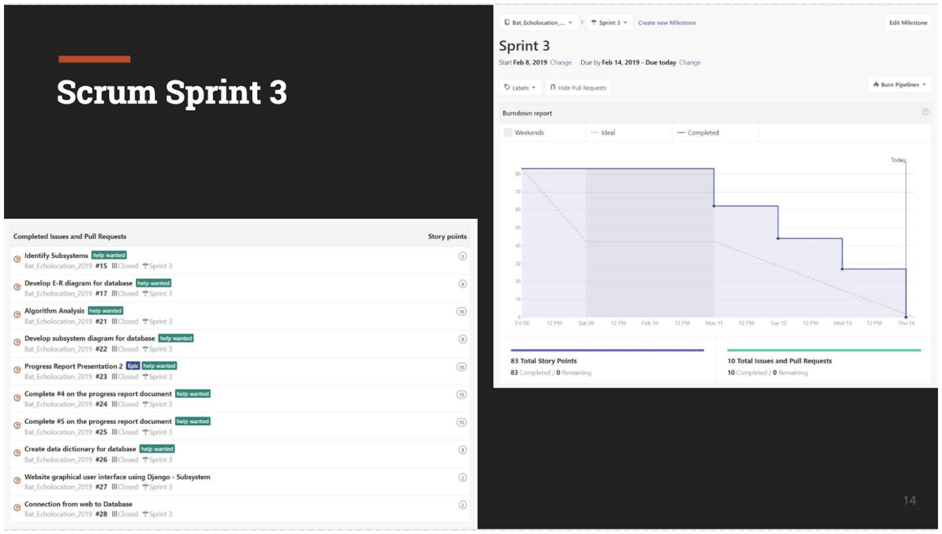


				Start Date	Sprint Day					
Sprint	ID	Backlog Item	Owner	Estimated	Friday	Saturday	Sunday	Monday	Tuesday	Wednesday
1	1 Major	Understand the Code	Thien	8.0	7	7	5	0	0	5
1	2 Major	Reasearch File interface	Thien	10.0	10	9	8	0	0	8
1	3 Major	Research CNN	Kevin	20.0	0	5	3	0	0	3
1	4 Major	Research CNN	David	20.0	19	0	16	14	0	11
1	5 Major	Research Python, Jupyter Notebook	David	10.0	9	6	3	0	0	2
1	6 Major	Requirement Doc	Everyone	3.0	0	0	0	0	1	0
1	7 Major	Research file system	Rahim	8.0	0	5	3	1	0	1
1	8 Major	Presentation	Everyone	2.0	0	0	0	0	1	0

ii. Sprint 2



iii. Sprint 3



Scrum Sprint 3

Bat_Echolocation_... › Sprint 3 Create new Milestone Edit Milestone

Sprint 3

Start **Feb 8, 2019** Change Due by **Feb 14, 2019 - Due today** Change

Labels Hide Pull Requests Burn Pipelines

Burndown report

Weekends — Ideal — Completed

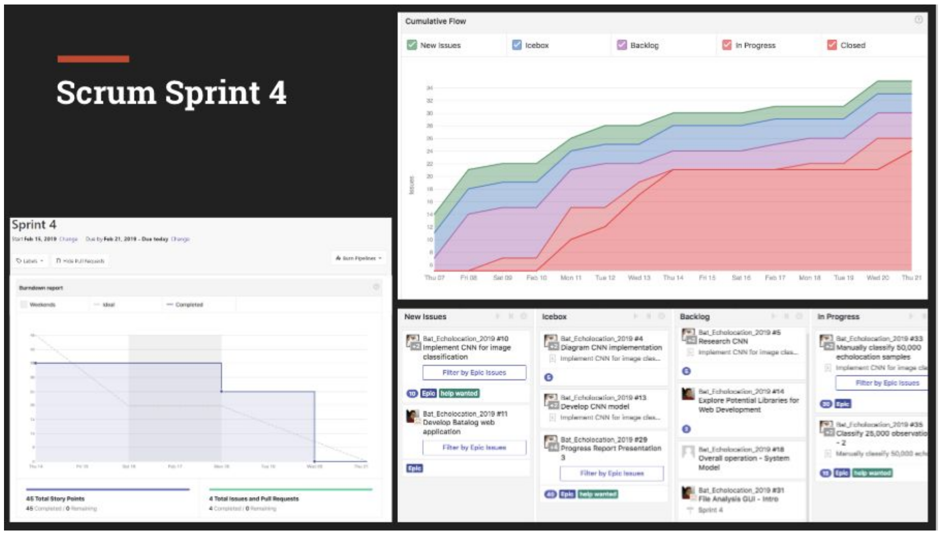
83 Total Story Points
83 Completed / **0** Remaining

10 Total Issues and Pull Requests
10 Completed / **0** Remaining

Completed Issues and Pull Requests	Story points
Identify Subsystems help wanted Bat_Echolocation_2019 **#15** Closed Sprint 3	5
Develop E-R diagram for database help wanted Bat_Echolocation_2019 **#17** Closed Sprint 3	8
Algorithm Analysis help wanted Bat_Echolocation_2019 **#21** Closed Sprint 3	10
Develop subsystem diagram for database help wanted Bat_Echolocation_2019 **#22** Closed Sprint 3	8
Progress Report Presentation 2 Epic help wanted Bat_Echolocation_2019 **#23** Closed Sprint 3	10
Complete #4 on the progress report document help wanted Bat_Echolocation_2019 **#24** Closed Sprint 3	15
Complete #5 on the progress report document help wanted Bat_Echolocation_2019 **#25** Closed Sprint 3	15
Create data dictionary for database help wanted Bat_Echolocation_2019 **#26** Closed Sprint 3	8
Website graphical user interface using Django - Subsystem Bat_Echolocation_2019 **#27** Closed Sprint 3	2
Connection from web to Database Bat_Echolocation_2019 **#28** Closed Sprint 3	2

14

iv. Sprint 4



Scrum Sprint 4

Sprint 4

Start **Feb 15, 2019** Change Due by **Feb 21, 2019 - Due today** Change

Labels Hide Pull Requests Burn Pipelines

Burndown report

Weekends — Ideal — Completed

45 Total Story Points
45 Completed / **0** Remaining

4 Total Issues and Pull Requests
4 Completed / **0** Remaining

Cumulative Flow

✓ New Issues ✓ Icebox ✓ Backlog ✓ In Progress ✓ Closed

New Issues

Bat_Echolocation_2019 #10 Implement CNN for image classification
Filter by Epic Issues
10 Epic help wanted

Bat_Echolocation_2019 #11 Develop Batalog web application
Filter by Epic Issues
Epic

Icebox

Bat_Echolocation_2019 #4 Diagram CNN implementation
Implement CNN for image clas...
5

Bat_Echolocation_2019 #13 Develop CNN model
Implement CNN for image clas...

Bat_Echolocation_2019 #29 Progress Report Presentation 3
Filter by Epic Issues
40 Epic help wanted

Backlog

Bat_Echolocation_2019 #5 Research CNN
Implement CNN for image clas...
5

Bat_Echolocation_2019 #14 Explore Potential Libraries for Web Development
3

Bat_Echolocation_2019 #18 Overall operation - System Model

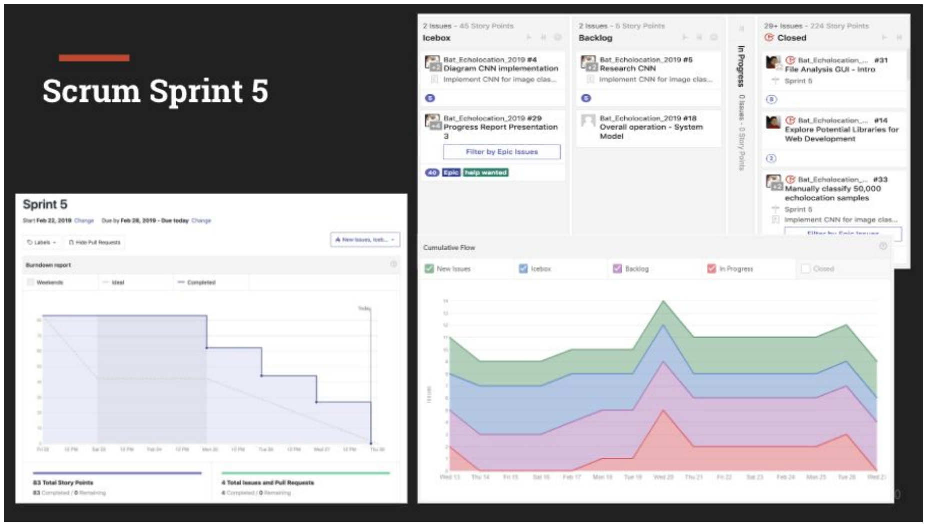
Bat_Echolocation_2019 #31 File Analysis GUI - Intro
Sprint 4

In Progress

Bat_Echolocation_2019 #33 Manually classify 50,000 echolocation samples
Implement CNN for image cla
Filter by Epic Issues
30 Epic

Bat_Echolocation_2019 #35 Classify 25,000 observatio - 2
Manually classify 50,000 ech
10 Epic help wanted

v. Sprint 5



Scrum Sprint 5

Sprint 5

Start **Feb 22, 2019** Change Due by **Feb 28, 2019 - Due today** Change

Labels Hide Pull Requests New Issues, Iceb...

Burndown report

Weekends — Ideal — Completed

83 Total Story Points
83 Completed / **0** Remaining

4 Total Issues and Pull Requests
4 Completed / **0** Remaining

2 Issues - 45 Story Points
Icebox

Bat_Echolocation_2019 #4 Diagram CNN implementation
Implement CNN for image clas...
5

Bat_Echolocation_2019 #29 Progress Report Presentation 3
Filter by Epic Issues
40 Epic help wanted

2 Issues - 5 Story Points
Backlog

Bat_Echolocation_2019 #5 Research CNN
Implement CNN for image clas...
5

Bat_Echolocation_2019 #18 Overall operation - System Model

In Progress 0 Issues - 0 Story Points

29+ Issues - 224 Story Points
Closed

Bat_Echolocation_... #31 File Analysis GUI - Intro
Sprint 5
8

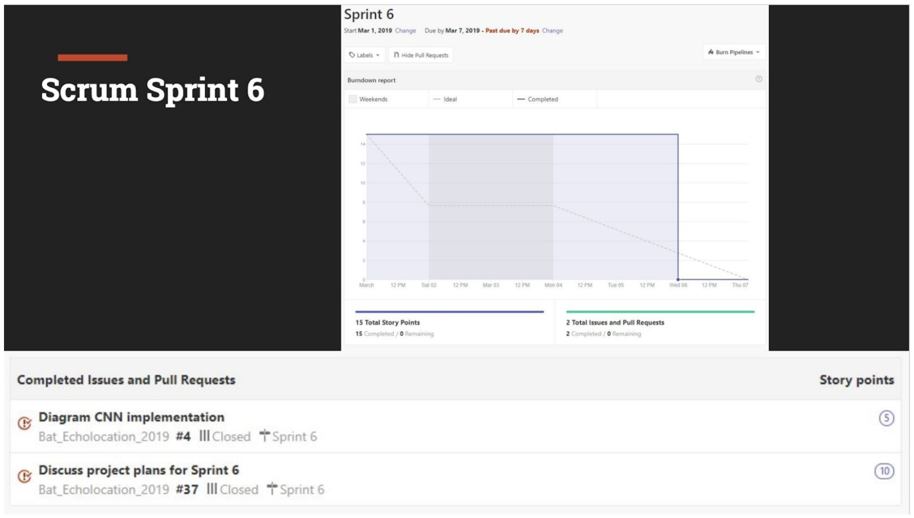
Bat_Echolocation_... #14 Explore Potential Libraries for Web Development
3

Bat_Echolocation_... #33 Manually classify 50,000 echolocation samples
Sprint 5
Implement CNN for image clas...
Filter by Epic Issues

Cumulative Flow

✓ New Issues ✓ Icebox ✓ Backlog ✓ In Progress Closed

vi. Sprint 6



Scrum Sprint 6

Sprint 6

Start **Mar 1, 2019** Change Due by **Mar 7, 2019 - Past due by 7 days** Change

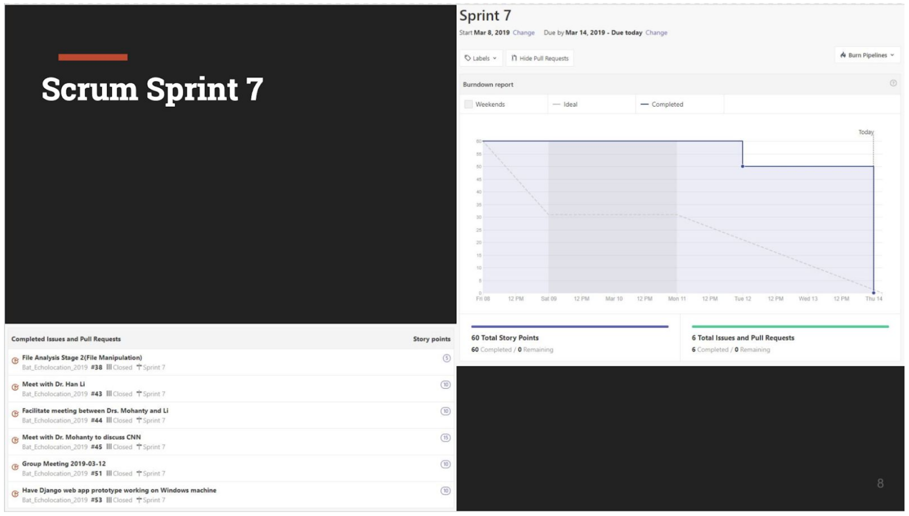
Burndown report

15 Total Story Points
15 Completed / 0 Remaining

2 Total Issues and Pull Requests
2 Completed / 0 Remaining

Completed Issues and Pull Requests	Story points
Diagram CNN implementation Bat_Echolocation_2019 **#4** Closed Sprint 6	5
Discuss project plans for Sprint 6 Bat_Echolocation_2019 **#37** Closed Sprint 6	10

vii. Sprint 7



Scrum Sprint 7

Sprint 7

Start **Mar 8, 2019** Change Due by **Mar 14, 2019 - Due today** Change

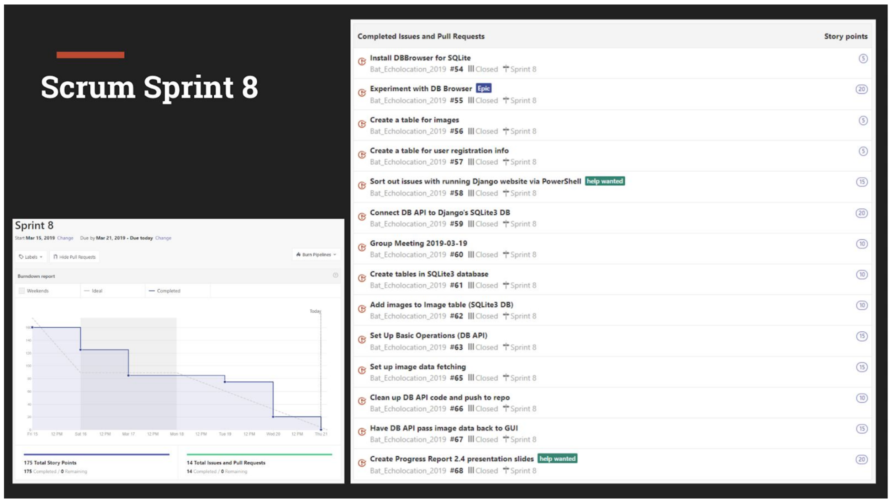
Burndown report

60 Total Story Points
60 Completed / 0 Remaining

6 Total Issues and Pull Requests
6 Completed / 0 Remaining

Completed Issues and Pull Requests	Story points
File Analysis Stage 2(File Manipulation) Bat_Echolocation_2019 **#38** Closed Sprint 7	5
Meet with Dr. Han Li Bat_Echolocation_2019 **#43** Closed Sprint 7	10
Facilitate meeting between Drs. Mohanty and Li Bat_Echolocation_2019 **#44** Closed Sprint 7	10
Meet with Dr. Mohanty to discuss CNN Bat_Echolocation_2019 **#45** Closed Sprint 7	15
Group Meeting 2019-03-12 Bat_Echolocation_2019 **#51** Closed Sprint 7	10
Have Django web app prototype working on Windows machine Bat_Echolocation_2019 **#53** Closed Sprint 7	10

viii. Sprint 8



Scrum Sprint 8

Sprint 8

Start **Mar 15, 2019** Change Due by **Mar 21, 2019 - Due today** Change

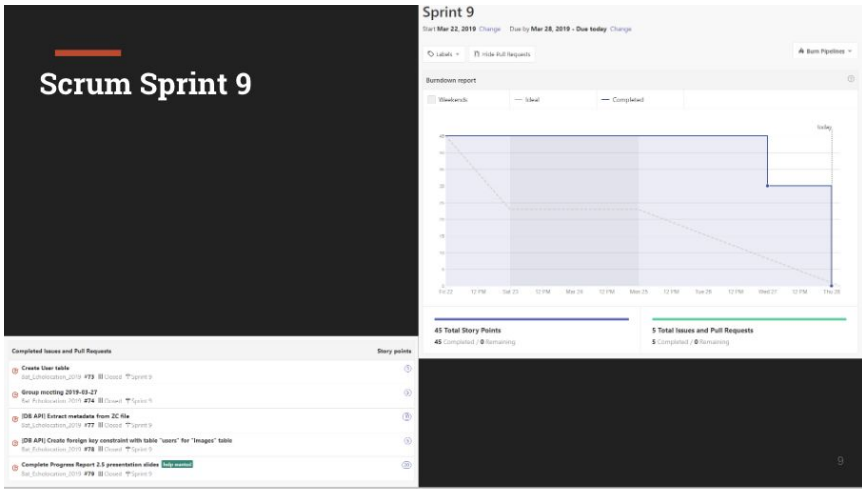
Burndown report

175 Total Story Points
175 Completed / 0 Remaining

14 Total Issues and Pull Requests
14 Completed / 0 Remaining

Completed Issues and Pull Requests	Story points
Install DBBrowser for SQLite Bat_Echolocation_2019 **#54** Closed Sprint 8	5
Experiment with DB Browser Epic Bat_Echolocation_2019 **#55** Closed Sprint 8	20
Create a table for images Bat_Echolocation_2019 **#56** Closed Sprint 8	5
Create a table for user registration info Bat_Echolocation_2019 **#57** Closed Sprint 8	5
Sort out issues with running Django website via PowerShell help wanted Bat_Echolocation_2019 **#58** Closed Sprint 8	15
Connect DB API to Django's SQLite3 DB Bat_Echolocation_2019 **#59** Closed Sprint 8	20
Group Meeting 2019-03-19 Bat_Echolocation_2019 **#60** Closed Sprint 8	10
Create tables in SQLite3 database Bat_Echolocation_2019 **#61** Closed Sprint 8	10
Add images to Image table (SQLite3 DB) Bat_Echolocation_2019 **#62** Closed Sprint 8	10
Set Up Basic Operations (DB API) Bat_Echolocation_2019 **#63** Closed Sprint 8	15
Set up image data fetching Bat_Echolocation_2019 **#65** Closed Sprint 8	15
Clean up DB API code and push to repo Bat_Echolocation_2019 **#66** Closed Sprint 8	10
Have DB API pass image data back to GUI Bat_Echolocation_2019 **#67** Closed Sprint 8	15
Create Progress Report 2.4 presentation slides help wanted Bat_Echolocation_2019 **#68** Closed Sprint 8	20

ix. Sprint 9



Scrum Sprint 9

Sprint 9

Start **Mar 22, 2019** Change Due by **Mar 28, 2019 - Due today** Change

Labels | Hide Pull Requests | Burn Pipelines

Burndown report

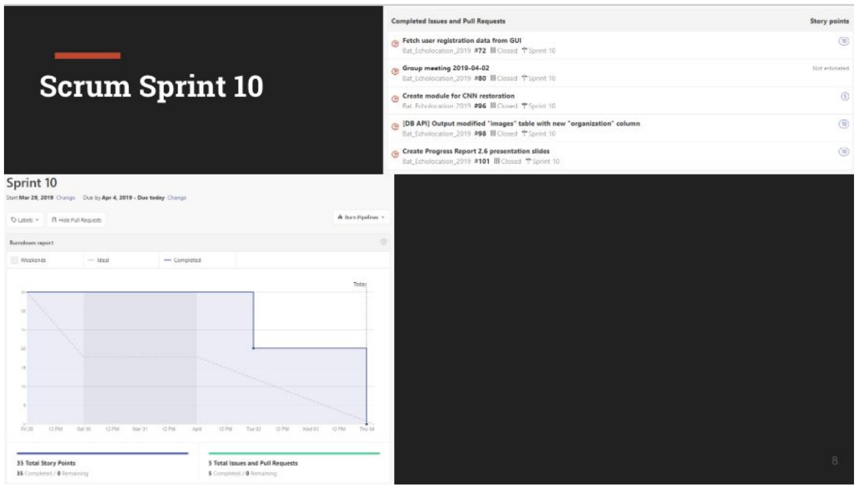
45 Total Story Points
45 Completed / 0 Remaining

5 Total Issues and Pull Requests
5 Completed / 0 Remaining

Completed Issues and Pull Requests	Story points
Create User table Bat_Echolocation_2019 **#73** Closed Sprint 9	5
Group meeting 2019-03-27 Bat_Echolocation_2019 **#74** Closed Sprint 9	5
[DB API] Extract metadata from ZC file Bat_Echolocation_2019 **#77** Closed Sprint 9	10
[DB API] Create foreign key constraint with table "users" for "images" table Bat_Echolocation_2019 **#78** Closed Sprint 9	5
Complete Progress Report 2.5 presentation slides help wanted Bat_Echolocation_2019 **#79** Closed Sprint 9	20

9

x. Sprint 10



Scrum Sprint 10

Completed Issues and Pull Requests	Story points
Fetch user registration data from GUI Bat_Echolocation_2019 **#72** Closed Sprint 10	10
Group meeting 2019-04-02 Bat_Echolocation_2019 **#80** Closed Sprint 10	Not estimated
Create module for CNN restoration Bat_Echolocation_2019 **#96** Closed Sprint 10	5
[DB API] Output modified "images" table with new "organization" column Bat_Echolocation_2019 **#98** Closed Sprint 10	10
Create Progress Report 2.6 presentation slides Bat_Echolocation_2019 **#101** Closed Sprint 10	10

Sprint 10

Start **Mar 29, 2019** Change Due by **Apr 4, 2019 - Due today** Change

Labels | Hide Pull Requests | Burn Pipelines

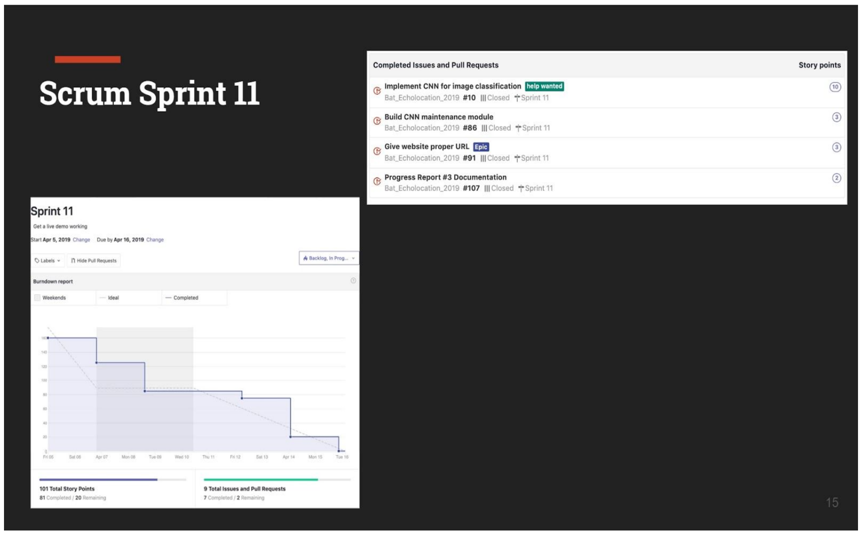
Burndown report

35 Total Story Points
35 Completed / 0 Remaining

5 Total Issues and Pull Requests
5 Completed / 0 Remaining

8

xi. Sprint 11



Scrum Sprint 11

Completed Issues and Pull Requests	Story points
Implement CNN for image classification help wanted Bat_Echolocation_2019 **#10** Closed Sprint 11	10
Build CNN maintenance module Bat_Echolocation_2019 **#86** Closed Sprint 11	3
Give website proper URL Epic Bat_Echolocation_2019 **#91** Closed Sprint 11	3
Progress Report #3 Documentation Bat_Echolocation_2019 **#107** Closed Sprint 11	2

Sprint 11

Get a live demo working

Start **Apr 5, 2019** Change Due by **Apr 16, 2019** Change

Labels | Hide Pull Requests | Backlog, In Prog...

Burndown report

101 Total Story Points
81 Completed / 20 Remaining

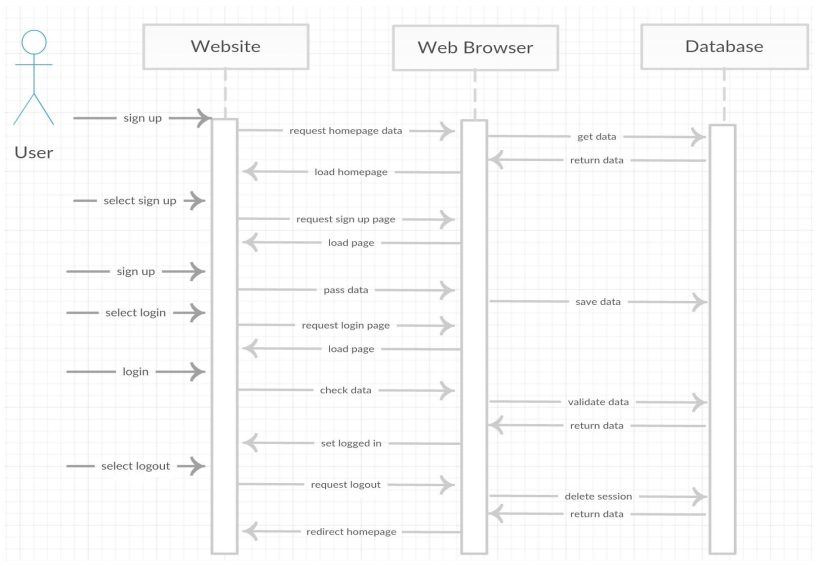
9 Total Issues and Pull Requests
7 Completed / 2 Remaining

15

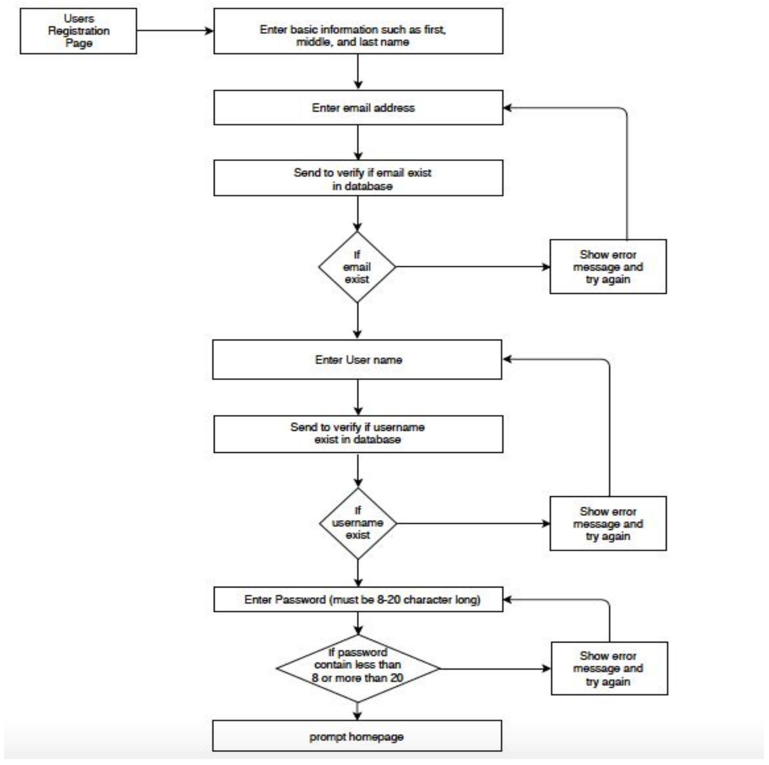
7. Subsystems

7.1 Subsystem 1 – User Account Management - *Rahim Iqbal*

i. Sequence diagram of User account system:



ii. Signup page diagram:


Users Registration Page
Enter basic information such as first, middle, and last name
Enter email address
Send to verify if email exist in database
If email exist
Show error message and try again
Enter User name
Send to verify if username exist in database
If username exist
Show error message and try again
Enter Password (must be 8-20 character long)
If password contain less than 8 or more than 20
Show error message and try again
prompt homepage

iii. Design choices

1. The first design plan only consist signup page and going to use SQL database in Django to create user table which

consist of several fields. It turns out that Django come with pre installed SQLite, and the previous django app that was built by our group members were already using sqlite.

2. The second design plan was to create signup, login, and logout feature as shown above in (Sequence diagram of User account system). The plan is to use Django sqlite pre auth_user table to store the user signup information such as username, email, password etc. in that table. For creating the login and logout function was to use Django built in view function that will help to create a login and logout feature.
3. The final design plan was to add a password reset feature using registered email address that lets users to reset the password in case the user forget the password. This feature not only email the password reset link, but also email the username in case the user forgets that too.

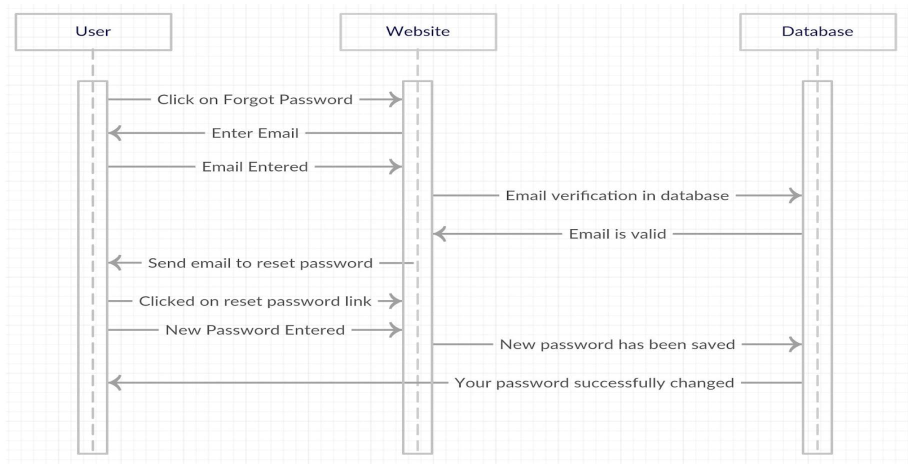
iv. Table: auth_user
    1. Data type: text (VARCHAR)
    2. This table contains the user registration information such as user name, first name, last name, email, and password in the database and it uses ID as a primary key. When the user enters their login information, the web application searches this table for the username and password and determines whether their credentials exist in this table. If so, then the user will be able to login and upload their data. The UID will help to clear the user data when the user logout.

v. There are several reasons for refinement the subsystem over the course of the project. There were mostly pros for the refinement. Such as adding an organization field let us to collect data of user organization for statistics purpose. Changing a password requirements will give more security to the user. Adding a reset password feature lets users to reset the password using their email, in case the user forget the password.

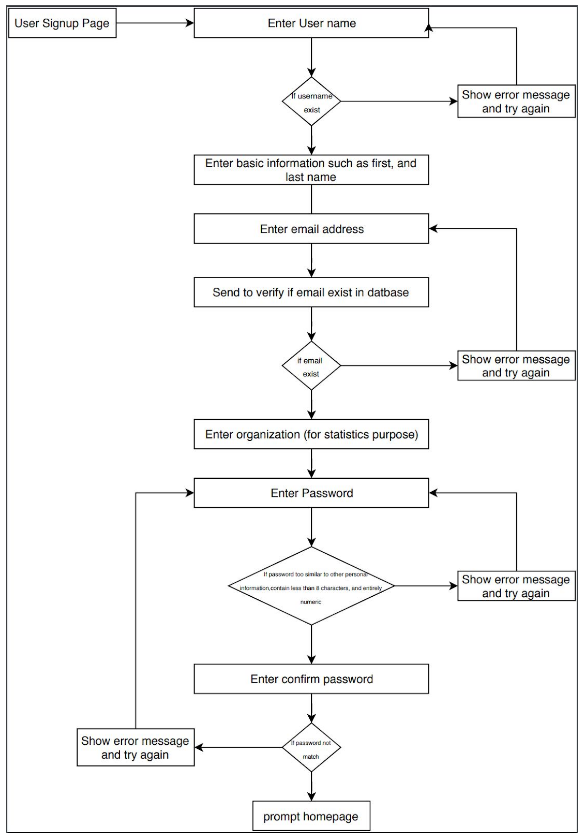
vi. There were several changes from the initial model such as adding a reset password feature, add an organization field, and changing password requirements, and adding a feature to see their username at the top right screen once the user login.

vii. Refinements

1. Forget Password Sequence Diagram:


User
Website
Database
Click on Forgot Password
Enter Email
Email Entered
Email verification in database
Email is valid
Send email to reset password
Clicked on reset password link
New Password Entered
New password has been saved
Your password successfully changed

2. The way the reset password feature work, when user click on forget password, it will prompt another page to enter registered email address. Once the email enter, it will look for the email in database and if the email is present in database it will send an email with reset password link. When user click on reset password link it will pop up another tab and ask to enter a new password and the new password will save in database.
3. Refined diagram of signup page:


User Signup Page
Enter User name
If username exist
Show error message and try again
Enter basic information such as first, and last name
Enter email address
Send to verify if email exist in datbase
if email exist
Show error message and try again
Enter organization (for statistics purpose)
Enter Password
If password too similar to other personal information,contain less than 8 characters, and entirely numeric
Show error message and try again
Enter confirm password
If password not match
Show error message and try again
prompt homepage

        4. In this refined diagram, created a field of organization for statistics purpose. Also change the password requirement for adding more security in user password.

viii. Scrum Backlog (Product and Sprint - Link to Section 6) https://docs.google.com/document/d/1jJgxoAWclTfXR5WuWl7eNxvBmqgRZZZUhYIx0BMW1Hs/edit#heading=h.rz9jqyppht2k

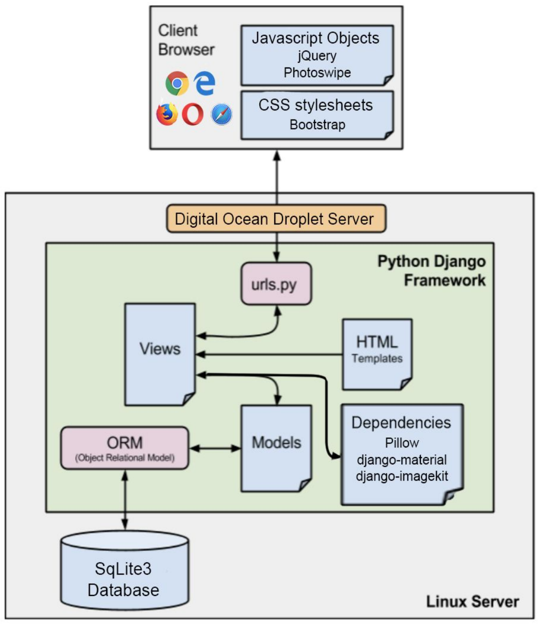
ix. Coding
    1. Approach- Functional
    2. Language
        a. Majority - Python
        b. Minority - Javascript, HTML

x. User training: If you don't have an account, click signup to create one. If you do, click login to log in.

xi. When testing, first we always make sure that the subsystems we make would work perfectly on local machine. I always make sure that when user enters their information for registering to website the information will save to database and prompt to homepage after successfully creating an account. Any changes we made to the code, we check it on local machine. If everything was working perfectly on local machine, then we make the final commit and push to the repo.

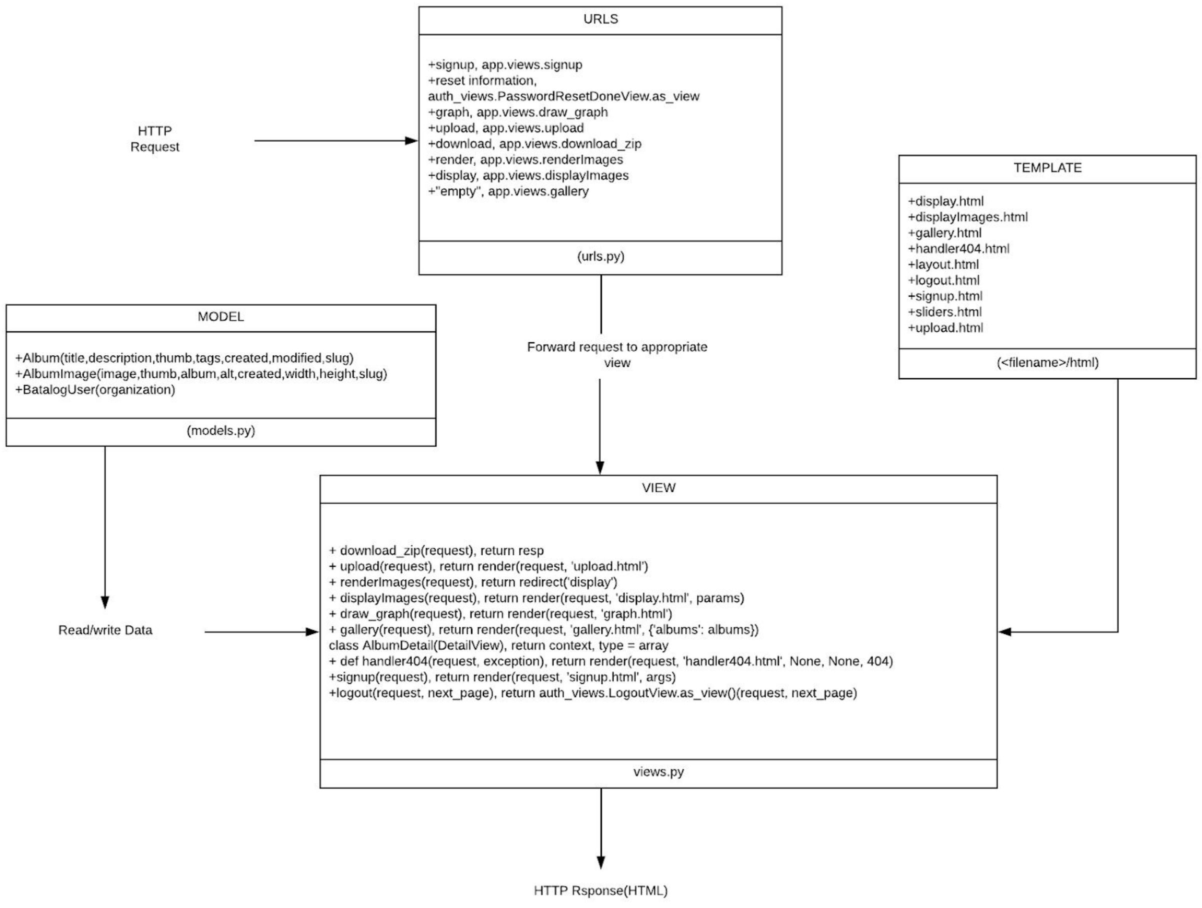
7.2 Subsystem 2 – Front End - *Thien Le*

- Initial design and model



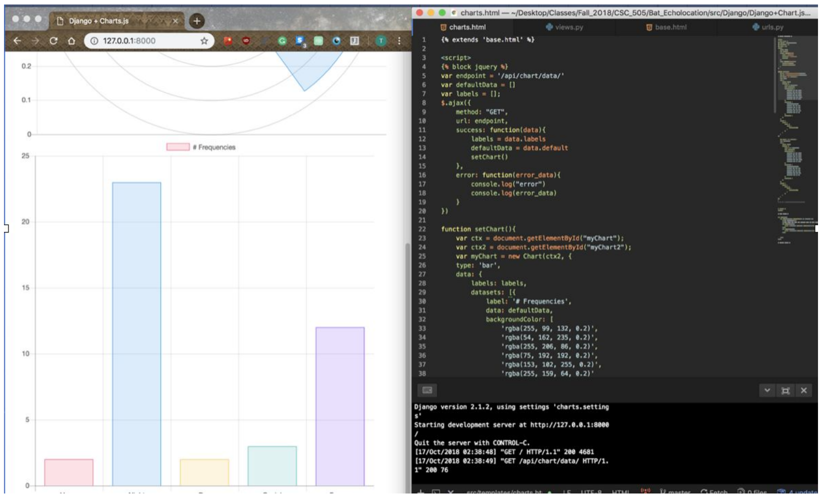
The first implementation of this project was in Fall 2018 as a Data Science project at UNCG. The initial design and model we have picked for the project is Django since it is a Python web based programming language. The design is fairly simple as our first priority is to develop a page where the user can allow to us our bat echolocation tool to analyze their own zero crossing file.
The front end is composed of mainly the templates which are HTML/CSS and javascript, plus the views functions. Once the front end is established through the template and views the next object was to connect to the back ends. These back end or functional modules are Display Images, Download as Zip, and Graph. Please refer to subsystem 3 to find out more information about these functions.

- The design choice for this front end the Django framework. Most of the implementations are similar to the Django documentation. There are many reasons that Django is the to do web platform we use for this project.
    - Django is a Python web framework which most of our group members are familiar with. This allows all of us to develop the website anywhere rather be restricted to any particular server platform, and can run your applications on many flavours of Linux, Windows, and Mac OS X.
    - Versatile - Django is very well known and have been used on almost any type of website so it was a good opportunity to pick it up.
    - Secure - Django user and admin management have to be one of the best functions that it provides to the developer. It also has built in clickjacking, cross-site scripting, SQL injection.


URLS
+signup, app.views.signup
+reset information, auth_views.PasswordResetDoneView.as_view
+graph, app.views.draw_graph
+upload, app.views.upload
+download, app.views.download_zip
+render, app.views.renderImages
+display, app.views.displayImages
+"empty", app.views.gallery
(urls.py)
HTTP Request
TEMPLATE
+display.html
+displayImages.html
+gallery.html
+handler404.html
+layout.html
+logout.html
+signup.html
+sliders.html
+upload.html
(<filename>/html)
MODEL
+Album(title,description,thumb,tags,created,modified,slug)
+AlbumImage(image,thumb,album,alt,created,width,height,slug)
+BatalogUser(organization)
(models.py)
Forward request to appropriate view
VIEW
+ download_zip(request), return resp
+ upload(request), return render(request, 'upload.html')
+ renderImages(request), return redirect('display')
+ displayImages(request), return render(request, 'display.html', params)
+ draw_graph(request), return render(request, 'graph.html')
+ gallery(request), return render(request, 'gallery.html', {'albums': albums})
class AlbumDetail(DetailView), return context, type = array
+ def handler404(request, exception), return render(request, 'handler404.html', None, None, 404)
+signup(request), return render(request, 'signup.html', args)
+logout(request, next_page), return auth_views.LogoutView.as_view()(request, next_page)
views.py
Read/write Data
HTTP Rsponse(HTML)

- Data dictionary
    - This Front End illustrates the following concepts on top of the Django Framework:
        - Using django-material and materializecss for building Django UI.
        - Using django-imagekit for building resizing images.
        - Using photoswipe javascript library for more rich image gallery user experience.
    - Javascript Objects
        - jQuery - a library that helps navigate through documents, create animations and handle events easier.
        - Photoswipe - an image gallery for web development.
    - CSS stylesheets
        - Bootstrap - responsive grid system, extensive pre built components
    - Digital Ocean Droplet Server
        - A scalable compute platform with add-on storage, security, and monitoring capabilities to easily run production applications.
    - Python Django Framework

        - ORM - can be used to interact with application data from various relational databases such as SQLite, PostgreSQL and MySQL.
        - Dependencies
            - Pillows - Python Imaging Library.
    - SqLite3 Database
        - An embedded database, server-less and can run within our app.
- If refined (changed over the course of project)
    - First Take on the Project: Django + Chart.js


Django + Charts.js
127.0.0.1:8000
# Frequencies
charts.html — ~/Desktop/Classes/Fall_2018/CSC_505/Bat_Echolocation/src/Django/Django+Chart.js...
charts.html
views.py
base.html
urls.py

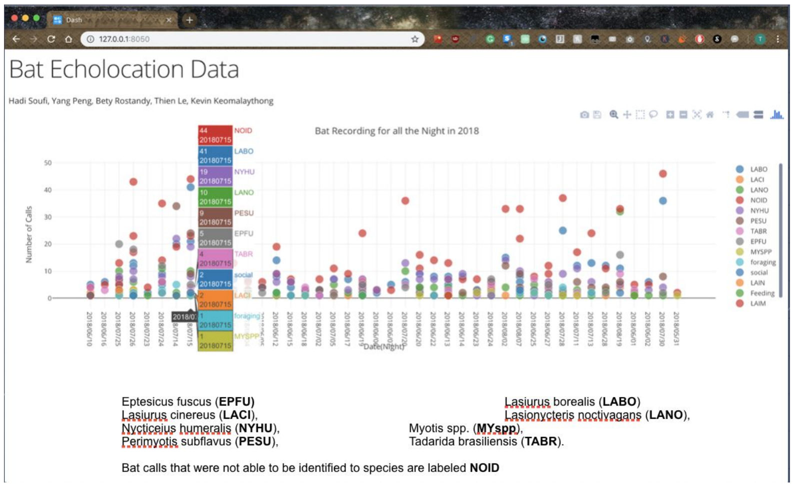
```
{% extends 'base.html' %}

<script>
{% block jquery %}
var endpoint = '/api/chart/data/'
var defaultData = []
var labels = [];
$.ajax({
    method: "GET",
    url: endpoint,
    success: function(data){
        labels = data.labels
        defaultData = data.default
        setChart()
    },
    error: function(error_data){
        console.log("error")
        console.log(error_data)
    }
})

function setChart(){
    var ctx = document.getElementById("myChart");
    var ctx2 = document.getElementById("myChart2");
    var myChart = new Chart(ctx2, {
    type: 'bar',
    data: {
        labels: labels,
        datasets: [{
            label: '# Frequencies',
            data: defaultData,
            backgroundColor: [
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)',
                'rgba(255, 159, 64, 0.2)'
```

```
Django version 2.1.2, using settings 'charts.setting
s'
Starting development server at http://127.0.0.1:8000
/
Quit the server with CONTROL-C.
[17/Oct/2018 02:38:48] "GET / HTTP/1.1" 200 4681
[17/Oct/2018 02:38:49] "GET /api/chart/data/ HTTP/1.
1" 200 76
```

        - This is the first take on displaying bat echolocation data. The benefits of using Chart.js is that is very straightforward and easy to use. However, it is very hard to incorporate with the zero crossing file analysis since it will only read data from csv format. There were no changes to the initial since this is the first implementation from the barebone django beginner project.
    - Second take on the Project: Django + Dash



Bat Echolocation Data

Hadi Soufi, Yang Peng, Bety Rostandy, Thien Le, Kevin Keomalaythong

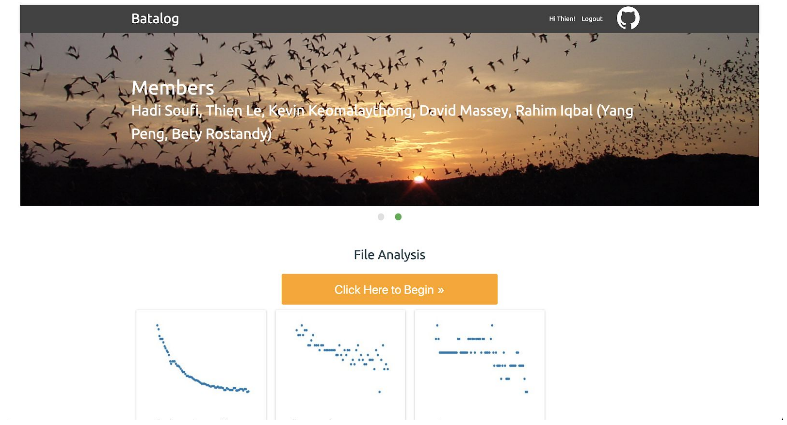
Eptesicus fuscus (**EPFU**)
Lasiurus cinereus (**LACI**),
Nycticeius humeralis (**NYHU**),
Perimyotis subflavus (**PESU**),

Lasiurus borealis (**LABO**)
Lasionycteris noctivagans (**LANO**),
Myotis spp. (**MYspp**),
Tadarida brasiliensis (**TABR**).

Bat calls that were not able to be identified to species are labeled **NOID**

        - The second refined on this project was with Dash, a productive Python framework for building web applications. Written on top of

Flask, Plotly.js, and React.js, Dash is ideal for building data visualization apps with highly custom user interfaces in pure Python. Dash is amazing at forming visualization on the data that have been fed into the views.py. However, this was refined since our focus on the project has changed. We wanted to focus more of the mass file analysis rather than just individual points on each graph.


Batalog
Hi Thien! Logout
Members
Hadi Soufi, Thien Le, Kevin Keomalaythong, David Massey, Rahim Iqbal (Yang Peng, Bety Rostandy)
File Analysis
Click Here to Begin »

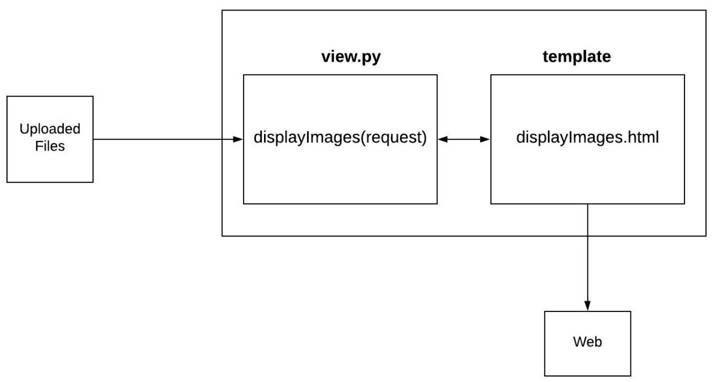
- With the focus been altered that was mentioned above, this is now our latest refinement to the website. We're now focused on providing tools for the end user. The main tool is composed of classification and CNN analysis, and allows the user to submit any zero crossing file or zip file. The user can then download and view the results.

- Scrum Backlog (Product and Sprint - Link to Section 6)
- Coding
  - Approach - Object oriented programming
  - Language(s):
    - Majority - Python
    - Minority - Javascript, HTML, CSS
- User training
  - Training / User manual (needed for final report)
- Testing
  - Before we make any final commit and push to the repo we would always to make sure it works perfectly on my local machine first. Another way we carry out the testing process to have a copy of the repo and test it before committing.

7.3 Subsystem 3 – Functional Modules- *Display Image(Thien), Download as Zip(Hadi)* Graph(Rahim).

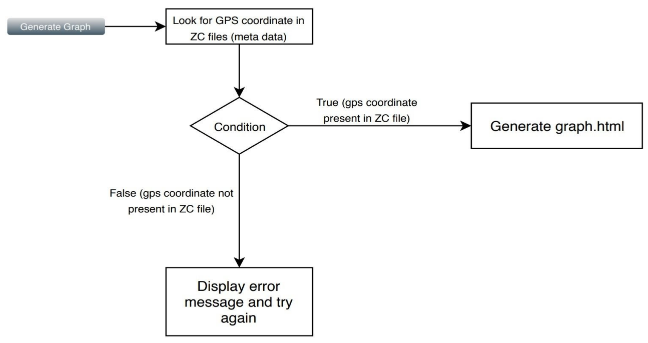
- Display Image
  - This sub-module's goal is to display the uploaded images into a user-friendly output where they can see the results on the web. Most of the implementation is from *views.py* and *displayImages.html.* The method *display_images* takes in the files from the uploaded section and return that to the template. The *display_images.html* then reformats the returned images into two columns, one for abnormal and the other echolocation.

DisplayImages Sub-module



- Graph:
  - The first graph design plan was to generate a plot line graph using gps coordinates in zero crossing files, the graph will help to locate the location of the bat echo.
  - The second design plan was to add an exception in case the zero crossing file contains no gps coordinates. That way it will lead to error message and ask to try again.

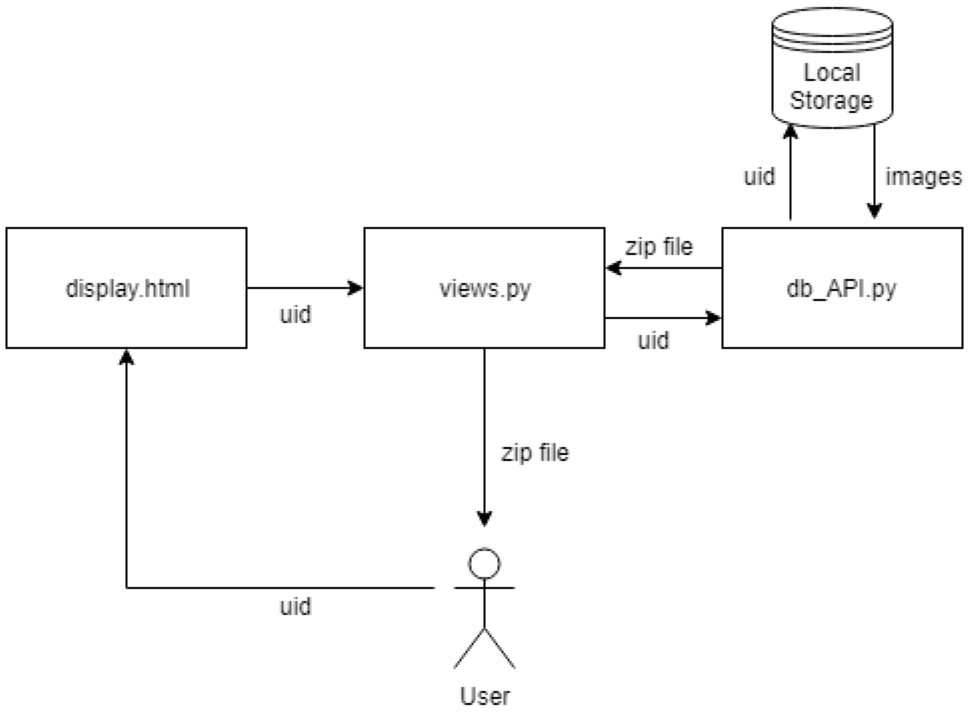
Generate graph Sub-module


Generate Graph
Look for GPS coordinate in ZC files (meta data)
Condition
True (gps coordinate present in ZC file)
Generate graph.html
False (gps coordinate not present in ZC file)
Display error message and try again

  - Data dictionary

        - Table: Images
        - When the user upload ZC files it will generate data inside metadata column. The data include several things from the files such as datetime, species, gps position/coordinates, etc. After generating a result the user can generate the graph by simply clicking on generate gps graph. The graph look for gps position/coordinate in metadata column. If the gps position/coordinate present in metadata then it will generate graph.
    - If refined (changed over the course of project)
        - Reason for refinement (Pro versus Con)
        - Changes from initial model
        - Refined model analysis
        - Refined design (Diagram and Description)
    - Scrum Backlog (Product and Sprint - Link to Section 6)
    - Coding
        - Approach - Object oriented programming
        - Language(s):
            - Majority - Python
            - Minority - Javascript, HTML, CSS
    - User training
        - Training / User manual (needed for final report)
    - Testing
        - The gps graph has been test in local machine using a random gps coordinates. After making sure that the gps graph working perfectly in local machine, then we push the code to repo.

- Download as Zip (Hadi)



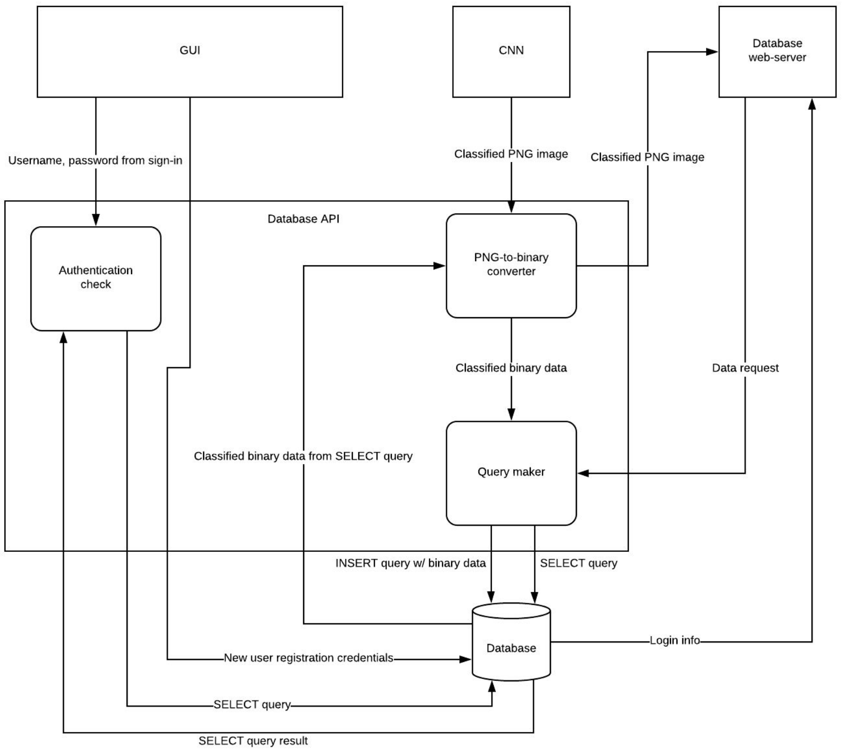
I chose a functional approach for this system. The user presses a button in display.html which sets off a chain of function calls ending in db_API.py. The zipping function in db_API.py was written in Python 3. The frontend was written in HTML with Django.

Testing was very straightforward. If I could successfully download a zip file, no errors were thrown, and no files were leftover, I considered it successful.

User training: After you've uploaded your data, you may want to download it for further analysis. Simply press the 'Download as Zip' button on the display page to receive a zip file with all of your pulses sorted by type.

7.4 Subsystem 4 – Database API - *Kevin*

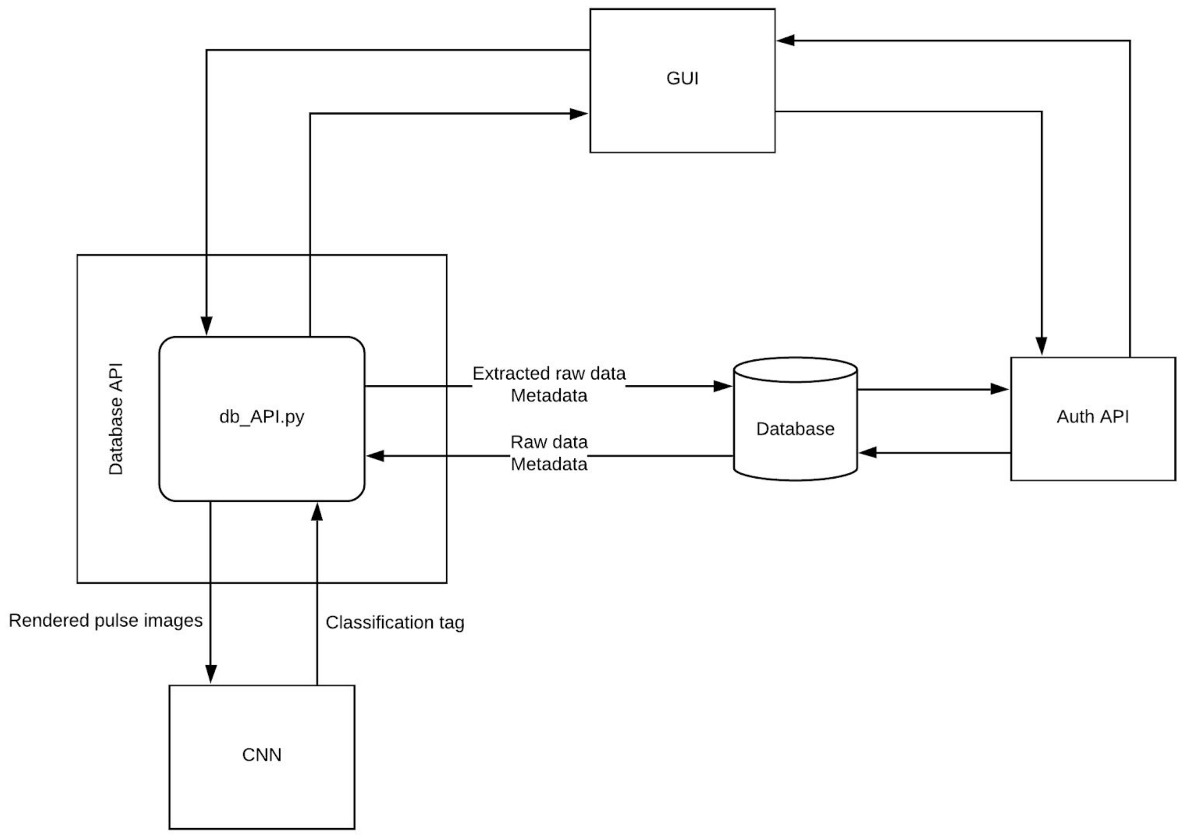
The initial design of the database API is shown below.



This design consists of three sub-modules: a PNG/binary converter, a query handler, and a user authentication handler. The converter would convert PNG to binary and vice-versa. The query handler would create SQL queries based on GUI data. The user authentication handler would perform two tasks:

- If a user creates an account, their registration information is sent to the query handler and inserted into a special table in the database.
- If a user tries to login, their request is checked against the “users” table and either authorized or denied depending on whether a match is found.

This design was expanded from a previous design of the database API used for the Fall 2018 Data Science version of this project. However, later in development the user authenticator was cut from the design in favor of Django’s built in system, and so the API will be modifying its own user authentication table. The updated design is shown below.



This subsystem was developed using a functional approach. The entire API consists of functions that prepare the data to be inserted into the database and convert ZC data into human-readable format. It uses Python's native SQLite3 library to support these operations. Said library supports a limited version of the SQL language, but is used here because the Django framework uses an SQLite3 database for local storage.

The database management itself is only concerned with two tables, "images", where the raw pulse data is stored, and "auth_user", Django's own user authentication table. The data dictionary for the tables is as follows.

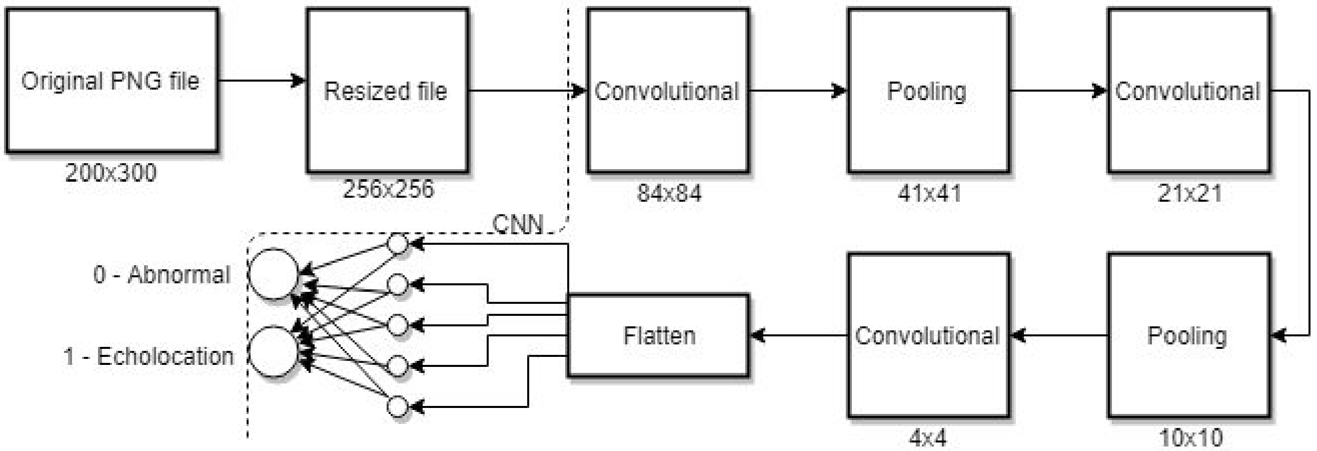
images		
Column Name	References	Description
name		Name of the ZC file with pulse number
raw		Raw pulse data in x/y coordinates
classification		Classification tag for a pulse; 0 for “echolocation”, 1 for “abnormal”
metadata		Data about the series of pulses, including date, timestamp, GPS coordinates, etc.
uid	auth_user	UID of the user that uploaded the file

auth_user	
Column Name	Description
username	Username of user
password	Password for user login
email	Email address associated w/ user
first_name	First name of user
last_name	Last name of user
organization	Organization affiliation of user (user is presumed to be a researcher)

Because the database API is more focused on the back-end, user training for this specific subsystem would be unnecessary. As for testing, it pertains more to the development of the database API and ensuring that its connections with other subsystems go through without any errors.

7.5 Subsystem 5 – CNN - *David*

- Initial design and model



- The first CNN design plan was to take other CNN designs and implement them. The result was low accuracy on the files tested and high training losses. The belief was that all CNN’s were swappable with one another. Further research proved this incorrect.
- Future CNN design plans were to make the layers based on the convolutional and pooling layer formulas below (Refined Model Analysis). The first design using those formulas was a CNN that resized images to 256x256 from the original 200x300. The squares represent the layers and the output image dimensions are listed below them.
    - Other CNN designs were constructed by resizing images to 216x324, 216x288, 250x250, and 512x512 in separate models.

- Refinements
    - Models with resized images:
        - Pro: Using the formulas for convolutional and pooling layers, the layers can be determined. The layers for CNN’s that use image sizes of 216x324, 216x288, 256x256, and 512x512 can be made from the factors which divide into these dimensions. More layers mean more features that may be detected by the CNN which may increase accuracy.
        - Con:
            - 1. Resizing removes or adds pixels to the images which may affect accuracy of the CNN.
            - 2. Convolutional layers require filters to have odd dimensions. For a model that requires images of 256x256 or 512x512, with both sizes being a power of 2, different filter sizes have to be calculated. Since output image sizes have

to be whole numbers, sometimes the only stride available is one.

- Model that uses original images with no resizing:
  - Pro: Images are unaltered which eliminates Con#1 above.
  - Con: It's difficult to calculate the layers of the dimensions 200x300 without using their factors. Both are divisible by 100 which makes their last dimensions to be 2x3 if only factors of 100 are used. If more layers are to be added, other values, other than the factors, will force some layers to have a stride of one. Two extended 200x300 models were planned and tested which made the last image dimension to be 2x7 and 1x26. The accuracies of both were lower than the model that used only the factors of 100.

- Changes from initial model
  - From the diagram in section 4, the Test/Train CNN section was added to reflect the use of external code to make a CNN model and incorporate a way to manually or automatically retrain it.
  - CNN's that required the original images to be resized were designed on paper and a few were tested. After testing, a CNN that didn't require any resizing had higher accuracy and was used.
- Refined model analysis
  - The following formulas were used:

    Convolutional Layer:

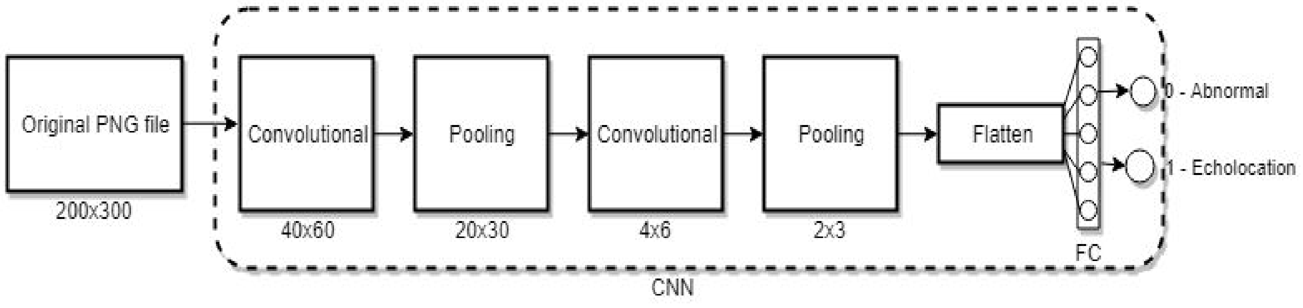
    Output size=Dimension size - Filter size+2(Padding)/Stride

    Pooling Layer:

    Output size=Dimension size - Filter size/Stride

The diagram below shows the layers along with the output image sizes. The layers of the CNN that are used in training are also present in the final model. Each image consists of three dimensions: height, width, number of channels. The convolutional layer filter size is required to be an odd number. The convolutional layers use a filter size of five, and the pooling layers use a filter size of two. These represent factors of 100. A dropout of 50% was added after all convolutional layers and after the flatten layer. The use of dropout is a regularization tool that minimizes and delays the presence of overfitting but is not present in the final model to process images. Dropout in training extends the region of low training losses and provides a region to use for models before the training starts to overfit. Following the dropout is an activation layer. Batch normalization was tested but provided high losses and low accuracy. The number of filters starts with 96 in the first convolutional layer and increases to 256 in the flatten layer. This process adds more types of filter grids to use for feature detection. The number of filters may be used as an added dimension to describe the images in the layers.

- Refined design (Diagram and Description)



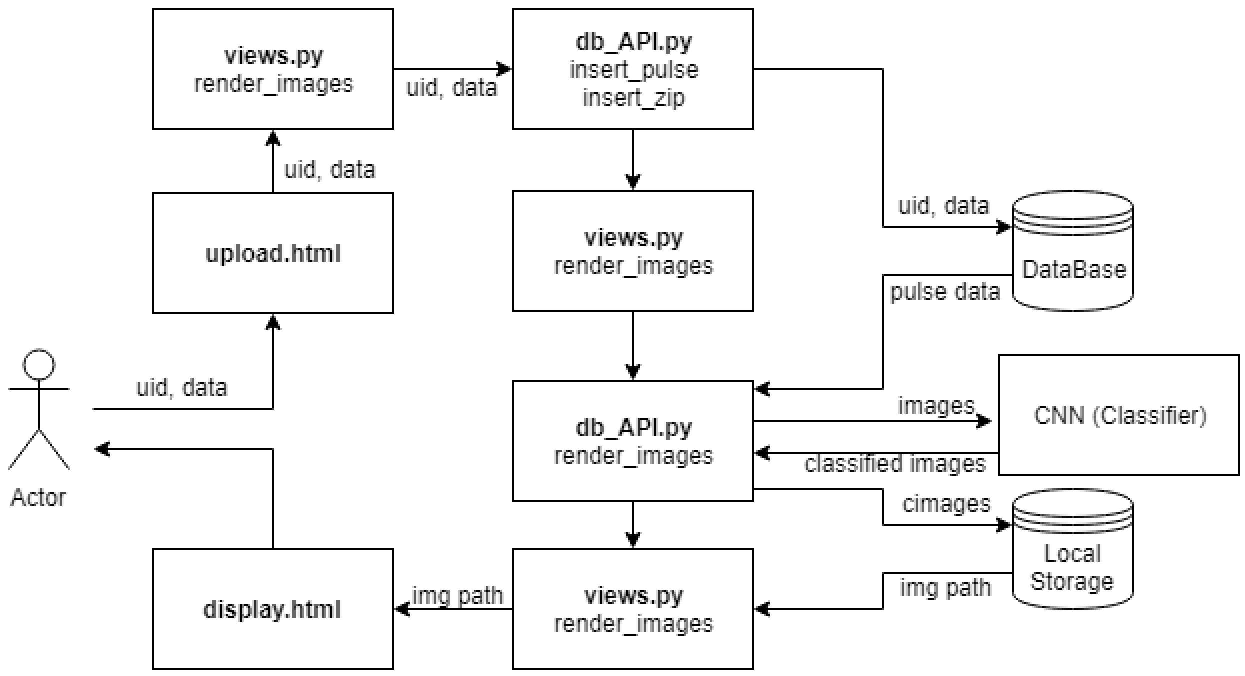
Original images are of dimensions 200x300 with the number of channels being 3 or RGB. The following describes the layers using the formulas above:

1. Convolutional: Filter size: 5 Stride: 5 Number of filters: 96
2. Pooling: Filter size: 2 Stride: 2
3. Convolutional: Filter size: 5 Stride: 5 Number of filters 128
4. Pooling: Filter size: 2 Stride: 2
5. Flatten
6. Dense fully connected layer: 256 neurons
7. Output dense fully connected: 2 neurons representing 2 classes. The output is a float that is 0 for abnormal call images and 1 for echolocation call images.

- Scrum Backlog (Product and Sprint - Link to Section 6) https://docs.google.com/document/d/1jJgxoAWclTfXR5WuWl7eNxvBmqgRZZZUhYIx0BMW1Hs/edit#heading=h.rz9jqypht2k
- Coding
  - Approach-Functional approach
  - Language
    - Python with Keras and Tensorflow as a backend
  - Testing
    - Testing consisted of training models and testing them with 37 abnormal validation images and all echolocation images. The Modelcheckpoint function provided models to test at various points in training. Matplotlib provided graphs to visualize training results. The goal was to test models that had low training losses and high accuracy as reported by Keras. The steps-per-epoch and validation steps parameters are known to be calculated from the formula:

Number of images/batch size. Batch size is adjustable by the tester. Lower batch sizes were used to minimize memory errors and maximize speed of the epochs. The tradeoff is the number of epochs required to process all images increases with smaller values. The valley of low training losses is extended in addition to using a higher dropout and smaller learning rate. The number of epochs required to process all training images was between 1.5 to 3 while the validation images required one. A copy of the abnormal training files were flipped, and echolocation training files were rotated two degrees and added to the files to train. The highest abnormal file accuracy was 83%, and echolocation accuracy was 99% using the same model.

7.6 Subsystem 6 - Zero Crossing File Analysis (Hadi)



Initially, we rendered the images and stored them in the database. However we realized that this was inefficient and would cause our database usage to rapidly bloom. Instead, we moved rendering the images to a user session and delete them after it ends. Now we only store the x/y coordinates of a ZC file, which is far more space efficient. The first model had a time complexity of O(n) + u, where n is the number of inserted images and u is the number of times a user logs in multiplied by the time complexity of each login. Our new model has a time complexity of O(n) * u.

We also incorporated classification into db_API.render_images. Immediately after it's rendered, it's sent to the classifier. Calling this a refinement or update to the original design is a bit of a stretch, since we always knew it was going to be incorporated into the system somehow. We didn't know exactly where it was going to go, so it wasn't explicitly designed into the system.

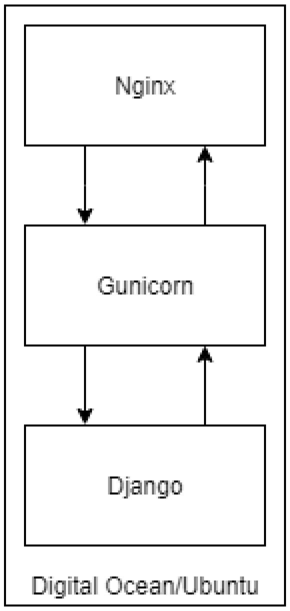
The other major update to the model was adding a handler for zip files. All it does is iterate through the zip and call insert_pulses on each file.

I chose a functional approach to this module. All of the functions were written in Python 3, the frontend was written in HTML with Django, and the database calls were written in SQL.

Testing was time consuming, as there were several features that all needed to be debugged. I started from the layer farthest away from the user and worked my way out. The functions farthest from the user were both the most complicated and the most isolated. So I wrote some, like db_API.render_images and db_API.insert_pulse, in Jupyter Notebook and tested them thoroughly. Then I tested each function in views.py and finally the HTML buttons. I considered my module complete if it received ZC data, put it into the DB, turned it into images, separated the data by user, and displayed the data after completion.

User Training: Once you've created an account and logged in, you will see a button labeled 'Click Here to Begin' on the home page. Clicking it will take you to the Upload page, where you can select your files. Currently we accept zero crossing and .zip files. After pressing 'Upload', our system will process your files. Depending on how much data you give us, it may take several minutes to process everything. When it's all done, you'll be redirected to the display page, where all of your data will be classified.

7.7 Subsystem 7 - Bring the website online (Hadi)



We didn't make a diagram of this initially since none of us had any experience with making a live website before. We knew it was possible & that our project didn't have any unique requirements, so we decided to wing it and see what happened.

We chose Digital Ocean to host our website because it's affordable, simple, and scalable. As a bonus, it provides a DNS hosting service. After loading the project onto Digital Ocean, I experimented with WSGI services and settled on Gunicorn because uWSGI wasn't working. I chose Nginx as my web server because Digital Ocean has a nice tutorial for setting it up.

This is an object oriented approach, but I didn't do any coding for this part. Most of my work involved fiddling around with the various modules and trying to get everything to click.

I tested my setup by building it from the inside out. I started by bringing Django online in a virtual environment to make sure there was nothing wrong with the project & that my Digital Ocean configurations were correct. Then I brought Gunicorn into the mix and tested to see if it worked, and finally Nginx. Nginx was the most satisfying for me, because I knew that once I got it working I'd have a proper domain name instead of an IP address.

The user shouldn't ever have to interact with this module except by using the url www.echobatalog.com

8. Complete System – *Group responsibility*

- Final software/hardware product
- Source code and user manual – screenshots as needed - Technical report
    - Github Link
- Evaluation by client and instructor
- Team Member Descriptions

This is just a guide, and use it to create/improve your report. Feel free to add sections. You are responsible for your own subsystem/s, not other members. You have to contribute to the team's goals and objectives, and develop your subsystem/s, write your documents and slides.