University of Arkansas – CSCE Department
Capstone II – Preliminary Report – Spring 2018

Salvador Sanchez, Zack Fravel, David Darling, Jace McPherson, Jonathon Raney

Abstract

oxDNA is an extensible DNA simulation and analysis software used by DNA researchers from various fields of study. While existing as a powerful tool for running simulations in molecular dynamics, oxDNA remains difficult to utilize for users without a computer science background. As a team, we want to bring these tools to a wider pool of users by simplifying the simulation process and building out a web-based user interface.

Along with the automation of the configuration of simulation input data, we have built in analysis and visualization tools that share immediately useful information with the user. We hope this framework represents a significant contribution to the field of molecular self-assembly by enabling researchers from different backgrounds to have access to these powerful DNA simulation tools.

1.0 Problem

oxDNA is currently clunky in its workflow. There are many isolated steps involved in the data generation and simulation process. With so many distinct, advanced processes, including compiling oxDNA, generating initial system state data using python scripts, converting the output to JSON files for visualization purposes, and running scripts to perform analysis, the simulation software is extremely inaccessible.

Even for advanced users, the entire simulation pipeline is bloated and inefficient. In many cases, the stock Python analysis scripts do not provide enough functionality to cover all the possible analysis cases. For advanced data generation and analysis, researchers are tasked with writing their own scripts. This is less than desirable in a research context where time could be better spent in other areas. The lack of process unification and simplicity also takes a toll on this useful software's accessibility.

2.0 Objective

The objective of this project was to wrap oxDNA's functionality in a simplified user interface that is accessible via the Internet. The problems mentioned in section 1.0 have mostly been solved by the final product. We feel the server and front-end have been able to effectively tackle the overall issue of accessibility. This project provides a unified portal to oxDNA functionality, as well as easier-to-use and fully-functional analysis tools (provided via built-in and user-defined analysis functions) which ensures a smooth user experience.

3.0 Background

3.1 Key Concepts

Undoubtedly, the most important technology this project works with is the oxDNA software, a molecular dynamics and modeling program used to compute complex simulations of nucleic acids. The software includes features for several different types of common simulations. These include: molecular dynamics, Brownian dynamics, and Metropolis Monte Carlo. Additionally, Lennard-Jones interactions, Kob-Andersen mixtures along with multiple patchy particle models can be generated [1]. This rich feature set makes oxDNA very useful for researchers looking to utilize computing power for predicting how molecular systems will interact in a wide variety of scenarios.

In order to implement the required server functionality, Django Python in conjunction with a PostgreSQL database is used. Django is an open-source web-server framework for use with Python [2]. This serves as our API and Job execution platform. For the frontend client, we used Angular2, a renowned framework for developing websites using HTML, CSS, and Typescript. It follows the Model-View-Template architecture with the goal of ease-of-use in mind. Angular2 is widely supported and extensible via open source packages that are downloadable and instantly available via the Node Package Manager (NPM). This flexibility allows easy conjunction with 3rd party 3D visualizers which are used to analyze simulation outputs.

oxDNA produces a trajectory file where all the relevant information to viewing DNA reaction simulation images lies. This info is translated into a pdb or xyz file using a converter provided in the "UTILS" directory. This file is analyzed in HTMoL, a molecular visualization program for displaying 3D representations of molecular systems [3], from xtc output format which is generated using another popular chemistry software called Gromacs. Gromacs is open source and developed by a team of researchers from around the world and provides conversion from pdb format to the standard xtc format [4]. HTMoL was developed by researchers for embedding 3D visuals into webpages and its documentation and download is free of charge for academic use. Some of its key features include but are not limited to automatic identification of atom types, high-resolution images, and molecular dynamics trajectory playback (xtc and pdb formats).

3.2 Related Work

The foundational research for DNA self-assembly systems was originally carried out by Erik Winfree. Winfree developed a model for artificial, self-assembling systems called the Tile Assembly Model which he introduced in his 1998 thesis [5]. This model could be split into two versions, namely the abstract Tile Assembly Model and the kinetic Tile Assembly Model (aTAM and kTAM respectively). Between the two versions of the model, the key differences lie in the fact that the aTAM version generally ignores realistic errors and provides a more high-level approach, while the kTAM version accounts for errors and provides means to analyze errors that can happen in reality. Because of this accounting for errors, the kTAM model has been used in actual lab experiments to predict the ways assemblies will form. Winfree's research into these types of models showed that the systems could be classified into the field of algorithmic self-assembly. Different simulators have been developed using these models. Xgrow, developed by Erik Winfree among others, actually implements both kTAM and aTAM [6]. Another simulator, ISU TAS, similarly features both assembly models, but is more focused on designing tile assemblies [7]. These softwares offer a more high-level simulation of self-assembly systems when compared with the oxDNA software which makes use of its own molecular dynamics model.

oxDNA itself was originally based around the research of T.E. Ouldridge, J.P.K. Doye, and A.A. Louis, who introduced the coarse-grained DNA model [8]. oxDNA has since been expanded by multiple research groups to be a framework for simulating and analyzing DNA and RNA. However, oxDNA is not the only molecular dynamics simulation software currently available. Another nucleic acid simulation software is Nanoscale Molecular Dynamics (NAMD), created by researchers from the University of Illinois at Urbana-Champaign; This software features scalable simulations that can utilize hundreds of processing cores in order to model massive molecular systems [9]. Similar to oxDNA, in order to correctly utilize NAMD requires extensive technical knowledge of the subject as well as a solid grasp of programming concepts. These requirements are generally too demanding of researchers not from a computer science background and causes unnecessary delays in getting simulations up and running. The development of a user-friendly graphical interface to oxDNA should mitigate many of these issues and attract more researchers to the software.

4.0 Design

4.1 Requirements and Design Choices

The culmination of the WebDNA software have accomplished the following:

- Provides the user with an interface for generating input data to a simulation environment.
- Allow users to control the execution parameters of a simulation. This includes, but is not limited to: simulation type, initial seed, time steps, and temperature. These parameters are based on the generic/simulation options
- Displays simulation progress and provides a visualizer to view the current state of the simulation.
- Provides a visualizer to render the system state at different points in time.

- Provides an analysis pipeline editor. Such an editor would allow users to perform analyses on the output of the simulation, mapping and/or reducing system states to other forms of analysis data. See below for more details.

One of the highlights of the aforementioned design choices is the *analysis pipeline editor*. This serves as a solution to the cumbersome analysis process that DNA researchers undergo every time they want to extract or calculate new information from simulation results. As mentioned in section 1.0, current analysis solutions require manually writing python scripts to analyze execution data. The *analysis pipeline editor* gives users the ability to visually pipe the raw simulation output through a series of scripts such that they can quickly process simulations meaningfully. See Figure 1 for an example of such a pipeline.

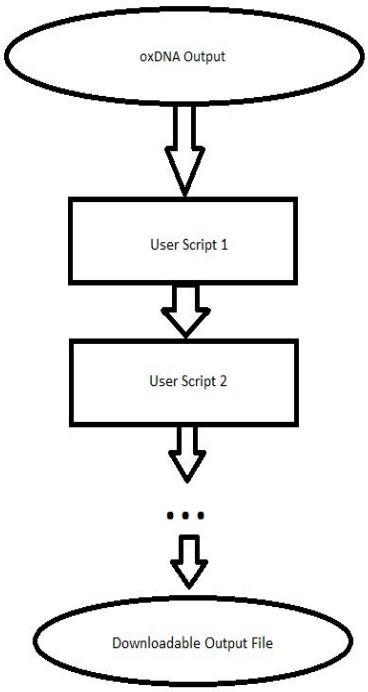


Figure 1. Mockup of the *analysis pipeline editor*. The WebDNA pipeline editor provides the ability for users to specify custom scripts to be executed in order.

The variety of analyses that may be performed is great, so it is conceivable that no visual editor can cover all the cases researchers may need to execute. In this case, it is useful to provide a "custom script" operator, where a researcher who is savvy with Python can manually perform a step of the pipeline without having to rely on built-in tools. We originally planned to allow Community Scripts where users could share scripts with each other, however this functionality did not make it into the final project. Community scripts are discussed in more detail later in section 4.2.2. The semantics of the custom script functionality imply script sandboxing, since users' custom scripts will be executed on our servers (which is a blatant system vulnerability if not properly safeguarded). This functionality was found to be very

difficult to correctly implement to a useful degree and so was also removed from the final project in the interest of implementing core functionality.

The implementation of the remaining key requirements mainly revolved around web application development. For example, all simulation configurations must be stored on a central server, then served to the user so they can view and modify the configurations. This data flow is also necessary for simulation visualizations, which are stored on the server and served to the user, rather than performing intense computation on the user's end. Section 4.2 covers the implementation details for these broad features.

4.2 High Level Architecture

As mentioned in section 4.1, the server and client perform distinct and separate tasks in order to fulfill the end-user requirements and functionality. Essentially, the client is a "pretty terminal" view on the server, granting a limited view of oxDNA execution configurations and output data. The server is responsible for handling requests for custom executions, performing standard parameterized oxDNA simulations, as well as executing utility scripts, both built-in and custom-uploaded by users. To further explain, we will cover the server architecture in section 4.2.1 and the client architecture in section 4.2.2.

4.2.1 Server Architecture

The server provides request endpoints to perform the following actions (the list is not comprehensive, but highlights the main requests clients will make):

1. Create and save simulation configurations and parameters.
2. Fetch previously saved simulation configurations and parameters.
3. Begin the execution of an oxDNA simulation based on a previously saved configuration.
4. Fetch visualization data (i.e. system state data) for a currently executing or previously executed oxDNA simulation.
5. Request the execution of previously made python scripts on generated input files.
6. Request the execution of premade/custom scripts on output data. The server provides a suite of generic analysis scripts that users can chain together to obtain meaningful, customized results.

The server runs using Python Django. The Django server is capable of redirecting HTTP/REST requests to python functions that can perform the actions listed above. Thus, Django comprises our RESTful web server for this project.

Data is stored in a PostgreSQL database, with the following data types:

- **User**: Contains standard user data, such as email, name, password hash, salt for the hash, etc. Users also possess Projects.
- **Project**: Contains simulation execution configuration data and references to current/past Jobs.
- **Job**: Contains information about a currently executing or previously executed simulation using the "oxDNA" executable. Jobs store much of the same information as a Project.

- **Script**: Contains information about a script that should be scheduled for execution by the server. Elements of this table contain data such as file names and script descriptions which are specified by users upon script upload. Scripts were originally planned to be specified as either private or public. Public scripts would appear in queries for community scripts, whereas private scripts would stay unique to the user, with the option to make them public at any time. This functionality was unfortunately not implemented.

The following UML diagram represents the basic Schema for each of these main data types as implemented in PostgreSQL:

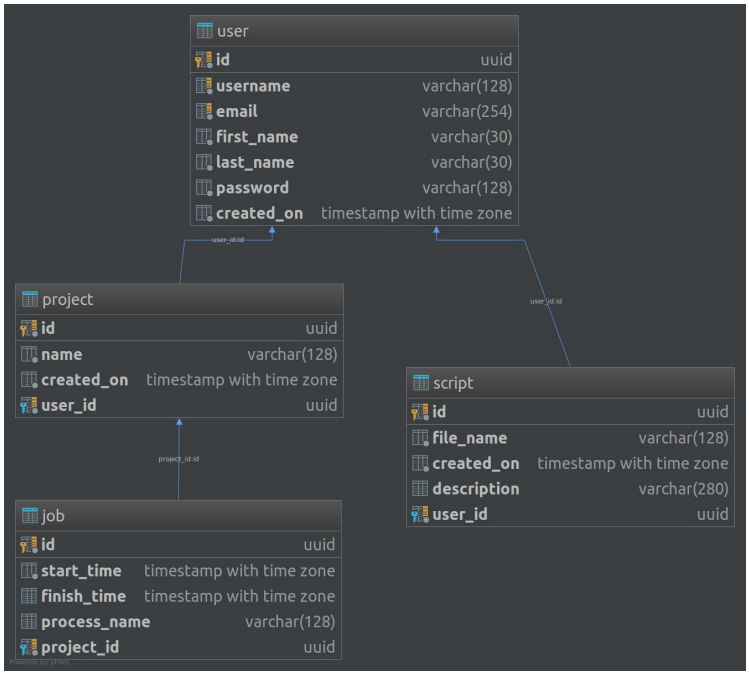


Figure 2. The WebDNA Database Schema in PostgreSQL. This UML diagram summarizes the key tables and relations used in the database.

A few of the structures of these tables have changed from the original proposal, but these four tables are still good representations of the most important aspects of the server's data storage.

Executing an oxDNA process involves the use of some sort of task queue, allowing us to execute oxDNA processes automatically, as well as check on their output command-line output and current state (i.e. running, errored, finished, etc.). Celery for Python is a Python module that wraps and abstracts other task queues like "redis", "rabbitmq", and others. Celery offers an API for executing bash commands in this manner. We have set up our REST endpoints to trigger the execution of various Celery functions that launch executions, run analysis pipelines, and convert output files to various formats. All of this is automatically load balanced which made implementing the main task queue for the server much simpler.

The final server functionality that was supposed to be implemented was the execution of client-defined Python scripts for performing processing on output data. We wanted to ensure that both our users and our server were kept safe from malicious scripts. This was planned to be implemented by using a "jailing" library, which would allow execution of scripts in sandboxed (or "jailed") environments, where we could control the system-level commands available to the jailed Python interpreter. Upon further research, this methodology would have cost too much time to implement correctly and was revealed to be still vulnerable to advanced attacks. Because of this, the functionality was not implemented in the final project.

4.2.2 Client Architecture

Our front end solution to the oxDNA simulation software contains, amongst other things, a web page that allows the user to create Projects. Projects, from a user standpoint, allow users to isolate distinct configurations and simulation results into manageable groups. Project creation/execution involves three main components: the *input configurator*, the *simulation visualizer*, and the *analysis pipeline*. We expedited the web development process by using one of many Angular2 website templates that provide visually pleasing components and layout functionality, so that we can focus more deeply on client functionality as opposed to frequently trying to figure out layout and visual issues.

With respect to the user experience and application layout, the project components will be split into two distinct pages per project; one page will contain only the *input configurator*, while the second page will contain the *output pipeline*. The *simulation visualizer* will live in a separate area of the website. This is because running a simulation is a big, time-consuming job, and should therefore be treated functionally as a "job submission" rather than a quickly adjustable visualization. This is not to say that a simulation could not be cancelled, adjusted, then restarted, but rather that a two-page layout emphasizes the magnitude of the work being done.

The *input configurator* contains settings of the following categories:

1. Generation Options: A module that will allow the user to upload a file of DNA sequences, as well as specify a box size for the default system generation.
2. Generic Options: A module that asks for generic options for the simulation. These are based on the "Generic Options" listed in the oxDNA documentation [10].
3. Simulation Options: A module that asks for custom simulation-specific options. These are based on the "Simulation Options" listed in the oxDNA documentation [10]. Some options have prerequisites for other option states before the original option can be set.
4. Molecular Dynamics Simulation Options: These options are specific to the execution of a Molecular Dynamics (MD) simulation. This is the only simulation type that WebDNA currently supports.

The settings and scripts modified in the *input configurator* are saved to their respective Project associated with a user account. Figure 3 on the next page shows a mockup interface for the *input configurator*.

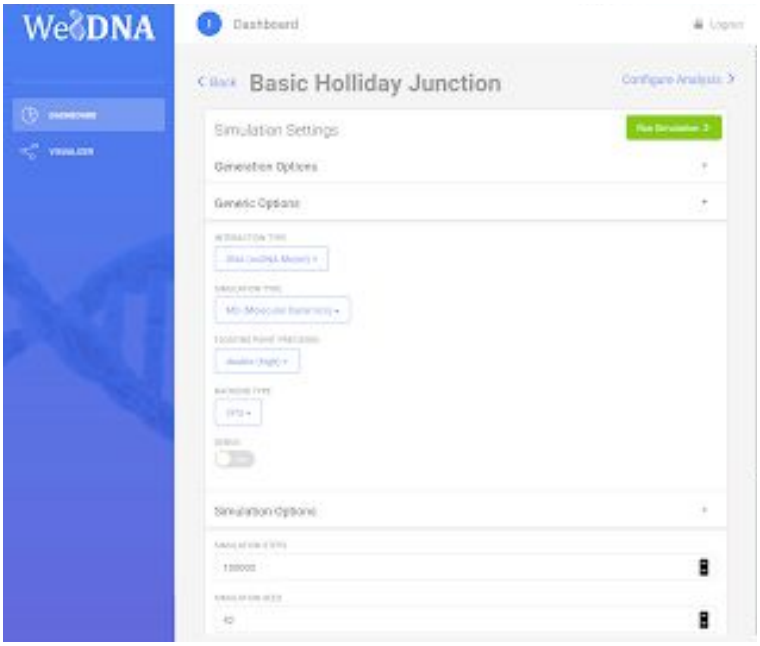


Figure 3. A mockup of the *input configurator*, the first of two pages for all projects.This represents the configuration settings of an already-created project.

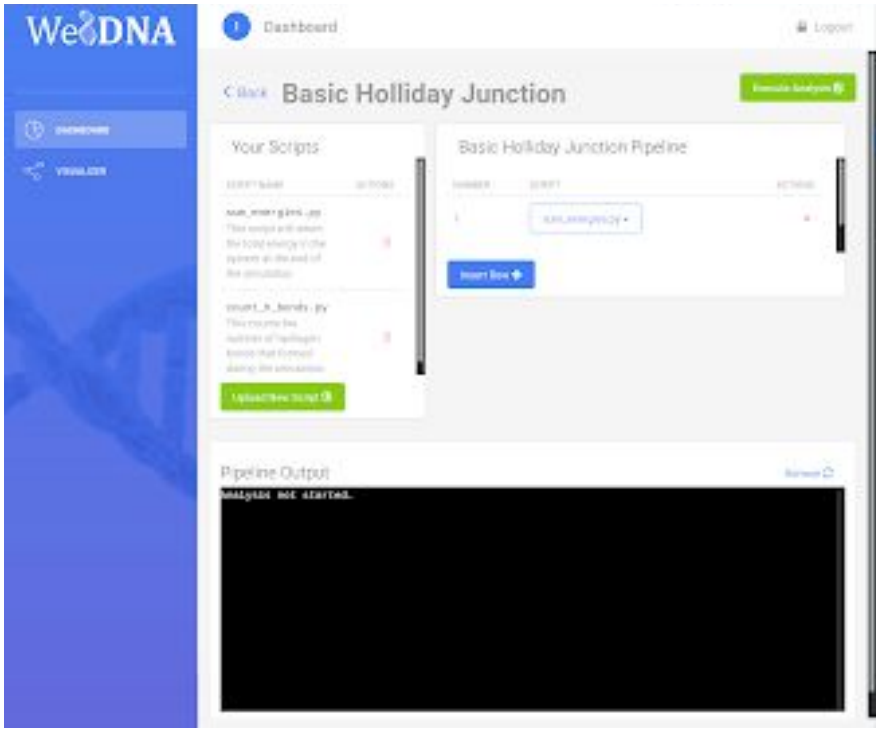
The next major component of a Project relates to the *simulation visualizer*. There are a couple semantics relating to simulations for projects in WebDNA, listed below:

- Simulations are slow, so the visualizer should display simulations as they are being calculated in near-real-time.
- All simulations ─ finished or not ─ should allow backward scrubbing to previous points in the simulation, if they were configured to save the system state throughout the simulation.

The visualizer we have chosen for our DNA visualizations is HTMoL (see [3]), which renders the DNA system state using WebGL. It is an MIT licensed project, so it makes a perfect fit into our similarly-open-source project.

The final component, the *analysis pipeline*, allows the user to specify a sequence of analysis scripts, which is created using a simple table interface to build a pipeline that specifies scripts to be executed in order. Figure 1 in section 4.1 gives a good visual for what this pipeline can look like for an example project. The specific requirements for this pipeline aren't entirely clear, as there are so many potential use cases. So to keep the functionality as general as possible, we simply provide guidance as to how users should expect to program their analysis scripts.

Figure 4 shows an example of the analysis editor



Unfortunately, users who are not as tech-savvy will likely not be able to make use of the custom script feature as it requires that they know Python programming. These uploaded scripts are made available for all of a user's projects to ensure maximum convenience. Python scripts uploaded by the user should include metadata specified on the front end that the server will then parse to give the script useful information. For instance, the purpose of a script in the repository named "generated_strands_23.py" with no other information might not be clear to user a few months after they upload it. On the other hand, a script with the name "Weighted Strand Generator" with a specific and helpful description describing its functionality and intended purpose will make it much easier to remember how to use and increase its usefulness for multiple projects.

4.3 Risks

Risk	Risk Outcome
Learning curve for the Custom Scripts Module	We wanted to make the custom scripts module more user-friendly and robust, however, due to poor time management, we had to limit ourselves to a fairly hands-on approach for end-users. We still find that the interface WebDNA provides cuts down on a lot of the hard work researchers undergo to perform simulations.
Unmanageable codebases	We are proud of much of the detail that has gone into implementing the server-side and client-side codebases. We feel that they will be easily extensible with proper high-level documentation.

File upload security risk	Due to similar constraints with the first item of this table, we do not screen scripts for safety. We are ultimately trusting that the researchers using the software will not run malicious code on their own servers.
Visualization setbacks	We encountered much trouble when trying to implement the client-side visualizer. We have since gotten it working by modifying the HTMoL source code to our needs, as well as using programs from oxDNA and Gromacs to aid in the process of generating visualization files that are friendly to HTMoL.

4.4 Tasks and Schedule

Below we have listed detailed tasks that were implemented over the course of this semester. These tasks are divided into two types: Server and Frontend. We initially wrote these tasks in a Trello board to keep ourselves organized, and have translated the tasks and their descriptions to this proposal document. The tasks are ordered in the order we presented in the "Done" list on our Trello board. The tasks listed below are also far more verbose than what we within our last report; with included completion dates.

Server Tasks

Task and Description	Member(s)	Due Date (2018)
Setup Server Hosting Setup server hosting on university owned server.	Jace	Done! Mar 11
Initialize Database Schema on Hosted PostgreSQL Database The UML for the database is shown in Figure 2 of section 4.2.1. The scripts have already been created, they will just need to be executed on the new server environment.	David	Done! Feb 7
Endpoint: POST User Authentication Users are authenticated by providing a username||email and password combination to a POST endpoint, then receive a JSON Web Token (JWT) if authentication passed. This token is used to approve all further requests to the API from that user.	Jace	Done! Feb 12

Password hashing for storage and retrieval to the database is handled by the PostgreSQL package "pgcrypto", using their implementation of bcrypt. More information for setting this up can be found here.		
Add "project" table to database-definition repo We came to an agreement about the project table's necessary columns, then committed it to the repo.	Jace, Zack	Done! Feb 12

Endpoint: GET Projects
This endpoint returns all of the user's project's top-level information for display on their project dashboard. Each object only contains simple information, such as the following:
`{`
`"id": …,`
`"name": ...,`
`"created_on": ...,`
`"job_running": ...`
`}`
	Zack	Done!
		Feb 12

Endpoint: POST User Registration This endpoint needs to do the following things: ● Checks that the username is not already in use, else responds with ErrorResponse ● Checks that the email is not already in use, else responds with ErrorResponse ● Checks that the password meets minimum strength requirements (your pick), else responds with ErrorResponse ● Hashes the password with the bcrypt methods added in the attached Merge Request ● Responds with AuthenticationResponse, added in the attached Merge Request	David	Done! Feb 12
Celery: Initial Setup In order to set up Celery for use with the Server the following tasks were carried out: ● Installed RabbitMQ ● This is our message broker and is the default for Celery	David	Done! Mar 11

• Installation instructions can be found here • Installed Celery in our virtual environment using Pip • Installation instructions can be found here • Once installation was completed, files were added to the Django server to implement the Celery library, specified in the getting started guide linked above it to the repo.		

Celery: Set up basic task scheduling
Once Celery had been installed, we needed to set up worker initialization and create a task for launching simulations. Tasks can be defined as functions and are decorated with a Celery API decorator. Documentation can be found here
These were defined in a file: tasks.py
An example:
`from .models import User`
 	
`@app.task`
`def create_user(username, password):`
`  User.objects.create(username=username, password=password)`
	David	Done!
		Mar 31

Add "Job" model to server Defined the "Job" table for the database	Jace, David	Done! Mar 31
Endpoint: GET Project Configuration Data The server needed to return the Project configuration data for a single project by project ID. This GET request essentially serves the stored project configuration JSON data in the body of the request's response.	Salvador, Jace	Done! Apr 25
Endpoint: PUT Project Configuration Data The frontend is able to upload files to our server according to the type of file it is, according to the following types: Sequence file (labelled strands)	Salvador, Jace	Done! Apr 25

External forces file
Sequence Dependent Parameters
The server can accept the uploads of these files and store them in the following fashion in some data directory:
`/data/{project_id}/{file_type}.dat`
 	
This file path should be saved in the Projects data file, which will be stored as follows:
`/data/{project_id}.json`
 	
Project Configuration JSON Example:
`{`
`"sequence_file": "/data/{project_id}/sequence.txt",`
`...`
`}`		

Map between ProjectSettingsSerializer and input.txt Provided endpoints and functionality for the following: 1. POST: Generate an input.txt given a project_id and a serialized set of simulation options (ProjectSettingsSerializer) 2. GET: Return a ProjectSettingsSerializer data from an existing input.txt, fetched by project_id	Salvador	Done! Apr 3
Endpoint: GET Check Job Status An endpoint specified for checking the current status of an executing simulation. This returns a boolean of whether the project is currently executing along with stdout log file for the project if it exists.	David	Done! Apr 3
Celery: Output Piping Once a Celery task is implemented to launch simulations, it was important to be able to connect to the stdout pipe to be able to provide the user with updates about a launched oxDNA simulation. Information about connecting tasks to standard output pipes can be found on the Celery documentation page See the following example: `import sys`	David	Done! Apr 1

logger = get_task_logger(__name__)
 	
@app.task(bind=True)
def add(self, x, y):
  old_outs = sys.stdout, sys.stderr
  rlevel =
self.app.conf.worker_redirect_stdouts_level
  try:
 	
self.app.log.redirect_stdouts_to_logger(logger,
rlevel)
  print('Adding {0} + {1}'.format(x, y))
  return x + y
  finally:
  sys.stdout, sys.stderr = old_outs		

Celery: Task Dictionary We needed to implement a dictionary to map Celery task ids to job ids stored in our PSQL database. Django was able to carry out database migrations automatically, so it did not require much configuration. This dictionary is used to keep track of what stored jobs are currently being executed on the Celery task queue.	David	Done! Apr 3
Endpoint: POST General File Upload to Project Endpoint to be able to upload any file necessary to the designated path within the project directory of a single user	Salvador	Done! Apr 2
oxDNA generation files utilities Generate generated.dat and generated.top when a sequence.txt file has been properly uploaded	Salvador	Done! Apr 25
Endpoint: GET to showcase any file available in project directory Created endpoint so that any file within a user's project directory can access the complete literal string value of the file. No files can be retrieved if the project doesn't exist, or if the project is executing a simulation.	Salvador	Done! Apr 25
Leverage Gromacs to Convert PDB files to XTC Trajectories for Use with HTMoL	David	Done! Apr 25

Gromacs has the capability of converting PDB files with multiple frames to XTC files which HTMoL makes use of.		
Endpoint: PUT Analysis script Uploaded a user created script into the newly created or existing file system under "{user_id}/scripts"	Salvador	Done! Apr 25
Script Checking for Script Upload Upon receiving, the upload of a custom script, we will run security scans. If the user has opted to make their script publicly available, the python file will be scanned for documentation and metadata (things such as script name, version, description, parameter types, etc.) to make for more descriptive community scripts.	Jonathon	Redacted May 6
Execute oxDNA Simulation from Project Data The user is be able to submit a Job for queuing to the server, which is initialized using only the desired project_id. An instance of oxDNA is executed according to the parameters specified in the Project corresponding to the supplied project_id. The user is also able to query the current progress of the simulation via another GET endpoint. This endpoint returns some state information including any output from oxDNA.	David	Done! Apr 1
Create Zip Archives for XTC and PDB Trajectories The original endpoints to get the executing simulation's current trajectories only return a single PDB file. It was more useful to convert the PDB file into an XTC file and create a cached ZIP archive of both to send as a response to the client.	David, Jace	Done! Apr 25
Create "Script" table to store necessary information about a script pipeline "Script" table with the fields id, file_name, user_id, created_on, description	David	Done! Apr 25
Convert multi-frame PDB to individual PDBs, then convert to XTC with Gromacs	Jace	Done! Apr 26
Endpoint: Modify DELETE Project to delete project directory Also, stop the oxDNA task if currently executing.	David	Done! Apr 26
Endpoint: get list of scripts, custom scripts,	Salvador	Done!

input options, and output files To enable the frontend to create selectors on which the user can build their analysis command with		Apr 26
Endpoint: GET download project file Endpoint to download complete files from any directory within the user's respective project	Salvador	Done! Apr 26
Celery Task: Execute analysis scripts in order 1. oxDNA executes 2. oxDNA writes trajectory.dat, energy.dat, etc... 3. User requests execute of analysis pipeline. 4. First script requests access to the information in trajectory.dat, etc. 5. First script returns a Python object with results from first analysis step 6. Subsequent script executes with previous output as input 7. Script returns *its* output 8. Repeat step 7 for all scripts Scripts run with access to the oxDNA files and a single string of command-line inputs, specified by the user in the frontend. Script table: ● id ● fileName ● commandLineArguments	David	Done! Apr 29
Endpoint: GET User Script Output File Users will now be asked to write to an output file with a standard name "output.txt". This output is made available for viewing/download via this endpoint.	David	Done! Apr 29
Endpoint: GET User Script Log An endpoint created to return the log file for user scripts which will contain their stdout along with any error messages generated during execution.	David	Done! Apr 29
Endpoint: POST Specify Analysis pipeline An analysis pipeline can simply be a list of distinct	David	Done! Apr 30

nodes.
`{`
`  "pipeline": [`
`  { /* analysis node */ },`
`  { /* analysis node */ },`
`  ...`
`  ]`
`}`		

Endpoint: Cancel an execution	David	Done! Apr 30
Endpoint: PUT scriptlist.txt	David	Done! Apr 30
Script Upload: Create row in Script table with corresponding description on script upload	Salvador	Done! Apr 30
Endpoint: Run analysis pipeline using output from already-finished simulation	David	Done! Apr 30
Endpoint: GET list of scripts from database	David	Done! Apr 30
Endpoint: DELETE script	David	Done! Apr 30
Endpoint: Download zipped project Take all the files within the project file system and place it within a zip file	Salvador	Done! Apr 30

Client Tasks

Task and Description	Member(s)	Due Date (2018)
Login/Registration Page Login: Designed a clean and simple login page which allows users to input usernames and passwords with clear input elements for each field, along with a submit button which sends the appropriate login request to the server.	Zack, Jace	Done! Mar 31

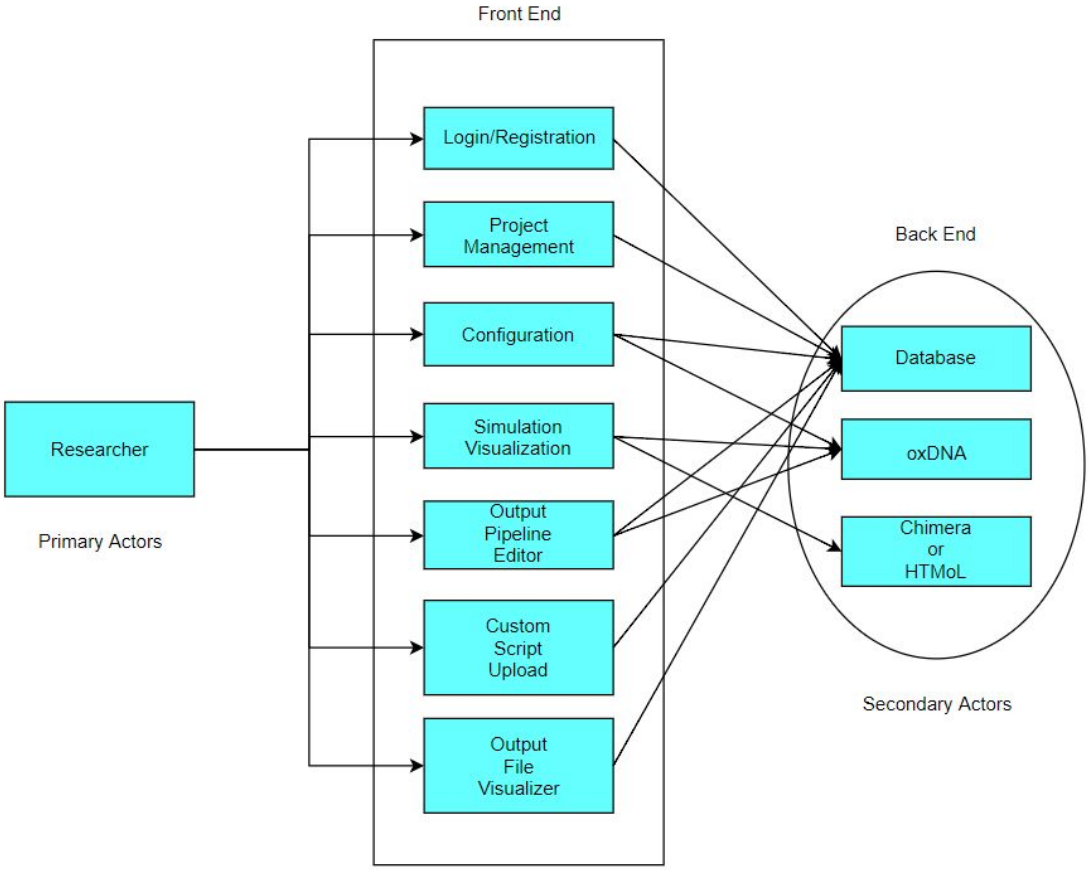
Registration: A register button changes the page's form to provide elements for entering necessary registration information (i.e. first name, last name, username, password). Appropriate requests are made to the server once the form submit button has been clicked.		
Project Management Page Project List: Designed a well organized list of projects associated with the currently signed-in account. Projects listed on this page have been created previously by the user. Project Options: Each project provides several options when clicked: "Visit Project Page", "Delete Project", and "Download Options". "Visit Project" takes the user to the selected project's specific configuration page. "Delete Project" provides a secondary popup box asking the user to confirm a projects deletion from their account. "Download Options" opens a small drop-down list of different files pertaining to the project that can be downloaded including project configurations and simulation outputs.	Jace	Done! Apr 1
Configuration Page Allows the user to configure a project for simulation.	Jace	Done! Apr 3
Show Log from Project Table ("View Output" button)	Jace	Done! Apr 3
Execute simulations from frontend: Setup endpoints and click events	Jace	Done! Apr 3
Check for user's JWT in cookies, then load the Dashboard	Jace	Done! Apr 25
Modify Output/Log Component to be contained in a auto-scrolling, fixed-height container	Jace	Done! Apr 30
Output File Visualizer This is based on the HTMoL visualizer, as mentioned in section 4. This made it both aesthetic and functional changes to the Visualizer, including: ● Allowing users to toggle automatic refresh of the scene (if the project is configured to output data for every step). ● Allowing users to download the raw data of the configuration (this will already have been	Jace	Done! Apr 30

loaded on the user's computer, since that raw data was needed to populate the visualizer).		
Output Pipeline Editor This was tricky, but we needed to give the user a visual way to edit an analysis pipeline.	Jace, Zack	Done! Apr 30
Custom Script Upload Manager This is a simple page that is included in the "Scripts" page of the Dashboard, with the option to upload python scripts generated by a researcher.	Jace, Zack	Done! Apr 30
Dashboard (Integration of All Pages) The Dashboard is the central location for all project management and script management. The user has access to their project to enable them to configure, visualize, and analyze their projects.	Jace	Done! Mar 31
Documentation Final documentation is written in detail to cover all client functionality for end users. The documentation is accessible for users without any sort of computer science background so that the majority of researchers are able to get up and running quickly.	All members	Done! Apr 30

4.5 Deliverables

- Design Document
    - This document (this very report) outlines front end functionality, database schema, and the scope of analysis.
    - Essentially, this details the functionality and architecture of the completed project to allow those unfamiliar with the project to understand it and give a good overview of the various frameworks involved.
    - This document serves as the Design Document deliverable.
- API/Job-running Server
    - This server is implemented using the Django REST Framework.
    - Tthe server provides all of the REST endpoints required to support the following functionality:
        - User registration
        - User login
        - Project configuration
        - Run simulations
        - Simulation visualization
        - Output analysis
        - Custom script execution/uploads
    - This server is also be where user simulations are executed using the Celery task queuing library.
- Website
    - Angular2 framework and Base Template (see [13]).
    - HTMoL used for molecular simulation viewing.
    - Users are able to experience the full life-cycle of a project - from creation to output statistics and visualisation.
- Data Analysis
    - The project provide users with an easy to use data analysis pipeline.
    - This is implemented through the aforementioned website and API server.
- Final Documentation
    - This is a detailed document meant to be read by a non-computer scientist that provides instruction on the use of the WebDNA front end for oxDNA simulations.
    - All group members contributed to the documentation for the respective tasks they completed.
    - The final documentation is detailed enough for someone completely unfamiliar with the project to be able to gain a solid understanding of the underlying frameworks and codebase.
    - List of api packages involved

5.0 Use Cases



5.1 Login/Registration

Use Case	Login/Registration
Author	Zack and Jonathon
Primary Actor	Researcher
Goal in context	For the Researcher to sign in or sign up
Preconditions	Database setup and GUI using Angular
Trigger	Upon first visiting page
Scenario	1. Researcher: Heads to URL of website

Exceptions	1. User does not exist in sign in 2. Account name already exists in sign up
Priority	Low
Channel to Actor	GUI
Usage Frequency	At each session use
Secondary Actors	Database
Channels to secondary actors	Database: Network
Open Issues	Security against password guessing and bots

5.2 Project Management

Use Case	Project Management
Author	Jonathon and Salvador
Primary Actor	Researcher
Goal in context	Help manage (edit and organize) multiple project simulations with essentially a file system with folders
Preconditions	Database setup
Trigger	Upon first logging in or being chosen from dashboard
Scenario	1. Researcher: logs in 2. Database: retrieves all projects of the logged in user
Exceptions	1. If user tries to delete a project, then the user will be asked by a pop-up if they actually want to do so
Priority	Medium
Channel to Actor	GUI
Usage Frequency	Any time the user logs in or goes clicks back

	through the dashboard
Secondary Actors	Database
Channels to secondary actors	Database: Network
Open Issues	Ability to make folders for better organization

5.3 Configuration

Use Case	Configuration
Author	Zack and Salvador
Primary Actor	Researcher
Goal in context	Generate an Input file for the execution of the simulator and facilitate changes (extra files) to the working directory within the project manager
Preconditions	Simulation software (oxDNA) be implemented and routed
Trigger	User chooses a project in the project manager
Scenario	1. Researcher: logs in 2. Database: retrieves projects 3. Researcher: chooses a project
Exceptions	1. Insufficient information to run simulation 2. Input variables are invalid
Priority	High
Channel to Actor	GUI
Usage Frequency	Each time a project is chosen within a user account
Secondary Actors	Database, oxDNA
Channels to secondary actors	Database: Network oxDNA: Network
Open Issues	Validation of sufficient and valid inputs

5.4 Simulation Visualization

Use Case	Simulation Visualization
Author	Jace and Zack
Primary Actor	Researcher
Goal in context	Have visualization software embedded into the webpage to visualize the simulation
Preconditions	Implementation and routing of visualization software and having output files to work with from execution of oxDNA
Trigger	When the pending execution screen is finished, the output files from oxDNA will give the data necessary for the visualization software
Scenario	1. Researcher: logs in 2. Database: retrieves projects 3. Researcher: chooses project 4. Database: sets up files for configuration and execution 5. Researcher: configures and executes their project 6. oxDNA, Visualization: outputs the visualization of the executed configuration
Exceptions	User tries illegal operations of the visualization software
Priority	High
Channel to Actor	GUI
Usage Frequency	After each execution of a project
Secondary Actors	Visualization, oxDNA
Channels to secondary actors	Visualization: Network oxDNA: Network
Open Issues	Having JSON files of the iterated steps of the simulation executed

5.5 Output Pipeline Editor

Use Case	Output Pipeline Editor
Author	Salvador and David
Primary Actor	Researcher
Goal in context	To create a drag and drop output analysis flow chart that will treat each block as a script. The script functioning on parameters from the flow going in and giving its output to the flow going out.
Preconditions	Pre-existing python scripts needed, along with executed simulation output to do analysis on
Trigger	Upon finishing simulation execution
Scenario	1. Researcher: logs in 2. Database: retrieves projects 3. Researcher: chooses project 4. Database: sets up files for configuration and execution 5. Researcher: configures and executes their project 6. oxDNA, Visualization: outputs the visualization of the executed configuration 7. Researcher: clicks on the icon to perform the output analysis from the dashboard
Exceptions	Invalid placements of the scripts in the flow chart
Priority	High
Channel to Actor	GUI
Usage Frequency	After each execution of the simulation if the user decides to click on the icon on the dashboard
Secondary Actors	Database, oxDNA
Channels to secondary actors	Database: Network oxDNA: Network

Open Issues	1. How to handle aggregate of python scripts 2. How to handle all the different outputs aggregated (make many flow charts?)

5.6 Custom Script Upload

Use Case	Custom Script Upload
Author	Jace and David
Primary Actor	Researcher
Goal in context	To upload the user's custom output analysis scripts for either their use only or to share with anyone with an account
Preconditions	Database setup
Trigger	Selected from dashboard
Scenario	1. Researcher: logs in 2. Database: retrieves projects 3. Researcher: clicks on an icon on the dashboard to go to the custom script upload page
Exceptions	1. The script is a security risk 2. The script doesn't follow our requirements (such as a description) if the user chooses to make the script public
Priority	Medium
Channel to Actor	GUI
Usage Frequency	Anytime user selects the script upload icon on the dashboard
Secondary Actors	Database
Channels to secondary actors	Database: Network
Open Issues	1. How many requirements 2. The scope of the security scan

5.7 Output File Visualizer

Use Case	Output File Visualizer
Author	David and Zack
Primary Actor	Researcher
Goal in context	Being able to take the output from the output analysis flow diagram that was executed, and display the text files and terminal prints given from it
Preconditions	Implementation of the output analysis flow chart
Trigger	Execution of the configuration of a project and then executing a valid output analysis flow diagram
Scenario	1. Researcher: logs in 2. Database: retrieves projects 3. Researcher: chooses project 4. Database: sets up files for configuration and execution 5. Researcher: configures and executes their project 6. oxDNA, Visualization: outputs the visualization of the executed configuration 7. Researcher: clicks on the icon to perform the output analysis from the dashboard 8. Researcher: executes output analysis pipeline 9. Server: displays output of pipeline
Exceptions	User cannot interact with display
Priority	High
Channel to Actor	GUI
Usage Frequency	Each time a user executes a flow diagram after the execution of a project
Secondary Actors	Database, Server
Channels to secondary actors	Database: Network Server: Network

Open Issues	1. Possibly taking in multiple diagrams at once 2. Organization of its display

6.0 Conclusions and Looking Forward

This project was designed and implemented to benefit any researches who use, or are looking to start using the oxDNA simulation software. Everything from its front-end user interface to its simple, but effective back-end was designed to streamline the process of setting up and running oxDNA simulations for people who would otherwise be forced to use the command-line software. We believe the finished project provides enough functionality and is simple enough to be usable by researches who might not have any prior programming or computer skills. The intended result of publishing this software is to introduce oxDNA to a wider audience and further the field of molecular dynamics.

Although the overall project was considered a success by team members, some important functionality wasn't successfully implemented. The main feature we missed was the ability for users to upload scripts to a community environment were scripts could be shared with others. This admittedly advanced feature simply wasn't possible given the timeline of the project. In addition, a good sandboxing environment wasn't implemented to ensure secure script execution. This means the server will have to be deployed only in "safe" environments where users can be trusted with arbitrary script executions. The most unfortunate result of this is the limitations it places on how the server can be deployed. It likely will not be able to be made publicly available by universities to avoid malicious scripting. Despite these drawbacks, we feel that the key functionality of being able to automatically execute custom scripts will still be very useful for users of the software. Another issue we ran into involved getting our visualizer to work correctly in our web environment. The oxDNA output files were not compatible with the HTMoL software by default. We ended up having to use a workaround method of using Gromacs to convert oxDNA output to the required xtc format. This conversion process is not without a few bugs that were not able to be resolved. These bugs can sometimes cause the HTMoL visualizer to work incorrectly and allocate a large amount of RAM on client computers. Additionally, the HTMoL visualizer currently only works correctly on Mozilla FireFox web browsers which is an obviously big problem for Chrome users. We were unable to mitigate these issues primarily due to a lack of understanding of the HTMoL source code.

We see many others areas with room for improvement as well. The layout of our REST API is not compliant with REST standards, and many of the endpoints simply dump the raw text content of files in data directories, which is a big load on the server-side, as well as on the client-side. For features like live console output, this alone can put a strain on the server, which constantly has to read and re-read a large output file to be sent to the user. We also note that we haven't made a strong effort to keep code well commented and specifically documented, which will make onboarding new developers more challenging. Finally, we wanted to be able to offer the most functionality possible with the least amount of user-effort when it came to the analysis pipeline, but we ended up requiring that the user write more boilerplate, complicated code to make their analysis scripts work. We plan on providing documentation about how

scripts should be written to handle the data flow properly, as well as to take full advantage of oxDNA's existing scripts.

7.0 Key Personnel

Dr. Matthew Patitz – Dr. Patitz performs research in self-assembling systems, and has published a number of papers related to theory, software, and experimental aspects of self-assembling systems [11].

Salvador Sanchez – Sanchez is a senior Computer Science and Mathematics major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Paradigms, Software Engineering, and Operating Systems. Front end development with Zabbix monitoring legacy PHP code, making server metric reporting pages for a Cerner systems engineering team.

Zack Fravel – Fravel is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed coursework in Embedded Systems, VLSI Synthesis, Low Power Digital System Design, Programming Paradigms, as well as independent study of Abnormal & Evolutionary Psychology.

David Darling – Darling is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Software Engineering and Programming Paradigms. He completed internships at Entergy Arkansas creating automated tools to streamline the design process for new electrical transmission structures.

Jace McPherson – McPherson is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has worked software development/engineering internships at Metova and Google, as well as researched and developed software to control multiple 3D printer robots to perform a single print cooperatively. He has an upcoming job as a software engineer at Google.

Jonathon Raney – Raney is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Programming Paradigms and Operating Systems. Currently undergoing further study in Genetics and Bioinformatics courses.

7.1 Facilities and Equipment

We wound up not requiring many resources for development. For the frontend web client, we were able to enormously speed up the process of our Angular2 website by purchasing a licensed template (at around $30-$40, see [12]). This template provided great visual components with immediate functionality and portability, so we didn't have to spend time nit-picking about component visuals and were able to focus more on overall layout needs for the site. Looking forward, researchers using the project will certainly

want a powerful VM with many CPU cores for hosting. Our provided VM with one processing core was usable, but certainly didn't speed up the testing process.

8.0 References

[1] oxDNA Features, https://dna.physics.ox.ac.uk/index.php/Features
[2] Django Documentation, https://docs.djangoproject.com/en/1.11/
[3] HTMoL Main Page, http://htmol.tripplab.com
[4] Gromacs, http://www.gromacs.org/About_Gromacs
[5] Self Assembly, http://self-assembly.net/wiki/index.php?title=Main_Page
[6] Xgrow, http://self-assembly.net/wiki/index.php?title=Xgrow
[7] ISU TAS, http://self-assembly.net/wiki/index.php?title=ISU_TAS
[8] oxDNA Documentation, https://dna.physics.ox.ac.uk/index.php/Documentation
[9] NAMD, http://www.ks.uiuc.edu/Research/namd/
[10] oxDNA Main Page, https://dna.physics.ox.ac.uk/index.php/Main_Page[
[11] UofA Directory, https://csce.uark.edu/directory/index/uid/patitz/name/Matthew-Patitz/
[12] Creative Tim, Light Bootstrap Dashboard Angular 2,
https://www.creative-tim.com/product/light-bootstrap-dashboard-angular2