University of Arkansas – CSCE Department
Capstone II – Preliminary Report – Spring 2022

Designing and Simulating the Self-Assembly of DNA Nanostructures and DNA Computers

Lindsey Albin, Ben Hughes, Kyle Sadler, Christopher Souvanouphong

Abstract

The design of DNA strands is an expensive and time-consuming process. Researchers in the nanotechnology field waste valuable resources transforming nanostructure designs into DNA strands. The goal of the project is to simplify the process of constructing DNA strand designs by creating a quick and easy way for researchers to convert nanostructure tile designs into DNA strand diagrams.

Approximately a dozen labs around the world have the capability of implementing tile-based self-assembly. The successful implementation of this project can impact these labs by reducing consumable and labor costs, allowing rapid development of new designs and allowing the quick export of a design for simulation to test strains and their stability.

1.0 Problem

DNA tile-based self-assembly is a process in which a collection of simple yet disorganized components coalesce to form complex structures. While this happens naturally all around us (take a snowflake for example), scientists have developed methods to artificially manipulate matter on the atomic level to replicate this phenomenon. For example, the abstract Tile Assembly Model (aTAM) is a high-level abstraction that ignores potential errors. The aTAM is a mathematical model used to implement tile sets via designed DNA strands. In this model, assembly starts from a given “seed” tile and grows non-deterministically and asynchronously.

One of the biggest issues facing researchers today when working with tile-based self-assembly is the cost overhead and the extensive number of software packages needed for these models. The current process of transforming a nanostructure design into a set of DNA strands is tedious and requires many independent software packages. This means that researchers must waste valuable time importing, exporting, transferring, and reformatting data. The other problem faced is the cost of self-assembly. Ordering the DNA in a tile-based format can get expensive very fast. If there was a way for scientists to minimize the number of errors and mismatches in the DNA strands, it could save them thousands of dollars in the long run.

2.0 Objective

The objective of this project is to simplify the DNA nanostructure design process. Instead of having to use multiple independent software packages, our project will allow researchers to easily design an abstract nanostructure tile assembly and transform it into a DNA strand diagram, readable by Scadnano. Scadnano is web-based software for visualizing DNA molecules as strand diagrams. Our software will be hosted on a web server so that it is accessible and does not require the user to manually install and compile software packages.

3.0 Background

3.1 Key Concepts

DNA Base Pairs. DNA base pairs are molecules called nucleotides, on opposite strands of the DNA double helix. They form chemical bonds with one another and act like rungs in a ladder to hold the two strands together. The four bases are adenine (A), cytosine (C), guanine (G), and thymine (T). Adenine forms a pair with thymine, and guanine forms a pair with cytosine. The binding of these pairs forms the structure of DNA.

DNA Nanotechnology. DNA nanotechnology is the study and design of DNA nanostructures. Strands of synthetic DNA can be designed so that they bind in a controlled and predictable process which allows arbitrary DNA structures to be created at the nanoscale. This is possible due to two technological advancements: a detailed understanding of the DNA binding process and the ability to manufacture synthetic DNA. Nanoscale DNA structures have applications in fields such as biophysics, diagnostics, nanoparticle and protein assembly, biomolecule structure determination, drug delivery, and synthetic biology.

DNA Tile Assembly. This project focuses on a DNA nanostructure fabrication process called *DNA tile assembly*. In tile assembly, DNA strands are conceptually organized into rectangular structures called *tiles* which are the fundamental building unit in tile assembly structures [Figure 1]. These tiles have four DNA binding domains (a, b, c, d) which are used to control the binding properties of the tile and determine the shape of the fabricated nanostructure. Multiple types of tiles can be designed so that when mixed in a solution, tiles attach to each other in a way that builds the intended DNA nanostructure. The entire set of tile types used in an assembly is called a tileset.

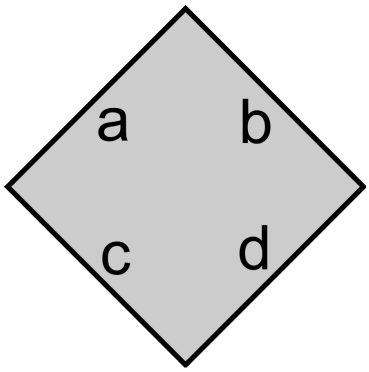

Figure 1. Abstract DNA tile used in tile assembly.

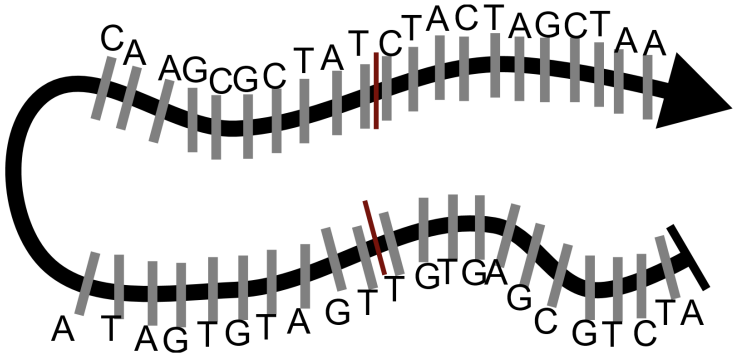

Figure 2. A single-stranded tile formed out of a single strand of DNA.

Figure 3. A core tile which contains two additional DNA strands which brace the structure and form its core.

DNA Tile Motifs. As tiles are merely conceptual representations of blocks of DNA binding domains, there are many ways to implement them in actual strands of DNA. The most common DNA tile motif is *single-stranded tiles*, in which each tile is physically realized as a single strand of DNA bent back on itself [Figure 2]. The other main motif is *core tiles*, in which two additional strands are added to the single-stranded tile to brace the structure and add more stability [Figure 3]. In this project, we handle both single-stranded tile and core tile motifs.

Scadnano. Scadnano is a software package for visualizing DNA molecules as strand diagrams and designing synthetic DNA nanostructures. It allows users to sketch out DNA nanostructures using strand diagrams via a web-based GUI or programmatically through a Python library. Scadnano supports exporting of the nanostructure file for coarse-grained molecular simulation, and simulation of twist and strain on the nanostructure.

3.2 Related Work

There are many existing software applications that are related to DNA nanotechnology and nanostructure design. When designing a tile assembly nanostructure, one of the first steps is to create a tileset using a tile assembly simulator [3]. First, the nanostructure is divided into tiles. The tile assembly simulator then helps simulate how a DNA tileset design will act when the tiles are physically created out of DNA and mixed in a test tube.

Tilesets are often visualized in Scadnano which is a software program for visualizing DNA molecules as strand diagrams [4]. While Scadnano is a useful tool, it is often time-consuming to import tileset designs.

The specific sequences of DNA used in a tile assembly are often picked using a sequence designer such as DNADesign [6]. However, these tools do not make it easy to combine these sequences with a tileset structure and Scadnano, which is the problem our project will solve.

4.0 Design

4.1 Application Requirements

- Input an abstract tile assembly, tile types, and binding domain lengths. All inputs will be defined through the interface
- All three inputs are required for the software to compute the DNA strand diagram
- Output a JSON file for the tile design for the user that can be uploaded to Scadnano to view the DNA strand diagram that was designed
- Software must be hosted on a web server
- A nice and clear transition from our webpage to Scadnano
- The user interface must be easy to understand and easy to use
- The process of inputting all of the data must be clear, organized, and provide the user with feedback on correct and incorrect inputs
- The user interface must keep the user's attention (show a loading bar, etc) while the computation is taking place

4.2 Detailed Architecture

The architecture consists of two primary components: the user interface written in React and the backend server written in Python. Communication between the user interface and the backend is facilitated by an API service written in JavaScript.

User Interface

The front end is responsible for providing users the ability to design and export abstract nanostructure tile assemblies. Through the interface, the user can define the canvas size and the binding domain lengths for the tile design, as well as define different tiles that can be added to the design [Figure 4]. With this design, the user is able to select the tiles they have defined and place them into the canvas where they see fit. The user is then able to download a JSON file for the designed tile assembly [Figure 6], which can be uploaded to Scadnano to view the strand diagram created [Figure 5]. The interface is also responsible for error-checking the input and providing feedback on incorrect uploads or errors. Specifically, the front end validates the tile design before it is uploaded to the API.

The user interface is built using React 16.4.0 and has two main components: the Tile Assembly Canvas and the Tile Menu. The front end uses React Material UI 5.5.0 to customize the look and feel of the interface.

Tile Assembly Canvas

The first major component of the front end is the Tile Assembly Canvas which is located in the middle of the screen [Figure 7]. This canvas allows the user to create abstract tile assemblies by placing tiles in a grid. Multiple tiles can be used in the same assembly and are represented with different colors as shown in [Figure 4]. The canvas was created using a Material UI Card component and HTML div elements. React onClick listeners were added to each tile location to add tiles when the user clicks on the canvas.

Tile Menu

The second main component of the front-end is the tile type selection element on the right-hand side of the screen [Figure 8]. This allows the user to select the type of tile to place on the canvas and shows the currently active selection. Users are also able to change the core length, color, and label of each tile as well as create new tiles using the Add+ button. When the “Export to Scadnano” button is clicked, the abstract tile assembly data from the canvas is validated and uploaded to the backend via the API service. The tile menu uses Material UI Card, Button, and TextField components.

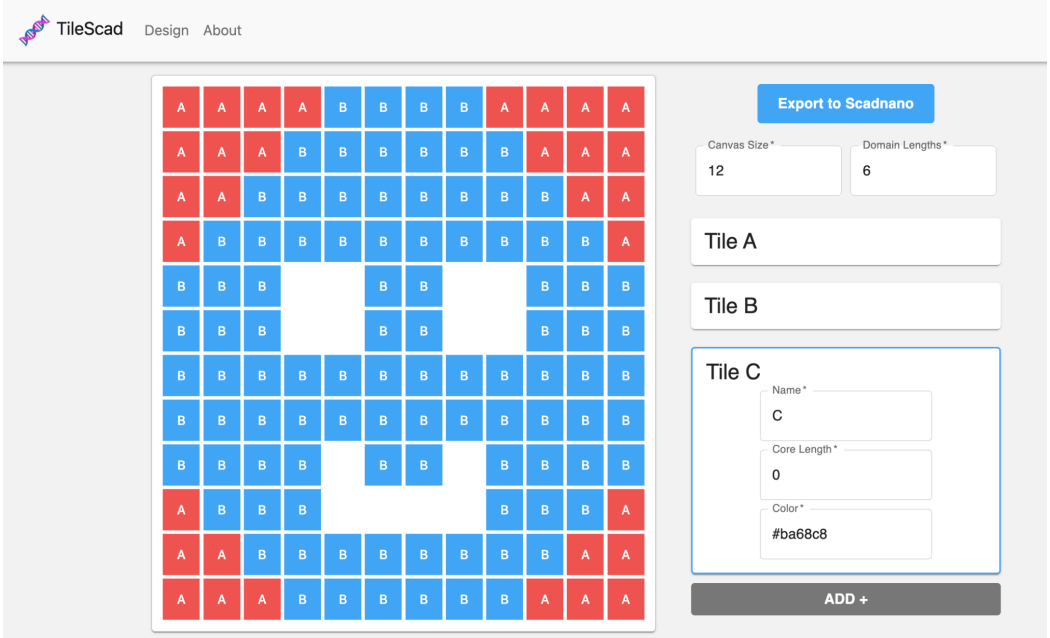

Figure 4. Example abstract tile assembly input with different tile types.

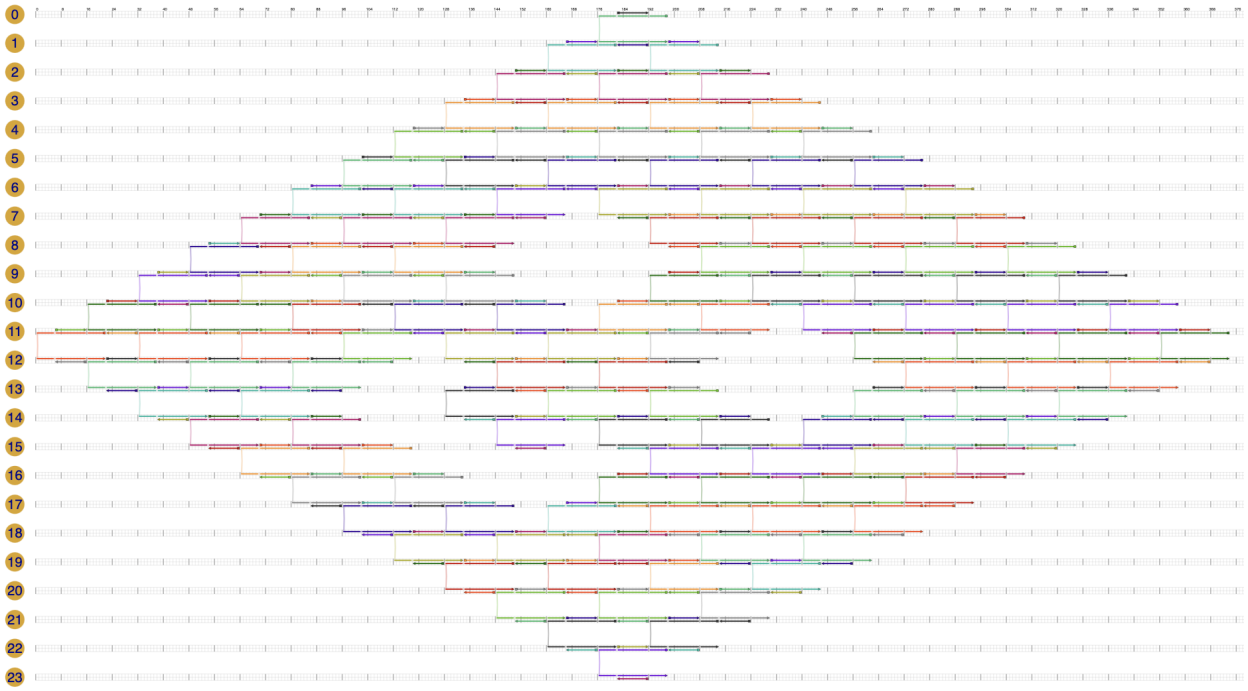

Figure 5. Resulting Scadnano strand diagram for abstract tile assembly shown in Figure 4.

canvas_design_16472818189001412 〉 No Selection


```
{
  "version": "0.17.1",
  "grid": "square",
  "helices": [
    {"max_offset": 1024, "grid_position": [0, 0]},
    {"max_offset": 1024, "grid_position": [0, 1]},
    {"max_offset": 1024, "grid_position": [0, 2]},
    {"max_offset": 1024, "grid_position": [0, 3]},
    {"max_offset": 1024, "grid_position": [0, 4]},
    {"max_offset": 1024, "grid_position": [0, 5]},
    {"max_offset": 1024, "grid_position": [0, 6]},
    {"max_offset": 1024, "grid_position": [0, 7]},
    {"max_offset": 1024, "grid_position": [0, 8]},
    {"max_offset": 1024, "grid_position": [0, 9]},
    {"max_offset": 1024, "grid_position": [0, 10]},
    {"max_offset": 1024, "grid_position": [0, 11]},
    {"max_offset": 1024, "grid_position": [0, 12]},
    {"max_offset": 1024, "grid_position": [0, 13]},
    {"max_offset": 1024, "grid_position": [0, 14]},
    {"max_offset": 1024, "grid_position": [0, 15]},
    {"max_offset": 1024, "grid_position": [0, 16]},
    {"max_offset": 1024, "grid_position": [0, 17]},
    {"max_offset": 1024, "grid_position": [0, 18]},
    {"max_offset": 1024, "grid_position": [0, 19]},
    {"max_offset": 1024, "grid_position": [0, 20]},
    {"max_offset": 1024, "grid_position": [0, 21]},
    {"max_offset": 1024, "grid_position": [0, 22]},
    {"max_offset": 1024, "grid_position": [0, 23]}
  ],
  "strands": [
    {
```


Figure 6. Resulting Scadnano JSON file for abstract tile assembly shown in Figure 4.

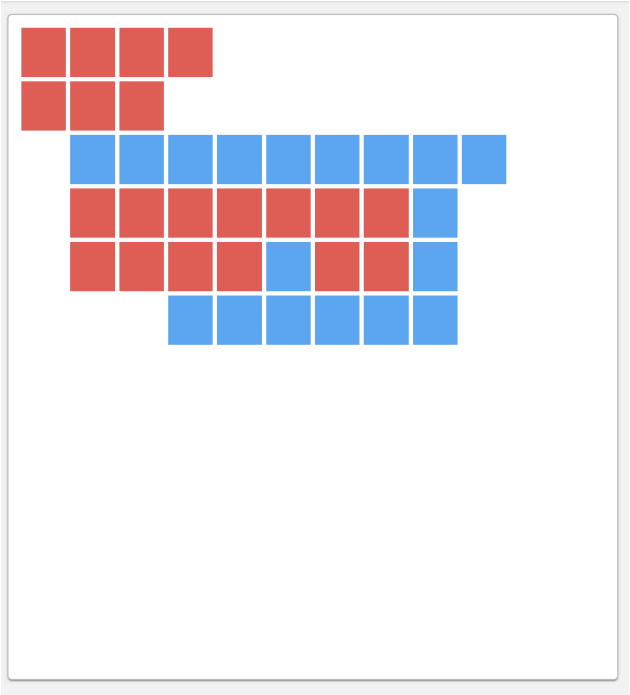

Figure 7. An abstract tile assembly using red and blue tiles displayed on the tile assembly canvas. The canvas is implemented using flexbox HTML div elements and onClick listeners.

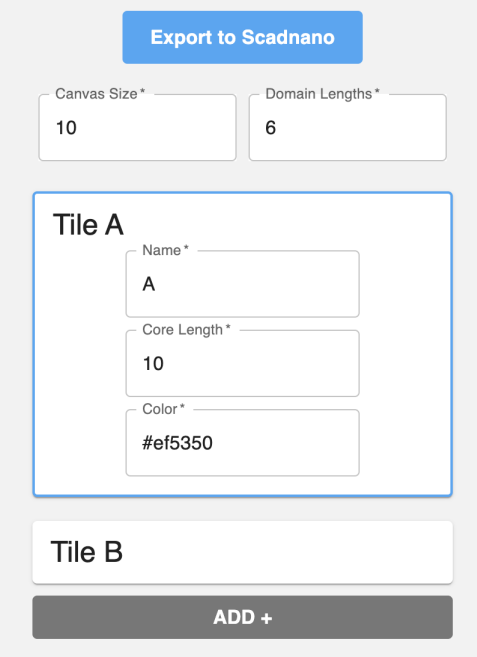

Figure 8. Tile menu with Tile A currently active. The tile menu is implemented using Material UI Card, Button, and TextField components.

Frontend Input Validation

As users are allowed to use differently-sized tiles in the same assembly, the tile assembly data must be checked before being sent to the backend. This ensures that tile assemblies are only uploaded if it is geometrically possible to arrange the tiles in a Scadnano DNA design. Note: when the data is uploaded, the backend also runs a similar validation function as described in the section Backend Input Validation and Error Types.

The two Scadnano designs below help illustrate how tile assemblies can be geometrically invalid. Figure 9 shows an abstract tile assembly that uses tiles of the same core length, while Figure 10 shows an assembly with two different tile core lengths. The red and blue tiles in each of the assemblies have core lengths of 10 and 0 respectively. Notice that the core length of the tiles influences the slope of the resulting shape of the Scadnano design. This is because a tile's width directly affects its width-to-height ratio as the height of each tile is fixed. If multiple tile sizes are used, we must ensure that tiles don't collide when being rendered in the final Scadnano design (for example if two lines of different widths cross).

Determining if a given tile assembly is geometrically valid reduces to determining if every vertical column in the resulting Scadnano design contains tiles of the same core width. As the abstract tile assembly is rotated 45 degrees from the resulting Scadnano design, we can ensure the assembly is valid by checking that each northwest-southeast diagonal in the assembly contains tiles with the same width.

If the tile assembly is invalid when the user clicks "Export to Scadnano", an error message appears on the screen and the data is not uploaded to the API [Figure 11].

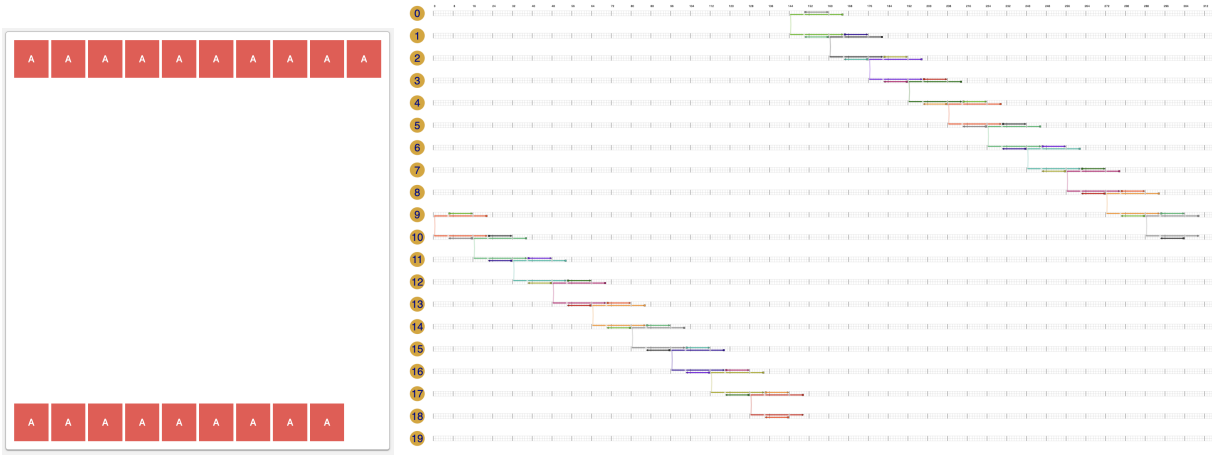

Figure 9. Abstract tile assembly with tiles of the same size (left) and the resulting Scadnano design (right).

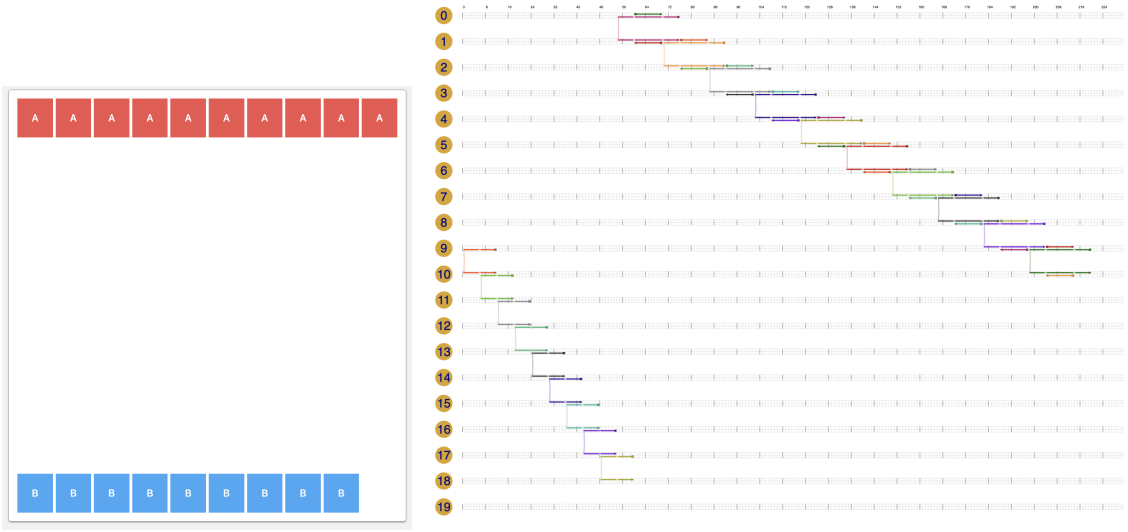

Figure 10. Abstract tile assembly with tiles of different sizes (left) and the resulting Scadnano design (right).

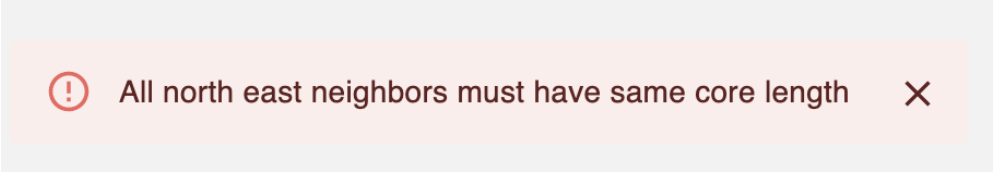

Figure 11. Error message appearing when the tile assembly is geometrically invalid.

API Service

The API service contains functions that abstract the underlying details of the backend API from the frontend. It contains one function, uploadData(), which takes validated tile assembly data, uploads it to the API, and returns the name of the resulting Scadnano file to download.

Python Backend

The main logic of our server is written in Python 3.10.0. This design choice was made because the Scadnano API is only accessible through a Python package. When the user uploads the input files, the backend will be responsible for receiving the files, processing them into the resulting strand diagram, and sending the processed file back to the user.

We chose to implement the backend using the Python framework Flask and the Web Server Gateway Interface tool gunicorn due to the tools' simplicity and ease of use. Members of our team were already familiar with Flask and gunicorn as well.

API Routes

The backend has two main API routes:

- **/api/upload-design** used to upload an abstract tile design to the server
- **/api/download-file** used to download the resulting Scadnano file from the server

These API routes are both written in flask and make use of a TEMP_DIRECTORY variable which is a server directory used to temporarily store Scadnano files as they are being created.

Upload Design API Route

After the user uploads an abstract tile assembly using the **/api/upload-design** API route, three pieces of data are extracted from the user's request:

1. A list of tile types and the lengths of their core strands, in nucleotides
2. The tile assembly grid specifies which tile is placed at each grid location if any
3. The binding domain length of all tiles in the assembly

This data is first checked to ensure that the abstract tile assembly is valid. Then the tile assembly grid is processed into a Scadnano file using the Scadnano library.

Backend Input Validation and Error Types

Before the user's uploaded data is processed into a Scadnano file, the data is first verified to ensure that it is valid. Below is a list of invalid data types and the returned errors

Condition	Error Message	Error Code
If the data is missing	Data is missing	400
If the tile assembly grid, binding domain length, or tile types are missing	Data is missing	400
If the tile assembly grid is not a 2D array	Invalid Data	400

If the tile assembly grid is not square	Invalid Data	400
If the binding domain length is more than 30*	Invalid Data	400
If the tile assembly grid is more than 60 x 60*	Invalid Data	400
If there are more than 50 tile types*	Invalid Data	400
If the core lengths differ in any NW-SE diagonal in the tile assembly**	Invalid Data	400
If the server has an internal error	Server Error	500

Figure 12. A table of the error types returned by the API and their triggering conditions.

* The conditions marked with a star indicate data validation checks that are implemented to protect our server from denial of service attacks.

** This ensures that each tile in a column has the same width when the tile assembly is drawn in Scadnano. This condition essentially ensures that the resulting DNA strand geometries are compatible and don't overlap.

Tile Assembly Data Processing

After the input data has passed all the validation checks, we know that the tile assembly is valid, and the resulting Scadnano tiles will be geometrically compatible. At this stage, we combine the tile assembly grid and the tile type data to compute the number of domains and their lengths at each location in the tile assembly grid. A sample conversion is shown below.

Tile Assembly Grid Location	**Full Tile object**
Tile 3	Tile(NW_domain=4, top_core=8, NE_domain=4, SE_domain=4, bottom_core=8, SW_domain=4)
null	null *(no tile)*

Figure 13. A sample conversion of two Assembly Grid locations, one where there is a tile present and the other where there is no tile present.

The grid of fully specified tiles is then transformed into a Scadnano design file by using the Scadnano Python package. Each domain in a tile is created using the scadnano.Domain() constructor. These domains are then organized into strands using scadnano.Strand(). Groups of strands are then organized into tiles and written to a Scadnano design file using the scadnano.draw() function. The file is then stored in the TEMP_DIRECTORY directory to be accessed by the /api/download-file API route.

Deployment and build scripts

To automate the deployment of this project, a build script written in bash was created to automatically install and configure our application on Ubuntu 20.04 servers. This installation script installs the following tools used for deploying our project

- **Node Package Manager (npm)** – used for installing the dependencies of our project
- **NGINX** – a web server used to create a reverse proxy between our frontend and backend
- **PM2** – a process management tool used to ensure our application is constantly running
- **Python 3.10.0 –** the programming language used in our backend Flask server

A bash script was also written to deploy the latest version of our application code. This script automatically pulls the latest changes from GitHub, builds the code, and restarts our updated application. These scripts allowed us to deploy our code with one command.

Finally, the domain https://tilescad.org/ was purchased from Google Domains and directed to the IP address of the Vultr server using a type A DNS record. This ensures that the tilescad.org domain always resolves to our application.

4.3 Risks

Risk	Risk Reduction
User concerns about research confidentiality	Provide a disclaimer that strand files are never stored on our servers and DNA research intellectual property remains with the user
Research liability for software bugs	Provide a disclaimer that we are not responsible for any incorrect results and all results should be manually checked before using as research material.
User enters data that is too big for our servers to process efficiently	Add checks to limit the size of user input to protect our servers

4.4 Tasks

1. Preliminary research
    a. Gain proper nanotechnology background knowledge required for the project
    b. Understand Scadnano, the software used for visualizing DNA molecules as strand diagrams
    c. Understand the process of creating DNA strands with the Abstract Tile Assembly Model (aTAM)
2. Design the software
    a. Design the frontend in Figma

        i. Design layout for the tile assembly editor so that it is easy for users to create abstract tile assemblies
        ii. Design a way for users to switch between tile types in the same assembly and a way for users to create new tile types
        iii. Include a button to submit the abstract tile assembly, the strand lengths and binding domain lengths to our backend
    b. Design the backend
        i. Choose a Python web framework
        ii. Design DNA strand routing algorithm to transform abstract tile assemblies into Scadnano DNA files
        iii. Design an API route specification to upload an abstract assembly
        iv. Design an API route specification to receive a resulting assembly Scadnano file. The assembly file will be downloaded as a json file containing every DNA strand position in the assembly, the name of its parent tile, and its position in a 3D integer lattice. An example list of DNA helices is shown in Figure 6.
3. Software implementation
    a. For this stage, we will split into two teams: one to implement the backend and one to implement the frontend.
    b. Frontend implementation in React
        i. Create tile assembly editor
        ii. Add user input validation and helpful error messages if the input is invalid
        iii. Create a loading animation and screen while the computation is taking place if needed
        iv. Create a button that allows the user to download the output Scadnano file
    c. Backend implementation in Python
        i. Set up a basic Python server that can respond to requests
        ii. Write functions to receive uploaded input files
        iii. Write functions to parse the input abstract tile assembly and check for errors
        iv. Write a function to take the input data and compute the resulting Scadnano strand diagram using the Scadnano Python library
        v. Write a function to write the output Scadnano data to a file
        vi. Create an API route to download the results of a submitted job
4. Integration of the frontend with the backend API
    a. Ensures that files and input data can be uploaded from the frontend to the backend
    b. Ensure that output data from the backend can be sent to the frontend both as JSON and as a file
5. Testing the software
    a. Test the abstract assembly editor with different inputs to ensure that the software is robust and can handle errors gracefully
    b. Test creating and selecting new tile types
    c. Verify that error messages are displayed when the tile assembly is invalid
    d. Verify that the system remains stable and does not crash under all sets of user inputs

4.5 Schedule

Tasks	Dates
Background research on the Scadnano libraries, software, etc	1/10 - 1/14
Make mockups of frontend pages in Figma	1/17 - 1/21
Setup Python server and write build scripts	1/24 - 1/28
Implement front-end tile canvas and input validation	1/31 - 2/11
Define API routes, interfaces, and behavior	2/14 - 2/17
Write backend functions to parse input files and data	2/18 - 2/22
Design DNA strand routing algorithm using Scadnano package	2/23 - 2/28
Implement backend API routes and functions	3/1 - 3/7
Integrate front-end interface and backend API	3/8 - 3/15
Polish user interface and add ease-of-use improvements	3/16 - 3/29
Add more descriptive error messages	3/30 - 4/2
Present project to project sponsor and receive feedback	4/3 - 4/9
Final changes based on sponsor feedback	4/10 - 4/17
Clean up code and work on the final presentation	4/18 - 4/22
Presentation and final report	4/25 - 4/29

4.6 Deliverables

1. Design Documentation: a document outlining the architecture and tech stack of this project and how each component works.
2. Python Backend: Python script that transforms abstract tile assemblies into DNA strand models.
3. React Frontend: The Javascript files that make up our web interface for this project and allow for file uploads
4. Build Scripts: scripts to build and launch the application on a web server
5. Live Web Address: URL to live version of our project on https://tilescad.org/
6. Final Report

5.0 Key Personnel

Kyle Sadler – Sadler is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He is also pursuing a Bachelor of Science in pure mathematics from the University of Arkansas. He currently works full-time for Pipedream, a robotics startup building underground delivery robots. He will be responsible for the application architecture and backend.

Lindsey Albin – Lindsey is a senior Computer Science and Graphic Design major with minors in Mathematics and Art history in the Computer Science and Computer Engineering Department and the Graphic Design Program at the University of Arkansas. She currently works at the McMillon Innovation Studio on campus as the student creative director and previously she worked teaching kids to code using Kotlin and Minecraft. She has taken a class on user experience design and computer graphics along with programming paradigms where she learned Python. She will be responsible for the front end.

Ben Hughes – Hughes is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has previously had an internship with the Walt Disney Company. He is currently working on the front end / UI for a mobile application in Mobile Programming. He has worked on Javascript and Python in the Paradigms course. He will be responsible for the front end.

Christopher Souvanouphong – Christopher is a senior Computer Science major at the University of Arkansas and pursuing a minor in Mathematics. He has previously developed a mobile ordering application for a local restaurant, allowing him to gain experience with both frontend and backend development. He will be responsible for the backend development.

Dr. Trent Rogers – Dr. Rogers is a postdoctoral researcher at Maynooth University researching self-assembling and self-organizing systems. He was a National Science Foundation Graduate Research Fellow and the recipient of the Doctoral Academy Fellowship as a Ph.D. student. He graduated from the University of Arkansas with a Ph.D. in computer science in 2019.

6.0 Facilities and Equipment

Vultr Server - The web application is hosted on a 2GB cloud computing instance rented from the cloud service provider Vultr.

Google Domain - The domain tilescad.org was purchased from Google Domains

GitHub Repository - The code for this project is stored in the GitHub repository located at https://github.com/Capstone-Team-4-UARK-2022/TileScad.org

7.0 References

[1] DNA Nanotechnology, https://www.nature.com/articles/natrevmats201768

[2] Scadnano, https://scadnano.org/

[3] Tile Assembly Simulator, http://self-assembly.net/mpatitz/papers/TAS-SASOW.pdf

[4] DNADesign, https://www.dna.caltech.edu/DNAdesign/

[5] Definition of base pair, https://www.cancer.gov/publications/dictionaries/cancer-terms/def/base-pair