TigerPath - Princeton Four-Year Course Planner

Team:

- Richard Chu (Project Leader): richardchu@princeton.edu
- Barak Nehoran: bnehoran@princeton.edu
- Adeniji Ogunlana: ogunlana@princeton.edu
- Daniel Leung: dl13@princeton.edu

I. Overview

Since Princeton affords such rich academic opportunities, students are always eager to get the most out of the four years they spend here. Without careful planning, however, it is often difficult for students to decide on which courses to take and when to take them. Whether they are ambitious incoming freshmen or juniors considering switching majors, there is currently no easy way for students to map out their courses to see if switching majors or pursuing one more certificate would be a viable option. Our goal is to create an application that would simplify the process of creating a four-year plan. The application would allow users to input the courses they have taken as well as those they are planning on taking during future semesters, and then indicate the major/certificate requirements that have been satisfied and those that still need to be satisfied.

II. Requirements and Target Audiences

The main intended users of our application are Princeton students. While the application can be beneficial to all Princeton students, it would be especially helpful for students who want assistance in maintaining a plan of what classes to take and when to take them, as well as those who are contemplating switching majors. This application might also help Princeton academic advisors (and Princeton faculty in general) gain a better sense of the departmental graduation requirements and the relative difficulty of satisfying them, which would in turn help them better advise students. For example, students could print a copy of their four-year plan to discuss with their advisor.

Our application benefits students by giving them an easier way to plan what courses to take during their four years at Princeton. Instead of manually comparing the courses they'd like to take with the requirements they need to satisfy, the application will handle the comparison for them. Current solutions to this problem are very rudimentary. Students typically navigate through different Princeton webpages to find the requirements they need to fulfill, and then use ad-hoc methods such as Excel spreadsheets or notes applications in order to plan out what courses they need to take in the future. This is time consuming and error prone, and could lead to problems if the student misses a certain requirement or if they misread the fine print about which classes count for which requirements. Any human error in planning could eventually require a student to take summer courses or an overloaded semester to graduate on time. Our application would simplify this experience, automate much of the process of building the four-year plan, and reduce the risk of human error.

III. Functionality

Main Features

- The user can search for and add courses to their four-year schedule
- The user can drag and drop courses between different semesters in order to customize their schedule
- The app tracks the major requirements that have been fulfilled by the user's four-year schedule as well as what requirements still need to be fulfilled
    - For the minimum viable product, we will be focusing on the BSE majors, Computer Science - AB, the Woodrow Wilson School of Public and International Affairs, Economics, History, Spanish and Portuguese, and Molecular Biology
    - As an additional feature, the app would also track certificate requirements
- The user can customize what requirements they have already fulfilled through other means, such as through study abroad, AP credits, summer classes, and waivers
- The app informs the user of when courses are typically offered (such as how frequently the course is offered and which semester it is typically offered in)

Use Case Scenarios

1) Deciding on which departmental/distribution requirements to take
    a) The user accesses the web app and authenticates through CAS.
    b) The user adds a major they want to track, and the website pulls the set of requirements needed for the major.
    c) The user inputs all of the courses they have taken so far or pulls the data from a previous session.
    d) The user looks at the requirements they have satisfied based on these courses and examines which ones are still needed to complete the degree.
    e) The user adds additional courses to the current semester or future semesters and the distribution/departmental counters or indicators update accordingly.
    f) The user repeats Steps d and e until all of the requirements are satisfied.
2) Making room for future courses/study abroad
    a) The user marks a semester (or more) as a potential study abroad period.
    b) The user performs Use Case 1 until they reach a plan satisfying most of their requirements.

    c) The user examines the number of empty course slots left and determines whether they can study abroad for the intended period, and which requirements they would need to satisfy while abroad.
    d) The user might print out their four-year plan to make their advising appointment easier when they decide to get approval for their classes.

3) Selecting or switching majors
    a) The user contemplates the possibility of switching majors, or of declaring a certain major.
    b) The user inputs their previous courses or pulls them from a previous saved session.
    c) The user switches the major they're tracking, which makes the app change the set of requirements being tracked. The app recalculates the distribution/departmental requirements that the user has satisfied with the previously taken courses.
    d) The user attempts to add in courses to the remaining semesters in order to check if it would be possible to satisfy the new major's requirements.
4) Plausibility of certain certificates
    a) The user adds the certificates they want to pursue. The website pulls the additional requirements for these certificates and tracks them for the user.
    b) The user performs Use Case 1 until they have satisfied all of their requirements for both their major and their certificates.
    c) Otherwise, the user determines that they cannot complete all of the certificates they added, and modifies them accordingly in order to readjust the requirements. They repeat Step b.
5) Student Advising
    a) The user is about to meet with their advisor to discuss the classes they would like to take this semester.
    b) The user performs one of Use Cases 1-4 until they have satisfied all of their requirements.
    c) The user exports the four-year-plan and either emails the PDF to their advisor or prints the hardcopy to bring to the advising appointment.

IV. Design

Server

We are planning on creating the backend of the web application using Python with the Django web framework. We will host the app using Heroku.

The server API will expose the following endpoints to the client (all of which require the user to be authenticated in order to be accessed):

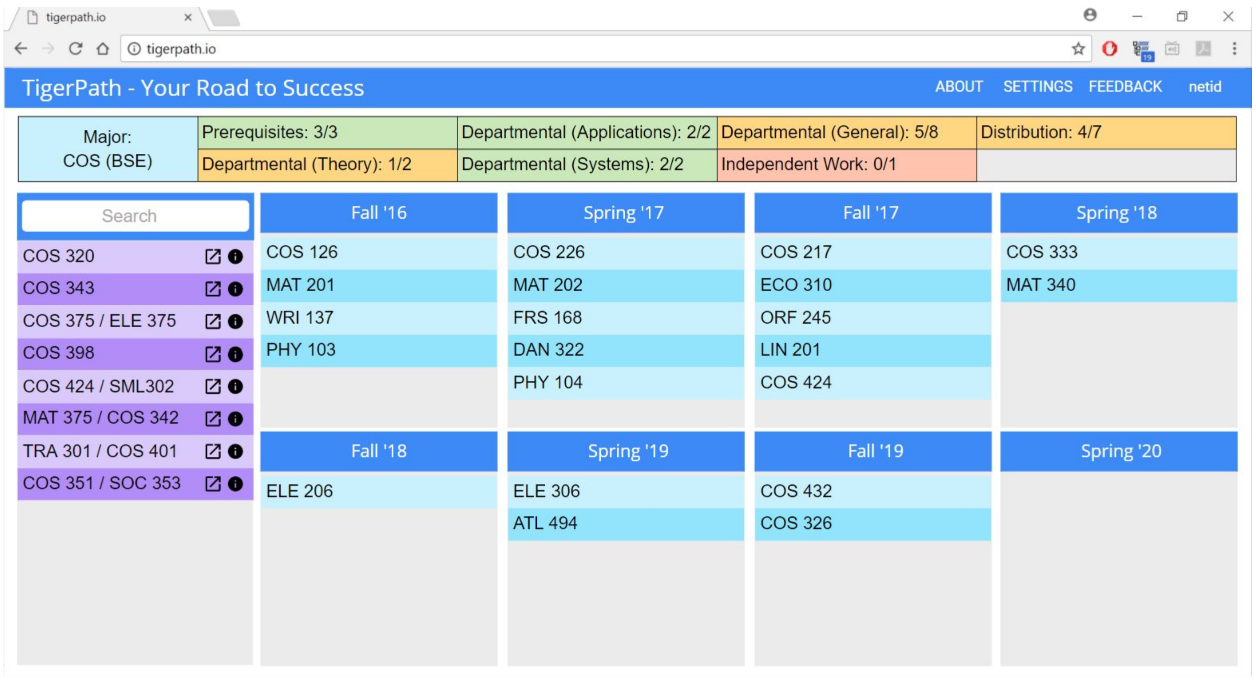
- **GET /api/courses**: Returns JSON containing a list of courses and the corresponding information from the database.

    ○ Can be fine-tuned by adding query parameters to specify the particular department or course number.

- **GET /api/require?major=[USER_MAJOR]**: Returns JSON of requirements for the major specified by USER_MAJOR from the database, where USER_MAJOR is the three-letter department code used to specify a major (such as COS).
- **GET /api/user/meta**: Gets user-specific metadata, such as what major the user is tracking and what class year the user is in.
- **GET /api/user/plan**: Gets the plan that the user has created in their last session, which includes what courses the user is taking, and in which semester.
- **POST /api/user/meta**: Changes user settings such as what major the user is tracking.
- **POST /api/user/plan**: Posts the plan that the user has created locally (that includes what courses the user is taking, and in which semester) to the server, and updates the user's database entry accordingly.

Client

For our front-end, we will be using HTML, CSS, JavaScript, and the Bootstrap front-end framework. Our application will consist of three pages: (1) the splash page, (2) the settings page, and (3) the course selection page. The splash page will present a welcome message, some information about the application, and request the user to login via CAS. If the user has never used our application before, they will be taken through a short tutorial of the app, and then will be asked to input their major in order to start using the application. If the user is a returning user, they will already have a major associated with their NetID, and they will be taken directly to the main course selection page. This is what the course selection page of the application will look like (note that this is only meant to show the user interface, and that the contents may not be accurate):



TigerPath - Your Road to Success

Major: COS (BSE)	Prerequisites: 3/3	Departmental (Applications): 2/2	Departmental (General): 5/8	Distribution: 4/7
	Departmental (Theory): 1/2	Departmental (Systems): 2/2	Independent Work: 0/1	

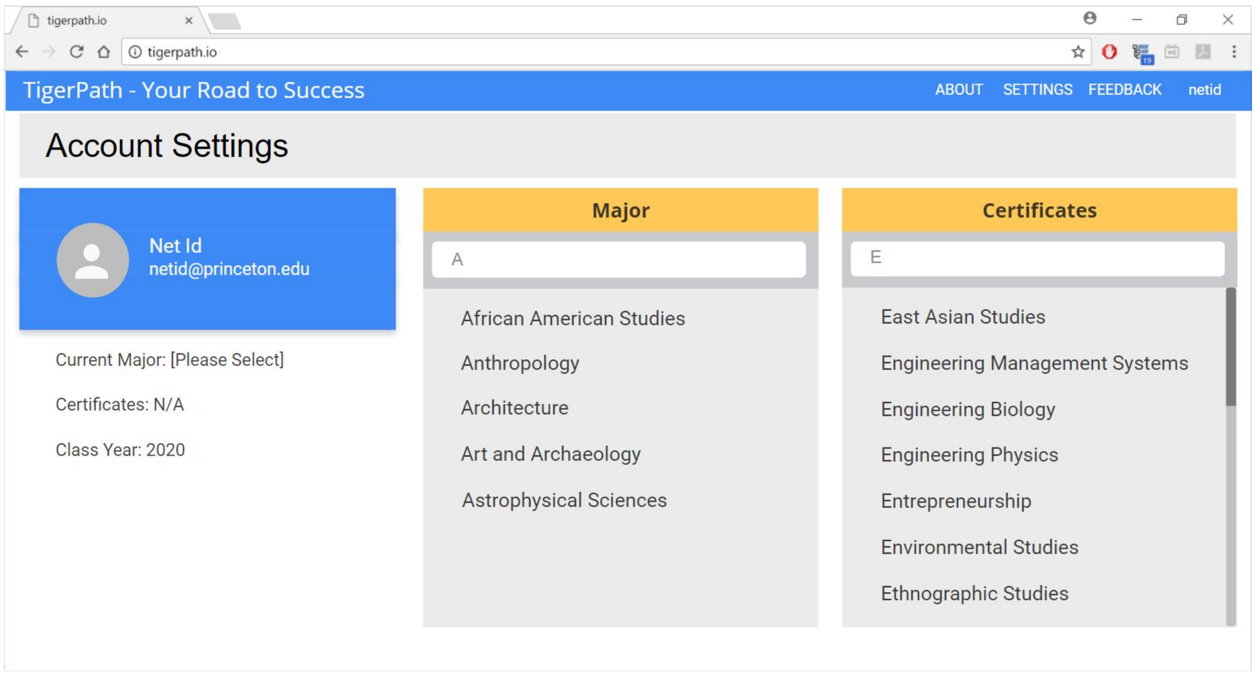
COS 320
COS 343
COS 375 / ELE 375
COS 398
COS 424 / SML302
MAT 375 / COS 342
TRA 301 / COS 401
COS 351 / SOC 353

Fall '16	Spring '17	Fall '17	Spring '18
COS 126	COS 226	COS 217	COS 333
MAT 201	MAT 202	ECO 310	MAT 340
WRI 137	FRS 168	ORF 245	
PHY 103	DAN 322	LIN 201	
	PHY 104	COS 424	

Fall '18	Spring '19	Fall '19	Spring '20
ELE 206	ELE 306	COS 432	
	ATL 494	COS 326	

This page will present most of our application's features. The user can search for classes on the left pane and drag and drop classes to certain semesters. The requirements are tracked at the top of the page. Each bar represents a requirement for the major, and it displays the number of satisfying courses the user already has in their schedule over the total number of courses they need to fulfill the requirement. If the user has not started completing a certain requirement, the bar will be red. If they have started the requirement but have not completed it, the bar will be orange. If they have completely satisfied the requirement, the bar will be green.

The following page represents the "Settings" page that the user can navigate to:



Account Settings

Net Id
netid@princeton.edu

Current Major: [Please Select]

Certificates: N/A

Class Year: 2020

Major

A

African American Studies
Anthropology
Architecture
Art and Archaeology
Astrophysical Sciences

Certificates

E

East Asian Studies
Engineering Management Systems
Engineering Biology
Engineering Physics
Entrepreneurship
Environmental Studies
Ethnographic Studies

On this page, the user will be able to see the current major and certificates they are currently tracking, and can change them if they so choose.

Database

We will be using MongoDB, a NoSQL database, to store all of our data, because of its flexibility and speed over its SQL counterparts. We will be using mLab to host our MongoDB database.

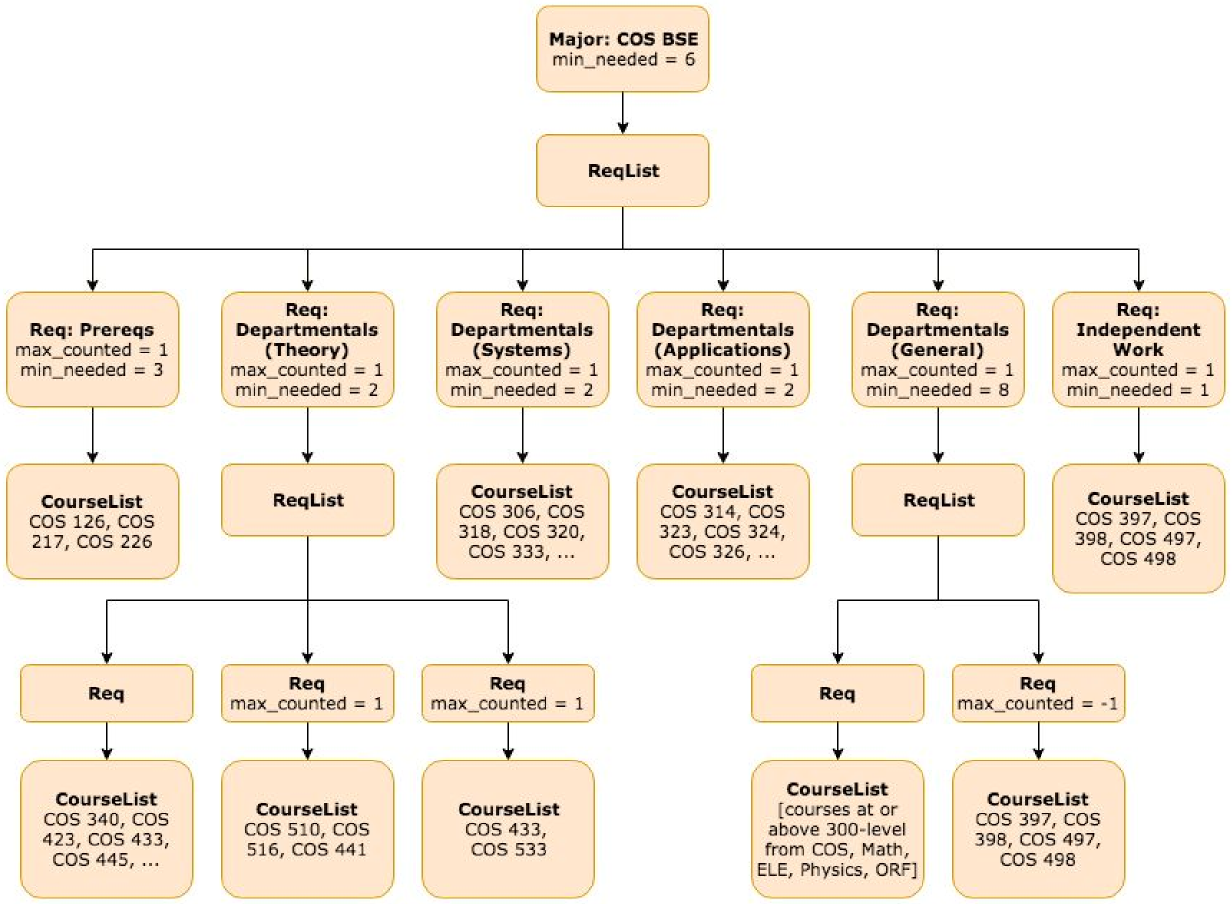
We believe that storing the major requirements data in the database will be a significant hurdle in completing our project; thus, we have outlined in detail how we are currently planning on doing it. We are creating a tree data structure for each major that will encode the required courses that a student would need to take. The data structure has the following grammar:

```
Major = Req
Req = min_needed, max_counted, (ReqList | CourseList)
```

```
ReqList = [Req, ...]
CourseList = [Course, ...]
```

Where `min_needed` is the minimum number of courses/subrequirements that need to be taken/satisfied for the requirement to be satisfied, and `max_counted` is the maximum number of courses/subrequirements that can be passed up the tree. If `max_counted` is not specified, then any number can be counted; if it is negative, then the number counted is the sum of the number taken/satisfied with `max_counted` (so for instance, "`max_counted = -2`" signifies that two less than the number taken/satisfied are passed up). If `min_needed` is not specified, then there is no minimum on the number of courses that are needed.

Here is an example of what such a tree would look like for COS BSE (this doesn't include the general BSE requirements, which would have its own separate tree):



In order for all of the requirements for the COS BSE `Major` to be satisfied, all `min_needed=6` of the `Req`s in its `ReqList` need to be satisfied. If we look at the Prereqs `Req` as an example, we see that we need `min_needed=3` of the courses in its `CourseList` to be satisfied in order for `max_counted=1` to be propagated up and be counted in the `min_needed=6` for the `Major`. This is basically how the entire tree should be interpreted.

We foresee that there may be some problems with this data structure. For instance, there is no way to represent a course that can only be counted for one requirement but not another. There may be other requirements that cannot be represented here; we're planning to revise this as necessary as we come across more erratic requirements from other majors.

We would store course data by semester for the past 4+ years. For each semester, we would have a separate entry for each course, and the attributes for the course would be stored within this entry. The attributes for each course could include the title, the description, the department and course number, and the times that the course meets at.

In order to store the user data, we would have an entry for each user with various attributes that pertain to user-specific data, such as the NetID, the major, the class year, and the courses that the user has put into each semester.

V. Timeline

March 18th:
- Design document submitted

March 25th:
- Start a Git repository
- Set up project website and link it to tigerpath.io
- Develop elevator pitch
- Server-side: Set up a Django app and connect it to the database
- Hosting: Host the app on heroku
- Client-side: Start working on searching for courses and adding them to different semesters
- Database: Start parsing course data and storing it in the database
- Database: Start parsing major requirements data and storing it in the database

April 1st:
- Database: Have all of the course data stored in the database
- Database: Have the requirements for our selected majors stored in the database
- Client-side: Setup CAS authentication
- Server-side: GET /api/courses is finished
- Client-side: Finish being able to search for courses and drag-and-drop them into semesters

April 8th:
- Server-side: GET /api/require/[USER_MAJOR] is finished

- Client-side: Checks the user four-year schedule with the requirements to see which requirements have been satisfied and which have not

April 13th: Prototype Finished

- By this time, we should have basic functionality of the app, including the following:
    - User can search for courses and add them to their four-year schedule
    - User can check what requirements are satisfied by their schedule, and what requirements have yet to be satisfied
    - User data is not yet stored in the database
- Start propagating the app to friends for them to test

April 15th:

- Server-side: GET /api/user/plan, POST /api/user/plan, GET /api/user/meta, POST /api/user/meta are finished
- Client-side: Store the user plan in the database and retrieve it during new sessions, retrieve/store user metadata

April 22nd:

- Clean up UI and make it look nicer, if it doesn't already look good
- Let the user customize requirements that they have fulfilled through other means (such as study abroad)
- Inform the user of when courses are typically offered (such as how frequently the course is offered and which semester is it typically offered in)
- Start putting in requirements for other majors

April 27th: Alpha test

- By this time, we should have full basic functionality of the app, including the following:
    - Being able to retrieve saved user data during a later session
    - Having a presentable UI
    - Requirements customization and course time predictions
- Continue propagating app for people to try out

April 29th:

- Continue putting in requirements for other majors
- Work on other additional features
- Start working on presentation/final documentation

May 3rd: Beta Test

- The app is fully usable and useful to a user that would want to use it
- Do a beta test release to the Princeton community

May 9th, 10th: Demo Days

- Finish presentation

May 13th: Project Due

- Finish final documentation

VI. Risks and Outcomes

We requested to get data from the registrar about what requirements need to be fulfilled for each major, but unfortunately our request was denied. Therefore, we have to manually gather data about the major requirements from what is posted online on the official Princeton department websites. This poses a risk because we have to spend time gathering this data, and because we need to make sure that the requirements we maintain are both complete and up-to-date. In order to ameliorate this risk, we've decided to only gather requirements for a few majors at first (namely, the BSE majors, Computer Science - AB, the Woodrow Wilson School of Public and International Affairs, Economics, History, Spanish and Portuguese, and Molecular Biology), which will allow us to focus on building the actual application. Once we have this initial set of majors added and working, we can add other majors later on.

For our project, we are planning on creating a web application using several different technologies, including Django. There are several people on our team who have not had experience building web apps before, and none of us have experience using Django. Therefore, in addition to the problems we have with data acquisition and building the application itself, we also need to learn the frameworks that we are using. This may take a bit of startup time, but we anticipate being able to learn it eventually; however, if it ends up taking too long, we could simply switch to a different framework that we've either used before or that is easier to learn.

There is also a risk involved regarding the maintainability of our project. During the start of each semester, when new courses are added, we need to manually update the course data that we have in our application. We also need to manually modify the major requirements whenever they are updated, which we hope will not be too often. We can probably build some automatic tools that will help us accomplish these tasks, such as an automated course data scraper and a tool that outputs the difference between the current version of a major requirements webpage with a saved version of the webpage, so that we can more easily find the changes and update with just those changes.