1. End-User Guide

1.0 Purpose

Purview is an anonymous public content-sharing platform that allows users to share short videos with everyone around them. Purview integrates the anonymous geolocation-based feed of Yik Yak and an extended version of the ‘Stories’ feature of Snapchat to offer university students and passionate video-makers a fast and fun way to share their experiences with everyone near them. It is the first video sharing platform allowing users to anonymously share content and quickly catch up with what is happening in their surrounding area by viewing through the eyes of their peers near them.

1.1 Installation

After release, one can simply search for ‘Purview’ on the App Store, and download the application using one’s Apple ID, as one would do for any other iOS app. For beta-phase installation, contact our team to receive a beta invite via email. Upon receiving the beta invite, simply install the app ‘Test Flight’ from the App store, and redeem the code sent from the beta invite; finish by installing Purview.

1.2 Login and Registration

To create an account, simply enter your username, and a valid password (at least 6 characters, 1 number, 1 punctuation mark), and press the ‘Sign Up’ button.

To login to an existing account, enter the username and password used at registration, and press the ‘Log In’ button.

1.3 User Settings

To access user settings, click on the profile icon (rightmost) in the tab bar navigator. On the page, you can see your current score (top bar), as well as your current active videos and recently liked videos (coming soon). At the bottom, you can change your password, logout, or delete your account.

1.4 Home Page

1.4.1 Information Bar

The top bar of the screen is the information bar; on the very left is the user’s current score. Score is increased by voting on videos, as well as receiving votes on videos that you post. A rewards system based on this score will be implemented shortly. On the very right are buttons to sort the video feed. The default (left) is sort by recent videos, and the ‘fire’ button on the right is sort by video score. Clicking on either button will refresh the feed and reorder the feed based on the specified criterion. Clicking on the Purview logo will scroll the user back to the top.
1.4.2 Video Feed

The video feed is comprised of a list of all videos posted within 2.5 km of your current location within the last 3 days. Each video contains several smaller components. The first is the video itself, located in the slightly rounded square with red outline. The video’s sound can be toggled on and off by tapping on the video, or by pressing the mute button. A double tap on the video switches to an expanded view. For full-screen, press the full screen button on the bottom right; this view allows the user to pause the video, and control playback directly.

Below the video are a series of red buttons with text on them: these are the tags of the video that the poster has attributed to it, which can be used to describe the contents of the video itself. Clicking on these tags will submit a search query for that tag (discussed in section 1.5). Right below the tags is an indication of when the video was posted.

To the very right is the score of video, as well as buttons that allow the user to vote on the video. To cancel your vote, click on your previous vote again. Your current vote is indicated by whichever button is highlighted with the red color. To view additional videos, simply scroll down through the feed, and the app will make requests for new videos as necessary. To refresh the feed to fetch new videos, simply navigate to the top of the feed, and drag down until the refreshing activity indicator becomes visible.

1.4.3 Navigation

The main navigation of the application is controlled by the tab bar at the bottom of the screen. The first tab opened after logging in or signing up is the home page described above. To navigate to other tabs, simply press on their icons in the tab bar. The current active tab is indicated by the highlighted icon. The tabs, in order, are: home, search, post, hall of fame, and user settings / profile.

1.5 Search Page

The second tab is the search page, where users can search for specific videos within their overall feed by tag. For example, typing ‘Princeton’ as the search query populates the feed with all nearby and recent videos with the ‘Princeton’ tag.

Just as in the home page feed, users can drag down to refresh, and sort the results by either recent or popularity. In the top bar, the score indicator is replaced by a back button, which returns the user to the home page. Users interact and vote with videos as normal on this page; pressing on a tag will simply submit a new query without any redirection.

pg. 2
1.6 Hall of Fame

The fourth tab in the tab bar is the hall of fame. The hall of fame tracks the top 10 videos of all time, ranked by overall score. Whereas videos found in the home feed are location-based and expire after three days, videos in the hall of fame can be accessed globally, and never expire. Although the user can control the video normally via toggling sound and full screen buttons, the videos in the hall of fame cannot be further voted on.

On the hall of fame feed, videos have an extra indicator in the top left corner showing what its current overall rank is within the hall of fame. The point-based rewards system will also contain rewards for owning videos both currently and previously in the hall of fame that vary based on its peak rank within the hall of fame.

1.7 Posting a Video

To post a video, click on the middle tab on the tab bar. The page will immediately display the camera’s current contents. The camera can be toggled between front and back-facing by pressing the flip button in the upper right hand corner. To record, press and hold the record button; a red indicator (second image) will fill up the circle as the video approaches the 10 seconds maximum duration.

After 10s, or if the record button is released, the flow transitions to the video confirmation page (third image), where the user can review the capture video, cancel by pressing the ‘x’ button, or continue by pressing the red arrow. In the last screen (fourth image), the user can attach up to 10 tags to the video, by either typing in custom text or pressing the suggested tags. Tags can be removed from the video by tapping the tag to be removed. To confirm and post, the user presses the check button which will be redirect them to the home page.
2. Developer Guide

The application is split up into two separate systems: the Purview server and the Purview client. The server is implemented using Python 2.7 and MySQL 14.14 and is hosted on an Ubuntu 14.04 Digital Ocean Droplet. The client is implemented using react native and has been tested on iOS 10.

2.1 Server Guide

2.1.a Setting up - Getting the server on your local development environment

After cloning the back-end git repo to your local, establish a virtual environment with virtualenv. Activate the virtual environment and install the dependencies. Then, initialize the SQLite database used for testing on your local using the manage.py executable located on the top-level directory. Cd into ftp and ensure there is a ‘videos’ directory and, within that, a ‘thumbnails’ and ‘hof’ directory. If these are not there, create them before starting development. The ftpserver.py executable in the ftp folder is used to run the ftp server. Run the ftp server and you are ready for development!

2.1.b.i Technology stack

The server stack builds on Python 2.7 and MySQL. The web server was implemented with Flask. Flask’s main use in the application is for automated dispatching of HTTP method requests to the appropriate function handlers. In addition to Flask, SQLAlchemy is used by the application to interact with the database layer. SQLAlchemy enables the definition of the SQL schema and query construction in a Pythonic syntax. It also performs input sanitization to mitigate SQL injection.

2.1.c Remote server configuration

The server is hosted on a Digital Ocean Droplet deployed with Apache. This setup was preferred due to the high-quality documentation and low pricing of Digital Ocean. The Droplet runs on Ubuntu 14.04.5, has 512MB RAM and 20GB storage and can serve over both IPv4 and IPv6 protocols.

Although currently these configurations are more than enough for beta testing, the current model is not scalable, as 20GB of storage can store at best 500 videos. Therefore, upon release we will need to upgrade our server to one with 40 or 60 GB storage 1024 or 2048 MB RAM to improve the rate at which our server handles request from the client.

2.1.d.i Directory structure

The server directory structure is simple yet allows for a range of features and extensibility. Below is a summary of the major files and directories.

2.1.d.i.a /app

The web server code is contained primarily under the /app directory. For each module, there is a folder under the app directory (for now, the API is the only module and can be found at /app/mod_api). Inside each module there is a models.py file describing the database schema for the entities needed by that module. api.py is a routing file that maps endpoints to handler functions. In the API module, the handler functions represent RESTful resources and are stored
under /resources. The tree output on the right shows the main resources provided by the API module: auth (authentication), hof (hall of fame), users, and videos.

2.1.d.i.b /ftp
The FTP server is responsible for storing and serving video data. After receiving a video upload request, the web server establishes a connection with the FTP server and pushes the video to it. A similar process occurs during a video retrieval request, except the web server retrieves a video from the FTP server. On the FTP server, videos reside under /videos. Videos which have made it to the hall of fame are stored under /videos/hof.

2.1.d.i.c config.py
The config.py file defines the possible configurations the server can be run with, namely, testing and production. Important application-wide variables are defined here, such as the number of threads which the server runs on and how many rounds user passwords are hashed before they are stored in the database.

2.1.d.i.d manage.py, run.py
The manage.py file is used for database initiation and migration. The run.py file starts the web server listening on port 80.

2.1.d.ii.a SQL Schema

<table>
<thead>
<tr>
<th>Table</th>
<th>Purpose</th>
<th>Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>Stores username and password hash</td>
<td>Videos (one-to-many)</td>
</tr>
<tr>
<td>videos</td>
<td>Stores filepath of video (used to fetch video from FTP server)</td>
<td>Users, (many-to-one), Votes (one-to-many), tags (many-to-many)</td>
</tr>
<tr>
<td>tags</td>
<td>Stores tags which have been put on videos by users</td>
<td>Videos (many-to-many)</td>
</tr>
<tr>
<td>votes</td>
<td>Stores votes on a per video basis</td>
<td>Videos (many-to-one)</td>
</tr>
<tr>
<td>hall_of_fame</td>
<td>Stores same information as video table, but used for hall of fame feature</td>
<td></td>
</tr>
<tr>
<td>blacklisted_tokens</td>
<td>Stores auth tokens which are no longer valid</td>
<td></td>
</tr>
</tbody>
</table>

2.1.e Cronjobs
Because videos expire every three days, we need to update the database and file system to reflect those changes. Similarly, the hall of fame needs to be updated every time a set of videos is deleted with the most upvoted expired videos of all time. To do this, we setup a cronjob on the server that performs the afore-mentioned operations.
2.1.f Testing
Tests are implemented via the python unittest framework. There is an autotest.sh script in the top-level directory of the repository that will run all the tests using the testing configuration of the server. As of writing this guide, there are over 81 tests written that should all pass upon successful installation of the server code. There is a rich set of helper functions for common actions such as registering a user, posting a video, voting on a video, etc. to keep the testing code DRY.

2.1.g Logging
Because testing may not always catch all the bugs that can occur when the server serves the client side, logging is necessary to detect undesired behavior. Specifically, we currently log all the requests made to the server and the output of our cronjobs to ensure they are running properly and with the proper frequency. Most importantly, whenever we deploy a new version of the server, we store our log files to create a running record of the server and the cronjobs to fall back on in case similar errors to the ones we have already tackled arise in the future.

2.2 Client Guide

2.2.1 Technologies Used
The front end is built with React Native, which is an open source, Javascript code library developed by Facebook and Instagram that allows developers to build mobile applications cross-platform, and near-natively. Apart from understanding Javascript fundamentals, developers should be familiar with JSX, a syntax for embedding XML within Javascript that behaves quite similarly to HTML. Xcode and a functioning simulator or iOS mobile device is also necessary to run the application.

2.2.2 Page Structure and Navigation
In the main directory of the application there is the file ‘index.ios.js’ that is, as the name suggests, the index of the application, within which all subcomponents are rendered. This index component controls the highest level navigational flow that occurs between loading, logging in, and the content of the application.

The rest of the screens of the application are contained in the folder ‘screens’. These are entire pages of the application between which page navigation occurs. The very first page is ‘LoadingPage.js’, which simply renders a splash screen, and sends a GET request to the ‘Auth/Status’ endpoint to check if the user is currently logged in. If so, it navigates directly to the content part of the application, and if not, to ‘LoginPage.js’. The login page renders basic text fields and buttons that allow the user to enter an email and password,
and then to log in and sign up as desired, which then send the appropriate requests to the server. After this is successful, the navigator transitions to the content pages.

The content pages are described by the file ‘ContentContainer.js’. Note that the name of this component is not a ‘Page’, but a ‘Container’. This is because this component has its own navigational flow, and contains all other pages that are considered content: FeedPage, PostPage, and ProfilePage. Navigation between these content pages is controlled by the tab bar at the bottom of each screen, as defined by the ScrollableTabView component. The ContentContainer also keeps track of all videos currently loaded, and other variables, such as current sort-by and user score, within its state. These state variables are then passed down to subcomponents, along with functions to modify them; essentially, Content Container acts as the primary information source.

The primary content page is FeedPage, which is responsible for not only the home page, but also the search page and hall of fame pages. This abstraction can be made efficiently since all three pages are simply video feeds with an information bar on the top. The differing behaviors of each page are achieved by simply passing as a property to FeedPage which kind of page it should be. This property is fed down to smaller components, that conditionally render different things based off this property. But in general, this component renders a TitleBar component, sometimes a SearchBar, but always a ListView of Video components that comprise the feed. A ListView was chosen over a ScrollView due to performance issues.

The next main page is PostPage.js. The post page, as described in the end-user guide, has three different subscreens: ready to record / recording, confirmation, and posting. Transition between these sub-screens is managed by the state of the PostPage component, and change based on user interaction with various buttons. The first screen renders a Camera component, which starts and stops capturing based off a PanResponder with defined behavior for touch and release.

Video capture is also limited by time: a setTimeout function is called at the start of video capture, which automatically ends recording and transitions to the next sub-screen at the end of ten seconds. At this screen, a Video component is rendered using the video data that the user just recorded. At the next screen, this same video is rendered in addition to a text field, and numerous Tag buttons that allow the user to attribute tags to the video. The last page is ProfilePage.js, which currently consists of a simply information bar, and many simple buttons that allow for user setting changes.

To make any changes to the overall navigational navigation, modify ‘index.ios.js’. To modify content specific navigation, modify ‘ContentContainer.js’. To change what is visible on a given screen, go to the corresponding page in the screens folder, and modify what is returned in the render function. In terms of styling, each component defines the styles for all of its subcomponents. In addition, there is also a global theme object defined in index.js that defines common colors for the application, which allows for easy thematic changes across multiple component layers.

### 2.2.3 Component Layout

The contents of each page can be described entirely within the given file, but some pages
contain smaller components that are either quite complex, or repeated multiple times. Such components are abstracted into their files, located in the ‘/components’ directory of the project.

The component TabBar.js, as outlined in purple, defines how the tab bar looks when rendered in the ContentContainer. This component is fed into ScrollableTabView as a property, which then integrates this property with its pre-defined behaviors. The other components are used primarily within the FeedPage screens. The largest one of these is VideoFeedContainer.js, of which each FeedPage component has exactly one. Each VideoFeedContainer contains a TitleBar, outlined in green, at the very top. This title bar, as described earlier, changes based off a feedType property, passed in by its parent.

The VideoFeedContainer is responsible for telling the FeedPage the status of getting and fetching videos: if it is doing its initial loading, currently refreshing, or loading additional videos. It is within this component that almost all requests to the server regarding videos are defined. For example, the fetchFeedVideos function makes a GET request to the appropriate endpoint, and populates a fetchedVideoData variable in its state. Next, a fillFeed function is called to fetch the actual video file for each video that should be rendered up to the current user scroll limit, and send all its data to the ContentContainer for rendering. Lastly, the component also defines behavior for refreshing feed, as well as score and voting related server requests.

The last main component is the ‘VideoContainer.js’, which defines both how a video should be displayed given all of its metadata, as well as how it should change based off of user interaction with the video. Each VideoContainer obviously renders the actual Video component itself, based off of the video source. In addition, it renders a couple of buttons on top: mute and fullscreen. Each VideoContainer also has ‘shouldMute’ and ‘active’ properties, that allows its parents to control which videos should be rendered on screen at a time, and to override the current sound state.

2.3 Module Dependencies

Front-end:

- React-native-camera
- React-native-circular-progress
- React-native-fetch-blob
- React-native-scrollable-tab-view
- React-native-search-bar
- React-native-vector-icons
- React-native-video

Back-end:

- Flask
- Bcrypt
- WTForms
- SQLAlchemy
- Jonschema
- openCV
- Ftplib
- Pyjwt
- Requests
- Werkzeug

2.4 Acknowledgements

- Annette Chu ’20 designed the logo of Purview.
- Sotiris Apostolakis was the Project Advisor throughout the spring semester of 2017.