PURVIEW – FINAL REPORT
A Team with an Idea

Having been roommates for two years, we had decided to work together on the final project early in the semester. We opted to stick with just the three of us despite that this meant greater individual workloads. However, we felt this was a cost worth paying for a greater degree of coordination and communication efficiency. Indeed, working with people living in the same room made it incredibly easy to coordinate coding sessions, assist each other when bugs surfaced and find convenient times for team meetings for future planning.

Figuring out what product to build was not as easy. We iterated over a plethora of ideas, including an efficient Eating Club Web Portal, a mobile app for outsourcing daily chores, and an online platform to streamline the student group recruitment process. However, we wanted to build something that would be widely used and had the potential to go big. For this reason, we decided not to settle with any of the above.

We ultimately came across the idea of Purview, an anonymous geolocation-based video sharing platform. Immediately, we unanimously decided this was the project to go for. It felt right for all of us and this resulted in a very enjoyable development process.

We determined our stack, data needs, the use cases and process flows of Purview and the core back-end entities the app would build upon. After getting them approved by Professor Kernighan, we submitted our design document and set-up our status page\(^1\) and Git repositories. We were ready for development. “If only we would turn on our computers and find the app fully coded and ready for launch.” was the running joke in our team at the time.

**Lessons learned:**

- Work with people you enjoy being with. You will spend long hours together and being friends with them makes the process much more enjoyable.
- Never work on something that you are not willing to put your heart and soul into it. You will build something notable only if you have a passion for what you are building.
- Start early; Figuring out what to build, how to build it and how to name it takes time.

\(^1\) https://purview-333.github.io/
Initial Planning

Woody Allen once said that a good way to make god laugh is to tell him about your plans. A good way to make university students laugh is to attempt to convince them to not put off a six week long project until the last moment.

Making a design document and timeline are certainly not the sexiest parts of development. The three of us were raring to bust out of the stables and cowboy code the next best thing, but in our way stood the daunting paperwork of planning. We were forced to consider milestones, alpha testing, use cases, and other icky details normally left to the managerial suite. Looking back at the design document we created, it appears that we bravely responded to this challenge; the document itself is about 11 pages long. The timeline, although detailed, reeks of naive eagerness; according to it, all of the main features were supposed to be completed one month before the final due date. However, with great foresight we included 2 weeks of slippage time which came in handy. By the time of writing this, we have met all the milestones described in our timeline and even implemented additional features such as thumbnails, score, and partial profile pages.

Lessons Learned:

- Whether the timeline and design document kept us on track is a grey area, but we cannot deny that composing them led to productive discussions about the goals, nuances and specifics of our application. In the words of Shia LaBeouf: just do it.
Technology Choices, Prototyping and Early Development

Very early into the project, we agreed to use Python for the backend, MySQL for the database and React Native for the frontend. These choices were a function of our prior experience (all of us had some Python experience) and portability (React Native’s promise of write once, run everywhere for mobile). A key decision was to split into frontend and backend developers. Matthew, having prior experience with React, took on the task of developing the client, and Gordon, extremely terrified of anything remotely artistically/design-oriented, committed himself to working on the backend. Yannis wanted to learn more backend technologies and so chose to work with Gordon on the server. In addition to this division of labor, we organized the codebase into two separate repositories, one for the backend and one for the frontend. This approach prevented any namespace cross-pollution between the two systems and ensured frontend changes would not break backend functionality and vice versa.

In terms of development methodology, we employed a combination of ticketing, code review and testing to crank out features. We tracked our progress via the ticketing system Trello; each feature or bug would have a ticket associated with it, a label designating which team should handle it, and a relative priority. Anytime one of us were free to work, they took a ticket off Trello and created a corresponding branch in our version control system. After dev work on that ticket was complete (including writing additional tests that would have failed had the code change not been implemented), we would request code review. This process worked extremely well as testing and code review caught many bugs and stymied poor coding practices (perhaps one of the few positive instances of peer pressure). Moreover, an ever-growing suite of tests ensured we were not regressing when adding new features or updating existing ones.

By the end of spring break, we were still on track with our timeline. On the backend, we provisioned the tools needed for developing with our web framework, Flask, set up end-to-end connectivity for all the API endpoints, and completely implemented the user creation endpoint. On the frontend, we deployed the project status page. Unfortunately, after classes started up again, development slowed down to a crawl as Princeton began doing what it does best: adulterating the youthful moments of students by filling them with PSETs, exams, and anxiety.
Core Feature Development and Alpha Version

Post-prototype we focused on the features that were naturally extending from the core entities of our prototype: the Voting System and Tagging/Tags Search. Having developed a good amount together for the prototype, this stage of development was particularly smooth. Everybody was aware of the agreed coding conventions and processes, everybody was properly using Github, code review had been well-integrated into the development process and unit testing of new modules was never skipped.

Alpha Testing was a particularly eye-opening experience. We had five testers test Purview, who we interviewed during and post testing. What we noticed was that our testers not only identified bugs we were not aware of, but also had a much different view of how specific features should work. Therefore, although we were praised for the UI Matthew had developed, we had to significantly adjust a lot of the features to better match user expectations and fix a black screen bug that would torture us for the week to come.

Unfortunately, we were not moving as fast as we desired, probably because other classes also required a significant amount of work at the time. This was the point where we felt we should have spent more time developing over spring break. Due to this time constraint, although we had decided early on to stick with Digital Ocean for hosting and although we successfully setup our remote server and MySQL database, the remote server had not been properly integrated with the front-end side at that point. This resulted in us having a fully functioning Alpha Version of Purview but with a locally hosted server.

Lessons learned:

- Unit Testing and Code Review are the bread and butter of a bug-free app. Establish your testing suite and get used code reviewing before you submit code early on. The returns of adhering to these practices are exponential!
- Work during spring break. If you do not, you will feel the consequences of it later.
Beta Testing and Publishing

After we wrapped up the core features of our application, it was time for the beta testing portion of the project. To accomplish this, we obtained the necessary development certifications, and submitted an archive of our Xcode project to Itunes Connect. Up to this point, we had only run the application on Matthew’s phone or on simulator, which is only a limited taste of how the app will actually perform ‘in the real world’. Simply uploading the build to Itunes Connect allowed us to install Purview on all developers’ devices; testing on Gordon’s iPhone 5 revealed some performance limitations and screen variation on different device generations. However, our goal was to have people outside of our team be able to try it too: this led us to Apple’s external beta testing feature.

Before our app could be distributed externally for beta testing, our build needed to pass some basic review criteria from Itunes, a process that usually takes about one business day. Our first build was rejected for the following reason: the app failed to load. It turns out that a server error had coincided almost exactly with their choice of review timing. After fixing this error and submitting again, we were finally approved for beta testing. Over the next few days, we sent out 36 invitations by email to various friends with iPhones, and accumulated 26 installations, with over 100 sessions among these users. With these users, Itunes connect could track basic session info, such as when and how many times the app would crash. This, in addition to manual feedback sent by most beta testers led to some more minor improvements for the application. The beta testing process also contributed to populating the content on our live video feed and let us evaluate our server performance under higher stress.

While submitting our app for approval for beta testing, we simultaneously submitted it for the App Store in hopes of publishing our current version of the app. However, despite being approved for beta, the build was rejected for official release due to more stringent measures on user safety regarding user-generated content. Our plan is to resolve these issues with a better reporting system, put together a terms of service document, and release the app officially at a later time.
The Future of Purview

It is May 14, 2017. Submission date for the COS 333 final project. This date signifies the shut down for many projects and the beginning of the end for many others that will be ultimately dropped at some point during the summer. We know the former will not be the case for Purview. We hope the same will hold for the latter.

Purview is an application that requires a wide user base over a small geographical area at which events that incentivize video postings regularly occur. University campuses are settings perfectly meeting this criterion. However, most students have already left their Universities and soon the rest will follow. This means that these event hubs that constitute perfect target groups for Purview will not be as prominent during the summer season.

For this reason, we decided to hold up and not release Purview yet. We feel we have a better chance of having Purview be successful if we launch and promote it in September, when all the students are back on campus. Timing is crucial and we want to make sure we push Purview at the time when we will have the greatest target user-base in our scope.

So, what is in store for the summer? First of all, we will continue with iterative beta testing to ensure our current features offer a smooth experience. We already have 21 beta testers who have committed to keep doing so over the summer.

Secondly, based on the feedback we have collected so far and the research we have performed ourselves, we have already accumulated a list of new features to implement over the summer. This includes Video Filters, Video Zoom-In/Out, “Location Teleporting”, Recently Liked Videos Feed and Password Retrieval. This list will be constantly reviewed and expanded as we come up with more ideas and receive feedback from our testers.

Our end-goal is to make our application Android-compatible and ready to launch by the beginning of September. We might get more people on board if needed, but for now we feel confident in our powers to achieve this goal. The clock is already ticking. It is now on us to make the most of the time we have at our disposal.
Words to next year’s 333 class

1. **Write code that you would not mind seeing again.** In COS 126, 226, and 217, we often finish an assignment in one or two sittings (bugs notwithstanding) and therefore do not have to routinely reread and maintain the code we write. For this project, we often had to work with code we wrote at the very beginning and build features on top of it over a span 6 weeks. Any deficiencies in the initial code would only have cascading effect and worsen future code.

2. **Keep your testing code modular.** The real advice here is to have automated testing, but Professor Kernighan should have belabored this point to an extent that you would prefer to talk to a Jehovah’s Witness instead of listen to the merits of testing one more time. So, the next best thing we can say is to make tests pleasant to write. Refactor common low-level actions such as registering a user into clean helper functions that promote spicy one-liners.

3. **Integration is hard.** So, we wrote tests for the backend, and they all pass. The frontend works as expected with dummy data. Suddenly, excrement hits the fan when we try to put the backend and frontend together even though they work in isolation. What gives? Even if you split your team into frontend and backend divisions, set aside time to debug important features together.

4. **Don’t be scared of pitching.** Don’t tell anyone but we managed to not get called on to pitch for the entirety of the semester.