University of Arkansas – CSCE Department
Capstone I – Final Proposal – Fall 2017

WebDNA Project Proposal

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

Abstract - Zack

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 DNA simulations, 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 standard simulation data, we also hope to build in analysis and visualization tools that share immediately useful information with the user. Our goal by the end of the project is to have made 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 - Jace

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 analysis scripts (written in Python) do not provide enough functionality to cover all the possible analysis cases. For advanced data generation and analysis, researchers are required to write their own python scripts, which is an overly advanced process. The lack of process unification and simplicity also takes a toll on this useful software's accessibility.

2.0 Objective - Jace

The objective of this project is to wrap oxDNA functionality in a simplified user interface that is accessible via the Internet. The problems mentioned in section 1.0 should ideally be solved by the final product. In summary, the software must solve the overall issue: accessibility. A unified

portal to the oxDNA functionality, as well as easier and fully-functional analysis tools (provided via built-in and user-defined analysis functions) will ensure a smooth user experience.

3.0 Background

3.1 Key Concepts - John, David

Undoubtedly, the most important technology this project will be working 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 will be used. Django is an open-source web-server framework for use with Python [2]. This will serve as our API and Job execution platform. For the frontend client, we will be using 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 will allow easy conjunction with the prewritten 3D visualizer using 3js which will eventually be used and updated to analyze simulation outputs. Our frontend will rely on the Django Server for all its data requests (served via the API portion) and Job queuing (performed by the Job execution portion).

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 can be analyzed in VMD, a molecular visualization program for displaying 3D representations of molecular systems [3], from xyz output format or by UCSF’s Chimera from pdb format to produce useful visualizations. Chimera is developed by the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco and provides extensive interactive DNA visualization [4]. Complete documentation and download is free of charge for noncommercial use. Some of its key features include but are not limited to automatic identification of atom types, high-resolution images, molecular dynamics trajectory playback (many formats), and distance and angle plots.

3.2 Related Work - David

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 Goals - Jace

The culmination of the WebDNA software should accomplish the following:

- Provide 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
- Display simulation progress and provide a visualizer to view the current state of the simulation.
- Provide a visualizer to render the system state at different points in time.

- Provide 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.
- Provide a repository of community data-generation/analysis scripts. Users who have uploaded custom scripts to the website for their personal purposes may share those scripts with other users of the software, if they so desire. A convenient “Script Repo” interface will be provided to the user, that lists scripts, with the option to view further details about the script’s functionality, use cases, and more. Saved scripts can be used in the user’s projects later.

One of the highlights of the aforementioned requirements 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* would give 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.

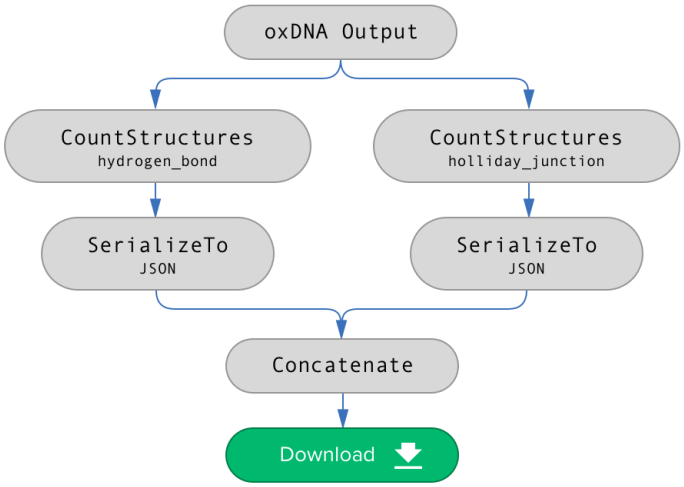


Figure 1. Mockup of the *analysis pipeline editor*. The WebDNA pipeline editor will provide default analysis nodes, as well as custom user-defined nodes.

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. In addition to this, users will be able to execute scripts submitted by other members of the community. 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).

The implementation of the remaining key requirements is highly dependent on the nature of 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** - Salvador and Jace

As mentioned in section 4.1, the server and client will be performing 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 needs to provide 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 sandboxed python scripts on generated input files.
6. Request the execution of premade/custom scripts on output data. The server should provide a suite of generic analysis scripts that users can chain together to obtain meaningful, customized results.

The server will be running with Python using Django. The Django server will be 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 will be stored in a PostgreSQL database. The following data types will be implemented:

- **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 (in the form of foreign key relationships from Jobs to Scripts). The Project table also contains output analysis pipeline data in the form of a linked list of python Scripts.
- **Job**: Contains information about a currently executing or previously executed simulation using the “oxDNA” executable. Jobs need to 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 need to know what Python script to execute, all parameters to that script, what files to use as input, where to save output, and if there are any subsequent Scripts that need executing. Scripts can be specified as either private or public. Public scripts will appear in queries for community scripts (as mentioned in the last design goal), whereas private scripts will stay unique to the user, with the option to make them public at any time.

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

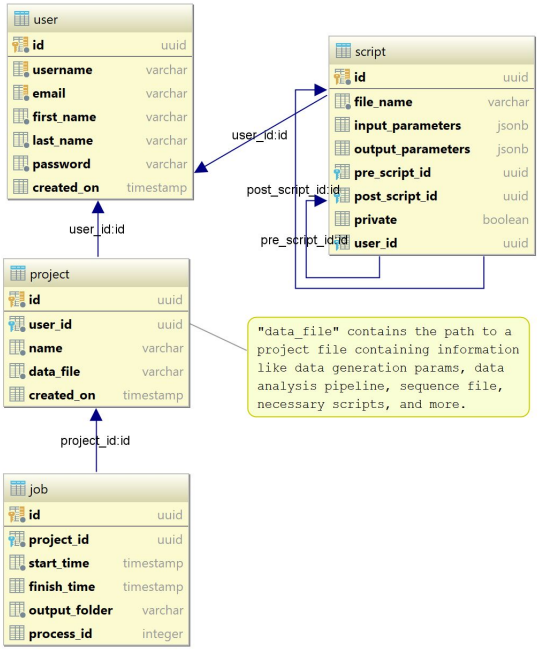


Figure 2. Database Schema in PostgreSQL. This UML diagram summarizes the key tables and relations needed in the database.

As project development continues, more tables will need to be implemented, and the structures of these tables may change from this original proposal, but these four tables represent the most important aspects of the server’s data storage.

4.2.2 Client Architecture

Our front end solution to the oxDNA simulation software will contain, 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 *output pipeline*. We can expect to expedite the web development process by using one of many Angular2 website templates that provide visually pleasing components and layout functionality, such that we can focus more deeply on client functionality as opposed to figuring 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 both the *simulation visualizer* and the *output pipeline*. 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* will be segmented into two categories: *required* and *optional* settings. Required settings include the following:

1. Sequence Input: A module that will generate a file of character-based sequences of DNA (or RNA) strands as the primary input or accepts a file/manual text input.
2. Generator Script: A module specifying the "generator", i.e. a Python script that will generate topology and configuration data from the sequence input (Required Setting 1).
3. 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].
4. 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.

Optional settings refer to any non-critical input-phase parameters that the user may want to make. These "parameters" are limited to modifications of the generated topology and configuration data created based on settings in the "Generator Script" setting (Required Setting 2):

1. Custom Scripts Module: This allows users to run pre-defined scripts (bundled with the oxDNA software on the backend) or run their own sandboxed Python scripts on the generated input data. Essentially, the user is allowed to transform the boilerplate generated topology and configuration files with these pre-defined/custom scripts. These scripts are stored with the user's account and can be made public.

The settings and scripts modified in the *input configurator* are saved to their respective Project associated with a user account. Figure 2 on the next page shows a mockup interface for the *input configurator*.

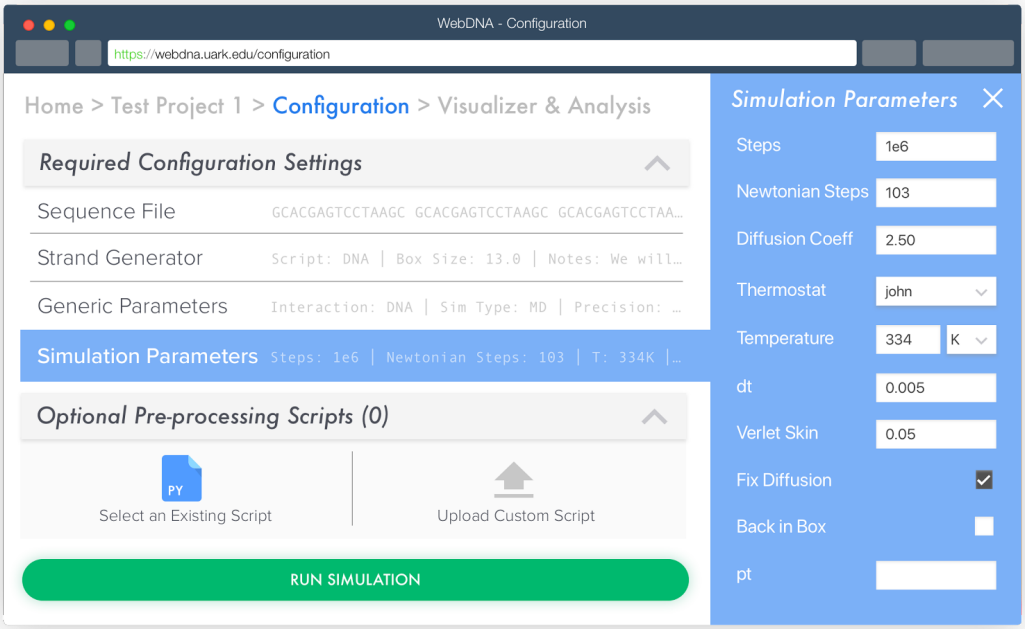


Figure 3. A mockup of the *input configurator*, the first of two pages for all projects.This does *not* represent the first page of the website, but rather the configuration settings of an already-created project.

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 will be based on the current implementation of a basic browser visualizer for oxDNA output files. This visualizer was written using a simple Python HTTP server, so it will need to be migrated to function with our Django HTML renderer. In addition, the current visualizer uses Three.JS to render the DNA system state using WebGL.

The final component, the *output pipeline*, will allow the user to pipe system states through user-defined analysis programs, which are created using a drag-and-drop style pipeline interface. Figure 1 in section 4.1 gives a good visual for what this pipeline will look like. The requirements for this pipeline are not entirely clear, as there are so many potential use cases.

Therefore, the pipeline will give access to the scripts in the oxDNA UTILS directory, as well as allow users to upload custom scripts (which will be sandboxed as mentioned in section 4.1).

Beyond project/simulation execution, the user is able to browse a repository of public scripts (visualized in a “Scripts” page on the main dashboard). that allow them to customize their simulation experience using the volunteered work of other programmers. Users who are not as tech-savvy can take advantage of public scripts to perform unique data-generation and analysis operations. All scripts uploaded by the user are private by default, but the user can opt to make these scripts public via a checkbox on the “Scripts” page next to each script. Python scripts uploaded by the user should include metadata in comments that the server will then parse to give the script more appeal to other users. For instance, a script in the repository named “generated_strands_23.py” with no other information does not appeal to users and will not get used. On the other hand, a script with the name “Weighted Strand Generator” with a multi-paragraph description describing the use cases, input/output parameters, and usage examples will have much more curb appeal, and give users a clear idea of what they are running.

4.3 Risks - Salvador

Risk	Risk Reduction
Learning curve for the Custom Scripts Module	Creating an easily accessible help page about a nearly intuitive scripting language.
Unmanageable codebases	Develop a framework and organization for source code beforehand, as well as an agreement on a rigorous style guide.
File upload security risk	Prescreening of all files.

4.4 Tasks and Schedule - Zack, Salvador, Jace, David

Below we have listed semi-detailed tasks to be completed next 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. Although tasks are sorted by due date, many of them can be completed concurrently. Assuming we can complete tasks by the listed due date, we will complete the design goals we have laid out in section 4.

Server Tasks

Task and Description	Member(s)	Due Date (2018)
Setup Server Hosting We will need to set up server hosting, most likely through the school servers, but it's also possible we use some 3rd party hosting provider like Heroku or AWS.	Jace	January 26
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	January 26
Endpoint: POST User Authentication Users should be authenticated by providing a username||email and password combination to a POST endpoint, then should receive a JSON Web Token (JWT) if authentication passed. This token will be used to approve all further requests to the API from that user. Password hashing for storage and retrieval to the database will be handled by the PostgreSQL package “pgcrypt”, using their implementation of bcrypt. More information for setting this up can be found here.	Salvador	January 29

Endpoint: GET Projects
This endpoint should return all of the user’s project’s top-level information for display on their project dashboard. Each object should only contain simple information, such as the following:
`{`
`  "id": …,`
`  "name": ...,`
`  "created_on": ...,`
`  "job_running": ...`
`}`
	Jonathon	January 29

Endpoint: GET Project Configuration Data The server needs to return the Project configuration data for a single project by project	Zack	February 2

ID. This GET request should essentially serve the stored project configuration JSON data in the body of the request's response. The format for this has not yet been determined, as there are many moving pieces. This will probably happen in 2 steps: 1. Basic project configuration data for minimal operation of oxDNA 2. Add custom scripting options		

Endpoint: PUT Project Configuration Data
This endpoint needs to accept various files related to simulation configuration. The frontend will be able to upload files to our server according to the type of file it is, according to the following types:
 	
Sequence file (labelled strands)
External forces file
Sequence Dependent Parameters
The server will accept the uploads of these files and store them in the following fashion in some data directory:
`/data/{user_id}/{project_id}/{file_type}.dat`
 	
This file path should be saved in the Projects data file, which will be stored as follows:
`/data/{user_id}/{project_id}.json`
 	
Project Configuration JSON Example:
`{`
`  "sequence_file": "/data/{user_id}/{project_id}/sequence.txt",`
`  ...`
`}`
	Jace	February 7

Endpoint: POST Specify Simulation Generation Scripts We will have a separate endpoint that allows the user to modify the project's datafile to specify a build-in or custom data generation script. This should only need to accept a single argument in	David	February 7

the JSON body. URL: `POST http://{webdna_url}.{domain}/apiv1/generation_script/{project_id}` Response Body: `{ "Generate_script_id": "..." }`		

Endpoint: POST Specify Analysis Pipeline
The format of the Analysis pipeline request needs to be specified still. To begin with, we can represent the pipeline as a directed, hierarchical graph of analysis nodes. More specifically, a node of the pipeline can be represented as follows:
 	
`/* example analysis node */ { "node_id": /* e.g. 1 */, "input_node_id": /* e.g. 0 */ "script": /* either generic or custom file path to python script */, "parameters": [ ... ], "name": /* any name */ }`
 	
An analysis pipeline would simply be a list of distinct nodes:
`{ "pipeline": [ { /* analysis node */ }, { /* analysis node */ }, ... ] }`
	Jonathon	February 14

Implement File Generation The files “sequence.txt”, “external_forces.txt”, and “sequence_dependent_parameters.txt” all need to be generated by the server using parameters specified on the client by the input modules.	Zack	February 16
Script Checking for Script Upload Upon receiving, the upload of a custom script, we will run security scans. If the user has opted to	Jace	February 28

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.		
Execute oxDNA Simulation from Project Data The user should be able to submit a Job for queuing to the server, which is initialized using only the desired project_id. An instance of oxDNA will be executed according to the parameters specified in the Project corresponding to the supplied project_id.	David, Jace	March 16

Client Tasks

Task and Description	Member(s)	Due Date (2018)
Login/Registration Page Login: Design 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. Registration: A register button will change the page's form to provide elements for entering necessary registration information (i.e. first name, last name, username, password). Appropriate requests will be made to the server once the form submit button has been clicked.	Zack	January 22
Project Management Page Project List: Design a well organized list of projects associated with the currently signed-in account. Projects listed on this page will have been created previously by the user. Project Options: Each project should provide several options when clicked: "Visit Project Page", "Delete Project", and "Download Options". "Visit Project" should take the user to the selected project's specific configuration page. "Delete Project" will provide a secondary popup box	Jace, Salvador	January 30

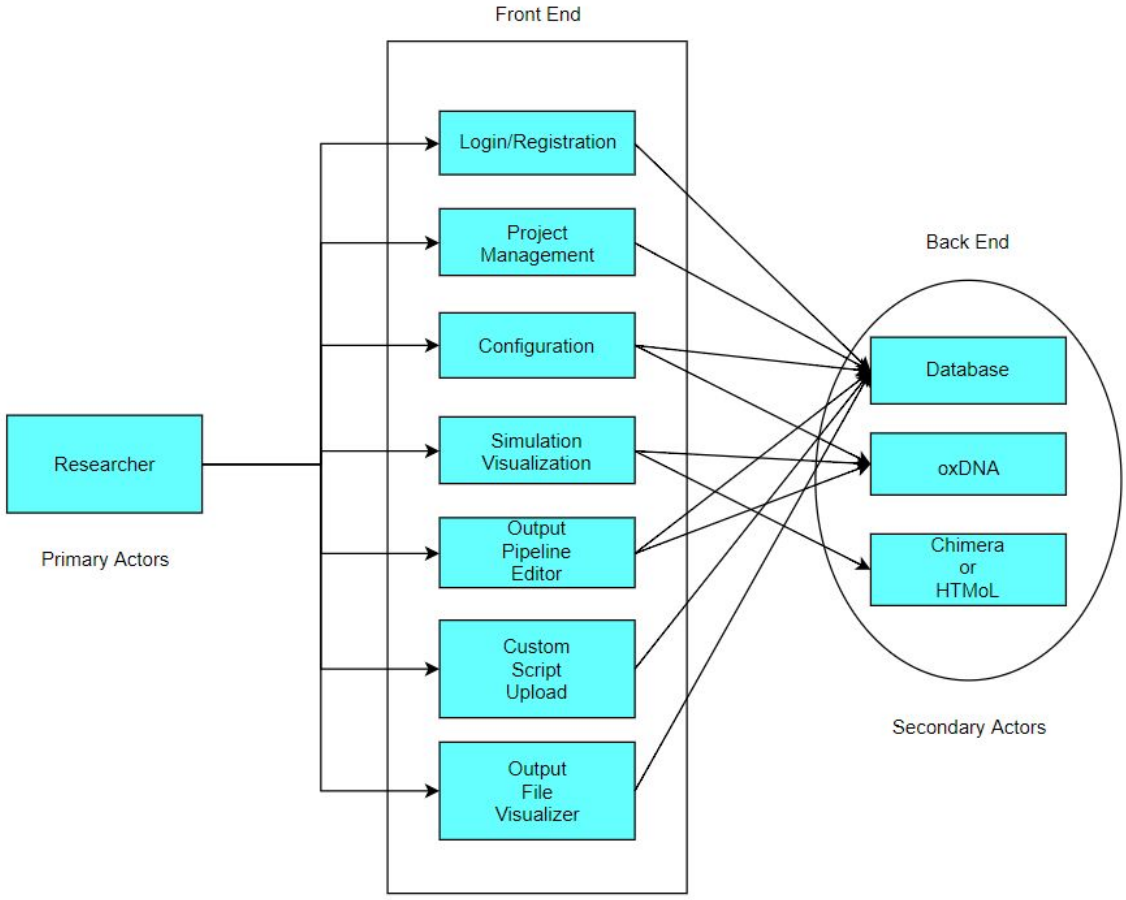
asking the user to confirm a projects deletion from their account. "Download Options" will open a small drop-down list of different files pertaining to the project that can be downloaded including project configurations and simulation outputs.		
Configuration Page Allows the user to configure a project for simulation. This page will end up looking much like the configuration page shown in Figure 3.	Zack	February 7
Output File Visualizer This will be based off the current Three.JS visualizer, as mentioned in section 4. We will be making 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 loaded on the user's computer, since that raw data was needed to populate the visualizer).	Salvador, Zack	March 9
Output Pipeline Editor This is tricky, but we need to give the user a visual way to edit an analysis pipeline. Essentially they need to be able to generate a pipeline with the same visual style as shown in Figure 1.	Salvador, Jace	March 23
Custom Script Upload Manager This will be a simple page that will be included in the "Scripts" page of the Dashboard, with the option to upload python scripts generated by a researcher. Upon upload, it will be scanned for security and for our specifications for it to be added to the community's scripts (if they choose to make their script public).	David, Zack, Jonathon	March 30
Dashboard (Integration of All Pages) The Dashboard needs to be the central location for all project management and script management. The user will have three main tabs in a sidebar: Projects, Scripts, and Profile/Settings. Projects will list the users projects and show the	All members	April 6

Project Management Page. Scripts will show the user's private scripts, as well as giving them the ability to search public community scripts and add those to their account. Profile/Settings will give the user basic control over their profile, such as changing their name, email, username, and password.		
Documentation Final documentation will need to be written in detail to cover all client functionality for end users. The documentation must be 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	April 13

4.5 **Deliverables** – Salvador and Zack

- Design Document
    - Outline front end functionality, database schema, and scope of analysis
- API/Job-running Server
    - Python/Django
- Website
    - Angular2 framework and Base Template.
- Data Analysis
    - Output documents
    - Work in existing Python scripts into front end analysis
    - Sandboxing of custom data analysis python scripts
- Visualization tools
    - Javascript
    - Chimera
- Final Documentation
    - Provide 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.

5.0 Use Cases - Salvador



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 flow diagram 9. Server: displays output of flow diagram
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 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.

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.

6.1 Facilities and Equipment - Zack

There isn't much hardware that we need for our purposes currently. Down the road it would be useful to own a few machines with powerful GPUs to perform quicker simulations. For the frontend web client, we can enormously speed up the process of our Angular2 website by purchasing a licensed template (usually around $30-$40, see [12]). This template would give us great visual components with immediate functionality and portability, meaning we wouldn't have to spend time nit-picking about component visuals and focus more on overall layout needs for the site. For such a small price, it justifies the efficiency gains.

7.0 References

[1] oxDNA Features, https://dna.physics.ox.ac.uk/index.php/Features
[2] Django Documentation, https://docs.djangoproject.com/en/1.11/
[3] VMD, http://www.ks.uiuc.edu/Research/vmd/
[4] UCSF Chimera, https://www.cgl.ucsf.edu/chimera/
[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